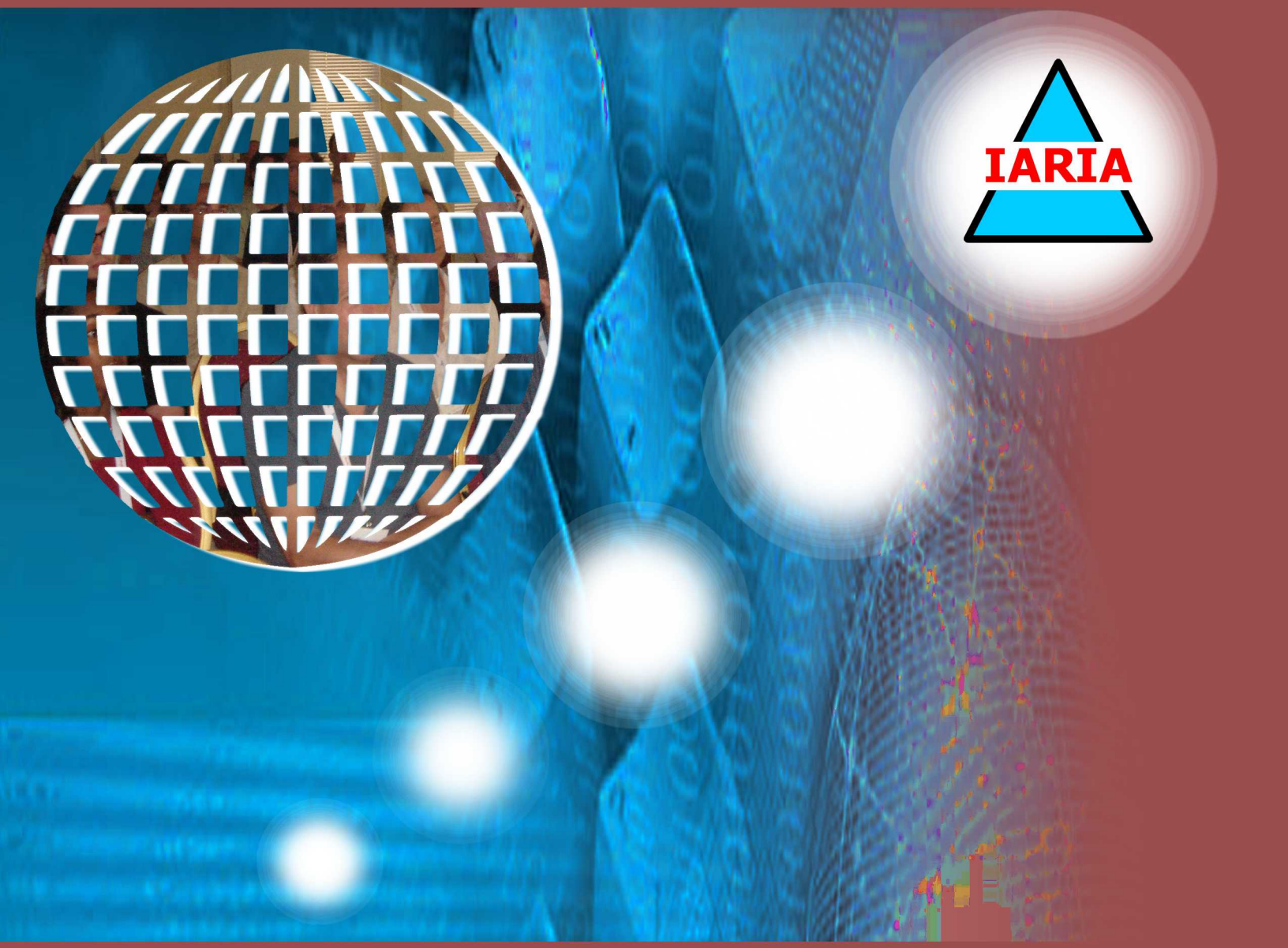
International Journal on Advances in Security



2013 vol. 6 nr. 1&2

The *International Journal on Advances in Security* is published by IARIA.
ISSN: 1942-2636
journals site: http://www.iariajournals.org
contact: petre@iaria.org

Sponsored by IARIA
www.iaria.org

International Journal on Advances in Security
Volume 6, Number 1 & 2, 2013

Shao-Ming Fei, Capital Normal University, Beijing, China
Eduardo B. Fernandez, Florida Atlantic University, USA
Anders Fongen, Norwegian Defense Research Establishment, Norway
Somchart Fugkeaw, Thai Digital ID Co., Ltd., Thailand
Steven Furnell, University of Plymouth, UK
Clemente Galdi, Universita' di Napoli "Federico II", Italy
Emiliano Garcia-Palacios, ECIT Institute at Queens University Belfast - Belfast, UK
Marco Genovese, Italian Metrological Institute (INRIM) -Torino, Italy
Birgit F. S. Gersbeck-Schierholz, Leibniz Universität Hannover, Certification Authority University of Hannover (UH-CA), Germany
Manuel Gil Pérez, University of Murcia, Spain
Karl M. Goeschka, Vienna University of Technology, Austria
Stefanos Gritzalis, University of the Aegean, Greece
Michael Grottke, University of Erlangen-Nuremberg, Germany
Ehud Gudes, Ben-Gurion University - Beer-Sheva, Israel
Indira R. Guzman, Trident University International, USA
Huong Ha, University of Newcastle, Singapore
Petr Hanáček, Brno University of Technology, Czech Republic
Gerhard Hancke, Royal Holloway / University of London, UK
Sami Harari, Institut des Sciences de l'Ingénieur de Toulon et du Var / Université du Sud Toulon Var, France
Dan Harkins, Aruba Networks, Inc., USA
Ragib Hasan, University of Alabama at Birmingham, USA
Masahito Hayashi, Nagoya University, Japan
Michael Hobbs, Deakin University, Australia
Neminath Hubballi, Infosys Labs Bangalore, India
Mariusz Jakubowski, Microsoft Research, USA
Ángel Jesús Varela Vaca, University of Seville, Spain
Ravi Jhawar, Università degli Studi di Milano, Italy
Dan Jiang, Philips Research Asia Shanghai, China
Georgios Kambourakis, University of the Aegean, Greece
Florian Kammueller, Middlesex University - London, UK
Sokratis K. Katsikas, University of Piraeus, Greece
Seah Boon Keong, MIMOS Berhad, Malaysia
Sylvia Kierkegaard, IAITL-International Association of IT Lawyers, Denmark
Marc-Olivier Killijian, LAAS-CNRS, France
Hyunsung Kim, Kyungil University, Korea
Ah-Lian Kor, Leeds Metropolitan University, UK
Evangelos Kranakis, Carleton University - Ottawa, Canada
Lam-for Kwok, City University of Hong Kong, Hong Kong
Jean-Francois Lalande, ENSI de Bourges, France
Gyungho Lee, Korea University, South Korea
Clement Leung, Hong Kong Baptist University, Kowloon, Hong Kong
Diego Liberati, Italian National Research Council, Italy
Giovanni Livraga, Università degli Studi di Milano, Italy
Gui Lu Long, Tsinghua University, China
Jia-Ning Luo, Ming Chuan University, Taiwan

Thomas Margoni, University of Western Ontario, Canada
Rivalino Matias Jr ., Federal University of Uberlandia, Brazil
Manuel Mazzara, UNU-IIST, Macau / Newcastle University, UK
Carla Merkle Westphall, Federal University of Santa Catarina (UFSC), Brazil
Ajaz H. Mir, National Institute of Technology, Srinagar, India
Jose Manuel Moya, Technical University of Madrid, Spain
Leonardo Mostarda, Middlesex University, UK
Jogesh K. Muppala, The Hong Kong University of Science and Technology, Hong Kong
Syed Naqvi, CETIC (Centre d'Excellence en Technologies de l'Information et de la Communication),Belgium
Sarmistha Neogy, Jadavpur University, India
Mats Neovius, Åbo Akademi University, Finland
Jason R.C. Nurse, University of Oxford, UK
Peter Parycek, Donau-Universität Krems, Austria
Konstantinos Patsakis, Rovira i Virgili University, Spain
João Paulo Barraca, University of Aveiro, Portugal
Juan C Pelaez, Defense Information Systems Agency, USA
Sergio Pozo Hidalgo, University of Seville, Spain
Vladimir Privman, Clarkson University, USA
Yong Man Ro, KAIST (Korea advanced Institute of Science and Technology), Korea
Rodrigo Roman Castro, Institute for Infocomm Research (Member of A*STAR), Singapore
Heiko Roßnagel, Fraunhofer Institute for Industrial Engineering IAO, Germany
Claus-Peter Rückemann, Leibniz Universität Hannover / Westfälische Wilhelms-Universität Münster / North-German Supercomputing Alliance, Germany
Antonio Ruiz Martinez, University of Murcia, Spain
Paul Sant, University of Bedfordshire, UK
Reijo Savola, VTT Technical Research Centre of Finland, Finland
Peter Schartner, University of Klagenfurt, Austria
Alireza Shameli Sendi, Ecole Polytechnique de Montreal, Canada
Dimitrios Serpanos, Univ. of Patras and ISI/RC ATHENA, Greece
Pedro Sousa, University of Minho, Portugal
George Spanoudakis, City University London, UK
Lars Strand, Nofas, Norway
Young-Joo Suh, Pohang University of Science and Technology (POSTECH), Korea
Jani Suomalainen, VTT Technical Research Centre of Finland, Finland
Enrico Thomae, Ruhr-University Bochum, Germany
Tony Thomas, Indian Institute of Information Technology and Management - Kerala, India
Panagiotis Trimintzios, ENISA, EU
Peter Tröger, Hasso Plattner Institute, University of Potsdam, Germany
Simon Tsang, Applied Communication Sciences, USA
Marco Vallini, Politecnico di Torino, Italy
Bruno Vavala, Carnegie Mellon University, USA
Mthulisi Velempini, North-West University, South Africa
Miroslav Velev, Aries Design Automation, USA
Salvador E. Venegas-Andraca, Tecnológico de Monterrey / Texia, SA de CV, Mexico
Szu-Chi Wang, National Cheng Kung University, Tainan City, Taiwan R.O.C.
Piyi Yang, University of Shanghai for Science and Technology, P. R. China

International Journal on Advances in Security
Volume 6, Numbers 1 & 2, 2013

CONTENTS

Maturing the Distribution of Supportive Tasks in Web Service Framework: Security and Reliability

Beytullah Yildiz
Department of Computer Engineering
TOBB Economics and Technology University
Ankara, Turkey
E-mail: byildiz@etu.edu.tr

***Abstract*—Security and reliability are the crucial features of distributed computing, of which one of the key enabling technologies is Web Service. In a service execution, the main task is carried out by endpoint logic, which is supported by additive functionalities and/or capabilities, called Web Service handlers. The handlers can be detached from the endpoint and distributed to suitable locations to improve availability, scalability, and performance. In this paper, security and reliability, which are among the most fundamental and essential requirements of the handler distribution, are investigated. The proposed environment contains a hybrid encryption scheme, digital signing, authentication, replication, and guaranteed message delivery. The benchmark results are presented to illustrate that the utilized reliability and security mechanisms for the handler distribution are reasonable and efficient.**

Keywords-Web Service; distributed computing; replication; reliability; security.

I. INTRODUCTION

Web Service is a technology providing seamless and loosely coupled interactions that help to build platform-independent distributed systems. Software standards and communication protocols offering common languages are at the foundation of Web Service. The strength of Web Service originates from its ability to hide platform-specific details of the implementations, to expose service interfaces, and to let these self-describing services be registered, published, and discovered dynamically across the Internet. Web Service utilizes the most basic distributed computing approach of client-server interaction. However, it also allows for the creating of complex service composed of many communicating services. Powerful Web Service applications can be assembled by combining the remote and local services.

A single Web Service application integrates endpoint logic and its handlers in a common framework. The main task is accomplished by the service endpoint logic. Supportive functionalities and capabilities, called Web Service handlers, are utilized to provide a full-fledged service. These capabilities might be related to security, reliability, orchestration, and logging, as well as any other necessary capabilities for a distributed system. These capabilities may help to compose a complex service as the handlers can deal with orchestration. They may also be utilized to provide high enough quality of services for a single client-server interaction as the handlers offer security or logging. A Web Service can employ several handlers in a single interaction; a chain of handlers can contribute to a service execution. A service can have a pair of handlers offering functions on both the server and client sides. Some handlers can be, in contrast, employed only in one side of the interaction.

Although handlers are required and inevitable in many cases, they may cause degradation in service quality if their numbers overload the service. A service endpoint with many handlers may suffocate in a single memory space. Hence, it is wise to use additional computing power. This raises the idea of distribution. There are different reasonable approaches for distribution of Web Service handlers. Some suggest that they can be deployed as services; others create a specific distributed environment for them. Distributing the handlers by using a designated setting provides a superior computing environment, especially when the concern is performance. On the other hand, the distribution requires certain features to ensure a suitable environment [1].

Security and reliability are among the most important criteria that need to be considered when a distributed system is being evaluated. Hence, this paper investigates reliability and message security for the distributed Web Service handlers and their effect on the system performance. A fundamental task of cryptography is to protect the secrecy of messages transmitted over public communication lines. For this purpose, in this research, an encryption scheme using a secret key is utilized to encode a message in such a way that an eavesdropper cannot make sense of it. To handle key exchange smoothly, the secret key is ciphered with a public key encryption algorithm. Moreover, a digital signature is used to verify the sender. Reliability mechanisms are also employed to attain a robust environment for Web Service handlers.

The rest of this paper is organized as follows: Section II provides information about related works on reliability and security. Distributed Web Service handler execution is briefly explained in Section III. Section IV investigates reliability. Section V gives details about message security. Finally, the paper is concluded in Section VI.

II. RELATED WORKS

Web Services, an ideal type of technology for distributed applications, benefit from several specifications for security and reliability purposes: WS-Security [2], WS-Trust [3], WS-Federation [4], and WS-ReliableMessaging [5]. These specifications provide the common language to develop secure, reliable, and interoperable interactions between clients and services.

WS-Security addresses security by leveraging existing standards and provides a framework to embed these mechanisms into a SOAP message. This happens in a transport-neutral fashion. WS-Security defines a SOAP header element, which contains the information defined by the XML signature that conveys how the message was signed, the key that was used, and the resulting signature value. Likewise, the encryption information is inserted into the SOAP header. WS-Trust explains the mechanisms to use security token and methods to establish trust relationships. It enables secure conversations between Web Services by defining how to issue, renew, and cancel the security tokens. WS-Federation defines mechanisms to let different security realms unite. Hence, authorized access to a resource handled in one realm can be provided to security principals whose identities are controlled in other realms.

In the Web Service reliability, the WS-ReliableMessaging specification offers an outline to ensure reliable message delivery between the sender and receiver for Web Services. The specification provides an acknowledgement-based scheme to guarantee that data are transferred between the communicating entities. Although it is mainly for point-to-point communication, the specification also supports service composition and transactional interaction.

Web Service specifications describe the syntax and do not define implementation mechanisms or APIs, which remain proprietary to individual vendors. Other than specifications, there are also works and research on security and reliability for Web services. Jayalath and Fernando describe a basic design and implementation approach for building security and reliability layers for Apache Axis 2 [6]. Moser et al. explain dependability features, including SOAP connection failover, replication, and checkpointing, in addition to reliable messaging and transaction management. Their paper also presents security technologies, including encryption and digital signatures for Web Services specifications, as well as other security technologies [7]. Pallemulle et al. present a middleware that supports interaction between replicated Web Services while providing strict fault isolation assurances [8]. Lei et al. propose greedy replica optimizers to improve reliability for a data-intensive online grid system [9]. Aghdaie and Tamir present a transparent mechanism that provides high reliability and availability for Web Services. Their paper explores fault tolerance even for the requests being processed at the time of server failure. The scheme can handle dynamic execution and does not enforce deterministic servers [10]. Zhang presents an integrated security framework based on the use of authentication, authorization, confidentiality, and integrity mechanisms for Web Services, and proposes a model to integrate and implement these security mechanisms in order to make Web Services robust [11]. Yamany et al. propose a metadata framework providing different levels to describe the available variations of the authentication, authorization, and privacy features. With the metadata, the security features are constructed to assist the service consumer and provider in reaching an agreement on how to meet their needs [12].

Security and reliability are not the concerns of only Web and Grid Service technologies. Other distributed computing applications also offer necessary reliability and security mechanisms. Fault tolerance approaches such as replication, recovery techniques, self-reconfiguration of systems, and dynamic binding are applied in various studies to improve reliability. Many research projects and applications utilize Secure Sockets Layer (SSL)/Transport Layer Security (TSL) protocols. Others prefer to utilize symmetric or asymmetric crypto systems. Some research projects offer a hybrid approach combining symmetric and asymmetric key encryption algorithms to offer superior solution.

Vaca et al. propose an automatic identification of faults by means of model-based diagnosis, which helps to establish particular fault tolerance mechanisms such as replications and checkpoints [13]. Zhao et al. design a scheduling algorithm offering reliability satisfying the user's requirement without exceeding the system capacity. The authors explain how to achieve the minimum number of replicas for each task while satisfying the user's reliability requirement with the minimum resources. They also target acceptable performance for execution time [14].

Desmedt et al. demonstrate a scheme that uses a public key to encrypt a random key, which is used to encrypt the actual message with a symmetric encryption algorithm [15]. Ramachandran et al. use a public/private key model for securely communicating messages [16]. Rizvi et al. present an implementation of a secure application syndicating symmetric and asymmetric key algorithms to minimize the execution time and maximize the security [17]. Ramaraj et al. describe a hybrid encryption based on AES and RSA for online transactions [18]. Palanisamy et al. propose the use of a symmetric key algorithm to encrypt and decrypt data and RSA for the symmetric key's encryption/decryption [19]. Damiani et al. discuss the applicability of outsourced Database Management System solutions to the cloud and provide an outline for management of confidential data in public clouds. It utilizes symmetric and asymmetric encryption for privacy and signing [20].

Kemathy et al. investigate a component-based security solution for XML messaging [21]. Ammari et al. provide architecture securing XML messages by encrypting flagged XML parts, each with a different type of encryption depending on data sensitivity and the defined importance level [22].

III. DISTRIBUTION

In this paper, the distribution of Web Service handlers is explored by utilizing a Message-Oriented Middleware (MOM), which is more narrowly focused on messaging. The MOM is designed to be as simple as possible while still offering robust support for messaging. Unlike SOAP messaging, MOM message headers and basic routing information are not contained in XML. This allows more efficient processing, since XML parsing is slow compared to the speed at which routing decisions are made in a specialized messaging system.

In general, the same proven security and reliability mechanisms from the related works are utilized. However, the present research differs in the use of a designated messaging system to improve efficiency and in having an end-to-end solution to complete the distributed handlers' secure and reliable execution in this setting.

The distributed handler execution is organized by a specialized management tool, which contains the orchestration engine explored in [23]. The constructs of the orchestration engine answer the wide range of the handler's execution configuration, such as serial, parallel, and conditional processing. In addition to orchestration, the distribution manager employs an efficient execution engine to meet the performance requirements. The details of the manager are provided in [24]. The engine utilizes a MOM to distribute the tasks to the handlers. The execution manager is so efficient that the overhead justifies the distribution, as investigated in [25]. This paper extends the required security and reliability mechanisms for the handler distribution explained in [1].

The execution of a message in the distributed environment is shown in Figure 1. The incoming requests to the Web Service are delivered to the distributed handler manager by a Web Service Container such as Apache Axis. The manager stores the requests, called messages, in the Message Execution Queue. The messages are sent to the distributed handlers and the responses are received after the successful handler execution. The manager ensures that each message is executed without being interrupted. Every message execution contains one or more stages. Several distributed handlers may construct a single stage for which handlers concurrently process the message. The manager awaits the completion of the handler executions before starting the delivery of the message to the next stage. This procedure continues until all stages of a message are completed. At the end, the successfully obtained output returns to the Web Service Container. All of the messages in the Message Execution Queue are executed concurrently.

Since the handlers are located in separate memory spaces, the message on the wire should be secured against unauthorized access. An adversary can see the important information and/or modify the message. Moreover, the handler distribution manager and the handlers should authenticate themselves to prevent an adversary from intervening in the interaction. Hence, the following issues need to be addressed for the purpose of security:

- Eavesdropping: Potential hackers who have access to the network are able to read the messages.
- Message modification: The message travelling between the handlers and distribution manager can be modified by an unauthorized person.
- False messages: It is fairly easy to produce false messages and send them as if from an actual computing node.
- Message replay: Similar to message modification, the message formed by a handler or the distribution manager can be saved by others and sent again.
- Repudiation: As messages can be forged, there is no way of validating that a message has been sent by a particular node.

The reliability of a handler itself is also essential for successful execution. The message must reach the distributed handlers and be executed without failure. Hence, the reliability of the message delivery is critical. Additionally, the system must have mechanisms in place to deal with the failure of the handlers.

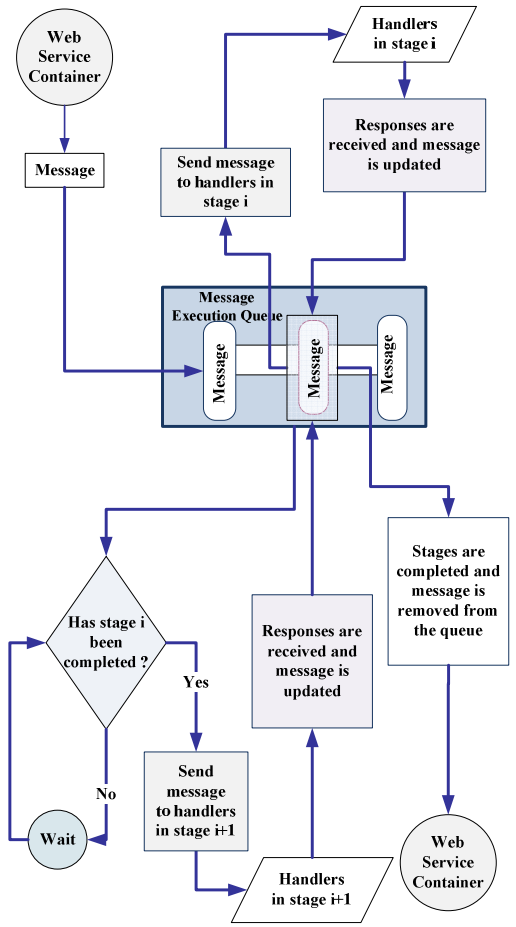


Figure 1. Executing the messages in the distributed Web Service handlers.

IV. RELIABILITY

Software reliability is described as the probability that the software functions without failure under given conditions during a specified period of time [26]. Reliability is also measured in terms of percentage of failure circumstances in a given number of attempts to compensate for variations in usage over time [27]. For Web Services, although reliability is viewed by some researchers as a non-functional characteristic [28], Zhang and Zhang describe one of the more comprehensive definitions of Web Service reliability as a combination of correctness, fault tolerance, availability, performance, and interoperability, where both functional and non-functional components are considered [29].

In this paper, reliability will be investigated in two categories: the reliability originating from the handler replication and the reliability coming from the utilization of a reliable messaging system.

A. Replicating handlers

Replication is critical for reliability, mobility, availability, and performance of a computing system. There are basically three replications: data, process, and message. These concepts are extensively explored in [30]. Data replication is the most heavily investigated one. However, the other replications are also very important in the distributed systems, especially for Service-Oriented Architecture.

The process replication is particularly of main interest in this paper because the intention is to investigate the replication of the handlers. There exist two main approaches in this area. The first one is modular redundancy [31]. The second approach is called primary/standby [32]. Modular redundancy has replicated components that perform the same functionalities. All replicas are active. On the other hand, the primary/standby approach utilizes a primary process to perform the execution. The remaining replicas wait in their standby state. They become active when the primary replica fails.

The processes can be classified into two categories: no consistency and consistency. The first category is the simplest one; the processes are stateless. They do not keep any information for the processed data. Therefore, consistency is not an issue between the processes. Replicated instances can be allowed to run concurrently. On the other hand, replicas may enter an inconsistent state if the process is not atomic and stateful. Inconsistency has been extensively investigated in [33].

Replication is a very important capability where a handler is inadequate. Sometimes, a handler may not be able to answer the incoming requests. The tasks may line up such that the overall performance degrades. This is similar to a shopping center, where customers are waiting in line to be served. The solution is to add one or more persons to serve when necessary. Similarly, adding a replica to help with the execution contributes to the overall performance.

In addition to the performance, a replica can be leveraged for fault tolerance. It is possible that a handler crashes. The replication contributes to the continuity of the execution and improves the availability and reliability. Without using handler replication in the case of an error, the whole computation cannot continue. The computation becomes more resilient with handler replication. The execution continues as long as at least one replica of every handler has not failed.

For n handlers with the replication factor of R, the execution can be successful for R-1 failures per handler. The maximum allowable number of errors is:

$$\sum_{i=1}^{n} R_i - 1 \quad (1)$$

where n is the number of Web Service handlers and R_i is the replication number of the *i-th* handler. The execution cannot continue even in a single handler fault, where $\forall i \in N: R_i = 1$.

In the distributed Web Service handler execution environment, a variation of the primary/standby approach is utilized. Dynamic binding ideas are employed for the replicated handlers using the primary/standby approach. Dynamic binding is a technique that allows services to be linked at run-time [34] [35]. The execution manager decides at run-time which distributed handler is invoked: the primary handler or a replica. The handlers are prioritized. The handler with the highest priority is assigned to execute a message. The other replicas wait until their priorities become highest. The system is able to change the priority during the execution. When a fault occurs, the handler priority is minimized.

If the replicated handlers were executed concurrently, a checkpoint mechanism must have been utilized. The checkpoint mechanism is based on the idea of saving the state of the system. In fault detection, the execution of the system is recovered from the checkpoint where the state was saved. The recovery mechanism is only launched when a fault occurs. Compensation handlers (Rollback) are specific computing nodes that limit and confine the effects created by a faulty handler. Compensation handlers allow the execution of the faulty replicated handlers to be rolled back to a specific point.

The checkpoint approach presents some drawbacks. It necessitates the introduction of additional elements into the distributed handler execution design. It requires extra time to check each important point, and recovery processes need to be activated when the rollback occurs. In fact, establishing the correct and minimal checkpoint and recovery structure is a highly complex task. The checkpoint solution is not, in short, suitable in view of the fact that the rollback mechanism could introduce a very high overhead in the case of a fault. Hence, the primary/standby approach is preferred for the distributed replica handlers.

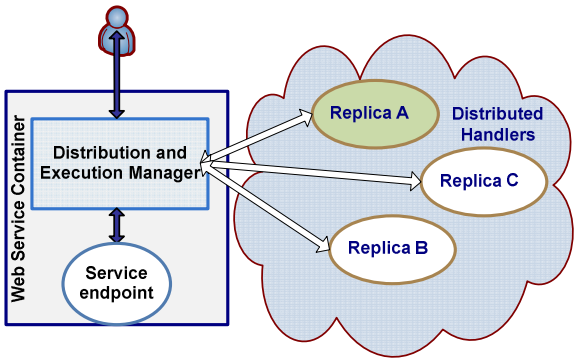


Figure 2. The execution of replicas using the primary/standby approach.

The replicas are never allowed to be executed concurrently, except in the case of stateless handlers. Even though they are allowed to run in a parallel manner, they cannot join the processing of the same message. The messages have to be different so that the parallel execution does not cause inconsistency. Figure 2 depicts a replicated handler processing the incoming message while the other replicas await their turns.

When only one of the several replicated handlers is executed, as shown in the square in Figure 3, the following formula works to compute the reliability value:

$$R_{RH} = \sum_{i=1}^{n} P_i \, R_i \qquad (2)$$

where R_{RH} is the reliability of the replicated handlers' execution, P_i is the execution probability of handler i, and $\sum_{i=1}^{n} P_i = 1$.

The reliability of the parallel handlers with the AND junction and the reliability of the serial handlers can be formulated as:

$$R_s = \prod_{i=1}^{n} R_i \qquad (3)$$

where R_s is the reliability of the handlers' execution and R_i is the reliability of handler i.

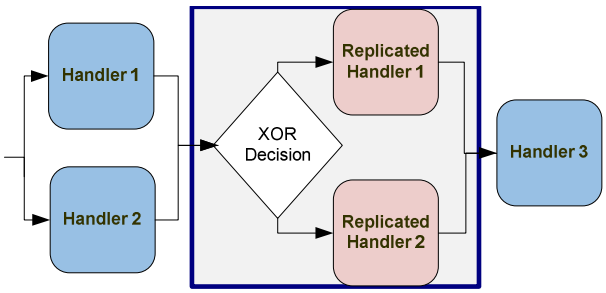


Figure 3. A sample configuration for the handlers' execution.

By using formulas 2 and 3, the reliability of the handlers' execution in Figure 3 can be formulated as:

$$R_s = \prod_{i=1}^{2} R_i \, . \sum_{R_i=1}^{2} P_{R_i} R_{R_i} \, . R_3 \qquad (4)$$

$$R_s = \prod_{i=1}^{3} R_i \, . \sum_{R_i=1}^{2} P_{R_i} \, R_{R_i} \qquad (5)$$

where R_s is the reliability of the handlers' execution. R_i is the reliability of the ith replica and $P_{R_i} = 1$ for only one replicated handler, which is executed; the value is 0 for the remaining handlers.

B. Reliable messaging

The distributed handler mechanism benefits from two different sources for reliable message delivery: a messaging broker and its own execution mechanism.

The messaging system, NaradaBrokering, provides message-level reliability. It also offers supportive functionalities to the messaging and a very reasonable performance [36]. The messages can be queued up to several thousands and are gradually delivered to their destinations to provide flow control for the messaging. Additionally, the system has a Reliable Delivery Service (RDS) component that delivers the payload even if a node fails [37].

RDS stores all the published events that match up with any one of its managed templates, which contain the set of headers and content descriptors. This archival operation is the initiator for any error correction, which is caused by the events being lost in transit to their targeted destinations and also by the entities recovering either from disconnect or a failure. For every managed template, RDS also maintains a list of entities for which it facilitates reliable delivery. RDS may also manage information regarding access controls, authorizations, and credentials of the entities that generate or consume events, which are targeted to this managed template.

When an entity is ready to start publishing events on a given template, it issues a discovery request to find out the availability of RDS, which provides the archival environment for the generated template events. The publisher will not circulate template events until it receives a confirmation that RDS is available.

The publisher ensures that the events are stored by RDS for every template event that it produces. After successful delivery of the event to RDS, the event is archived and a message is sent to the publisher to verify that the message was received by RDS successfully. Otherwise, a failure message with the related event id is sent back to the publisher. After verification, the suitable matching engine is utilized to compute the destinations associated with the template event.

A subscriber registers with RDS. A sequence number linked with the archival of the interaction is recorded. The number can be also described as an epoch, which signifies the point from which the registered entity is authorized to receive events conforming to the template. Once a template event has been archived, RDS issues a notification. The notifications allow a subscribing entity to keep track of the template events while facilitating error detection and correction. Upon receipt of the notification, the subscribing entity confirms the reception of the corresponding template event.

When an entity reconnects to the broker network after failures, the entity retrieves the template events that were issued and those that were in transit before the entity's leaving. After the receipt of the recovery request, RDS scans the dissemination table starting at the sync related to the entity and then generates an acknowledgment-response invoice event outlining the archival sequences that the entity did not previously receive. Accordingly, the missing events are provided to the receiver.

In addition to this, a reliable mechanism for the Web Service handler execution environment is built on top of the reliable messaging that NaradaBrokering provides. The distributed Web Service handler mechanism is able to repeat the execution of a specific handler in the event of a failure. Failure is declared when a response is not received from a distributed handler. There are several possible reasons behind an unsuccessful response. For example, the communication link may be broken, or the handler may not have successfully processed the message because of either an error or a crash. The distributed Web Service handler mechanism checks the possibilities by sending the message several times to its destination. In each attempt, it waits for a specific amount of time. This duration is either assigned or calculated by the system. After several unsuccessful attempts, the message processing may switch to a replica, depending on its priority. As discussed previously, handlers can populate their replicas to improve availability and reliability.

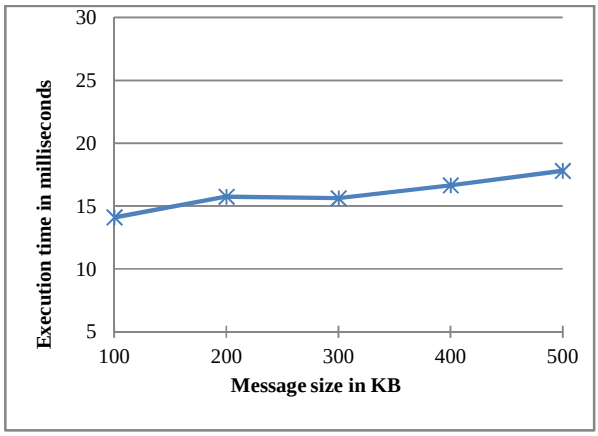


Figure 4. The cost of the reliability mechanism for the distributed Web Service handlers.

For a reliable messaging benchmark, two HP DL 380 G7, 2 x Xeon Six Core, 2.93 GHz, 48 GB memory physical machines are utilized. The machines are virtualized to create four 4-core and 16 GB memory machines and one 8-core 32 GB memory machine. These machines are connected to each other via a LAN and share a common storage system. Virtual machines use Windows Server 2008 R2 64-bit operating systems. The cost of the reliable mechanism of the messaging for the distributed handlers is shown in Figure 4. The cost contains the time needed for reliability procedures to send the tasks to the distributed Web Service handlers or receive the responses back. The time for the handlers' executions and the time for the messaging are excluded to illustrate only the reliability cost for varying message sizes. Figure 4 shows that the message size does not affect the cost of the reliability of the messaging very much. The cost is very reasonable when reliability is a necessity for the distribution.

V. Message Security

Security is one of the most important issues for computing systems. Critical data can be seen or altered by an unauthorized person. This is increasingly important if the data are transferred through the network, which is a more vulnerable environment.

Local computing does not expose its data to the outside world very much, but this is not the case for distributed computing. The computation is shared between nodes, which may be physically dispersed in the distributed environment. The transmission of the data among the nodes may expose critical information to dangerous vulnerabilities. Hence, the transportation channels between the computing nodes must be secured in addition to the security of the nodes.

NaradaBrokering, which is utilized for messaging, has a security framework that is able to support secure interactions between the distributed handlers [38]. The security infrastructure consists of a Key Management Center (KMC), which provides a host of functions specific to the management of keys in the system. The KMC stores the public keys of the interacting entities. It also provides authentication and authorization mechanisms to offer an enhanced environment for secure messaging.

Authentication is an elementary security requirement to prove that an entity possesses a claimed identity [39]. The basic tool that a person can use to prove a claimed identity is generally something that the person knows (e.g., a password), something that the person has (e.g., an authentication token), or a biometric property (e.g., fingerprints or iris recognition). Different mechanisms can be used for cryptographic authentication. Keyed hash functions or symmetric ciphers that utilize a specific key are among the examples. The key is only available to the entity to be authenticated and the verifier. One requirement for authentication mechanisms using the key is obviously the protection of the applied key. The leakage of the key causes

the collapse of the security mechanism. Another requirement is that the key should preferably be unique for the interacting entities. An attack is confined with this specific communication if the key of an interaction gets compromised.

Asymmetric cryptography can also be used for authentication. A proper procedure based on elliptic curves is described in [40]. A cryptographic operation is performed, to be authenticated using the entity's private key. The verifier checks the received response using the corresponding public key by performing a cryptographic verification operation on the received value. NaradaBrokering utilizes an authentication mechanism for the publishers and subscribers, which are the computing nodes for the distributed handler execution. For the authentication, the publisher or subscriber sends its signed request by using private key. The broker verifies this request by using the public key of the entities.

The KMC incorporates with an authorization module to manage the usage of the messaging. Every topic has an access control list that authorizes the subscribers. Similarly, an access control list exists for the publishers. After verification of the signature, the publisher or subscriber is permitted to access the entity according to the relevant access control lists.

The message traveling between the computing nodes is described in Figure 5. It contains a unique id, properties, and a payload. The unique message id is a distinctive name for a message. The handler execution mechanism may host many messages being executed at one moment. Hence, an identifier is necessary to achieve the correct executions; a Universally Unique Identifier (UUID) generated id is assigned to every message. The generator assures that there will not be more than one of the same id in the system. Thus, the design gives enough assurance that the message executions are not blended.

```
<context>
  <id>4099d6dc-0b0e-4aaa-95ff-2e758722a959</id>
    <properties>
     <encKey> bèø3DKUQ …</encKey >
     <sender>
       <senderId>12345… </ senderId >
       <signed>šZQSİÑU,k…</signed >
     </sender >
      ….
  </properties>
  <payload>
     ….
  </payload>
</context>
```

Figure 5. The message format for the distributed Web Service handlers.

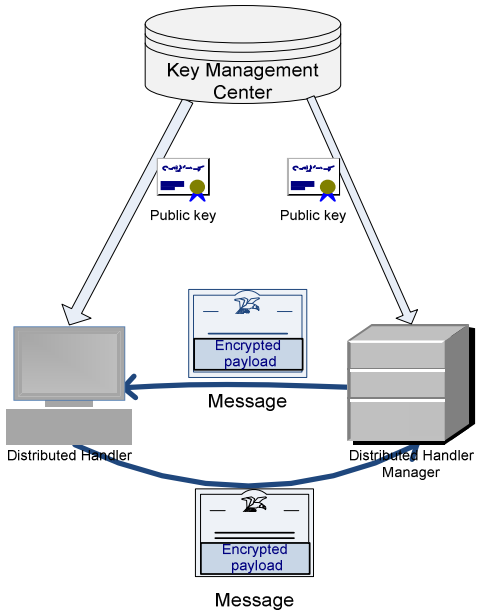


Figure 6. Security mechanism for a distributed handler execution.

The second important part of the message format is the properties section. This part conveys the required additional information to the computing nodes. The information can be specific to a single handler or generic for all handlers. There is a property that contains the encryption key. It is a symmetric key that is created for a single message. The key size is usually selected to be large enough to provide the necessary security. On the other hand, it must be kept in mind that generating larger keys is time-consuming. The average time taken for key generation for different bit sizes is presented in [41]. Therefore, the same symmetric key can be utilized to send a group of messages for a period of time. Additionally, the properties section contains the sender's signature to prove the sender's authenticity. The sender signs its unique id with its private key. Both the sender's id and signature are added to the properties section. The last part of the distributed handler message format is the payload, which contains the encrypted message.

The performance of the asymmetric key encryption is worse than the performance of the symmetric key encryption [42]. It can take about 1000 times more CPU time to process an asymmetric encryption or decryption than a symmetric encryption or decryption. Nevertheless, an important advantage of asymmetric ciphers over symmetric ciphers is that no secret channel is necessary to exchange the keys. The receiver needs only to be confident about the authenticity of the public key provider. Asymmetric ciphers also cause fewer key management problems than symmetric ciphers. Only $2n$ keys are needed for n nodes to communicate securely with each other in an asymmetric key encryption system. However, in a system based on a symmetric cipher, $n(n - 1)/2$ secret keys are needed for secure interaction among n nodes. Because of these features,

asymmetric ciphers are typically used for non-repudiation, for authentication through digital signatures, and for the distribution of symmetric keys. Thus, the asymmetric key algorithm is able to support the solving of the key exchange and management problem of the symmetric keys. On the other hand, symmetric ciphers are used for bulk encryption.

To use the best part of the algorithms, a hybrid approach is utilized. Figure 6 demonstrates a secure messaging architecture for the distributed handlers. The payload of the message is encrypted by a symmetric cipher algorithm. Advanced Encryption Standard (AES) is used for the encryption. AES is a natural choice for the symmetric key algorithm because it has been analyzed extensively and used worldwide. The cryptography scheme is a symmetric block cipher that encrypts and decrypts blocks of data. The AES key generation algorithm takes a random seed as an input. A 256-bit session key is created and passed within the properties section of the message to the other computing node for decryption.

The RSA algorithm is utilized for the asymmetric key encryption. The sender encrypts the symmetric session key with the 2048-bit public key of the receiver to present the confidentiality. The RSA algorithm requires two large prime numbers as the input along with a random seed. All of these inputs, which are created randomly, are provided for key generations. The created private keys are then kept locally, and the public key is stored in the KMC.

With the commonly used RSA implementations, doubling the RSA key length means that encryption will be more than two times slower and decryption will be almost four times slower, as shown in Table I. In general, RSA encryption is much faster than RSA decryption. The fast encryption relies on the use of a short public exponent. The RSA algorithm is commonly used in this way. It is possible to have an RSA public key with a long public exponent, which will make encryption as slow as decryption. However, because a long public exponent does not improve security, short public exponents are widespread. Some well-known RSA implementations do not even support long public exponents. Hence, the decryption exponent is typically huge, whereas the encryption exponent is small.

TABLE I. PUBLIC KEY ENCRYPTION AND DECRYPTION RESULTS.

	Encryption time in milliseconds		Decryption time in milliseconds	
Plain text size in KB	1024-bit	2048-bit	1024-bit	2048-bit
100	262	470	4703	14906
200	563	892	8844	29594
300	784	1298	13703	44542
400	1048	1735	17766	59064
500	1298	2391	22454	73737

TABLE II. COMPERISON OF CIPHER TEXT SIZE IN PUBLIC KEY ENCRYPTION.

Plain text size in KB	Cipher text size in KB	
	1024-bit	2048-bit
100	109	105
200	218	209
300	329	314
400	438	418
500	547	522

The key length of an RSA key specifies the number of bits in the modulus. A larger key increases the maximum number of bytes that we can encrypt in a block at once, as well as the security of the encryption. On the other hand, with every doubling of the RSA key length, decryption becomes much slower. Key length also affects the speed of encryption, but the speed of decryption is usually of greater concern.

Moreover, depending on the padding scheme, the cipher text size increases in RSA encryption. This is another factor that is taken into consideration while using this system. Table II shows the text sizes for 1024-bit and 2048-bit key encryptions. When using a 1024-bit RSA key with PKCS #1 padding, it is not possible to encrypt a string that is longer than 117 bytes. Increasing the size of the RSA key to 2048 bits will allow the encrypting of 245 bytes of data, but longer RSA keys are expensive and they take more time to generate and operate. On the other hand, the AES encryption algorithm does not show the size increase in the encrypted text even though it also does block ciphering with 128 bytes.

The size of cipher text can be calculated with the following formula:

$$c = p + b - (p \bmod b) \quad (6)$$

where c is the cipher text size, p is the plain text and b is the block size.

As mentioned earlier, the authentication of the sender and receiver and authorization to access the message are established by the security mechanisms of the messaging broker. Figure 7 shows the tasks happening between the sender and receiver for a single interaction. The sender generates the symmetric session key for a message or a group of messages. The payload containing the message is encrypted with this symmetric session key with the AES algorithm. The sender looks up the receiver's public key in the KMC. The RSA algorithm is used to encrypt the symmetric session key with the receiver's public key. Hence, only the node that has the right private key can decrypt the session key to get the encrypted payload. At the same time, the sender authenticates itself by signing its id with its private key.

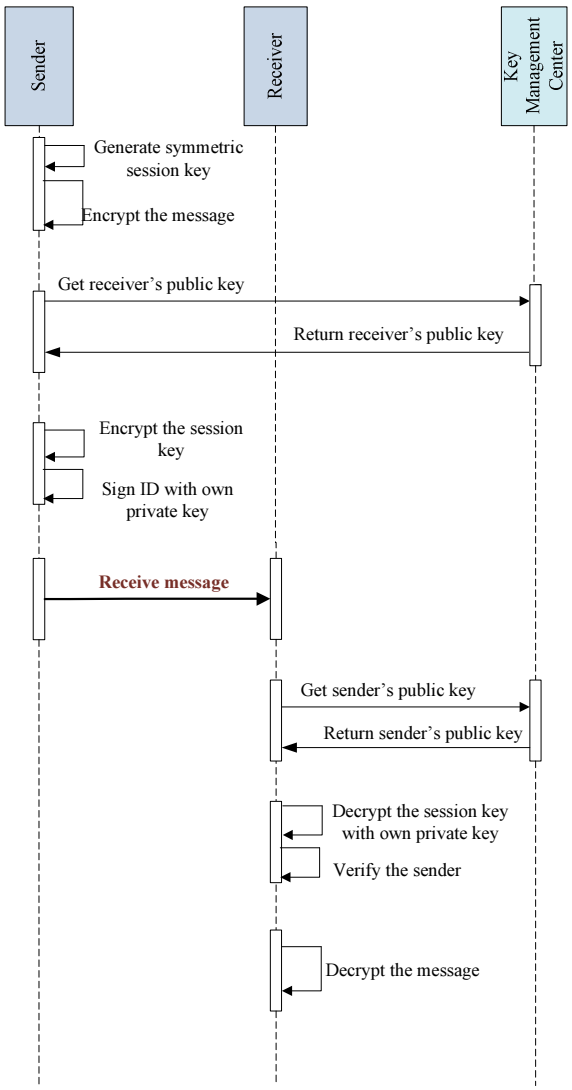


Figure 7. Interactions while sending a message between computing nodes.

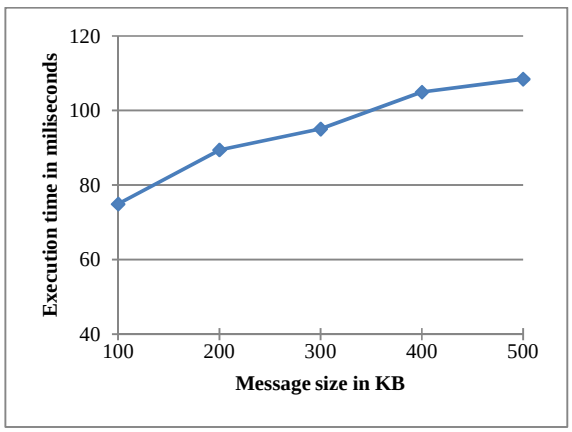


Figure 8. The cost of the security mechanism for the distributed Web Service handlers.

When the subscriber receives the message, it identifies and verifies the sender by using the sender's public key, which is retrieved from the KMC. The session key carried within the "encKey" tag is then decrypted by the receiver's private key. The retrieved session key is used to decrypt the payload to get the original message. In this interaction, the senders and receivers are either the distributed Web Service handlers or the distribution manager.

The benchmarks showing the cost of the aforementioned security mechanism and the results of the tables containing public key encryption are determined in the same environment as the reliability benchmark, as discussed in Section IV.B. Figure 8 shows the cost of the secure environment for varying payload sizes. The signing of the sender's id to present the authentication causes very small overhead. Instead of asymmetric key encryption, the usage of the symmetric key to encrypt the messages provides reasonable execution time. As stated earlier, even though asymmetric key encryption solves the problem of key exchange, it does not accomplish the message encryption and decryption for large sizes at an affordable cost. In short, a hybrid approach using both asymmetric and symmetric ciphers helps to improve security at a reasonable cost.

VI. CONCLUSION

Although the distribution of Web Service handlers provides many advantages in terms of scalability, availability, and performance, it necessitates a reliable and secure atmosphere. The instruments explained in this paper for secure and reliable handler distribution and the support tools of the utilized messaging broker grant the necessary features for this atmosphere. Utilized reliability mechanisms deal with the distributed computing node failures by using replication and ensure the message delivery. The hybrid security approach advances the environment by offering a solution for the key exchange problem of the symmetric encryption and by reducing the cost of the asymmetric cipher algorithm. The design also delivers the authentication and authorization mechanisms for the distributed handlers. The benchmark results show that the costs originating from the utilized instruments are acceptable and affordable. In short, the design of the distributed execution with the security and reliability tools offers a satisfactory environment for Web Service handlers. Moreover, it should be kept in mind that a secure and reliable environment must be employed in many mission-critical tasks.

REFERENCES

[1] B. Yildiz, "Reliability and message security for distributed web service handlers," Proc. the Seventh International Conference on Internet and Web Applications and Services (ICIW 2012) , Stuttgart, Germany, May 2012, pp. 17-22.

[2] Web Service Security (WS-Security), Available: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss, [Retrieved: May 20, 2013].

[3] Web Service Reliable Messaging (WS-ReliableMessaging), Available:http://public.dhe.ibm.com/software/dw/specs/ws-

rm/ws-reliablemessaging200502.pdf, [Retrieved: May 20, 2013].

[4] Web Service Trust (WS-Trust), Available:http://docs.oasis-open.org/ws-sx/ws-trust/v1.4/errata01/os/ws-trust-1.4-errata01-os-complete.html, [Retrieved: May 20, 2013].

[5] Web Service Federation (WS-Federation), Available:http://docs.oasis-open.org/wsfed/federation/v1.2/os /ws-federation-1.2-spec-os.html, [Retrieved: May 20, 2013].

[6] C. M. Jayalath and R. U. Fernando, "A modular architecture for secure and reliable distributed communication," Proc. the Second International Conference on Availability, Reliability and Security (ARES07), Washington, DC, USA, 2007, pp. 621-628, DOI=10.1109/ARES.2007.7.

[7] L. E. Moser, P. M. Melliar-Smith, and W. Zhao, ''Building dependable and secure web services,'' Journal of Software, vol.2, no.1, 2007, pp. 14-26.

[8] S. L. Pallemulle, H. D. Thorvaldsson, and K. J. Goldman, "Byzantine fault-tolerant web services for n-tier and service oriented architectures," Proc. the 28th International Conference on Distributed Computing Systems (ICDCS '08), Washington, DC, USA, 2008, pp. 260-268. DOI=10.1109/ICDCS.2008.94.

[9] M. Lei, S. V. Vrbsky, and Z. Qi, "Online grid replication optimizers to improve system reliability," Proc. IEEE International Parallel and Distributed Processing Symposium (IPDPS 2007), March 2007, pp.1-8.

[10] N. Aghdaie and Y. Tamir, "CoRAL: a transparent fault-tolerant web service," Journal of System Software, vol. 82, no. 1, January 2009, pp. 131-143, DOI=10.1016/j.jss.2008.06.036.

[11] W. Zhang, "Integrated security framework for secure web services," Proc. the Third International Symposium on Intelligent Information Technology and Security Informatics (IITSI '10), Washington, DC, USA, 2010, pp. 178-183, DOI=10.1109/IITSI.2010.8.

[12] H. F. EL Yamany, M. A. M. Capretz, and D. S. Allison, "Quality of security service for web services within SOA," Proc. Congress on Services (SERVICES '09). Washington, DC, USA, 2009, pp. 653-660, DOI=10.1109/SERVICES-I.2009.95.

[13] A. J. Varela-Vaca, R. M. Gasca, D. B. Nunez, and S. P. Hidalgo, "Fault tolerance framework using model-based diagnosis: towards dependable business processes," International Journal on Advances in Security, issn 1942-2636 vol. 4, no. 1 & 2, year 2011, pp.11-22.

[14] L. Zhao,Y. Ren, Y. Xiang, and K Sakurai, "Fault-tolerant scheduling with dynamic number of replicas in heterogeneous systems", Proc. 12th IEEE International Conference on High Performance Computing and Communications (HPCC), Melbourne, September 2010, pp. 434-441, DOI=10.1109/HPCC.2010.72.

[15] Y. Desmedt, R. Gennaro, K. Kurosawa, and V. Shoup, "A new and improved paradigm for hybrid encryption secure against chosen ciphertext attack," Journal of Cryptology, vol. 23, iss. 2, January 2010, pp. 91-120, DOI=10.1007/s00145-009-9051-4.

[16] K. Ramachandran, H. Lutfiyya, and M. Perry, "Chaavi: a privacy preserving architecture for webmail systems," Proc. the Second International Conference on Cloud Computing, GRIDs, and Virtualization, 2011, pp. 133–140.

[17] S. S. Rizvi, A. Riasat, and K. M. Elleithy, "Combining private and public key encryption techniques for providing extreme secure environment for an academic institution application," International Journal of Network Security & Its Application (IJNSA), vol.2, no.1, January 2010, pp. 82-96.

[18] E. Ramaraj, S. Karthikeyan, and M. Hemalatha, "A Design of security protocol using hybrid encryption technique (AES-Rijndael and RSA)," International Journal of the Computer, the Internet and Management, vol. 17, no. 1, January 2009, pp. 78-86.

[19] V. Palanisamy and A. M. Jeneba, "Hybrid cryptography by the implementation of RSA and AES," International Journal of Current Research, vol. 3, iss. 4, April 2011, pp.241-244.

[20] E. Damiani, F. Pagano, and D Pagano, "iPrivacy: a distributed approach to privacy on the cloud," International Journal on Advances in Security, vol. 4, no. 3&4, 2011, pp. 185-197.

[21] K. Komathy, V. Ramachandran, and P. Vivekanandan, "Security for XML messaging services: a component-based approach," Journal of Network and Computer Applications, vol. 26, iss. 2, April 2003, pp. 197-211, DOI=10.1016/S1084-8045(03)00003-1.

[22] F. T. Ammari and J. Lu, "Advanced XML security: framework for building secure XML management system (SXMS)," Proc. the Seventh International Conference on Information Technology: New Generations (ITNG '10), Washington, DC, , 2010, pp. 120-125, DOI=10.1109/ITNG.2010.124.

[23] B. Yildiz, G. Fox, and S. Pallickara, "An orchestration for distributed web service handlers," Proc. International Conference on Internet and Web Applications and Services (ICIW08), June 2008, Athens, Greece, pp. 638-643.

[24] B. Yildiz, "Distributed handler architecture," Ph.D. Dissertation. Indiana University, Bloomington, IN, USA. Advisor: Geoffrey C. Fox. 2007.

[25] B. Yildiz and G. Fox, "Measuring overhead for distributed web service handler," Proc. the Third IEEE International Conference on Computer Science and Information Technology (ICCSIT 2010), July 2010, pp. 566-570.

[26] H. Zo, D. Nazareth, and H. Jain, "Measuring reliability of applications composed of web services," Proc. 40th Annual Hawaii International Conference on System Sciences (HICSS '07), 2007, pp. 278- 288.

[27] J. D. Musa, Software reliability engineering, McGraw-Hill, New York, NY, 1999.

[28] A. Arsanjani, B. Hailpern, J. Martin, and P. Tarr, "Web services: promises and compromises, "ACM Queue, 1 (1), pp. 48-58, March 2003.

[29] J. Zhang and L.-J. Zhang, "Criteria analysis and validation of the reliability of Web Services-oriented systems," Proc. the IEEE International Conference on Web Services (ICWS'05), Orlando, Florida, July 2005, pp. 621-628.

[30] A. Helal, A. Heddaya, and B.K. Bhargava, "Replication techniques in distributed systems," Advances in Database Systems, vol. 4, 2002, pp. 61-71, DOI: 10.1007/0-306-47796-3_3.

[31] P.A. Lee and T. Anderson, Fault tolerance: principles and practice, Springer-Verlag New York, Inc. Secaucus, 1990.

[32] P. P. W. Chan, M. R. Lyu, and M. Malek, "Making services fault tolerant," Proc. 3rd International Conference on Service Availability (ISAS'06), Berlin, Heidelberg, 2006, pp. 43-61, DOI=10.1007/11955498_4.

[33] P.T.T. Huyen and K. Ochimizu, "Toward inconsistency awareness in collaborative software development," Proc. 18th Asia Pacific Software Engineering Conference (APSEC), Dec. 2011, pp. 154-162.

[34] A. Erradi and P. Maheshwari, "Dynamic binding framework for adaptive web services," Proc. the 2008 Third International Conference on Internet and Web Applications and Services. Washington, DC, USA, 2008, pp. 162–167.

[35] U. K¨uster and B. K¨onig-Ries, "Dynamic binding for BPEL processes - a lightweight approach to integrate semantics into web services," Proc. 4th International Conference on Service

Oriented Computing (ICSOC06), Chicago, Illinois, USA, 2006, pp. 116–127.

[36] S. Pallickara and G. Fox, “NaradaBrokering: a distributed middleware framework and architecture for enabling durable peer-to-peer grids,” Proc. the ACM/IFIP/USENIX International Conference on Middleware (Middleware '03), 2003, pp. 41-61.

[37] S. Pallickara and G. Fox, "A scheme for reliable delivery of events in distributed middleware systems," Proc. the IEEE International Conference on Autonomic Computing (ICAC'04), New York, NY, May 2004, pp. 328-329.

[38] S. Pallickara, M. Pierce, G. Fox, Y. Yan, and Y, Huang, "A Security framework for distributed brokering systems," Available:http://www.naradabrokering.org, [Retrieved: May 20, 2013].

[39] R. Falk and S. Fries, “Advances in protecting remote component authentication,” International Journal on Advances in Security, issn 1942-2636 vol. 5, no. 1 & 2, year 2012, pp. 28-35.

[40] M. Braun, E. Hess, and B. Meyer, “Using Elliptic Curves on RFID Tags,” International Journal of Computer Science and Network Security, vol. 2, February 2008, pp. 1-9.

[41] K. Ramachandran, H. Lutfiyya, and M. Perry, “A Privacy preserving solution for web mail systems with searchable encryption,” International Journal on Advances in Security, issn 1942-2636 vol. 5, no. 1 & 2, year 2012, pp. 26-45.

[42] C. Narasimham and J. Pradhan,“Evaluation of performance characteristics of cryptosystem using text files”, Journal of Theoretical and Applied Information Technology, vol. 4, iss. 1, 2008, pp. 56-60.

Model-Based Design of Dependable Systems: Limitations and Evolution of Analysis and Verification Approaches

Jose Ignacio Aizpurua and Eñaut Muxika
Department of Signal Theory and Communications
University of Mondragon
Spain
{*jiaizpurua,emuxika*}*@mondragon.edu*

***Abstract*—Designing a dependable system successfully is a challenging issue that is an ongoing research subject in the literature. Different approaches have been adopted to analyse and verify the dependability of a system design. This process is far from obvious and often hampered due to the limitations of the classical dependability analysis and verification approaches. This paper provides an overview of model-based dependability analysis, design and verification approaches. Firstly, model-based analysis approaches are grouped by the limitations of the classical approaches. Secondly, design approaches have been classified looking at their underlying recovery strategies: hardware replication and hardware reuse. Then, the ins and outs of model-based verification approaches are identified starting from fault injection approaches towards their evolution into model-based integrative approaches. Finally, a model-based hybrid design process is presented making use of the reviewed analysis, design and verification approaches.**

Keywords-Model-based dependability analysis, System design, Heterogeneous redundancy, Dependability verification.

Acronyms and Abbreviations

AADL	Architecture Analysis and Design Language
BDMP	Boolean logic Driven Markov Process
CFP	Compositional Failure Propagation
CFT	Component Fault Tree
(D)FTA	(Dynamic) Fault Tree Analysis
(D)RBD	(Dynamic) Reliability Block Diagram
(D(S))PN	(Deterministic and (Stochastic)) Petri Nets
(D)(C)TMC	(Discrete)(Continuous) Time Markov Chain
DOE	Design of Experiments
FI	Fault Injection
(FM)E(C)A	(Failure Mode) and Effect (Criticality) Analysis
FPT(C)(N)	Failure Propagation and Transformation (Calculus)(Notation)
HiP-HOPS	Hierarchically Performed Hazard Origin and Propagation Studies
IMA	Integrated Modular Avionics
MC	Model Checking
MCS	Monte Carlo Simulations
SEFT	State-Event Fault Trees

I. Introduction

Designing a dependable system presents many challenges throughout the development phase - from system specification to system validation and verification. This process is further complicated owing to the increasing complexity of the current systems, which use many and different components. Model-based design approaches provide mechanisms to manage this complexity effectively. However, these approaches together with dependability analysis and verification approaches, add some limitations to the system design process. Hence, those who are not familiar with the field of model-based design of dependable systems will find it useful to have a list of sources characterized by their limitations and evolutions. Our goal is not to exhaustively evaluate the specific features of these approaches, neither to give means to study these techniques in great detail. Instead, we are aimed at aggregating a comprehensive list of works grouped by their main characteristics.

We have completed our previous work [1] providing the reader a guided collection of sources to indicate where to learn about the model-based design of dependable systems. Namely, we have extended previous concepts and characterized three additional aspects: (1) tool-support of the analysis and verification approaches, (2) classification of dependable design approaches with respect to their recovery strategies, and (3) evolution of model-based fault injection approaches.

In computing systems, dependability is defined as "ability of a system to deliver a service that can be justifiably trusted" [2]. Such trustworthiness is based on the assurance of dependability requirements. These requirements are defined through dependability attributes: *Reliability*, *Availability*, *Maintainability*, *Safety (RAMS)*, *confidentiality* and *integrity*. The scope of this overview focuses on RAMS attributes. Consequently, security aspects (confidentiality and integrity) are not addressed.

Reliability is the ability of an item to perform a required function under given conditions for a stated period of time. It is usually characterized through Mean Time To Failure (MTTF) [3]. *Maintainability* is the ability to undergo repairs and modifications to restore to a state in

which the system can perform its required actions and it is commonly characterized by Mean Time To Restore (MTTR). *Availability* is the readiness for correct service defined by the ratio between MTTF and MTTF plus MTTR. *Safety* is the absence of catastrophic consequences on the user(s) and the environment. These four attributes are closely interrelated in such a way that if an attribute fails to meet the requirements, systems dependability is seriously threatened.

This survey concentrates on three main phases: dependability analysis, system design and verification. Despite being aware of the relevance of software code for system dependability in each of these phases, we will consider software code as a black box component to limit the extension of this paper (interested readers can refer to [4] [5] as an example).

Dependability analysis techniques can be organised by looking at how different system failures are characterized with its corresponding underlying formalisms. On one hand, *event-based* approaches reflect the system functionality and failure behaviour through combination of events. This analysis results in either Fault Tree (FT) like [6] or Failure Mode and Effect Analysis (FMEA) like [7] structures, which emphasizes the reliability and safety attributes. On the other hand, *state-based* approaches map the analysis models into a state-based formalisms (e.g., Stochastic Petri Nets (SPN) [8]). Those approaches analyse system changes with respect to time and concentrate on reliability and availability attributes.

Concerning the design of dependable systems, there exist alternative recovery strategies that add redundancies to the system design in order to avoid single points of failure and thus, provide fault tolerance. Traditionally, hardware replication have been viewed as a feasible solution to recover from failures. However, there is also another design strategy which compensates for component failures through the reuse of hardware components. We will address these design strategies and their influence on dependability and cost.

Fault Injection (FI) and Model-Checking (MC) [9] approaches have been widely adopted for the verification of design decisions. Since these approaches enable to learn about the system behaviour in the presence of faults and verify its correctness relying on a fully automated model-based approach, we concentrate on model-based FI approaches. They evaluate the dependability requirements using functional and failure behaviour models. Moreover, we also cover the recent evolution of these approaches towards the integration multiple approaches using a single reference model, i.e., model-based integrative verification approaches. We are conscious that there are other verification approaches, e.g., correct by construction paradigm aimed at ensuring the validity of the system by design [10] (i.e., formal verification). Nonetheless, we have decided to limit the overview to model-based analysis and verification approaches due to their potential to integrate in the system design process seamlessly.

The remainder of this paper is organized as follows: Section II classifies dependability analysis techniques based on the limitations of classical dependability analysis techniques. Section III groups dependable design approaches characterized by the replication and reuse of hardware components. Section IV discusses the characteristics of the verification approaches when designing a dependable system. Section V outlines a model-based hybrid design process based on the reviewed analysis, design and verification approaches. Finally, Section VI draws conclusions remarking different challenges for the model-based design of dependable systems.

II. Review and Classification of Model-Based Dependability Analysis Techniques

Event-based approaches analyse the failure behaviour of the system by investigating the logic succession of faults. They identify an event sequence leading to equipment or function failure. Differences are mainly based on representation and analysis structures. Two of the most widely used techniques are: FT Analysis (FTA) [6] and FMEA [7].

- *FTA* is a top-down deductive analysis technique aimed at finding all the ways in which a failure can occur. Starting from an undesirable system-level failure (i.e., top-event), its immediate causes to occur are identified until reaching the lowest component-level (i.e., basic-event). The top-event is broken down into intermediate and basic-events linked with logic gates organised in a tree-like structure. The resulting FT, is a qualitative model in the form of combinations of events which are necessary to the top-event to occur. This model can be quantified by ascribing probabilities to the basic events and combining them to evaluate the probability of the top-event. Moreover, importance measurements can be carried out to quantify the contribution of the basic-event occurrences to the top-event failure. There exist different importance measurement methods, e.g., Fussell-Vessely, Birnbaum [6].
- *FMEA* is a bottom-up inductive analysis technique. Starting from the different ways that a system can fail (i.e., known Failure Modes (FM)), it evaluates the effects that these failures can have on a process. Its main objective is an early identification of critical failure possibilities within the system design. Results are organized in a table identifying at least component's failure modes, effects, safeguards and actions. FMEA is characterised for being a qualitative technique for design analysis, while quantification of failure modes is carried out by Failure Mode and Effect Criticality Analysis (FMECA). The criticality analysis ranks critical failure mode effects by taking into account their occurrence probability.

Both techniques focus on the identification of events that jeopardize the system design objectives. However, their initial assumptions as well as their logical orientation are different: while FTA moves from known effects towards unknown causes in a deductive manner, FMEA progresses from known causes towards unknown effects inductively. They are not orthogonal techniques, indeed they are complementary and in some cases they overlap [11]. The extended usage of these approaches for dependability related tasks have lead to the identification of the main limitations. Subsequently, there has been a long list of works aimed at overcoming them:

- L1: FMEA and FTA are static representations of the system, neither time information nor sequence dependencies are taken into account [12].
- L2: Orientation of FTA and FMEA concentrate on the analysis of failure chain information. Consequently, their hierarchy reflects failure influences without considering system functional architecture (design) information [13].
- L3: FMEA and FTA depend on the analyst's skill to reflect the aspects of interest. Failure modes and undesired events must be foreseen, resulting in a process highly dependent on analyst's knowledge of the system [14].
- L4: Manageability and legibility of FTA and FMEA models is hampered when analysing complex systems. Model size, lack of resources to handle interrelated failures and repeated events, in conjunction with few reusability means, are its main impediments [13] [15].

L1 refers to the capability of the technique to handle temporal notions. This is of paramount importance when analysing fault tolerant systems. L2 emphasizes the interdisciplinary work between dependability analysis and architectural design. Joining both procedures helps obtaining a design, which meets dependability requirements consistently. L3 entails a trade-off solution between the time consuming analysis resulted from understanding the failure behaviour of the system and the acquired experience. A substantial body of works have been oriented towards the automatic generation of analysis models from design models (refer to groups 3, 5 in Table IV) addressing limitations L2 and L3. These approaches promote the reuse of design models showing the effects of design changes in the analysis results. However, note that the correctness of the analysis lies in the accuracy of the failure annotations. Finally, L4 underlines the capability of the model to handle the component-wise nature of embedded systems. This permits obtaining a model that better adheres to the real problem and avoids confusing results.

Many authors have developed new alternatives or extended existing ones. Three groups are identified in order to gather the approaches and limitations strategically. Firstly, L1 is addressed in the Subsection *Dynamic Solutions for Static-Logic Approaches*. Secondly, L2 and L4 are covered in the Subsection *Compositional Failure Propagation Analysis Approaches*. Finally, specifically focusing on L3 and generally addressing the remainder of limitations *Model-Based Transformational Approaches* are studied. Note that some approaches cannot be limited to a specific group, hence they are classified accordingly to its main contribution.

A. Dynamic Solutions for Static-Logic Approaches

The limitation concerning the lack of time information has been addressed by several authors to deal with system *dynamics* such as redundancy or repair strategies.

Dugan et al. [12] paved the way to cope with configuration changing analysis using FTs by defining Dynamic Fault Tree (DFT) methodology. New gates were introduced accounting for event ordered situations like common cause failures and redundancy strategies. In [16], temporal notions and FT models were integrated in order to handle the probabilistic timed behaviour of the system. The model reflects how modular FT models are switched through discrete points in time taking into account time dependant basic events.

Other alternatives to analyse DFT models are based on Monte Carlo Simulations (MCS) by specifying temporal failure and repair behaviours of components through state-time diagrams [17]. In [18], MatCarloRe approach is presented based on Simulink [19] for DFT modelling and reliability analysis. The technique integrates MCS and FTA methodologies resulting in a intuitive model-based simulating environment.

Following the way of DFTs, an approach emerged based on Reliability Block Diagrams (RBD) [3]. RBD is focused on the analysis of the success of the system, instead of the failure analysis orientation of FTs. Dynamic RBDs (DRBDs) [20] model failures and repairs of components based on their dependencies using state machines. To analyse these models, an algorithm which converts DRBDs into Coloured Petri Nets was presented in [21]. However, an integrated modelling and analysis toolset for DRBDs is lacking. Signoret et al. in [22] presented an approach called RBD driven Petri nets (RdP) which uses RBDs as an interface to build large Petri nets systematically. The modular characterization of Petri nets enables the intuitive creation of RdP models from predefined module libraries. Aligned with these formalisms, the OpenSESAME modelling environment connects RBDs and state-based formalisms [23]. It enables an straightforward evaluation of highly available system's fault tolerant behaviour including system *dynamics*. Its input models are based on RBDs and failure dependency diagrams, while component and repair tables are used to indicate component-specific characteristics (MTTF, MTTR) and repair groups. To carry out system analyses, these models are transformed into state-based SPN and Stochastic Process Algebra (SPA) models.

Lopatkin et al. [24] utilise FMEA models for system formal specifications. The approach defines generic patterns establishing direct correspondence between FMEA and state-based Event-B [25] formalism. The characterization of error detection and recovery patterns lead to analysing and verifying whether safety properties are preserved in the presence of faults. The use of these patterns, allows tracing from static FMEA considerations towards system *dynamics*.

Progression in the conjoint use of event and state formalisms is reflected with Boolean logic Driven Markov Processes (BDMP) [26]. BDMP employs static FT as a structure function of the system and associates Markov processes to each leaf of the tree. Similarly, State-Event Fault Tree (SEFT) [27] formalism combines elements from FT with both statecharts [28] and Markov chains, increasing the expressiveness of the model. SEFT deals with functional and failure behaviour, accounts for repeated states and events and allows straightforward transformations of SEFT models into Deterministic and Stochastic Petri Net (DSPN) models for state-based analysis.

In [29], Steiner et al. described a process to create and analyse SEFTs based on the ESSaRel tool [30]. Subsequently, the SEFT models are exported and analysed by means of a DSPN analyser tool called TimeNET [31]. The model described in Figure 1, shows some of the SEFT's strengths and characteristics in a simple behaviour model: (1) combination of statecharts with Markov chains by means of states (S) and events (E); (2) compositional characterization of system functionalities connected through required inports and provided outports (event_out_vel_ok, event_out_vel_high), which enables linking components in a FT-like structure while managing system's complexity; (3) characterization of functional and failure behaviour of the system, i.e., functional state (out_vel_ok), failure state (out_vel_high), and transition events between these states (no_obstacle_detected and sensors_working); and (4) underlying transformable logic to analyse with a target state-based formalism.

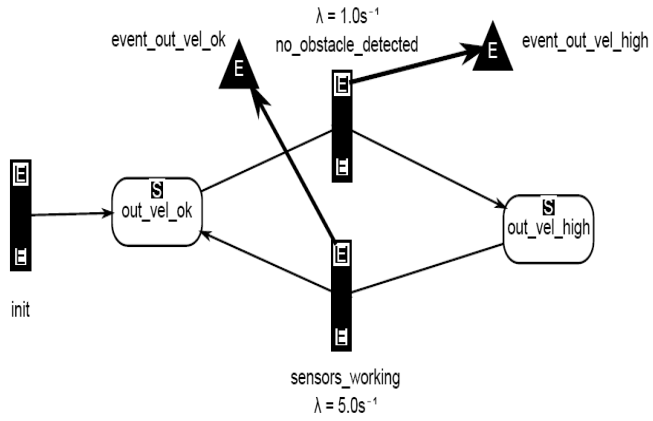


Figure 1. SEFT model example [29].

Likewise, El Ariss et al. in [32] presented an approach called Integrated Functional and Safety Specification (IFSS) model, in which they provide a systematic transformation from FT models into statecharts accounting for the temporal behaviour of faults. As a result of this integration process, they eliminate inconsistencies when using functional and non-functional models and account for the order of failure occurrences.

Table I displays among the addressed approaches those which has tool support for the specification and analysis of the dynamic behaviour of systems.

Table I
TOOL-SUPPORT OF THE DYNAMIC APPROACHES

Approach - Work	Tool Support	Type of Tool	Latest Release
DFT	e.g., Galileo [33]	Commercial, Educational	2003
Rao et al. [17]	DRSIM tool	Internal	2009
MatCarloRe	RAATSS [34]	Academic evaluation copy	2012
RdP	BStoK [35]	Comercial	2011
OpenSESAME	OpenSESAME [36]	Available	2009
Lopatkin et al. [24]	Rodin Plugin [25]	Available	2012
BDMP	KB3 Workbench [37]	Available	2012
SEFT	ESSaRel extension & transformation [29]	Internal*	2013

* Available for research purposes under agreement

DFT is a well-known mature approach for the evaluation of the system's *dynamics*. It has been adopted with different tools over the last years (e.g., Galileo [33]). In contrast, approaches such as IFSS, DRBD or SEFTs do not have a integrated tool support. The compositional and transformational features of the SEFT approach, provide an adequate abstraction of the system structure and behaviour. Developing a model-based tool which extracts DSPN models from SEFT models automatically would create an adequate environment for building a sound approach for manageable and consistent dependability analyses.

B. Compositional Failure Propagation Analysis Approaches

The common factors for Compositional Failure Propagation (CFP) approaches are: the characterization of the system architectures by design components; the annotation of the failure behaviour of each of them and the system failure analysis based on inter-components influences. Conceptually they all are very similar: the main objective of CFP approaches is to avoid unexpected consequences resulting from the failure generation, propagation and transformation of components.

Generally, CFP approaches characterise the system as component-wise developed FT-like models linked with a

causality chain. System architectural specifications and subsequent dependability analyses of CFP approaches rely on a hierarchical system model. This model comprises components composed from subcomponents specifying system structure and/or behaviour. CFP approaches analyse the system failure behaviour through characterizations of individual components, which lead to achieving a manageable failure analysis procedure. Failure Propagation and Transformation Notation (FPTN) [38], Hierarchically Performed Hazard Origin and Propagation Studies (HiP-HOPS) [39] and Component Fault Tree (CFT) [13] are the principal CFP approaches. Their main difference is in the failure annotations of components, which specify incoming, outgoing and internal failures to each component. In order to annotate the logical combinations of these failures, FPTN uses logical equations, HiP-HOPS makes annotations using Interface Focused FMEA (IF-FMEA) tables and CFT associates to each component individual FTs. Subsequently, the connections between system components determines the *failure flow* of the system, linking related failure annotations of each component.

Concerning the different contributions of CFP approaches, FPTN first addressed the integration of system-level deductive FTA (from known effects to unknown causes) with component-level inductive FMEA (from known causes to unknown effects). HiP-HOPS integrates design and dependability analysis concepts within a hierarchical system model. However, instead of exclusively linking functional components with their failure propagations like in FPTN, first the hierarchical system model is specified and then, compositional failure annotations are added to each component by means of IF-FMEA annotations. These annotations describe the failure propagation behaviour of the component in terms of outgoing failures specified as logical combinations of incoming and internal failures (cf. Figure 2).

From the IF-FMEA annotations shown in Figure 2, the outgoing failures at the port *out_1* will be specified as follows: *omission-out_1 = omission-in_1 AND omission-in_2 OR Stuck at 0*. Once characterized all the outgoing failures of all the ports, a FT synthesis algorithm analyses the propagation of failures between connected components. Traversing the hierarchical system model, while parsing systematically the IF-FMEA annotations of its constituent components, allows the extraction of the system FT and FMEA models.

CFTs are a model-based extension of FTA models. The component FTs can be combined and reused to systematically obtain the FT for any failure without having to create and annotate a FT for each failure. In order to integrate analysis and design concepts, it has been extended in [40] resulting in the Safe Component Model (SCM) approach. The approach separates components' functional/failure specification and realization views and through the integration of the failure propagation and hierarchical abstraction, SCM allows obtaining a hierarchical component based abstraction

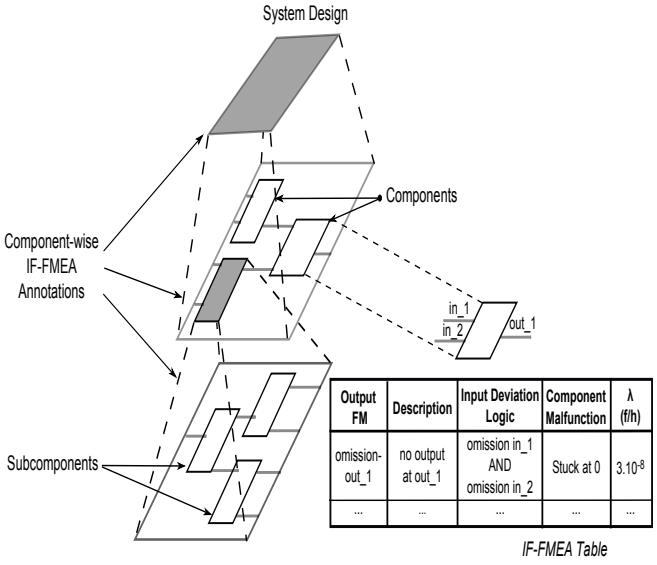


Figure 2. Hierarchical structure and CFP annotations in HiP-HOPS.

of CFTs.

They all have been extended to cope with occurrences of temporal events. Temporal extensions for FPTN [41] and HiP-HOPS [42] have been influenced by the DFT methodology. They concentrate on the analysis of non-repairable systems examining the order of events so as to identify specific sequence of events leading to failures. Integration of CFT concepts with state-based techniques resulted in the SEFT formalism, which is able to handle availability and maintainability properties of repairable systems.

Transformation of CFP approaches into dependability analysis formalisms is an ongoing research subject (see Subsection II-C). HiP-HOPS extracts FTA and FMEA models thanks to its underlying logic and SEFT applies translation schemes to generate DSPN models.

Other interesting extensions include mechanisms to automate and reuse analysis concepts. Failure Propagation and Transformation Calculus (FPTC) [43] approach adds the characterization of the nominal behaviour to FPTN models and generalizes the FPTN equations to improve the manageability and analysability. Moreover, an algorithm is implemented to handle the general inability of CFP approaches to cope with cyclic dependencies of feedback structures. In [44], general failure logic annotation patterns were defined for HiP-HOPS. Similarly, the CFP approach presented in [45] by Priesterjahn et al., emphasizes the reuse of failure propagation properties specified at the port level of components. These specifications centre on the physical properties of different types of flows, which allow reusing failure behaviour patterns for functional architectures.

Concerning the model-based generation of FMEA models, in [46] Struss and Fraracci presented an approach to extract FMEA models mechanically. To do so, they based on devia-

tion models which describe formally the constraints of system functionalities and it provides the necessary predictions for extracting FMEA models. The approach implements reuse mechanisms through the use of generic libraries and it has been implemented in the Raz'r tool [47].

The evolution of CFP approaches focus on reusability, automation and transformation properties. Since the annotations of the components' failure behaviour depend upon designers experience, reusing failure annotations leads to reducing the error proneness. Based on the fact that different dependability analyses have to be performed when designing a system, definition of a unique consistent model covering all analyses would benefit these approaches. This is why recent publications in this field centre on integrating existing approaches (cf. Subsection II-C and Section IV).

Regarding the tool support of the CFP approaches (cf. Table II) we can see that all approaches have been turned into toolsets. Nonetheless, the CFP approaches are moving one step further, integrating design languages in order to link the design and analysis processes (cf. Subsection II-C).

Table II
TOOL-SUPPORT OF THE CFP APPROACHES

Approach - Work	Tool Support	Type of Tool	Latest Release
FPTN	SSAP Toolset [38]	Unavailable	2006
HiP-HOPS	HiP-HOPS Tool [39]	Available	2012
CFT	ESSaRel tool [30]	Available	2009
SCM [40]	ComposeR	Internal	2012
FPTC	Epsilon [48]	Available	2009
Priesterjahn et al. [45]	MechatronicUML, Fujaba [49]	Available	2012
Struss & Fraracci [46]	Raz'r Tool [47]	Comercial	2012

C. Model-Based Transformational Approaches

The main aim of the transformational approaches is to construct dependability analysis models (semi-)automatically. The process starts from a compositional design description using computer science modelling techniques. The failure behaviour is specified either by extending explicitly the design model or developing a separate model, which is allocated to the design model. Transformation rules and algorithms extract dependability analysis models from it.

These approaches lead to adopting trade-off decisions between dependability design and analysis processes. On one hand, the automation and reuse of analysis techniques in a manageable way makes it a worthwhile approach for design purposes. The impact of design changes on dependability attributes are analysed systematically. On the other hand, from purist's point of view of classical analysis techniques, the automation process removes the ability of these techniques to identify and analyse hazards or malfunctions in a comprehensive and structured way.

Architectural Description Languages (ADLs) provide an adequate abstraction to overcome the limitations. Simulink [19], AADL [50] and UML [51] have been used for both architectural specification and failure behaviour specification. UML is a widely used modelling language, which has been extended for dependability analyses following Model Driven Architecture (MDA) concepts [52]. Namely, profiles allow extending and customizing modelling mechanisms to the dependability domain [53].

Lately, a wide variety of independently developed extensions and profiles have come up for dependability analysis [54]. However, some generally applicable metamodel is lacking. In an effort to provide a consistent profile CHESS ML emerged [55]. CHESS ML provides all necessary mechanisms to model dependable systems and extract either event-based (FMECA, FPTC) or state-based (SPN) analysis models. Recently in [56], a model-driven failure logic analysis method called CHESS-FLA has been presented. This approach is built upon Formalism for Incompletion, Inconsistency, Interference and Impermanence Failures' Analysis (FI^4FA) approach [57] which is based on the concepts of FPTC. FI^4FA differs from FPTC in that it offers two additional features. Namely, it provides mechanisms to analyse more types of failures than FPTC and it allows modelling and analysing the mitigation behaviour. The CHESS-FLA approach supports back-propagation of the results to ease the readability of the analysis results.

CFP approaches have been shifted towards the transformational paradigm. The toolset for FPTC approach [43] relies on a generic metamodel in order to support transformations from SysML and AADL models. In [58], a metamodel is developed so as to extract CFT models from functional architecture models specified in UML. This process permits the generation of reusable CFT models consistent with the design model. In the same line, integration of HiP-HOPS model with EAST-ADL2 automotive UML profile is presented in [59]. Translations from high-level ADLs to well established CFP analysis techniques, enable an early dependability analysis and allow undertaking timely design decisions.

Architecture Analysis and Design Language (AADL) captures the system architectural model in terms of components and their interactions describing functional, mapping and timing properties among others. The core language can be extended to meet specific requirements with annex libraries. Behaviour and error model annexes are provided with the tool. The error annex links system architecture components to their failure behaviour specification making possible the analysis of the dependability attributes of the system. It has been used for both state-based [60] and event-based [61] analysis.

AltaRica [62] is a dependability language, which enables describing the behaviour of systems when faults occur. The model is composed of several components linked together representing an automaton of all possible behaviour scenarios, including those cases when reconfigurations occur due to the occurrence of a failure [63]. It is possible to process such models by other tools for model-checking or generation of FTs [64]. Transformations from AADL to AltaRica are presented in [65], based on MDA concepts so as to perform dependability analyses and avoid inconsistencies while working with different formalisms.

Riedl and Siegle presented a language for the specification of reconfigurable and dependable systems called LARES [66]. It expresses system's fault tolerant behaviour using a generic language in which any kind of discrete-event stochastic system can be specified. It is based on fully automated model transformations to measure systems dependability. Namely, transformations into TimeNET [31] and CASPA [67] tools are carried out in order to solve state-based SPA and SPN models respectively.

In [68], Cressent et al. defined a method for RAMS analysis centred on SysML [69] modelling language from where a FMEA model is deduced. SysML diagrams define a functional model connected to a dysfunctional database enabling the identification of failure modes. This database contains the link between system architecture and failure behaviour giving the key for FMEA extraction. Further, the methodology for dependability assessment is extended using AltaRica, AADL and Simulink models. They address reliability and timing analyses and simulation of the effects of faults respectively.

Definition of a model for the extraction of all necessary formalisms for dependability analysis is the common goal for the aforementioned works. Interconnections between different formalisms in order to take advantage of the strengths of each ADL, allow analysing dependability properties accurately. AltaRica and AADL cover adequately the analysis of reliability, availability and maintainability attributes. Extraction of the main CFP approaches from ADLs should help to analyse comprehensively system safety properties. Moreover, Simulink model simulations allow evaluating the effects of failure and repair events in the system. Thereby, integrations between language specific models like in [68] helps evaluating accurately all dependability aspects of a system.

The acceptance of the transformational approaches depends on the availability of toolsets capable of performing (automatic) transformations. As it is shown in Table III, all ADLs have their own implementation toolsets. Namely, transformations from ADL models into CFP models have been carried out through metamodels and profiles implemented as plugins.

Table III
TOOL-SUPPORT OF THE TRANSFORMATIONAL APPROACHES

Approach - Work	Tool Support	Type of Tool	Latest Release
Simulink	Matlab [19]	Comercial	2012
UML, SysML	e.g., Eclipse Papyrus [70]	Available	2012
AltaRica	e.g., AltaRica Tools [71]	Available	2013
AADL	e.g., Osate [72]	Available	2012
CHESS-ML	CHESS Plugins [73]	Partially available	2012
FPTC	Epsilon [48]	Available	2009
Adler et al. [58]	CFT UML Profile	Internal	2012
HiP-HOPS	EAST-ADL2 Eclipse Plugin [74]	Available	2010
LARES [66]	LARES toolset	Internal	Ongoing
Cressent et al. [68]	MéDISIS Framework	Internal	Ongoing

D. Classification of Techniques

In order to classify the covered approaches, Table IV groups them taking into account limitations specified in Section II.

Table IV
SUMMARY OF LIMITATIONS OVERCOME BY APPROACHES

Group	Approach	Limitations
1	[12] [16] [17] [18] [23] [26]	L1
2	[13] [38] [40] [43] [46]	L2, L4
3	[39] [45] [56] [61]	L2, L3, L4
4	[20] [22] [32] [41] [63]	L1, L2, L4
5	[24] [27] [42] [55] [60] [66] [68]	L1, L2, L3, L4

Approaches gathered within the group 5 contain all necessary features in order to analyse dynamic systems consistently and in a manageable way. Compositional failure annotation, dynamic behaviour and automatic extraction of analysis models are the key features addressed by these approaches. Utilization of failure annotation patterns promote flexibility and reuse and consequently, reduce the error proneness. Nevertheless, as noted in [75], characterization of the failure behaviour of components depends on the component context, which conditions compositional and reuse properties. Moreover, automatic generation of the analysis model does not completely alleviate the dependency on the knowledge of the analyst. However, it lets managing and specifying the failure behaviour in a clear and consistent way.

III. Design of Dependable Systems: Trade-Off Between Dependability and Cost

Generally, dependability design decisions and objectives are related to trade-off decisions between system dependability attributes and cost. Dependability requirements often conflict with one another, e.g., safety-availability compromise when a faults leads the system to a safe shutdown in order to prevent it from propagating. The time at which design decisions are taken determines the cost that the design process can incur.

Designing a dependable system within considered requirements requires a process to match and tune combination of architectural components so as to find an optimal solution satisfying design constraints. There are other approaches concentrated on the design of dependable systems under the correct-by-construction paradigm. For instance, the approach presented in [76] creates a formal system specification preserving the correctness through gradual refinements of the system design model. However, instead of addressing formal correct-by-construction dependable design approaches, we will overview those approaches which are aimed at characterizing at design time the implications of design decisions (combination of components) on dependability and cost.

More specifically, we group dependable design approaches by looking at how system recovery strategies are implemented. Traditionally, hardware replication has been viewed as a feasible solution to recover from failures, e.g., N modular redundancy. Nonetheless, there are other kinds of recovery strategies which make the repair possible reusing already existing hardware components. On one side, in Subsection III-A we group those approaches that replicate the nominal functionality by aggregating additional hardware resources, i.e., *homogeneous redundancies*. On the other side, in Subsection III-B we group those approaches which are aimed at reusing hardware components to provide a compatible functionality and reduce hardware costs, i.e., *heterogeneous redundancies* [77]. Finally, in Subsection III-C, we have summarized and characterized these approaches with respect to our design criteria and identified key design activities to progress in the use of *heterogeneous redundancies*.

A. Design of Dependable Systems by Means of Homogeneous Redundancies

The principal question addressed by the approaches grouped in this subsection is the evaluation of the effect of design choices (e.g., robustness level of components, redundancy configurations) on dependability and cost.

Cauffriez et al. [78] and Clarhaut et al. [79] focused on designing a dependable system based on a design methodology presented in [80]. The main focus of this methodology relies on the early and systematic characterization of dependability criteria during the system design activities. As it is shown in Figure 3, the approach comprehends three types of architectures: functional architecture, equipment architecture and operational architecture.

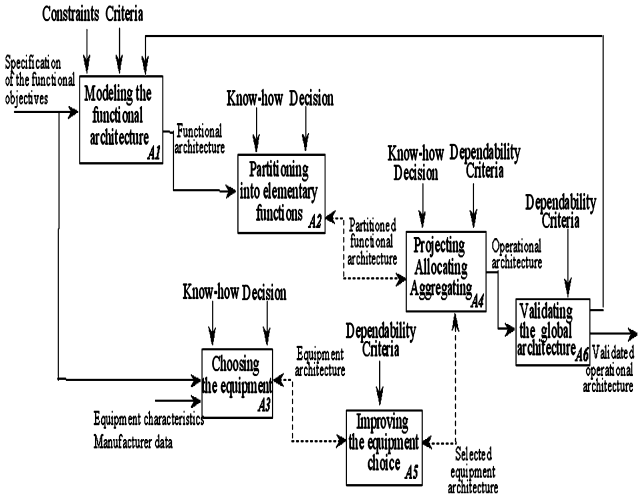


Figure 3. Methodology for designing distributed control systems [80].

The design process starts from the characterization of functional and equipment architectures addressing functional and dependability criteria. Subsequently, the allocation of the functional architecture onto the equipment architecture is evaluated in relation to dependability. As a result, the operational architecture is produced which could require reconsidering functional and/or equipment decisions in order to obtain a validated operational architecture with respect to dependability requirements.

Cauffriez et al. in [78] concentrated on the analysis of repairable architectures evaluating how the use of alternative hardware components affects system functionality and dependability. To do so, they characterized system-level functions in a top-down manner until lowest level subfunctions are reached. At the bottom layer, failure and repair rates of hardware components permit analysing system top layer's performance, reliability and availability using Monte Carlo simulations. In this way, a structural function is characterized which links functions and hardware resources and allows evaluating alternative operational modes by associating different subfunctions to perform the system-level function. The overall design methodology for modelling and analysing alternative architectural design choices has been integrated within a design tool.

Clarhaut et al. described a design approach overcoming the static-logic limitation of event-based analysis techniques (cf. Subsection II-A) by identifying sequential component-wise contributions to system-level failures [79]. During the design process, a *functional hierarchical tree model* characterizes dependencies between functions and hardware resources. This model accounts for alternative resources and hence, architectures to perform the modelled control func-

tions. Subsequently, the *Improved Multi-Fault Tree* (IFT) is constructed characterizing sequential failure relationships between components' failure modes (FM) and system functions which lead to the system-level undesired effects designated as dreaded events.

As shown in Figure 4, the structure of the design methodology revolves around the characterization, analysis, and optimization of system architectures so as to adopt optimal design decisions regarding dependability and cost. The IFT determines the dependability level of the overall architecture weighting the contribution of each component to the system-level dreaded events. Architectural design choices cover active and passive redundancies. The cost associated with each hardware component enables progressing between alternative architectures toward an optimal architecture which maximizes dependability and reduces hardware cost.

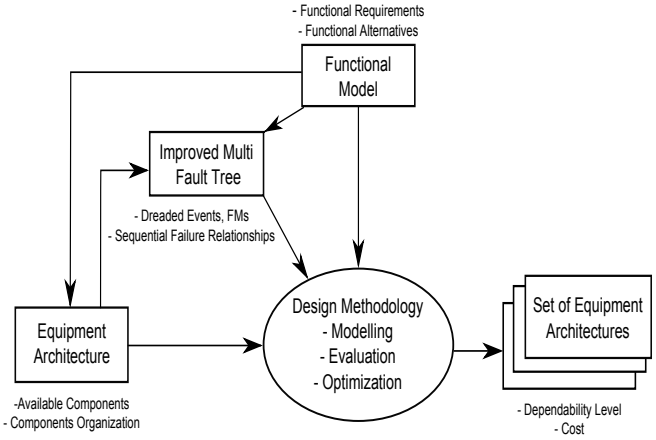


Figure 4. Clarhaut et al.'s design approach.

Adachi et al. in the work presented in [81], extended the HiP-HOPS approach with recovery strategies in order to design optimal architectures reducing cost and increasing dependability. The recovery strategies are formally represented using patterns. These patterns characterize the potential to detect, mitigate, and block affecting component failures which are previously identified with HiP-HOPS and analysed by means of FTA and FMEA. Finally, starting from an abstract architecture, recovery strategies are introduced without violating user constraints and an evolutionary optimization algorithm allows converging through dependability and cost requirements.

All the covered approaches aim at increasing system dependability through the replication of nominal components. This design decision implies a cost increase. Consequently, this drawback needs to be justified through an exhaustive and adequate analysis of how the system design meets functional and dependability requirements.

B. Design of Dependable Systems by Means of Heterogeneous Redundancies

One of the key properties of the systems which exercise *heterogeneous redundancies* is the ability to successfully accommodate changes in case of failure occurrences. Consequently, the system design approaches that we will cover in this subsection not only address dependability issues, but also adaptation capabilities. Thus, we group them as adaptive dependable design approaches.

Shelton and Koopman first worked on the concept of *heterogeneous redundancies* by means of *functional alternative* strategies. It allows to compensate for component failures changing the system functionality [77]. The approach models alternative system configurations and assigns a relative utility value to each of them weighing their contribution to the system performance and dependability. From this model, system's overall utility value is calculated which enables the evaluation and comparison of design choices as to where allocate resources for functional alternatives or redundancy. This characterization make it possible to evaluate how component failures affect system utility.

Wysocki et al. in [82] addressed the same design strategy under the *shared redundancy* concept. They concentrated on the reuse of processing units through the strategic distribution of software modules. Consequently, given the failure occurrence of a software component, it is possible to still continue operating through the reconfiguration of communication routes. To evaluate the reliability and safety of the alternative architectures, first a FTA is carried out. This analysis permits extracting minimal combination of events which leads the system to failure. Additionally, this information is used as input for further analysis through Design Of Experiments (DOE) to calculate system cost and failure probabilities. Based on the same design concept, Galdun et al. in [83] analysed the reliability of a networked control system structure using Petri Nets (PN) and Monte Carlo simulations.

Rawashdeh and Lump in [84] presented a framework for designing reconfigurable architectures for fault tolerant embedded systems called ARDEA. The approach is based on processing units' reconfigurations to achieve *graceful degradation* and cope with hardware failures. A gracefully degrading system tolerates system failures by providing the same or equivalent functionality with the remaining system components. Dependency Graphs (DGs) are used to model the functional flow of information from input to output considering alternative implementations. A centralized system manager uses DGs and a hardware resource list to find a viable mapping of software on the available processing units. It decides when to schedule/un-schedule software modules moving object code among available processing units without exceeding processor time and bandwidth.

In the MARS project, Trapp et al. [85] proposed a component based modelling and analysis method to exploit *implicit redundancies* so as to react to system failures by reusing hardware resources. It provides methodological support for modelling and gathering system configurations. Moreover, reasonable system configurations are elicited from a set of possible candidates. The system's adaptive behaviour is modelled based on quality types, which drive the system's graceful degradation possibilities. The quality type system determined at design time and modelled with a system inheritance tree, defines the possibilities for exchanging quality types between components.

Each system component operates under different configurations determined by Quality Attributes (QAs). These QAs are attached to each component's every I/O port. Configuration activation rules are defined over these ports based on the needed QAs to activate a configuration (preconditions) and provided QAs by this configuration (postconditions) (cf. Figure 5).

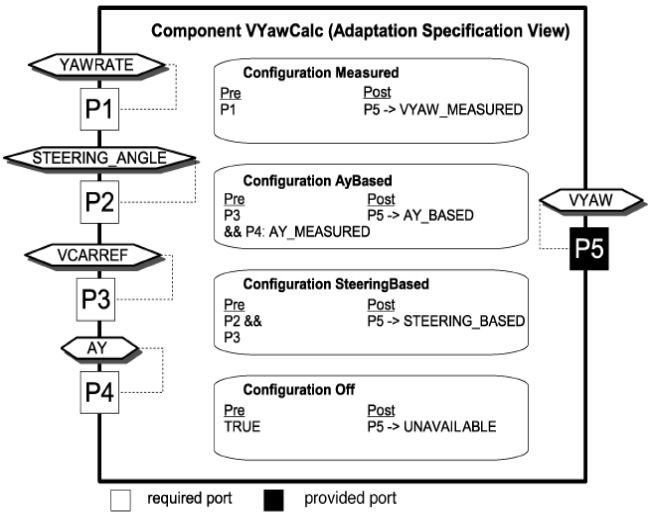


Figure 5. Example of an adaptation specification view [86].

As it is shown in Figure 5, for each component within the hierarchical system architecture, its possible configuration variants are defined. Each port has linked its own quality type which will determine the corresponding constraints over the port. Each configuration has its own activation preconditions and propagation postconditions defined over the component ports' QAs. This characterization leads to determining compatible components based on QAs. As a result, component compositions are abstracted into hierarchical components with a explicitly defined adaptation behaviour.

From this modelling paradigm (MARS modelling), different analyses have been carried out. In [87], transformations of MARS models into hybrid-CFT were performed in order to calculate configuration probabilities (cf. Subsection II-A). Moreover, in order to ensure the causality of the reconfiguration sequences and safety-related properties, verification activities have been carried out in [86]. Last but not least, methodological support for identifying an adaptation model meeting availability-cost trade-off were addressed in [88].

Similar design concepts are addressed in the avionics field. Namely, the Integrated Modular Avionics (IMA) design paradigm defines robust partitioning in on-board avionic systems so that one computing module (Line Replaceable Unit - LRU) is able to execute one or more applications of different criticality levels independently. The standardised generic hardware modules forming a network leads to looser coupling between hardware and software applications [89].

SCARLETT project [90] aims at designing reconfigurable IMA architectures in order to mitigate the effect of failures of functional, fault detection and reconfiguration implementations. Monitoring and fault detection functions aim at detecting component failures. Once a permanent failure is detected, the reconfiguration supervisor performs two key activities. Firstly, it manages the modifications given the current configurations and failed module and secondly, it checks the correctness of the system configuration and the loaded data in the LRU. The centralized supervisor determines a suitable configuration based on a reconfiguration graph, which contains all possible configurations. Reconfiguration policies and real-time and resource constraints, define the set of reachable safe transitions and states. In order to analyse the reconfiguration behaviour when failures occur, a safety model leads to finding the combinations of functional failures [91]. Based on the same concepts, DIANA project [92] aims at distributing these functionalities. This approach improves the availability of the reconfiguration mechanisms at the expense of relying on a complex, resource consuming communication protocol.

C. Summary of the Design Approaches

In order to characterize the reviewed approaches within this section, the following design properties have been described in the Table V:

1) Type of recovery strategy.
2) Dependability analysis approach.
3) Cost evaluation.
4) Other tasks: optimization or verification.

Since the use of *heterogeneous redundancies* requires considering system *dynamics*, the dependability analysis approaches described so far address system's temporal behaviour either by linking event-based static-logic approaches with state-based formalisms (e.g., Hybrid-CFT) or evaluating through approaches which integrate the temporal behaviour explicitly (e.g., MCS, DFT, PN) (cf. Subsection II-A). Moreover, given the extra design complexity of the systems which use *heterogeneous redundancies*, the mechanisms which help structuring the analysis and

Table V
APPROACHES AND ADDRESSED DESIGN PROPERTIES

Works	1	2	3	4
[78]	Homogeneous Redundancies	MCS	HW cost	NA
[79]	Homogeneous Redundancies	IFT	HW cost	Optimization
[81]	Homogeneous Redundancies	HiP-HOPS	HW & SW cost	Optimization
[77]	Heterogeneous Redundancies	Utility Values	NA	Optimization
[82] [83]	Shared Redundancies	FTA, MCS, DOE; PN, MCS	Mainten. cost	NA
[84]	Graceful Degradation	NA	NA	NA
[85]	Implicit Redundancy	Hybrid-CFT	HW & SW cost	Optimization, Verification
[90] [92]	Reconfigurable IMA	Safety analysis, AltaRica	NA	NA

Legend: *NA*: Not addressed; *Mainten*: Maintenance

reusing the models are necessary (e.g., libraries, hierarchical abstractions).

In order to obtain a predictable system design and avoid unexpected failure occurrences, all the approaches address design-time determined reconfigurations. Nonetheless, it is necessary to go beyond and overcome their underlying assumptions concerning the system's critical functionalities to perform reconfigurations effectively. Namely, among all the reviewed approaches only [91] takes into account a possible faulty behaviour of the fault detection and reconfiguration implementations. Similarly, the faulty behaviour of the communication network is only explicitly addressed in [83]. The evaluation of the possible faulty behaviour of these implementations leads to obtaining an approach which better adheres to the reality and consequently, more reliable results. Despite not addressing *heterogeneous redundancy* like concepts directly, in [93] an approach called component logic models is presented which does address the faulty behaviour of fault detectors. To this end, it associates modes with services and in the case of detection measures each service-port evaluates the veracity of the fault detectors (e.g., modes of false veracity: false positive and false negative).

As a result, from our perspective it is necessary cover the following design activities:

- *A1*: Systematic identification of *heterogeneous redundancies* and extraction of system configurations to react in the presence of failures.
- *A2*: Design of the system architecture to make the use of *heterogeneous redundancies* possible, i.e., fault detection and reconfiguration implementations.
- *A3*: Evaluation of the system dependability with respect to dependability and adaptivity constraints.

The first design activity calls for an approach which allows identifying systematically existing hardware components to provide a compatible functionality. To the best of our knowledge, only the work we presented in [94] works towards this goal. In [86], Adler et al. worked on the systematic extraction of system configurations annotating component by component their adaptive behaviour. During this process they evaluate in a ad-hoc manner if it is possible to provide another configuration variant using alternative hardware components and finally extract system configurations based on inter-component influences.

The second activity requires addressing design decisions regarding the organization of fault detection and reconfiguration implementations, i.e., their distribution and replication. On one hand, when implementing the fault detection function within a networked control system, it is possible to allocate it either on the source Processing Unit (PU) where the information is produced (e.g., sensor, controller) or in the destination PU, which is the target PU of the source information (e.g., controller, actuator) or in both PUs. On the other hand, when dealing with reconfiguration implementations, its distribution influences the overall dependability and cost of the system (cf. Table VI).

Table VI
DESIGN DECISIONS AND INFLUENCED ATTRIBUTES

Design / Attribute	Fault Detection		Reconfiguration	
	Source	*Destination*	*Centralised*	*Distributed*
Dependability	Det. at origin, unable to cope with comm. failures	Det. of wrong value & omission. Prone to CCF	SPOF	Multiple reconfig. redundancies
Cost	HW/SW implementation costs	Costly id. of all failures: failure transf.	Single reconfig. HW/SW costs	Higher cost: multiple reconfigs
Complexity	Direct failure handling	Further failure sources	Less comm. overhead	Complex comm; resource handling

Legend: *SPOF*: Single Point of Failure; *CCF*: Common Cause Failure; *comm*: communication; *det*: detection; *reconfig*: reconfiguration; *id*: identification; *transf*: transformations.

Additionally, when adopting design decisions within the second activity, it is necessary to address adaptivity constraints which also has influence on dependability, e.g., *timeliness constraints*: maximal duration in which the adaptation of one component can be performed [95], *dependency constraints*: dependencies between system components, where adapting one component requires further adaptation on other components [86] or *hardware resource constraints*: limit the use of hardware resources, e.g., processing power, memory [84].

In the third activity, previously adopted system design decisions are analysed with respect to dependability and cost. To this end, we include the alternative configurations and possible faulty behaviour of the fault detection, the reconfiguration and the communication implementations.

IV. Model-Based Verification: From Fault Injection to Integrative Approaches

Fault Injection (FI) approaches concentrate on verifying system behaviour in the presence of faults according to target dependability properties. The outcome of this process may lead to considering design changes. However, changes adopted late in the design process are costly. This is why we focus on model-based FI approaches adopted at the preliminary design phase. This process is based on the analyst's knowledge to reason about the functional, failure and recovery behaviour of the system. Timely evaluation of these properties provides a valuable feedback for design purposes. However, difficulties arise from the accuracy of the system behaviour, which requires an accurate knowledge of the system.

Within the model-based FI approaches, the verification process is characterized as follows: first, a functional model is specified which is then converted into an *extended system model* accounting for the functional and failure behaviour of the system. Temporal logic languages are used to define system *requirements*. They describe how the truth values of assertions change over time, either qualitatively (Computation Tree Logic (CTL), Linear Time Logic (LTL)) or quantitatively (Continuous Stochastic Logic (CSL), probabilistic CTL (PCTL)). Model-checking (MC) engine assesses whether such requirements are met or not by the extended system model, using a *analysis model*. To do so, it is necessary to transform the extended system model into the analysis model. When the analysis model fails to meet these requirements, its effects are deduced automatically identifying the paths that violate the conditions (counter-examples (CEs)) [9]. The logical orientation of this analysis process results in FMEA-like cause-effect analysis.

Automatic transformation from extended system models to MC analysis models and definition of predefined libraries for correct and complete failure specification and injection are the main points in order to obtain a consistent and robust FI process. ESACS project [96] addressed these characteristics prominently providing mechanisms to extend the model straightforwardly and extract FTA models from it. Other approaches concentrated on the generation of (probabilistic) FMEA (pFMEA) models [97] [98]. Application of pFMEA models, improvement in the interpretation of counter-examples [97] and automated tool support for injecting faults and analysing its consequences [98] are their main contributions.

Albeit these approaches provide a means to extract classical dependability models from the FI process, few of them concentrate on integrating existing CFP approaches. There are some incipient works linking CFP and verification approaches. They are influenced by HiP-HOPS [99] and FPTC [100]. Both approaches address the integration of qualitative design models with quantitative analysis via probabilistic MC. In [99], Gomes et al. described an approach to verify quantitatively system's dependability requirements. To do so, they did a systematic transformation of Simulink models into CTMC models by means of PRISM model checker [101]. The annotation of the failure behaviour is carried out as in HiP-HOPS using IF-FMEA. In [100] Ge et al. proposed a probabilistic variant of FPTC called Failure Propagation and Transformation Analysis (FPTA). The approach links architectural models with probabilistic model checkers specified in PRISM. The ins and outs of these approaches are grouped in the Table VII.

Results extracted from this process helps verifying if a system behaves as intended in the presence of failures. Concerning the analysis of dependability attributes, generated counter-examples as well as extraction of classical dependability analysis models, provides useful mechanisms to evaluate these attributes. However, there are some limitations hampering the analysis and interpretation of these approaches. Representation structures of the results, state-explosion problems, technical specification difficulties and model inconsistencies are some challenges to be addressed.

Due to the complexity and difficulties emerged from these approaches there have been a shift in the use of model-based FI approaches. Instead of developing purely verification oriented FI approaches, model-based *integrative verification* approaches are gaining support. These works result from the integration of design, analysis and verification tasks. They are aimed at combining dependability analysis techniques examined within the group 5 (cf. Table IV) with FI approaches. They express system behaviour using a compositional model, which gathers nominal, failure and recovery behaviours. Integrating approaches using model transformations, allows using a single design model for dependability and verification analyses. As a result, limitations concerning the ease of use, consistency and completeness of the analyses and automated tool support are addressed.

Within the model-based integrative verification approaches, the overall verification process is specified as follows: the system *design model* specified using a Architectural Description Language (ADL) characterizes functional, failure and recovery behaviour and structure of system components. This design model needs to be transformed in a target *analysis model* which allows analysing and verifying system requirements. There are two options to carry out this transformation: (1) transformation of the design model to the target analysis approach through direct transformation rules; (2) intermediate transformation into a tool independent *Intermediate Model (IM)*, so that consistency and traceability between different design, analysis and verification approaches

Table VII
SUMMARY OF MODEL-BASED FAULT INJECTION APPROACHES

Works	*Extended Model*	*Analysis Model*	*Reqs.*	*Results*	*Specific Features*	*Tool Support*	*Limitations & Improvements*
[96]	NuSMV	NuSMV	CTL, LTL	FT	Model construction facility: library of FM and requirements; Unification of formal verification & safety analysis	FSAP/NuSMV-SA, Internally Available [102]	State explosion; FT structure; Probabilistic analysis; Failure ordering
[97]	CTMC (PRISM)	PRISM	CSL	pFMEA, CE	CE improvement; Auto. probability calculations; Multiple failures	PRISM [101]	State explosion; Pattern based FI
[98]	BT	SAL*	LTL	FMEA	FMEA automatic generation from BT models; Analysis reduction strategies; Effect analysis of multiple failures	BTE tool [103], SAL Symbolic MC [104]	CE readability; Timed FI; pFMEA; Reduction techs.
[99]	Simulink	PRISM*	CSL	Prob. calc.; CTMC	Systematic generation of analysis models from CFP design models	Partialy available [105]	No CE; State-explosion; Dynamic behaviour
[100]	Epsilon	FPTA, PRISM	CSL	CE	Integration of CFP approach and probabilistic model checking	FPTC toolset, available [48]	Failure prone manual transformation; Translate CE to design model

Legend: *CTL*: Computation Tree Logic; *LTL*: Linear Time Logic; *CSL*: Continuous Stochastic Logic; *CE*: Counter-Example; *BT*: Behaviour Tree; *FM*: Failure Modes; **Symbols**: *: Automatic Transformation;

are attained. In the second case, these models enable the reuse of the same high level models for different target approaches. Subsequently, the analysis of the corresponding approach (either FI, state-based or event-based analysis) makes use of the respective underlying characteristics (cf. Figure 6). FI approaches allows extracting counter-examples, which in turn can be transformed into dependability analysis models (indicated with dashed lines in Figure 6).

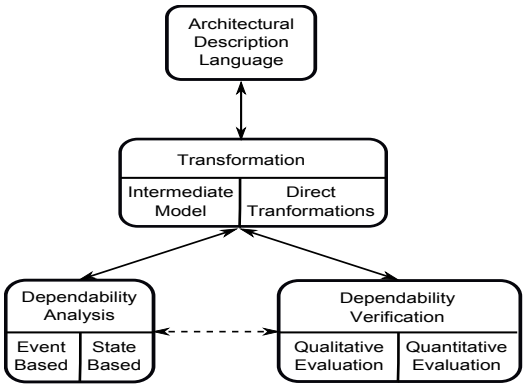


Figure 6. Structure of model-based integrative approaches.

The COMPASS project [106] is an integrative approach based on System-Level Integrated Modelling (SLIM) language which uses direct transformations [107]. The semantics of SLIM cover the nominal and error behaviour of AADL. The complete specification of SLIM consists of a nominal model, a failure model and a description of the effects of failures in the nominal model (*extended model*). Due to its underlying formal semantics, various types of analyses are possible: validation of functional, failure, and extended models via simulation and MC; dependability analysis and performance evaluation; diagnosability analysis and evaluation of the effectiveness of fault detection, isolation and recovery strategies.

Similarly, Güdemann and Ortmeier [108] proposed an intermediate tool-independent model called Safety Analysis Modelling Language (SAML). SAML describes a finite state automata, which is used to characterise the extended system model. This model specifies the nominal behaviour, failure occurrences, its effects and the physical behaviour of the surrounding environment. From this single model, quantitative and qualitative MC analyses are performed. The quantitative analysis, based on Deductive Cause Consequence Analysis (DCCA) [109], identifies minimal critical sets using CEs to indicate time-ordered combinations of failures causing the system hazard. The qualitative analysis, focusing on probabilistic DCCA (pDCCA), calculates per-demand and per-time failure probabilities.

Topcased project [110] aims at developing critical embedded systems including hardware and software. Topcased integrates ADLs and formal methods. The approach transforms high-level ADL models (SysML, UML and AADL) into an IM model specified in FIACRE language [111]. FIACRE specifies behavioural and timing aspects of high-level models making use of timed Petri nets primitives. Subsequent transformations of the IM model into MC tools (TINA and CADP) make it possible the formal verification and simulation of the specified requirements. TINA Petri Nets analyser [112] evaluates requirements specified in the state variant of LTL proposition logic (State/Event LTL (SELTL)) focusing on timeliness properties. CADP toolbox [113] transforms FIACRE models into LOTOS programs, which are handled by its underlying tools for validation via MC and simulation.

The pros and cons of the covered integrative works are summarized in the Table VIII.

The addressed works integrate well-known tools and formalisms. However, integration of analysis and verification

Table VIII
SUMMARY OF MODEL-BASED INTEGRATIVE APPROACHES

Works	*Design Model*	*Transf. Type*	*Analysis Model*	*Reqs.*	*Results*	*Specific Features*	*Tool Support*	*Future works*
[107]	SLIM	Direct	NuSMV*, MRMC*, RAT*	CTL, LTL, CSL	DFT, FMEA, prob. calc.	Req. patterns; Integrated verification, dependability and performance analyses of extended models	Toolset available for ESA members states [114]	Manual extension of the nominal model; Redundant FTA-FMEA results; State-explosion
[108]	SCADE, Simulink	IM: SAML	NuSMV*, PRISM*, MRMC*, Cadence SMV*	CTL, PCTL	DCCA, pDCCA	Combination of qualitative and quantitative analyses on the same model	S^3 tool [115]; publicly available	Manual extension of the nominal model; Transf. of ADLs (Simulink, Scade) into SAML; Req. patterns.; Library of FMs
[111]	SysML, UML, AADL	IM: FIACRE	TINA, CADP	LTL, SELTL	Timing, prob. calc.	Req. patterns; Integrated design, analysis and verification approaches	Topcased toolset [110]	State explosion; Back annotation of results

Legend: *IM*: Intermediate Model; *(P)CTL*: (Probabilistic) Computation Tree Logic; *LTL*: Linear Time Logic; *CSL*: Continuous Stochastic Logic; *SELTL*: State/Event LTL; *FMs*: Failure Modes; *Req*: requirements; *Transf*: transformations; **Symbols**: *: Automatic Transformation;

approaches when designing a dependable system is an ongoing research subject. There is an increasing interest in reusing and generalizing CFP approaches (e.g., transformation of CFP approaches into metamodels [43] [58] [59] and integration of CFP and verification approaches [100] [116]). Additionally, there exist other approaches closely related to the model-based integrative verification approaches, which cover the design process using formal analysis and verification approaches. For instance, under the correct by construction paradigm, the work presented in [117] matches with the idea of developing dependable systems by integrating specific approaches well-suited to each development phase. This methodology is based on a portfolio of different formalisms: starting from initial informal requirements, passing through formal requirements specification, modelling in Event-B and verification, towards the generation of executable code.

Interestingly, there still remain some challenges to be addressed. Back-annotation of the results into the design models would permit gaining more consistency between models. Additionally, it will lead to identifying the influence of the outcomes on the system components straightforwardly. Another issue is the size of the verification model extracted from the design model, which suffers from state-explosion problems.

The construction of a user friendly toolset integrating all these approaches is an issue itself. The last breakthroughs among integrative approaches' toolsets focus towards two main directions. On one hand, the automatic translations of design models into target specification formalisms is an unceasing goal [115]. These "push-button" toolsets avoid the designer to be exposed to the laborious, difficult and fail-prone process of creating a analysis model. For instance, in the COMPASS toolset [114], the designer is exposed to the SLIM formal specification language directly instead of using a ADL. However, one of the subtle problems underlying these tools is the size of the target models as a result of the design model transformations [110]. Another issue when creating these toolsets is related with the analysis results: the outcomes of the analysis results need to be displayed in a intuitive way to be understood by the designers and if necessary, take the corresponding countermeasures. To do so, they would benefit from an automatic transformation or back-propagation of the analysis results to the design model. Taking the automated tool support as an indicator to measure the potential to successfully integrate these approaches into the industrial practice, it is recognizable that all the covered approaches are working towards the construction of "push-button" automated toolsets to gain higher acceptance.

V. MODEL-BASED HYBRID DESIGN PROCESS

The goal of this section is not to provide a new design approach. Our aim is to make use of the reviewed analysis, design and verification approaches so as to outline a consistent and reusable model-based design process. This process arises from the structure of the model-based integrative verification approaches (cf. Section IV).

The separation of dependability analysis and verification tasks may lead to hampering the system design since results identified from either task need to be reconsidered during the design process (cf. Section III). On one hand, dependability analyses characterized by transformational approaches (cf. Subsection II-C), allow tracing from design models toward dependability analysis models. These approaches evaluate the dynamic system behaviour, as well as the effect of particular component failure occurrences at the system-level. On the other hand, purely verification oriented approaches mainly focus on the verification of the adequacy of the design model with respect to RAMS requirements. This is why we centred on covering integrative verification approaches.

When matching and tuning design components so as to find optimal design solutions satisfying design constraints, possible inconsistencies may arise due to the independent considerations of these approaches. This is why we should

focus on outlining a model-based hybrid design process, which unifies design, analysis and verification tasks. This process relies on initial system requirements, models, transformations and reuse of designer's considerations and results extracted from analysis and verification tasks (cf. Figure 7).

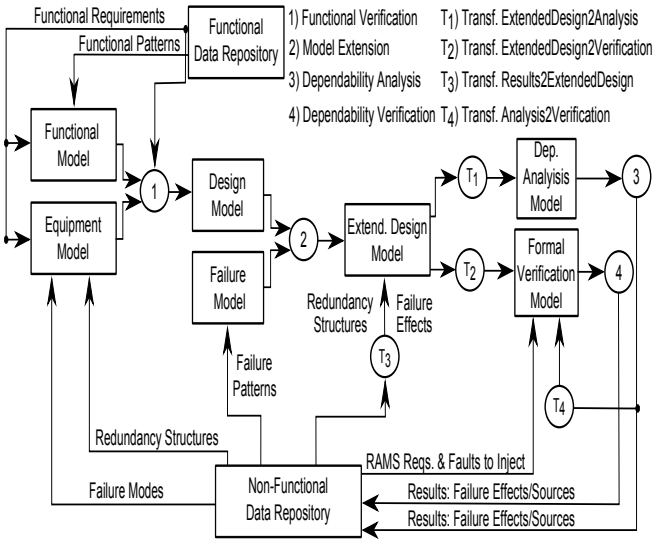


Figure 7. Model-based hybrid design process.

This design process starts from initial functional and physical characterizations. *Functional verification* analysis evaluates the adequacy of the allocation of the *functional model* into the *equipment model* according to functional requirements. The outcome of this process allows considering the verified *design model*. Subsequently, this model is *extended* with the *failure model* accounting for failure occurrences of the system. Failure patterns aid in the construction of the failure model allowing the reuse non-functional characterizations. Further, the effects of the considered failures and recovery strategies are annotated in the *extended design model* in order to counteract failure occurrences and its effects. With the aim to carry out *dependability analysis* and *formal verification* evaluations of the extended design model, twofold transformations need to be performed. The means to perform these transformations have been presented in Subsection II-C and Section IV respectively. Transformations of these models make the evaluation of the adequacy of the extended design model respect to RAMS requirements possible. Dependability analysis and verification tasks enable finding further failure effects and failure sources (apart from occurrence probabilities) either by CEs or dependability specific models. These results need to be transformed for design and analysis purposes. For the sake of reusing and refining the design process, data repositories have been considered consisting of annotation patterns for requirements and models (both functional and non-functional) and reusable recovery strategies.

On one hand, the outlined design approach enables benefiting from consistent design considerations. Moreover, data repositories allow the reuse of designer's characterizations as well as analysis results. Furthermore, user-friendly means make the annotation processes more evident. On the other hand, the automation of the extraction of dependability models hides information about the failure behaviour. Additionally, the flexibility of the approach depends on the system context, which would determine the reuse of functional and non-functional considerations.

VI. Conclusion and Future Work

Designing a dependable system, poses a wide variety of challenges on all its phases. This paper groups different approaches in order to identify and classify them.

The listed limitations guided the evolution of the analysis techniques towards Compositional Failure Propagation (CFP) and transformational analysis approaches. Automatic extraction of analysis models from design models is an ongoing research field, which leads to achieving consistency between design and analysis models.

However, this is not the cure-all remedy, which alleviates analysts from identifying and analysing failure behaviours, but helps obtaining a manageable analysis compared to the difficult and laborious traditional process. User friendly resources, such as design components or failure annotation libraries, enable the reuse of nominal and failure models.

Dependable design strategies have been characterized grouping them with respect to their underlying recovery strategy. Reuse of the existing hardware components is a promising solution to design dependable systems at the expense of reducing hardware costs. In this way, we have characterized the existing approaches addressing this design concept. Moreover, we have identified some of the activities which will help extending their use. Mainly, we have concentrated on the systematic identification of reusable resources and overcoming the assumptions of existing approaches, i.e., perfect fault detection, reconfiguration and communication.

For verification purposes, firstly Fault Injection (FI) approaches have been studied. Since the adoption of purely verification oriented FI approaches may incur complexities and difficulties in the system design process, then we have addressed model-based integrative verification approaches. Their main objective is to address consistently dependability analysis, design and verification tasks at the preliminary design phase. An early integration of these tasks would add value to the dependable design process. There are many challenging tasks to address when constructing an end-to-end dependable design methodology. Integration of the CFP approaches within this methodology or validation of the correctness of the faults to be injected are some of the subjects to be addressed.

Finally, we have outlined a model-based hybrid design process integrating addressed design, analysis and verifi-

cation approaches. The main purpose of this methodology is to sketch an intuitive model-based dependable design process attaining consistency and reuse among different models. The integration of the approaches should allow undertaking timely design decisions by reducing costs and manual failure-prone annotations. Additionally, it will alleviate the need to clutter a model with redundant information. Nevertheless, note that when designing a new system, special care should be taken, since reuse properties depend on the system context. The reuse of failure annotations in the design process, eases the architectural iterative refinement process. This makes possible the analysis of different implementations using the same component failure models.

Therefore, we foresee that instead of developing independent approaches to identify, analyse and verify dependability requirements, future directions will focus on integrating different approaches. This process requires tracing verification results to the initial dependable design model and vice versa. In this field, challenging work remains to do sharing information between existing approaches so as to take advantage of complementary strengths of different approaches.

Our current work focuses on the (re-)design of dependable systems by means of exploiting the benefits of heterogeneous redundancies, which may exist in some systems, e.g., trains or buildings. Our research challenge concentrates on the integration of dependability design and analysis activities, systematizing all the design steps, and overcoming assumptions adopted by other approaches for the system's operation and recovery process.

REFERENCES

[1] J. I. Aizpurua and E. Muxika, "Design of Dependable Systems: An Overview of Analysis and Verification Approaches," in *Proceedings of DEPEND 2012*, 2012, pp. 4–12.

[2] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Trans. Dependable Secur. Comput.*, vol. 1, pp. 11–33, January 2004.

[3] M. Rausand and A. Høyland, *System Reliability Theory: Models, Statistical Methods and Applications Second Edition*. Wiley-Interscience, 2003.

[4] A. Arora and S. Kulkarni, "Component based design of multitolerant systems," *IEEE Trans. on Sw. Eng.*, vol. 24, no. 1, pp. 63–78, 1998.

[5] M. Hiller, A. Jhumka, and N. Suri, "An approach for analysing the propagation of data errors in software," in *Proc. of DSN'01*, 2001, pp. 161–170.

[6] W. Vesely, J. Dugan, J. Fragola, Minarick, and J. Railsback, "Fault Tree Handbook with Aerospace Applications," NASA, Handbook, 2002.

[7] US Department of Defense, *Procedures for Performing, a Failure Mode, Effects, and Criticality Analysis (MIL-STD-1629A)*. Whasington, DC, 1980.

[8] M. A. Marsan, "Advances in petri nets 1989." Springer-Verlag, 1990, ch. Stochastic Petri nets: an elementary introduction, pp. 1–29.

[9] C. Baier and J.-P. Katoen, *Principles of model checking*. MIT Press, 2008.

[10] Y. Prokhorova, L. Laibinis, E. Troubitsyna, K. Varpaaniemi, and T. Latvala, "Derivation and formal verification of a mode logic for layered control systems," in *Proc. of APSEC'11*, 2011, pp. 49–56.

[11] P. Fenelon, J. McDermid, and M. Nicolson, "Towards integrated safety analysis and design," *ACM SIGAPP Applied*, 1994.

[12] J. Dugan, S. Bavuso, and M. Boyd, "Dynamic fault-tree models for fault-tolerant computer systems," *IEEE Trans. on Reliability*, vol. 41, no. 3, pp. 363–377, 1992.

[13] B. Kaiser, P. Liggesmeyer, and O. Mäckel, "A new component concept for fault trees," in *Proc. of SCS'03*, 2003, pp. 37–46.

[14] A. Galloway, J. McDermid, J. Murdoch, and D. Pumfrey, "Automation of system safety analysis: Possibilities and pitfalls," in *Proc. of ISSC'02*, 2002.

[15] C. Price and N. Taylor, "Automated multiple failure FMEA," *Reliability Eng. & System Safety*, vol. 76, pp. 1–10, 2002.

[16] M. Ĉepin and B. Mavko, "A dynamic fault tree," *Reliability Eng. & System Safety*, vol. 75, no. 1, pp. 83–91, 2002.

[17] Rao, K. Durga, V. Gopika, V. V. S. Sanyasi Rao, H. S. Kushwaha, A. K. Verma, and A. Srividya, "Dynamic fault tree analysis using Monte Carlo simulation in probabilistic safety assessment," *Reliability Eng. and System Safety*, vol. 94, no. 4, pp. 872–883, 2009.

[18] G. Manno, F. Chiacchio, L. Compagno, D. D'Urso, and N. Trapani, "MatCarloRe: An integrated FT and Monte Carlo Simulink tool for the reliability assessment of dynamic fault tree," *Expert Systems with Applications*, vol. 39, no. 12, pp. 10 334–10 342, 2012.

[19] "MathWorks," http://www.mathworks.com; Last access: 15/06/2013.

[20] S. Distefano and A. Puliafito, "Dynamic reliability block diagrams vs dynamic fault trees," *In Proc. of RAMS'07*, vol. 8, pp. 71–76, 2007.

[21] R. Robidoux, H. Xu, L. Xing, and M. Zhou, "Automated modeling of dynamic reliability block diagrams using colored petri nets," *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, vol. 40, no. 2, pp. 337–351, 2010.

[22] J.-P. Signoret, Y. Dutuit, P.-J. Cacheux, C. Folleau, S. Collas, and P. Thomas, "Make your petri nets understandable: Reliability block diagrams driven petri nets," *Reliability Engineering & System Safety*, vol. 113, pp. 61 – 75, 2013.

[23] "OpenSESAME: the simple but extensive, structured availability modeling environment," *Reliability Engineering & System Safety*, vol. 93, no. 6, pp. 857 – 873, 2008.

[24] I. Lopatkin, A. Iliasov, A. Romanovsky, Y. Prokhorova, and E. Troubitsyna, "Patterns for representing FMEA in formal specification of control systems," in *Proc. HASE'11*, 2011, pp. 146–151.

[25] "Event-B and the Rodin platform," http://www.event-b.org; Last access: 15/06/2013.

[26] M. Bouissou, "A generalization of Dynamic Fault Trees through Boolean logic Driven Markov Processes (BDMP)," in *Proc. of ESREL'07*, vol. 2, 2007, pp. 1051–1058.

[27] B. Kaiser, C. Gramlich, and M. Forster, "State-Event Fault Trees - A Safety Analysis Model for Software-Controlled Systems," *Reliability Eng. System Safety*, vol. 92, no. 11, pp. 1521–1537, 2007.

[28] D. Harel, "Statecharts: A visual formalism for complex systems," 1987.

[29] M. Steiner, P. Keller, and P. Liggesmeyer, "Modeling the effects of software on safety and reliability in complex embedded systems," in *Computer Safety, Reliability, and Security*. Springer Berlin Heidelberg, 2012, vol. 7613, pp. 454–465.

[30] TU Kaiserslautern, AG Seda and Fraunhofer IESE, "Embedded system safety and reliability analyzer (ESSaRel)," 2009, http://essarel.de; Last access: 15/06/2013.

[31] TU Berlin, Real-Time Systems and Robotics group, "TimeNET 4.0," 2007, http://tu-ilmenau.de/TimeNET; Last access: 15/06/2013.

[32] O. el Ariss, D. Xu, and W. E. Wong, "Integrating safety analysis with functional modeling," *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, vol. 41, no. 4, pp. 610–624, 2011.

[33] University of Virginia, "Galileo," 2003, http://http://www.cs.virginia.edu/~ftree/; Last access: 15/06/2013.

[34] Ferdinando Chiacchio, "RAATS Tool," 2012, http://www.dmi.unict.it/~chiacchio/?m=5&project=raatss; Last access: 15/06/2013.

[35] GRIF Workshop, "BStoK Module," http://grif-workshop.com/grif/bstok-module/; Last access: 15/06/2013.

[36] Max Walter, "OpenSESAME - Simple but Extensive Structured Availability Modeling Environment," 2009, http://www.lrr.in.tum.de/~walterm/opensesame/; Last access: 15/06/2013.

[37] EDF, "KB3 Workbench," 2012, http://research.edf.com/research-and-the-scientific-community/software/kb3-44337.html; Last access: 15/06/2013.

[38] P. Fenelon and J. A. McDermid, "An integrated tool set for software safety analysis," *J. Syst. Softw.*, vol. 21, pp. 279–290, 1993.

[39] Y. Papadopoulos, M. Walker, D. Parker, E. Rüde, R. Hamann, A. Uhlig, U. Grätz, and R. Lien, "Engineering failure analysis and design optimisation with HiP-HOPS," *Engineering Failure Analysis*, vol. 18, no. 2, pp. 590–608, 2011.

[40] D. Domis and M. Trapp, "Component-based abstraction in fault tree analysis," in *Computer Safety, Reliability, and Security*, ser. LNI. Springer, 2009, vol. 5775, pp. 297–310.

[41] R. Niu, T. Tang, O. Lisagor, and J. McDermid, "Automatic safety analysis of networked control system based on failure propagation model," in *Proc. of ICVES'11*, 2011, pp. 53–58.

[42] M. Walker and Y. Papadopoulos, "Qualitative temporal analysis: Towards a full implementation of the fault tree handbook," *Control Eng. Practice*, vol. 17, no. 10, pp. 1115–1125, 2009.

[43] R. Paige, L. Rose, X. Ge, D. Kolovos, and P. Brooke, "FPTC: Automated safety analysis for Domain-Specific languages," in *MoDELS Workshops '08*, vol. 5421, 2008, pp. 229–242.

[44] I. Wolforth, M. Walker, L. Grunske, and Y. Papadopoulos, "Generalizable safety annotations for specification of failure patterns," *Softw. Pract. Exper.*, vol. 40, pp. 453–483, 2010.

[45] C. Priesterjahn, C. Sondermann-Wölke, M. Tichy, and C. Hölscher, "Component-based hazard analysis for mechatronic systems," in *Proc. of ISORCW'11*, 2011, pp. 80–87.

[46] P. Struss and A. Fraracci, "Automated model-based fmea of a braking system," in *IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes*, vol. 8, 2012, pp. 373–378.

[47] OCC'M Software GmbH, "Raz'r Model Editor Ver. 3," http://www.occm.de; Last access: 15/06/2013.

[48] R. F. Paige, L. M. Rose, X. Ge, D. S. Kolovos, and P. J. Brooke, "Fptc: Automated safety analysis for domain-specific languages." in *MoDELS Workshops*, ser. Lecture Notes in Computer Science, M. R. V. Chaudron, Ed., vol. 5421. Springer, 2008, pp. 229–242.

[49] University of Paderborn, "FUJABA Tool Suite," 2012, http://www.fujaba.de/no_cache/home.html; Last access: 15/06/2013.

[50] P. Feiler and A. Rugina, "Dependability Modeling with the Architecture Analysis & Design Language (AADL)," Technical Note CMU/SEI-2007-TN-043, CMU Software Engineering Institute, 2007.

[51] "The Unified Modeling Language," http://www.uml.org/; Last access: 15/06/2013.

[52] OMG, "MDA Guide Version 1.0.1," http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf, 2003. [Online]. Available: http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf

[53] L. Fuentes-Fernández and A. Vallecillo-Moreno, "An Introduction to UML Profiles," *Journal of UML and Model Engineering*, vol. 5, no. 2, pp. 5–13, 2004.

[54] S. Bernardi, J. Merseguer, and D. Petriu, "Dependability modeling and analysis of software systems specified with UML," *ACM Computing Survey*, In Press.

[55] L. Montecchi, P. Lollini, and A. Bondavalli, "An intermediate dependability model for state-based dependability analysis," University of Florence, Dip. Sistemi Informatica, RCL group, Tech. Rep., 2011.

[56] B. Gallina, M. A. Javed, F. U. Muram, and S. Punnekkat, "Model-driven dependability analysis method for component-based architectures," in *Euromicro-SEAA Conference*. IEEE Computer Society, 2012.

[57] B. Gallina and S. Punnekkat, "FI^{4}FA: A Formalism for Incompletion, Inconsistency, Interference and Impermanence Failures Analysis," in *Proc. of EUROMICRO*, ser. SEAA '11. IEEE Computer Society, 2011, pp. 493–500.

[58] R. Adler, D. Domis, K. Höfig, S. Kemmann, T. Kuhn, J. Schwinn, and M. Trapp, "Integration of component fault trees into the UML," in *MoDELS'10*, 2010, pp. 312–327.

[59] M. Biehl, C. DeJiu, and M. Törngren, "Integrating safety analysis into the model-based development toolchain of automotive embedded systems," in *Proc. of LCTES '10*. ACM, 2010, pp. 125–132.

[60] A. Rugina, K. Kanoun, and M. Kaâniche, "A system dependability modeling framework using AADL and GSPNs," in *Architecting dependable systems IV, LNCS*, vol. 4615. Springer, 2007, pp. 14–38.

[61] A. Joshi, S. Vestal, and P. Binns, "Automatic Generation of Static Fault Trees from AADL models," in *DNS Workshop on Architecting Dependable Systems*. Springer, 2007.

[62] A. Arnold, G. Point, A. Griffault, and A. Rauzy, "The AltaRica formalism for describing concurrent systems," *Fundamenta Informaticae*, vol. 40, no. 2-3, pp. 109–124, 1999.

[63] B. Romain, J.-J. Aubert, P. Bieber, C. Merlini, and S. Metge, "Experiments in model based safety analysis: Flight controls," in *DCDS'07*, 2007, pp. 43–48.

[64] P. Bieber, C. Castel, and C. Seguin, "Combination of fault tree analysis and model checking for safety assessment of complex system," in *Proc. of EDCC'02*, vol. 2485. Springer, 2002, pp. 624–628.

[65] K. Mokos, P. Katsaros, N. Bassiliades, V. Vassiliadis, and M. Perrotin, "Towards compositional safety analysis via semantic representation of component failure behaviour," in *Proc. of JCKBSE'08*. IOS Press, 2008, pp. 405–414.

[66] M. Riedl and M. Siegle, "A LAnguage for REconfigurable dependable Systems: Semantics & Dependability Model Transformation," in *Proc. 6th International Workshop on Verification and Evaluation of Computer and Communication Systems (VECOS'12)*. British Computer Society, 2012, pp. 78–89.

[67] M. Riedl, J. Schuster, and M. Siegle, "Recent Extensions to the Stochastic Process Algebra Tool CASPA," in *Proceedings of the 2008 Fifth International Conference on Quantitative Evaluation of Systems*, ser. QEST '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 113–114.

[68] R. Cressent, V. Idasiak, F. Kratz, and P. David, "Mastering safety and reliability in a Model Based process," in *Proc. of RAMS'11*, 2011.

[69] "OMG Systems Modelling Language," http://www.omgsysml.org/; Last access: 15/06/2013.

[70] Eclipse Foundation, "Eclipse Papyrus," 2012, http://www.eclipse.org/papyrus/; Last access: 15/06/2013.

[71] Labri, "Altarica," http://www.altarica.labri.fr/; Last access: 15/06/2013.

[72] Carnegie Melon University, "OSATE," 2012, https://wiki.sei.cmu.edu/aadl/index.php/AADL_tools; Last access: 15/06/2013.

[73] CHESS partners, "CHESS Project," 2012, http://www.chess-project.org/; Last access: 15/06/2013.

[74] "ATESST2 Homepage," 2010, http://www.east-adl.fr; Last access: 15/06/2013.

[75] O. Lisagor, "Failure logic modelling: A pragmatic approach," Ph.D. dissertation, Department of Computer Science, The University of York, 2010.

[76] I. Lopatkin, A. Iliasov, and A. Romanovsky, "Rigorous development of dependable systems using fault tolerance views," in *Proceedings of the 2011 IEEE 22nd International Symposium on Software Reliability Engineering*, ser. ISSRE '11. IEEE Computer Society, 2011, pp. 180–189.

[77] C. P. Shelton and P. Koopman, "Improving system dependability with functional alternatives," in *Proceedings of the 2004 International Conference on Dependable Systems and Networks*, ser. Proceedings of DSN '04. IEEE Computer Society, 2004, pp. 295–304.

[78] L. Cauffriez, D. Renaux, T. Bonte, and E. Cocquebert, "Systemic modeling of integrated systems for decision making early on in the design process." *Cybernetics and Systems*, vol. 44, pp. 1–22, 2013.

[79] J. Clarhaut, S. Hayat, B. Conrard, and V. Cocquempot, "Optimal design of dependable control system architectures using temporal sequences of failures," *Ieee Transactions On Reliability*, vol. 58, no. 3, pp. 511–522, 2009.

[80] L. Cauffriez, J. Ciccotelli, B. Conrard, and M. Bayart, "Design of intelligent distributed control systems: a dependability point of view," *Reliability Engineering & System Safety*, vol. 84, no. 1, pp. 19–32, 2004.

[81] M. Adachi, Y. Papadopoulos, S. Sharvia, D. Parker, and T. Tohdo, "An approach to optimization of fault tolerant architectures using hip-hops," *Softw. Pract. Exp.*, 2011.

[82] J. Wysocki, R. Debouk, and K. Nouri, "Shared redundancy as a means of producing reliable mission critical systems," in *Proc. of RAMS'04*, 2004, pp. 376 – 381.

[83] J. Galdun, J. Ligus, J.-M. Thiriet, and J. Sarnovsky, "Reliability increasing through networked cascade control structure - consideration of quasi-redundant subsystems," in *IFAC Proc. Volumes*, vol. 17, 2008, pp. 6839–6844.

[84] O. Rawashdeh and J. Lumpp Jr., "Run-time behavior of ardea: A dynamically reconfigurable distributed embedded control architecture," in *IEEE Aerospace Conference Proceedings*, 2006.

[85] M. Trapp, R. Adler, M. Förster, and J. Junger, "Runtime adaptation in safety-critical automotive systems," in *Proc. of International Conference on Software Engineering*, 2007.

[86] R. Adler, I. Schaefer, M. Trapp, and A. Poetzsch-Heffter, "Component-based modeling and verification of dynamic adaptation in safety-critical embedded systems," *ACM Trans. Embed. Comput. Syst.*, vol. 10, no. 2, pp. 20:1–20:39, Dec. 2010.

[87] R. Adler, D. J. Domis, M. Förster, and M. Trapp, "Probabilistic analysis of safety-critical adaptive systems with temporal dependences," in *Proc. of RAMS'08*. IEEE Computer Society, 2008, pp. 149–154.

[88] R. Adler, D. Schneider, and M. Trapp, "Engineering dynamic adaptation for achieving cost-efficient resilience in software-intensive embedded systems," in *Proc. of Engineering of Complex Computer Systems*. IEEE Computer Society, 2010, pp. 21–30.

[89] J. Moore, *The Avionics Handbook*. CRC Press, 2001, ch. Advanced Distributed Architectures.

[90] P. Bieber, E. Noulard, C. Pagetti, T. Planche, and F. Vialard, "Preliminary design of future reconfigurable IMA platforms," *SIGBED Rev.*, vol. 6, no. 3, 2009.

[91] P. Bieber, J. Brunel, E. Noulard, C. Pagetti, T. Planche, and F. Vialard, "Preliminary Design of Future Reconfigurable IMA Platforms - Safety Assessment," in *27th International Congress of the Aeronautical Sciences*, 2010.

[92] C. Engel, A. Roth, P. H. Schmitt, R. Coutinho, and T. Schoofs, "Enhanced dispatchability of aircrafts using multi-static configurations," in *Proc. of ERTS'10*, 2010.

[93] M. Förster and D. Schneider, "Flexible, any-time fault tree analysis with component logic models," in *ISSRE*. IEEE Computer Society, 2010, pp. 51–60.

[94] J. I. Aizpurua and E. Muxika, "Dependable Design: Trade-Off Between the Homogeneity and Heterogeneity of Functions and Resources," in *Proceedings of DEPEND 2012*, 2012, pp. 13–17.

[95] C. Priesterjahn, D. Steenken, and M. Tichy, "Component-based timed hazard analysis of self-healing systems," in *Proceedings of the 8th workshop on Assurances for self-adaptive systems*, ser. ASAS '11. ACM, 2011, pp. 34–43.

[96] M. Bozzano and A. Villafiorita, "The FSAP/NuSMV-SA Safety Analysis Platform," *Int. J. Softw. Tools Technol. Transf.*, vol. 9, pp. 5–24, February 2007.

[97] H. Aljazzar, M. Fischer, L. Grunske, M. Kuntz, F. Leitner-Fischer, and S. Leue, "Safety analysis of an airbag system using probabilistic fmea and probabilistic counterexamples," in *QEST'09*. IEEE Computer Society, 2009, pp. 299–308.

[98] L. Grunske, K. Winter, N. Yatapanage, S. Zafar, and P. A. Lindsay, "Experience with fault injection experiments for fmea," *Software: Practice and Experience*, 2011.

[99] A. Gomes, A. Mota, A. Sampaio, F. Ferri, and E. Watanabe, "Constructive model-based analysis for safety assessment," *International Journal on Software Tools for Technology Transfer*, vol. 14, pp. 673–702, 2012.

[100] X. Ge, R. Paige, and J. McDermid, "Probabilistic failure propagation and transformation analysis," in *SAFECOMP'09*, 2009, vol. 5775, pp. 215–228.

[101] M. Z. Kwiatkowska, G. Norman, and D. Parker, "Prism 4.0: Verification of probabilistic real-time systems." in *Computer Aided Verification*, ser. Lecture Notes in Computer Science, G. Gopalakrishnan and S. Qadeer, Eds., vol. 6806. Springer, 2011, pp. 585–591.

[102] FBK, "The FSAP/NuSMV-SA Web Page," https://es.fbk.eu/tools/FSAP/; Last access: 15/06/2013.

[103] "Behaviour Tree Editor (BTE)," http://www.sqi.griffith.edu.au/gse/tools/overview.html; Last access: 15/06/2013.

[104] "Symbolic Analysis Laboratory (SAL)," http://sal.csl.sri.com; Last access: 15/06/2013.

[105] Alexandre Mota, "Simulink to prism," http://www.cin.ufpe.br/~acm/simulinktoprism/; Last access: 15/06/2013.

[106] "Correctness, Modelling and Performance of Aerospace Systems," http://compass.informatik.rwth-aachen.de; Last access: 15/06/2013.

[107] M. Bozzano, A. Cimatti, J.-P. Katoen, V. Nguyen, T. Noll, and M. Roveri, "Safety, dependability and performance analysis of extended AADL models," *Computer Journal*, vol. 54, no. 5, pp. 754–775, 2011.

[108] M. Güdemann and F. Ortmeier, "Towards model-driven safety analysis," in *Proc. of DCDS 11*, 2011, pp. 53 – 58.

[109] M. Güdemann, F. Ortmeier, and W. Reif, "Using Deductive Cause-Consequence Analysis (DCCA) with SCADE," in *Proc. of SAFECOMP'07*, vol. 4680, 2007, pp. 465–478.

[110] "The Open-Source Toolkit for Critical Systems," http://www.topcased.org; Last access: 15/06/2013.

[111] B. Berthomieu, J.-P. Bodeveix, P. Farail, M. Filali, H. Garavel, P. Gaufillet, F. Lang, and F. Vernadat, "Fiacre: an intermediate language for model verification in the topcased environment," in *ERTS'08*, 2008.

[112] B. Berthomieu, P.-O. Ribet, and F. Vernadat, "The tool TINA - Construction of abstract state spaces for petri nets and time petri nets," *International Journal of Production Research*, vol. 42, no. 14, pp. 2741–2756, 2004.

[113] J.-C. Fernandez, H. Garavel, A. Kerbrat, L. Mounier, R. Mateescu, and M. Sighireanu, "CADP - A Protocol Validation and Verification Toolbox," 1996.

[114] COMPASS Consortium, "Compass toolset," http://compass.informatik.rwth-aachen.de/download.html; Last access: 15/06/2013.

[115] M. Lipaczewski, S. Struck, and F. Ortmeier, "Using Tool-Supported Model Based Safety Analysis - Progress and Experiences in SAML Development," in *IEEE 14th International Symposium on High-Assurance Systems Engineering (HASE 2012)*, 2012.

[116] A. Gomes, A. Mota, A. Sampaio, F. Ferri, and J. Buzzi, "Systematic model-based safety assessment via probabilistic model checking," in *ISoLA'10*. Springer, 2010, pp. 625–639.

[117] R. Gmehlich, K. Grau, M. Jackson, C. Jones, F. Loesch, and M. Mazzara, "Towards a formalism-based toolkit for automotive applications," Computing Science, Newcastle University, Tech. Rep., 2012.

Towards Next Generation Malware Collection and Analysis

Christian Martin Fuchs
Department of Computer Science
Technische Universität München (TUM)
Munich/Garching, Germany
christian.fuchs@tum.de

Martin Brunner
Martin Brunner Security
Munich, Germany
martin.brunner@x90.eu

***Abstract*—The fast paced evolution of malware has demonstrated severe limitations of traditional collection and analysis concepts. However, a majority of the anti-malware industry still relies on such ineffective concepts and invests much effort into temporarily fixing most obvious shortcomings. Ultimately fixing outdated concepts is insufficient for combating highly sophisticated future malicious software, thus new approaches are required. One such approach is AWESOME, a novel integrated honeypot-based malware collection and analysis framework. The goal of our collection and analysis system is retrieval of internal malware logic information for providing sufficient emulation of protocols and subsequently network resources in real time. If protocol emulation components are trained sufficiently, a larger setup could even allow for malware analysis in an isolated environment, thus offering side-effect free analysis and a better understanding of current and emerging malware. In this paper, we present in-depth information on this concept as well as first practical results and a proof of concept, indicating the feasibility of our approach. We describe in detail many of the components of AWESOME and also depict how protocol detection and emulation is conducted.**

Keywords-malware collection; malware analysis and defense

I. Introduction

Cyber crime has become one of the most disruptive threats today's Internet community is facing. The major amount of these contemporary Internet-based attacks is thereby attributed to malware, which is usually organized within a botnet in large-scale scenarios. Such botnet-connected malware is on their part utilized for infecting hosts and instrumenting them for various malicious activities: most prominent examples are Distributed Denial of Service (DDoS) attacks, identity theft, espionage and Spam delivery [6][24][39]. Botnet-connected malware can therefore still be considered a major threat on today's Internet.

Due to the ongoing spread of IP-enabled networks to other areas we expect the threat posed by botnet-connected malware to increase and moreover reach further domains in public and private life. Thus, there is a fundamental need to track the rapid evolution of these pervasive malware based threats. Especially timely intelligence on emerging, novel threats is essential for successful malware defense and IT early warning. This requires both, acquisition and examination, of current real-world malware samples in sufficient quantity and variety, commonly acquired through meticulous analysis of the most recent samples. The evolution of malware over time has led to the development of intensive obfuscation and anti-debugging mechanisms, as well as a complex and multi-staged malware execution life cycle [37]. Each phase may include numerous measures aimed at maximizing installation success and reliability. In order to comprehensively analyze the full life cycle, the malware under analysis must have unhindered access to all requested resources during runtime. While this could easily be achieved by allowing full interaction with the Internet, this is not a viable approach in setups, which are forced to consider liability issues.

In this paper, we introduce AWESOME [1], which is short for: Automated Web Emulation for Secure Operation of a Malware-Analysis Environment. It is a novel approach for integrated honeypot-based malware collection and analysis. The overall goal of AWESOME is to capture and dynamically analyze novel malware on a large scale. To identify trends of current and emerging malware, we aim to cover the entire execution life cycle. That is, we want to track malware communicating via both known and unknown (C&C) protocols in an automated way within a controlled environment. In order to minimize harm to third parties, malware should by default have no Internet access during analysis. The whole procedure intends to trick a sample into believing it is running on a real victim host with full Internet access.

II. Background and Related Work

A. Malware Life-Cycle

Due to the predominant economic motivation for malicious activities backed by organized cyber crime also the sophistication of malware and the respective propagation methods continuously evolved, hence increasingly impeding malware defense. Thereby, the invested effort and the achieved result must be in a reasonable relation for a professional attacker. Thus, we experience the phenomena of a moving target. That is, cyber criminals chose their targets and attack vectors according to the best economic relation and an ongoing paradigm shift towards client-side and targeted attacks has been witnessed in recent years [6].

Widely spread malware is most effectively managed within a botnet, therefore, a newly compromised host is still to become a botnet-member. Many findings from recent research indicate an ongoing specialization of the various groups in the underground economy offering ”Malware as a Service” and ”pay per install” schemes including elaborated models for pricing, licensing, hosting and rental [9][18][24][26]. This often includes professional maintenance, support and service level agreements for the purchasable malware itself as well as innovations in the maintenance of infected victim hosts.

Therefore, there is

1) one group specializing on the development of the actual malware,
2) a second group deals with the operation platform and the distribution of malware (i.e., to establish botnets) and
3) a third group focuses on suitable business models.

As a result, also the actual malware itself evolved with respect to obfuscation techniques and anti-debugging measures. Thus, current malware checks for several conditions before executing its malicious tasks, such as hardware resources of the victim host, Internet connectivity or whether it is executed within a virtualized environment [16][34][46].

With respect to the facts outlined previously and findings of related work [9][37] we model the execution life cycle of today's (autonomous spreading) malware as depicted in Figure 1 [5]. Particular stages within this life cycle may differ depending on the malware type and are accordingly addressed in separate work (e.g., the life cycle of Web-based malware has been analyzed in [40]). A common setting consists of three phases:

1) **Propagation and exploitation**
   Within this initial phase a worm spreads carrying a malicious payload that exploits one or multiple vulnerabilities. In this context, a vulnerability encompasses also the human using social engineering techniques thus possibly requiring user interaction. Furthermore, a vulnerability may be OS based (e.g., a flaw in a network service) or - more commonly - application based. The latter includes specifically vulnerabilities in browsers, their extensions (such as Adobe's flash), Email clients (e.g., attachment, malicious links, Emails containing malicious script code, etc.) and other online applications such as instant messaging clients.
   The malicious payload of the worm may instrument a variety of attack vectors reaching from (classical) buffer and heap overflows to recent return oriented programming techniques [47], while evading appropriate countermeasures such as address space layout randomization (ASLR), data execution prevention (DEP) and sandboxing [30]. After successfully exploiting a vulnerability, a shellcode is placed on the victim host, which gets then extracted and executed, including decryption and deobfuscation routines when necessary.
2) **Infection and installation**
   As a result of executing the injected shellcode, a binary is downloaded and placed on the victim host. This binary typically is a so-called *dropper*, which contains multiple malware components. It is supposed to disable the security measures on the victim host, to hide the malware components and to obfuscate its activities before launching the actual malware. It is synonymously referred to as *downloader*, which has the same features except that it does not contain the actual malware but downloads it from a remote repository resulting in a smaller size [37]. As there is an emerging trend that multiple cyber criminals instrument a single victim host for their malicious purposes also several droppers may be installed (in parallel) within this step.
   Once the dropper is executed it extracts and installs further components responsible for hardening and updating tasks. That is, they prepare the system for the actual malware using embedded instructions. These tasks include, e.g., disabling security measures, modifying configurations and contacting a remote site for updates ensuring that the actual malware is executed after every reboot and impeding its detection and removal. After the update site has verified the victim host as ”real“ and probably worth getting compromised, it provides the dropper components with information on how to retrieve the actual malware (e.g., via an URL) and updated configurations, when necessary. Again, this may include multiple binaries each representing a different botnet.
   Once downloaded, the malware is executed by the dropper component installing its core components. Finally, these core components remove all other (non-vital) components resulting from previous stages and the malware is operational.
3) **Operation and maintenance**
   Initially, the malware launches several actions, which are intended to directly gain profit from the victim in case the attacker loses control over the compromised host later on. Therefore, the malware harvests valuable information such as credit card numbers and all kinds of authentication credentials and sends it as an encrypted file to a remote server under the control of the attacker. Next, the malware attempts to establish a communication channel to the attackers command and control (C&C) infrastructure awaiting further instructions. These may include commands to launch different malicious actions but also maintenance operations such as retrieving updates, further propagation or even to terminate and remove the malware.

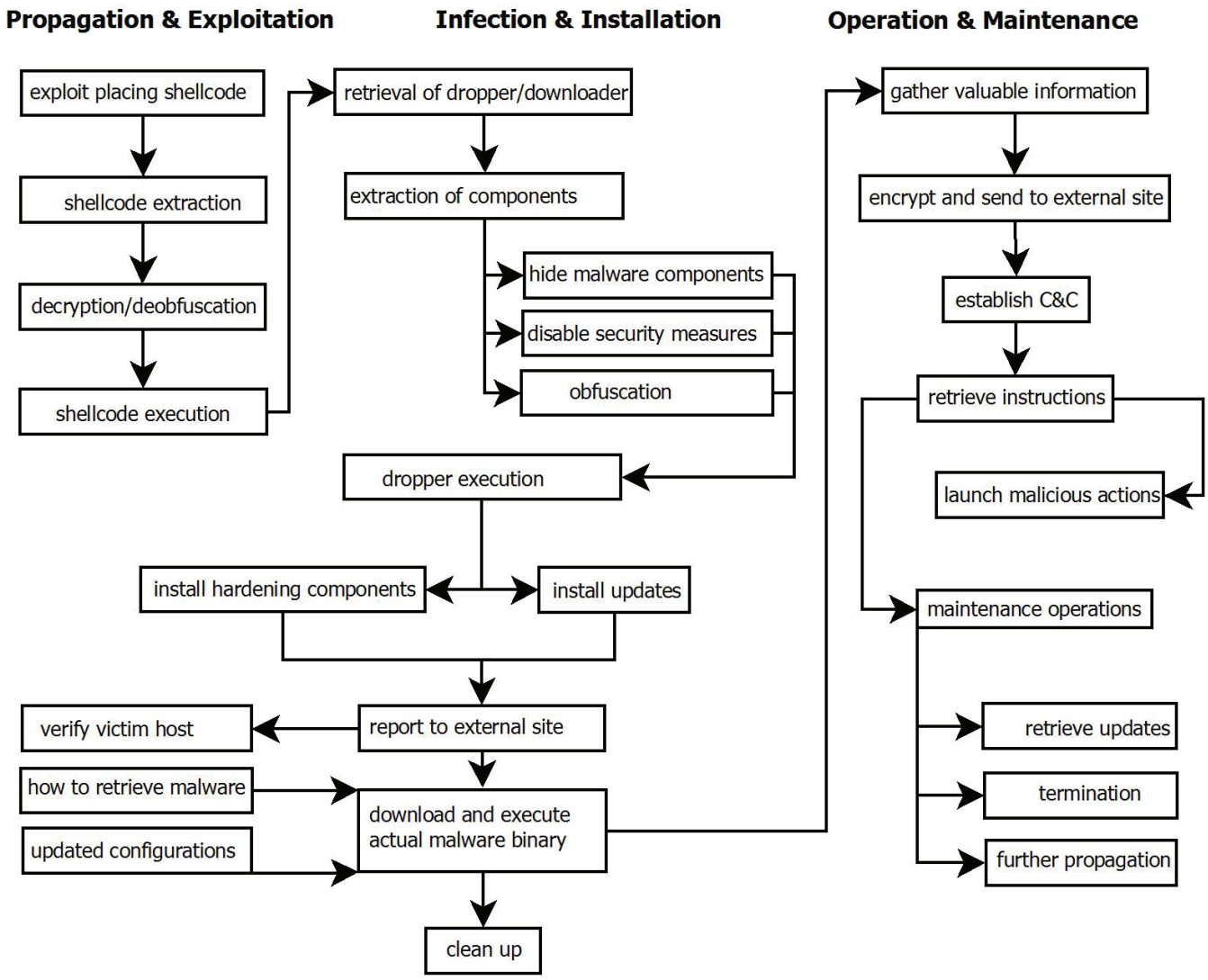


Figure 1. Today's multi-staged, complex malware life-cycle.

The various steps of the outlined, complex malware life cycle include many checks and measures to increase the resilience of the various features each intended to maximize the success of the malware installation, ensuring a reliable operation and to protect the cyber criminals from being tracked down. Thus, the reasons for this complex life cycle, especially the use of droppers in an intermediate step, are apparent:

- The dropper components evade discovery of system compromise. Also, the adversary has no need to distribute the core malware components in the first phase thereby impeding the successful collection and thus detection and mitigation of the malware.
- In addition, he can distribute the malware more selective and targeted.
- Finally, the attacker can ensure, that the victim host is real (i.e., not a honeypot or virtualized analysis system) and that it is worth being compromised (e.g., by checking available resources).

Among others, this implicates, that for a comprehensive analysis of a given malware it must receive all resources that it requests during its life cycle, since it may behave different or refuse to execute at all otherwise.

B. Combating Malware

One major issue that makes malware analysis a very challenging task, is the ongoing arms race between malware authors on the one hand and malware analysts on the other hand. That is, while analysts use various techniques to quickly understand the threat and intention of malware, malware authors invest considerable effort to camouflage their malicious activity and impede a successful analysis.

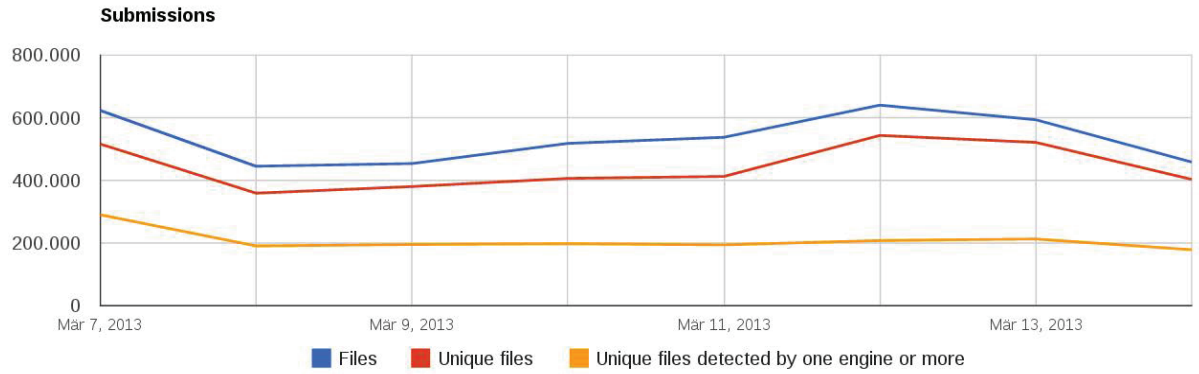


Figure 2. Amount of malware sample submissions to virustotal.com over a recent randomly chosen period.

While in past decades an explosive spread of identical malware has been observed, nowadays malware is mostly created utilizing creation kits. As a result, we experience a diffusion of malware variants, which are designed to be difficult to identify and analyze. Accordingly, a vast amount (i.e., hundreds of thousands) of new malware shows up *per day*. This is illustrated in Figure 2, which outlines the number of malware samples submitted to Virustotal based on a randomly chosen recent period.

Thereby the correlation between the total number of samples and the unique ones is notable, since they correspond to each other. This suggests that the majority of new malware can be attributed to be just new variants of already existing malware. That is, while most of the malware samples are unique (i.e., have different file hashes) they are in fact not. But since they are considered to be unique all of them need to be analyzed.

Only once analyzed and the threat posed by a given malware sample is estimated a corresponding anti-virus (AV) signature (i.e., a characteristic byte sequence) can be created. As a result, the vast amount of new malware introduces the need for automation of the entire malware estimation process (i.e., collection and analysis).

Advanced large-scale malware collection and analysis infrastructures, such as [2][10][15], can satisfy the requirements for automated tracking of malware, but suffer from several limitations:

1) Despite being executed within an isolated environment, samples must be supplied with requested network services during analysis. Otherwise, the ability to achieve high-quality results is impeded, potentially causing different malware behavior, refusal of execution or even blacklisting of the collector's address space.
   While certain services can be offered using sinkholing techniques, existing approaches are purely network-based, and reactions to malware-initiated connection attempts remain static during runtime. Predefined commonly-used services are offered by these infrastructures to the malware; however, other requests can not be handled accordingly.
2) High-interaction (HI) honeypots pose, aside from their complexity and maintenance issues, high operational risks. These is often inadequately addressed. While there are many methods of mitigation, the remaining risk is still higher than with low-interaction (LI) honeypots, resulting in ethical questions and possibly even legal and liability issues for the operating organization.
3) Existing approaches separate collection and analysis, thus forfeiting the system context (i.e., file handles, requests, sockets) of the victim host. While such separation is not necessarily a limitation (it may not be mandatory to gain qualitative analysis results), we argue that this loss of information hinders analysis and may degrade analysis results or prevent analysis of certain malware altogether.

C. Logic Analysis Using Virtualization

Like modern intrusion detection systems, protocol detection and sinkholing frameworks should not be limited to knowledge about traffic passing through the network. Instead, such systems should incorporate information on the involved systems as well. The tracking of library functions using API-hooks in user or kernel space is one possibility to do so and very popular in malware analysis [27][31][50].

System call tracing has long been known as a solution for profiling and detecting software behavior [17]. While access to privileged operations are usually implemented in the same way on most modern operating systems (OS) (figure 3), not all OSes offer an interface for system call auditing. For operating systems without such an interface, a monitoring functionality has to be implemented by hand, requiring complex and extensive modifications to guests' internal system call handling mechanisms [7].

Hooking of library functions allows finer-grained tracking of a sample's activities than system call tracing. However,

if implemented within the same context of execution as malware is run in, API-hooking is equally susceptible to detection and circumvention by adversaries but requires fewer changes to the operating system's internal mechanisms [7]. Thus, this method of behavior analysis is preferred in most sandboxes [27][31][50].

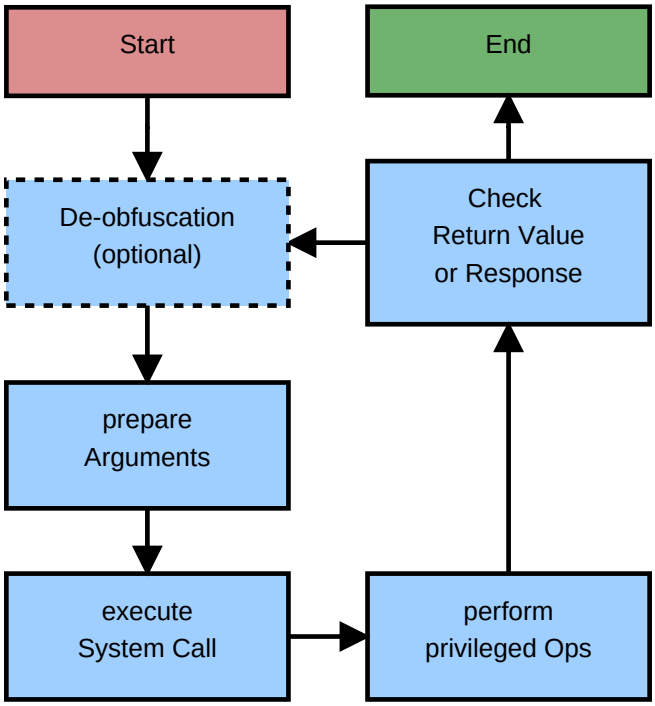


Figure 3. Privileged operations, such as network or disk operations, cannot be executed in user space. Software uses system calls to perform these operations in kernel space.

Due to possibilities offered by virtualization techniques, system call tracing became more popular again. Initially, Tal Garfinkel and Mendel Rosenblum proposed the use of Virtual Machine Introspection (VMI) to extract certain information from a virtual machine (VM) [21]. VMI can also be used to create and enforce behavioral policies [17]. Process activity and the state of a guest's (virtual) hardware components may be used for analysis, too.

We can utilize VMI to extract internal information from a process, monitor and control its behavior. If a program exhibits unexpected behavior or uses system calls or library functions, which are outside of the programs specifications, an intrusion can be assumed.

III. Approach

A. Basic Concept

To identify the services and protocols required in the next step of the execution life cycle of a sample, we intend to harvest information on internal malware logic during execution. In contrast to purely network-based approaches, our method also operates at the binary level, directly interacting with the malware's host system. Therefore, it aims to integrate network-based analysis and binary analysis, as in [48].

As depicted in Figure 4, the presented AWESOME approach [1] is based on an HI honeypot and a virtual machine introspection framework. We enhance our architecture with a transparent pause/resume functionality, which is instrumented to determine and, if needed, interrupt the program flow. Hence, we enable the extraction and alteration of program logic and data within the victim environment during runtime. This is specifically valuable for extracting protocol information and cryptographic material embedded within malware in order to determine the protocol type and intercept encrypted communication.

After checking, extracted information is forwarded to a service handler (SH) and sinkholing service in order to maintain full control over all interactions between the malware and the outside world. For handling unknown traffic as well, finite state machines (FSM) are automatically derived from the observed traffic and used for service emulation. An important goal of automating of the whole collection and analysis process is to handle large amounts of malware while allowing scalability.

B. Added Value

The system context of the malware collection facility persists and is also used in the subsequent analysis. The capabilities resulting from the merge of collection and analysis is similar to the approach used in HI honeypots. Thus, it is more closely aligned to real-world scenarios than LI honeypots. In addition, we achieve increased transparency during analysis due to the use of VMI. We consider this to be a benefit, since we argue that VMI based analysis is more likely to remain undetected by malware.

Compared to other techniques, VMI requires no trusted support components, which could be compromised [14] inside the sample's context of execution. Hence, the approach is more likely to observe the entire malware execution life cycle.

We are able to extract and inject data as well as instructions from or into the memory of the infected virtual machine (VM) during runtime (for example, in order to tap and manipulate encrypted C&C traffic). Since our approach does not depend on analysis components within the VM, we believe it to be more secure while also expecting better overall performance. Moreover, we are able to control any interaction between malware and third party systems. Thus, our architecture can fulfill legal and liability constraints.

Since our approach is applied directly at the instruction level, we are aware of the actions initiated by the malware, thus allowing us to provide matching services and even to service novel communication patterns. Subsequently, the risk resulting from HI honeypot operation is minimized.

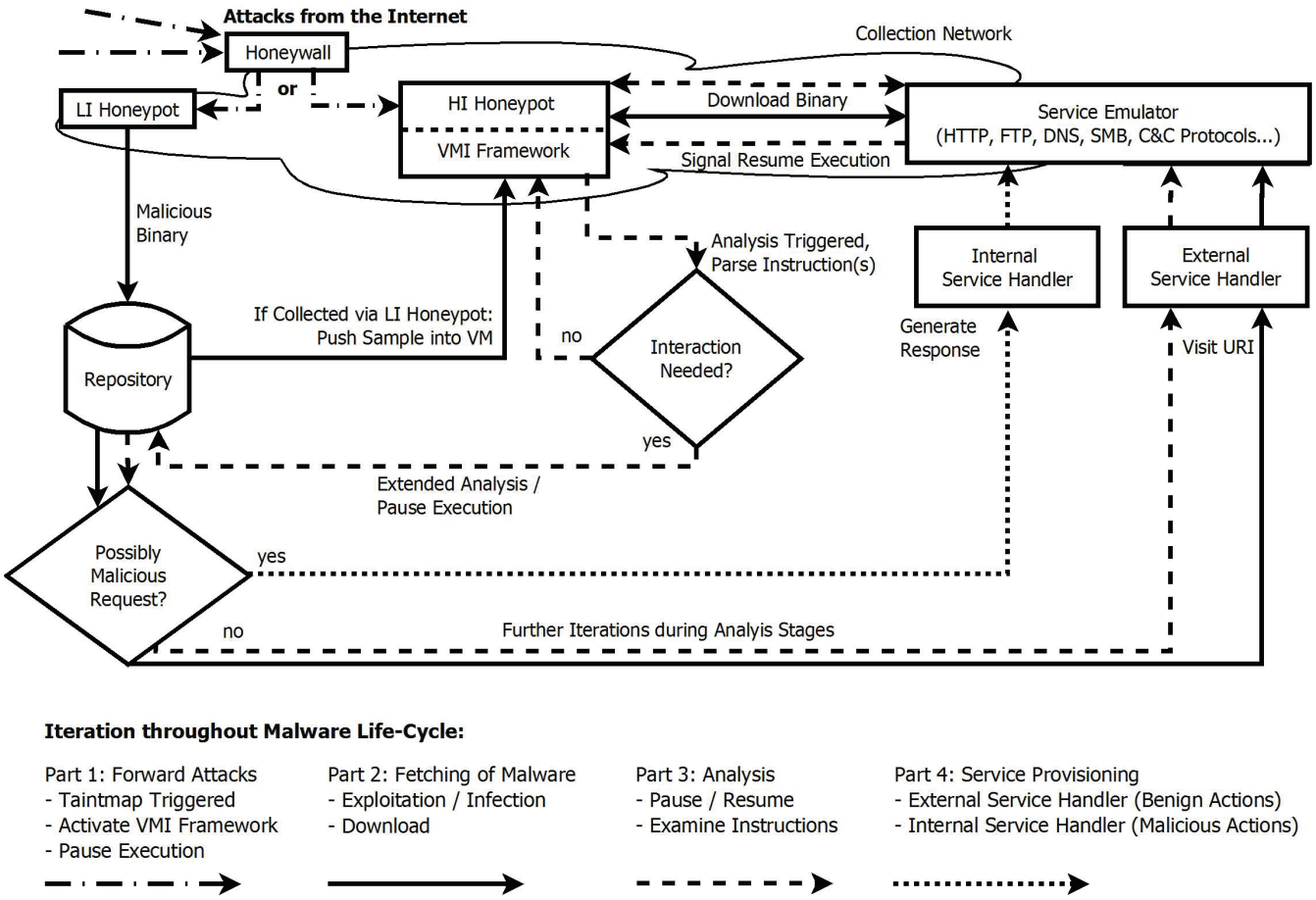


Figure 4. General design of the presented approach.

IV. Design and Implementation

The AWESOME approach [1] utilizes the following components:

- For *malware collection*, a modified ARGOS HI honeypot [41] is used.
- *Malware analysis* is conducted based upon Nitro [38], a KVM-based framework for tracing system calls via VMI. In particular, we determine whether a given action initiated by the currently-analyzed malware requires Internet access and thus apply a complex rule-set to the tracing component.
- Our *service provisioning* component manages all malware-initiated attempts to request Internet resources. Malicious attempts are handled via an appropriate sinkholing service spawned by Honeyd [44], and unknown traffic patterns may be handled utilizing ScriptGen [33].

While most popular LI honeypots have proven to be efficient for malware collection, their knowledge-based approach has also drawbacks regarding the quantity and diversity of the collected malware [51]. With respect to our primary goal (to handle unknown malware), we chose to apply the taint-map-based approach of ARGOS, since it allows the detection of both known and unknown (0-day) attacks. In addition, it is independent of special collection mechanisms. Moreover, it can cooperate with the KVM based VMI framework Nitro. Hence, several components were modified:

1) The victim VM's RTC is detached from the host's clock, since ARGOS is more time-consuming than traditional approaches and thus detectable by an abnormal latency and timing-behavior;
2) Once the taint-map reports tainted memory being executed, we activate the analysis functionality provided by the VMI framework; and,
3) Simple interpretation and filtering of system calls and their parameters is conducted directly within hypervisor space, while more complex analysis is performed via the VMM in the host environment [19].

The entire process consists of three parts (collection, analysis, and service provisioning) and is structured as described below. The steps are repeated iteratively throughout the entire life cycle of the malware.

A. Malware Collection

As a HI honeypot, ARGOS requires much effort in deployment and maintenance. Furthermore, one of the main drawbacks of ARGOS is its poor performance, which is among others related to the overhead caused by the taint mapping technique. To overcome this limitation, we deploy a two stage malware collection network (i.e., a hybrid honeypot system similar to the ones described in [3][28][49]) as outlined in Figure 5.

We take advantage of our existing honeyfarm infrastructure ([4][20]). This malware collection network consists of various honeypots and honeypot-types. Especially, client honeypots play an increasingly important role since the approach of this honeypot type covers the detection of state of the art attack vectors thus enabling client honeypots to capture current malware that may have not been collected using server honeypots.

The honeyfarm utilizes a large-scale network telescope (in particular a /16 darknet) serving the various honeypots. We use this infrastructure in order to filter noise and known malware (in particular everything that can be handled by the LI honeypots or their vulnerability handling modules respectively). The so collected binaries (which we consider to be mostly shellcode containing URLs and droppers) are stored in the central repository. Based on the file-hash known files are distinguished from novel ones. Only novel attempts are forwarded to the ARGOS HI honeypot, which then does the further processing. By doing so we minimize the load on ARGOS and thus justify its operation.

B. Malware Analysis

Dynamic malware analysis utilizing virtualization can be detected and thus evaded by environment-sensitive malware [16][19][34][46]. Hence, our goal is to achieve a reasonably transparent dynamic malware analysis without claiming the approach to be completely stealthy. However, we also consider VMI as the most promising available approach to evade malware's anti-debugging measures due to its minimal footprint. Thus, in order to provide the best chance at evading detection while still gaining the benefits of VMI, we have chosen Nitro since it offers several advantages regarding performance and functionality in comparison to other publicly available tools such as Ether (see [38]).

As Nitro is based on KVM, we have - in addition to guest portability - full virtualization capability, thanks to the host CPU's virtualization extensions. Thus, we can expect reasonable performance.

During the analysis process, we expect a malicious binary to be shellcode or a dropper rather than the actual malware binary. This initially retrieved binary is then decoded and usually contains a URL pointing at the resource used for deploying the next stage of the malware. In the second iteration, execution of this binary continues after it has been downloaded and the VM has been resumed. The resulting system call trace is then examined for routines related to connection handling (e.g., NTConnectPort). If present, we transparently pause execution of the VM and forward related traffic to the service provisioning component. The following sections outline the methods for dynamic malware analysis in more detail.

1) Subroutine Logic Analysis: Based on the executed system call's associated parameters and the memory of the calling process, certain information can be extracted directly from a thread. Traditional semantics checking may prove to be efficient for detecting known subroutines and protocol implementation [42].

Promising candidates for detection are operations and checks performed on the expected peer's response as well as common library functions and routines [50]. A sample's code could be checked for well-known standard algorithms, like routines used to generate symmetric checksums and cryptographic hash functions imported from libraries such as the de-facto standard implementations in OpenSSL or GnuTLS.

If the library function or algorithm being used can be detected, the logical next step is to extract input data. A function should perform sanity checks on its parameters, and on data returned from called functions. Operating systems APIs for use by customers are usually documented; thus, their expected input and return values are known.

Based on this information, the information returned by a function can be analyzed. Such data might be a checksum or signature against which a response was checked or simply a string or sequence of bytes.

The currently decoded part of malware can be analyzed. Sequences of system calls can then be used to reveal more and more deobfuscated parts of malware. Sometimes, it may be easier to not immediately start analysis, but continue execution and wait for a pre-known event to occur. In the case of network communication, analysis may be delayed until malware has sent a packet, and it leaves the virtual machine's (VM) network interface.

2) Delayed Analysis Triggering: Current virtualization solutions try to keep the software layer between physical hardware and the VM as slim as possible in order to improve overall performance. Auxiliary components like network interfaces, however, exist purely in software. To increase the quality of results, subsystems of the VMM and virtual hardware could be used as additional information sources. System call tracing could be used to activate secondary analysis functionality integrated within emulated hardware.

When using hardware assisted virtualization, certain physical devices can be forwarded to a guest exclusively. Ignoring this feature, the emulation code for virtual network interfaces (VIFs) can be extended to hold or trigger analysis components. In the case of qemu, code related to delayed analysis checking could reside, for instance in the virtual network interface (nic.c), as shown in Figure 6.

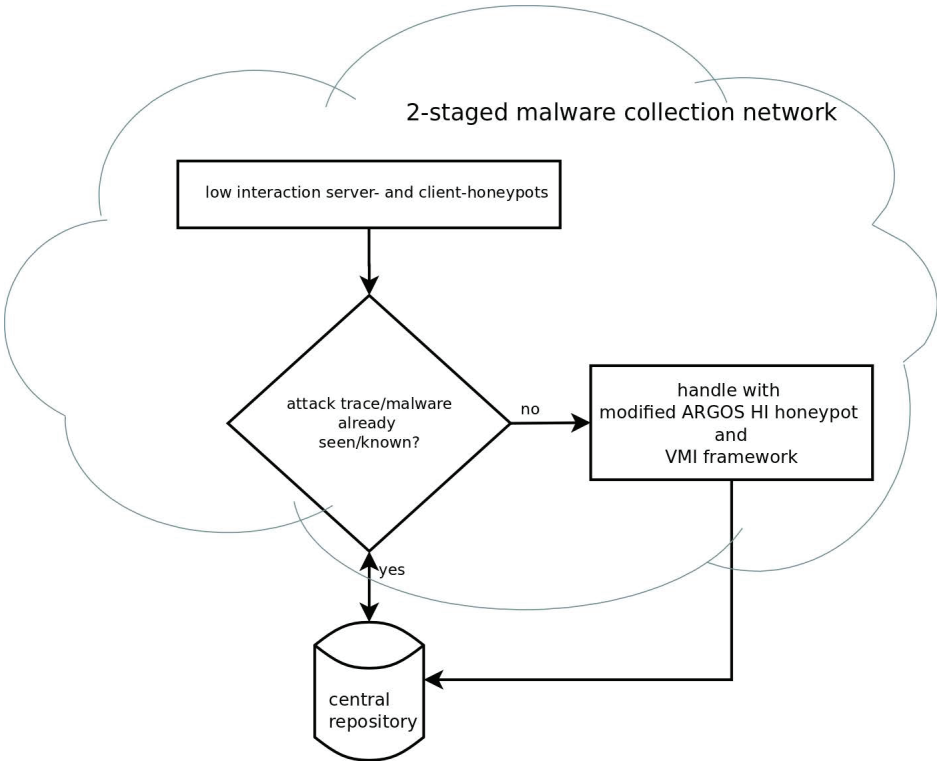


Figure 5. Scheme of the two-stage malware collection network.

Compared to immediate analysis upon execution of a system call, delayed analysis may ensure that malware executes a little bit more of its functionality, finishing execution of its privileged operation (e.g., opening a socket and connecting to a remote host) and returns to ring 3. Malware will normally fall back into a wait state (waiting for a peer's response) after sending a packet, resulting in more code being deobfuscated.

3) System Call Sequence Based Behavior Identification: The approach used for tracing system calls implies that evaluation is possible not just for a single system call and its associated memory, but for series of system calls. As proposed in [17][25][29][43], the behavior of a program or its deviation from its standard behavior can be detected based on series of system calls.

As only relatively short runs of system calls are needed for profiling algorithms, it is possible to detect segments of code instead of complete applications. Behavior detection can also be used to identify state machines present in malware, if context information is examined during analysis. In conjunction with fuzzing techniques, the state machine used in a command and control protocol could also be explored this way.

4) Latency and a Virtual Machine's Real Time Clock: Together, all these analysis steps and operations require a great deal of processing time and a rather sophisticated analysis subsystem. Complex parsing can be done within the hypervisor; however, this would slow down the whole system considerably. As such, the hypervisor should be kept as slim as possible and trigger functionality located within the userland part of the VMM [38].

While being trapped inside hypervisor, a system call and the VM issuing it will remain stalled. Once analysis continues outside the hypervisor, the VM will resume execution. The guest system should remain paused while it is accessed by external analysis components, to keep the guest in a consistent state. The VM should preferably not be able to detect that it is halted; it should retain its internal clock and run detached from the host's real-time clock.

C. Service Provisioning

Malware-driven outbound requests are evaluated to prevent harm to third party systems. For these checks, we rely upon existing measures, such as IDSs or a web application firewall. We are aware that such measures will not be sufficient to tell benign and malicious flows apart in every case; thus, we may build on existing approaches. such as [32]. We assume that a purely passive request (e.g., a download) does not cause harm to a third party. It is thus considered to be benign and handed over to the *external service handler* (SH, see Figure 4).

Since the external SH has Internet access, it resides in a dedicated network segment separated from the analysis environment. If a given request cannot be determined to

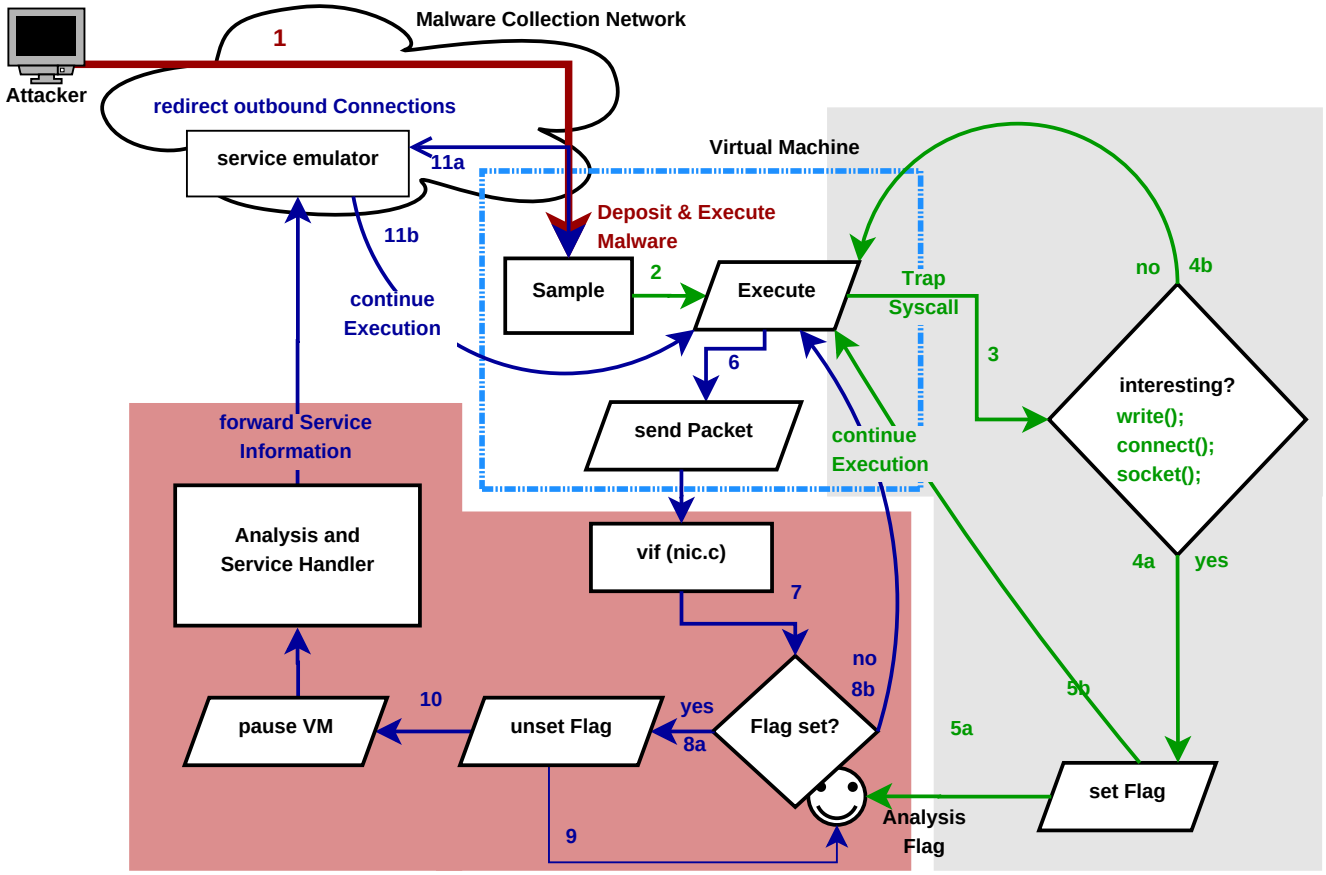


Figure 6. The gray area contains the system call tracing logic and will set a flag if a matching system call has been encountered. The red area holds the analysis components, which are activated only if the flag has been set.

be benign, it is redirected to the *internal service handler*. The sole task of these SHs is to fetch, prepare and provide information for the *service emulator* (SE). The SE launches the requested service in order to deliver the appropriate payload supplied by the SH.

Afterwards, execution is transparently resumed. Since these services can be extremely heterogeneous, the SE is based on Honeyd. It is a very flexible and scalable tool, which is able to emulate or spawn arbitrary services, given that a protocol template exists.

The creation of templates for novel protocols is a much more challenging task. Therefore, we instrument *ScriptGen*, which can derive FSMs from observed traffic, however, it could be replaced by other tools implementing similar approaches [8][12][35].

Each FSM represents the behavior of a given protocol at an abstract level while not depending on prior knowledge or protocol semantics. Based on the generated FSMs, service emulation scripts for the SE can be derived. By integrating such a tool into our approach, we aim toward adding 'self-learning capabilities' to the service provisioning element. Obviously this requires (at least) one-time observation of a given communication between the honeypot and the external system. Hence we need a (supervised) back-channel for learning about novel protocols. Once a corresponding communication has been recorded and the appropriate FSM has been generated, we are able to handle the new protocol as well. While the need for a back-channel is a clear limitation, we consider it to be a reasonable trade-off. The following sections describe the techniques for traffic redirection in more detail.

1) Connection Redirection Techniques: When packets leave a collection or analysis system and the protocol being used has been identified, the respective packet or connection should be forwarded to an appropriate SH within the analysis environment. The SH can either be hosted remotely in the analysis and collection network or locally. Depending upon the approach employed for trapping a sample's actions, redirection of generated traffic can happen in different ways, as shown in Figure 7.

DNS can be used to redirect a connection, if malware relies on using the domain name system to resolve the IP of its peer. The traditional approach is to redirect connections on a per-packet level using network address translation

(NAT), as not all malware will rely on DNS. Network based approaches deploy packet rewriting either on the local virtualization host, or the gateway. In a honeyfarm, the honeywall [11] would redirect these connections as they pass through.

2) Socket Modification at a Binary Level during Runtime: A novel approach to traffic redirection is to rewrite a newly-created socket upon creation in memory, as can be seen in Listing 1. Based on specific system calls, analysis components cannot just extract the parameters used for setting up a new socket or connection. As manipulation of these parameters is possible at runtime, these can be replaced, assuming the location and the format of the expected parameters is known.

While this approach might be more complex than the traditional network-based approach, it offers some interesting advantages, depending on where the destination related data is replaced or manipulated. Using this approach, *IPSec authentication header* functionality might be bypassed relatively easily. Therefore, the SH could use a standard *IPSec* implementation without the need to modify or disable AH integrity checking.

Listing 1. A network related function, for instance the connect command as shown here, could upon execution be rewritten.

```
Trapped API Call:
CALL, cr3: 0xe936000 pid: 1672,
CONNECT(ip: 184.170.X.Y, port: 3127);

Executed API Call:
CALL, cr3: 0xe936000 pid: 1672,
CONNECT(ip: 10.0.1.254, port: 3127);
```

Redirecting a socket at the binary level also reduces side effects occurring due to in-depth analysis. For instance, network-based connection redirection, classification and logging may lead to changed timing behavior; this change of behavior could be detected by malware, lead to connections timing out, or cause certain protocols to stop functioning. Socket redirection at the binary level also allows local system information to be changed more easily.

Naturally, protocol redirection by rewriting socket parameters at runtime is only possible, as long as malware used the operating systems interfaces. If malware would circumvent the operating system, execute completely in kernel mode, or run on a higher privilege level than the kernel, this would no longer be possible. As malware first has to gain the access to these restricted locations sections through system calls, the attempt to do so could in turn be detected. Additional research should be done in this area to investigate further advantages.

As described in section IV-B2, system calls can also be used for initiating delayed analysis functionality in subcomponents of the virtual machine. As network traffic passing through VIFs can be evaluated and manipulated, connection redirection could also be implemented in this location.

All further network related operations should subsequently be performed and handled by the relevant SH, as described in the next subsection.

D. Protocol Sinkholing

For sinkholing several advanced approaches exist on which we can base on. For example *Truman*[1] *(The Reusable Unknown Malware Analysis Net)* and *INetSim (Internet Services Simulation Suite)* simulate various services that malware is expected to frequently interact with. To this end, common protocols, such as HTTP(s), SMTP(s), POP3(s), DNS, FTP(s), TFTP, IRC, NTP, Time and Echo are supported. In addition, INetSim provides dummy TCP/UDP services, which handle connections at unknown or arbitrary ports. Hence these approaches can interact with a given malware sample to a certain level.

Trumanbox [22] enhances the state of affairs by transparently redirecting connection attempts to generic emulated services. To this end, it uses different information gathering techniques and implements four different modes of operation, which allow the application of different policies for outgoing traffic. Thus, Trumanbox can provide different qualities of emulation and addresses issues in protocol identification, transparent redirection, payload modification and connection proxying.

In addition, several work has been conducted in the context of malware analysis to address the issues of detecting, observing and intercepting malicious (C&C-) traffic [10][39][23][45] resulting in publicly available tools.

Hence we concentrate on the issue of handling unknown traffic patterns, such as C&C protocols, within our service provisioning element by instrumenting ScriptGen [33] .

In order to ensure defined test conditions, we chose to build our own malware, since this provides full control over all test parameters thus assuring reproducibility [5]. To this end, we base upon the source code of the Agobot / Phatbot family and compile it using a custom configuration. We chose Agobot, since it is one of the best known bot families and widely used. In addition, it provides a variety of functions. For the sake of easiness, we use unencrypted IRC as the C&C protocol. We deploy a minimal botnet consisting of only one infected host and one C&C server. In addition, we use a third host, which is responsible for the service emulation part. Our resulting test setup consists of three distinct machines:

1) The *victim host* resides on a machine running Microsoft Windows XP SP2. Traffic is captured on this host using WinDump v3.9.5 (the Windows port of tcpdump) based on WinPcap v4.1.2. This host is infected with our malware in order to capture the

[1] http://www.secureworks.com/research/tools/truman/ 10.06.2013

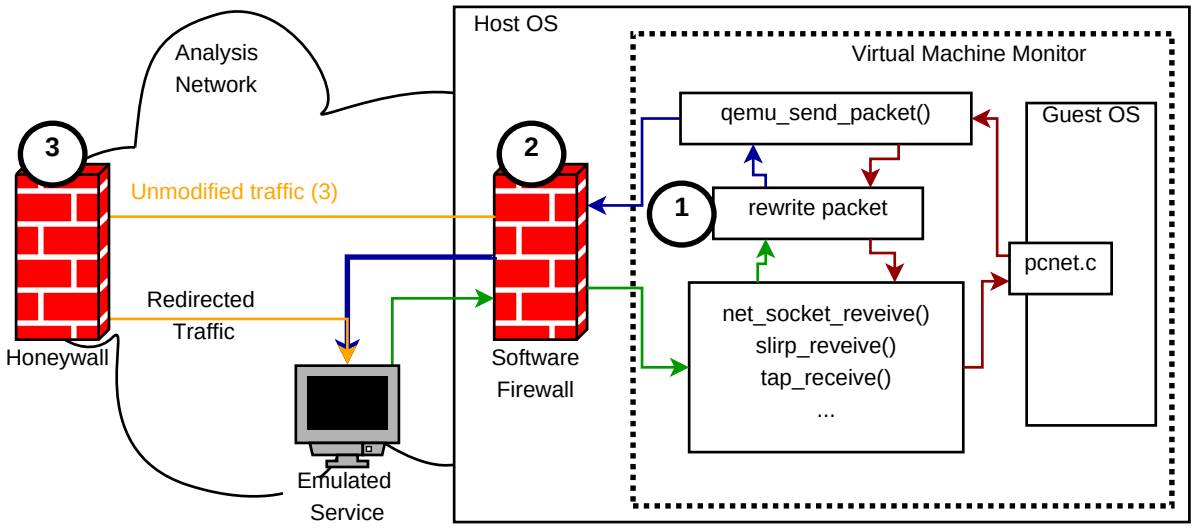


Figure 7. A connection could be redirected within virtual hardware, or the system call itself could be rewritten(1). The packets could be redirected using NAT, either by the virtualization host(2) or the honeywall(3).

malware initiated network traffic as a reference for real C&C traffic.

2) The *C&C server* resides on a machine running Microsoft Windows XP SP3 and is updated to the latest patch level. This is necessary, since this host is also used to build our customized version of Agobot. For a successful build of Agobot, a full featured Microsoft Visual C++ environment including the latest Visual Studio service pack and the latest platform SDK is required. The main purpose of this host is to act as the C&C server. Therefore, we use UnrealIRCd, a well-known IRC daemon widely used by botmasters. There are several customized versions, which are optimized for botnet usage (e.g., designed to serve a vast number of bots). However, for our simple test case the latest standard version (UnrealIRCd 3.2.9) is sufficient.
3) The *service emulator* runs a standard installation of Ubuntu Server 11.10 32bit. Its main task is to process the recorded traffic dump using ScriptGen and to generate a service emulation script out of the resulting FSM, which is then used by Honeyd. To this end, this host is equipped with Honeyd and the dependencies of ScriptGen (i.e., python 2.7, python-dev, cython, python-numpy, python-pcapy, python-setuptools and the nwalign package). Finally, this machine replaces the original C&C server by running Honeyd with the previously created script.

The overall test procedure of our experiment consists of the following steps, which are described in the sections below.

First, we deploy our botnet using a real IRC daemon as C&C server. We launch several commands to generate and record distinct traffic patterns between the infected machine and the C&C server.

Second, we derive FSMs out of this recorded traffic, which are then used to generate a Honeyd service emulation script incorporating the abstract protocol behavior.

Finally, we replace the original C&C server with the service emulation host running Honeyd and the generated script. We launch different commands to evaluate, whether the created script can emulate sufficient responses to fool our bot.

1) Generation of C&C Traffic: First, on the C&C host, we build our custom version of Agobot. Beside some other basic settings, we instruct the bot to connect to our C&C server and join a channel using the respective login credentials and a randomly generated nick. Thereby the nick consists of a random combination of letters prefixed by the string ”bot-” in order to generate data, which is variable on the one hand, but has a significant meaning on the other hand. In addition, we build the bot in debug mode in order to be able to track its activities. The IRC daemon is setup accordingly.

Listing 2. An excerpt of a FSM-recorded conversation.

```
=== Item 201
{ <CONV TCP 80
src:('192.168.1.1', 64459)
dst:('192.168.1.2', 80)>
        [MSG d:I 1:127]
        [MSG d:O 1:49]
        [MSG d:I 1:178]
        [MSG d:O 1:39]
        [MSG d:I 1:262]
        [MSG d:O 1:39 f:Cc]
}
```

Listing 3. Part of the definition of a FSM

```
...
<S [TCP:31337:] f:6 kids:1 label_len:158 new:3 >
{ <TR REG f:6,\#r:1> |F|
 <S [TCP:31337:1] f:2 kids:1 label_len:6 new:4 >
 { <TR REG f:2,\#r:13> |F||Mf||F||Mf||F||Mf||F||Mf||F||Mf||F|
  <S [TCP:31337:1|1] f:2 kids:1 label_len:16 new:0 >
  { <TR NULL f:2>
   <S [TCP:31337:1|1|1] f:0 kids:0 label_len:1054 new:2 >
    {...
```

Next, we execute our customized Agobot on the victim host, which connects back to the configured C&C server and attempts to enter the programmed channel awaiting further commands. On the victim host we record the traffic. Thereby we set additional parameters to avoid a limitation of the recorded packet size, since important information may be truncated otherwise. In order to command the bot we launch a conventional IRC Client, connect as botmaster to our C&C Server and join the previously configured channel as well. We instruct the bot to execute a given command by performing a query. Thereby we use a full stop as command prefix.

After login to the bot (*.login User password*) we instruct it to perform some harmless actions, such as displaying status information (*.bot.about, .bot.sysinfo, .bot.id, .bot.status*). Thereby we launch a diverse set of commands in order to obtain representative data. In particular, we use specific commands, such as .bot.sysinfo, several times, since they also query random and regularly changing values (e.g., uptime).

Furthermore, we also send some dummy commands and random strings intended to insert ”noise”. This is afterwards used to examine the result, i.e., none of these commands should appear in the resulting script. We perform a number of such sessions using randomly chosen commands in an arbitrary order. In doing so, we simulate a number of distinct bots, since the bot generates a different nick for every session. Finally, the recorded traffic of these conversations is filtered to remove traffic produced by other applications running in background.

2) Traffic Dissection and FSM Generation: On the service emulation host we dissect the previously recorded traffic dump and extract the used ports within the communication. Since ScriptGen is port based, this analysis is necessary to determine for which ports a corresponding FSM needs to be generated. As a result, we receive a list of identified ports and generate a FSM for each port. Therefore, we apply the existing functions implemented by ScriptGen:

First, a simplified FSM is built by parsing the traffic dump file for the corresponding data-link type and reassembling the packets and the respective conversations. A conversation is composed of messages, whereas a message is the longest set of bytes going in the same direction. Thus it is the starting point to build the FSM. The rebuild of the resulting conversations is based on the unique tuple of source- and destination-address and the corresponding ports, as can be seen in Listing 2. Thus there is no check, whether a given port actually corresponds to the expected protocol but one FSM per port is built.

It outlines the mentioned unique tuple (source- / destination-address, source-/destination port) along with the messages, where
d is the direction from the server perspective (I: incoming, O: outgoing),
l is the length of the payload in bytes and
f describes the set flags (if any). In this example ”Cc” refers to ”client close”.

Next, functionality to attach data contained in the traffic dump to an eventually existing FSM are called. In addition, these functions could be used to infer content dependencies between known conversations and an existing FSM. The output is a serialized, updated FSM (see Listing 3) serving as input to our converter, which implements the Region Analysis and builds the actual FSM based on the chosen thresholds for macroclustering and microclustering.

It contains information about the used protocol, the observed states and edges as well as the respective transitions, where
S is the self-identifier (i.e., the protocol and port) followed by the path,
f is frequency of the state,
kids describes the number of transitions the state has,
label_len is the length of the state labels,
new is the amount of conversations and
TR (Type Region) describes the identified region type. A region can thereby be *NULL* describing a transient state, i.e., a state with an outgoing NULL transition. That is, a state that immediately leads to a new future state after label generation without expecting a client request.

In addition, a region can be identified as *Fixed* (containing repeatedly the same data), *Mutating* (containing varying

data) or as *REG* (i.e., a regular expression). In turn, a region described via a regular expression may consist of several regions having varying characteristics. This is optionally indicated via corresponding flags, such as *"F"* (fixed region) or *"Mf"* (mutating region).

Listing 4. For a given path the corresponding strings and regular expressions representing the incoming and outgoing messages as identified by ScriptGen are extracted.

```
<<< Incoming
> Fixed region
0000    50  41  53  53  20  31  32  33          PASS 123
0008    0D  0A                                  ..
%regexp: 'PASS\\ 123\\\r\\\n'
<<< Incoming
> Fixed region
0000    4E  49  43  4B  20  62  6F  74          NICK bot
0008    2D                                      -
> Mutating region
Content candidate
...
> Fixed region
0000    77                                      w
Content candidate
...
> Fixed region
0000    0D  0A  55  53  45  52  20  62          ..USER b
0008    6F  74  2D                              ot-
...
>>> Outgoing (len: 16)
0000    50  49  4E  47  20  3A  35  44          PING :5D
0008    38  42  34  37  34  34   D   A          8B4744..
0010
**********************
```

Finally, every generated FSM is inspected and basic information such as port, IP address and the identified protocol is extracted for further processing (e.g., a rule-based decision, whether the FSM for the respective port should be enabled in the service emulator). We incorporate the described functionality for traffic dissection and FSM generation into a small Python script.

3) FSM Traversal and Script Building: In order to generate a service emulation script out of a given FSM, information about its elements is required. Specifically, we need to determine the total number of states and for each state

- all state labels (the messages sent by the server),
- the total number of edges (the number of possible transitions towards the next state) and for each edge
  - the respective edge label (the message representing the client request triggering the transition)

Thereby we assume that the first seen state label represents the initial message sent by the server (e.g., a service banner). Thus it is defined as the default state. To this end, a certain FSM is inspected by traversing through its paths, which are derived from the identified regions, as depicted in Listing 4. Based on this information the sequence of regular expressions and the respective strings to respond with is reassembled. We find, that we can recognize our previously exchanged messages, i.e., the launched commands and their according responses. In particular, we conclude, that ScriptGen is able to map also variable values correctly. For instance, it identifies *"USER bot-"* and *"NICK bot-"* as fixed regions while defining single letters and their combinations as mutating regions. This matches our configuration instructing the bot to use a variable nick consisting of random letters prefixed by "bot-". Thus, ScriptGen abstracts the protocol semantics correctly in this example. The output of the FSM traversal is then included in the service emulation script file. To this end, the static part of the script is created in a first step consisting of a header and some basic functions for echoing fake messages. For basic data exchange we rely upon basic functionality of Honeyd. In a second step, we created a modified version of the traverse function as originally implemented in ScriptGen so that it now traverses through a given FSM path and extracts the regular expressions of the exchanged messages.

Listing 5. A (simplified, primitive and reduced) FSM generated by AWESOME.

```
# State S0)
S0)
  if [[ $_ =~ /'PASS \\$PASS\\\r\\\n' ]];
  then
    echo -e "OK"
  else
    echo -e "ERROR"
    state="S2"
  fi
;;
# State S2:
S2)
  if [[ $_ =~ /'NICK\\ bot\\-untuv\\\r\\\nUSER
       \\ bot\\-untuv\\ 0\\ 0\\ \\:bot\\-un' ]];
  then
    echo -e "^@^@^@^@^@^@"
  else
    echo -e "ERROR"
    state="S3"
  fi
;;
# State S3:
# transient state (immediately leads to a new
# future state without expecting a client request)
S3)
  # therefore no response is generated
  # -> traverse directly to S4
  state="S4"
;;
... omitted
# State S9:
S9)
  if [[ $_ =~ /'USERHOST\\ bot\\-untuv\\\r\\\n' ]];
  then
    echo -e ":$IP 302 bot-untuv :bot-untuv=+"
  else
    echo -e "ERROR"
    state="S10"
  fi
;;
# State S10:
S10)
  if [[ $_ =~ /'JOIN\\ \\#$CHANNEL\\#\\ $PASS\\\r\\\n' ]];
  then
    echo -e ":$IP 302 bot-untuv :bot-untuv=+"
  else
    echo -e "ERROR"
    state="S11"
  fi
;;
```

4) Results: As a result of the steps described above a service emulation script is generated and can be used with Honeyd. Thereby the regular expressions intended to cover

a given class of requests are of special interest.

After replacing the original C&C server with the service emulation host we launched different commands to evaluate, whether the created script can emulate sufficient responses. By interacting with the emulated IRC service we found that it is capable of generating appropriate responses to a set of very basic requests. However, slight deviations of these basic requests cause the emulated service to not respond at all thus leading to insufficient emulation. We believe, that this is related to the limited amount of traffic, that we have generated for this experiment. In fact, the size of the generated traffic dump file is less than 100kB, which seems clearly insufficient for ScriptGen to learn all interactions properly. While we were able to demonstrate that generating service emulation scripts using the ScriptGen approach is essentially possible, further experimentation will be necessary to produce more accurate results.

Further experimentation is necessary and we are confident, that a larger amount of traffic will produce more accurate results. However, we found that generating service emulation scripts using the ScriptGen approach is essentially possible, an example is depicted in Listing 5. Specifically, we believe, that the basic assumption of ScriptGen (i.e., an exploit performs a limited number of execution paths) can be applied to our use case of service emulation for C&C traffic, since a bot performs a limited number of commands as well. Thus we conclude that the application of ScriptGen within the service provisioning component of our presented approach is feasible.

V. Discussion and Future Work

From a conceptional perspective the main limitation of our approach is that it can only capture samples of autonomous spreading malware due to the use of a taintmap. The server based approach with taintmaps, i.e., passively waiting for incoming exploitation attempts, implies that this type of honeypot can not collect malware, which propagates via other propagation vectors such as Spam messages or drive-by downloads.

With respect to the outlined paradigm shift in attack vectors we will need to consider such propagation vectors as well in future. However, since this is a sensor issue, this limitation may be overcome by integrating corresponding honeypot types (i.e., client honeypots) into our approach. This is left for future work. Moreover, due to the ongoing spread of IP-enabled networks to other areas (e.g., mobile devices and SCADA environments) our approach will be required to integrate malware sensors covering these attack vectors as well in future.

Beside the necessary further experiments to improve the accuracy of service emulation for C&C traffic, enhancements in malware analysis need to be tested. In particular, the interaction between all stated components will be evaluated, once all of them are readily deployed. At the time of writing the evaluation of system calls produced by Nitro needs to be finished and measures for checking malware initiated outbound attempts need to be evaluated. In a next step, we will test the use case of tracking and intercepting encrypted C&C protocols as well as making use of malware calling library functionality to perform subroutine analysis.

When it comes to malware analysis itself, multiple paths of execution could be traversed. The state of a virtualized guest could be saved at multiple times during execution and, later on, the analysis environment could revert the guest to different states saved during processing. Traversal of multiple paths of execution would allow automated fuzzing to take place during analysis. Multiple possibly valid responses may be tried, if few candidates exist. Additionally, unknown state machine versions used for C&C traffic may be explored.

Future work will also include the completion and evaluation of the service emulator and the measures to prevent harm to third party systems.

VI. Conclusions

In this paper, we have presented a novel approach for integrated honeypot-based malware collection and analysis, which extends existing functionalities. Specifically, it addresses the separation of collection and analysis, the limitations of service emulation, and the operational risk of HI honeypots.

The key contribution of the approach is the design of the framework as well as the integration and extension of the stated tools. While this is an ongoing research activity and thus still under development, several modifications to ARGOS and Nitro have already been implemented and successfully tested, indicating the feasibility of our approach.

System call tracing alone is a rather finite source of information for malware analysis, as relatively little information worth analyzing is transferred between a system call routine and the caller. System call parameters, such as path names or URIs, are extremely valuable in certain situations, but to make full use of all available information, the virtual machine's memory has to be analyzed too. Static analysis tools can subsequently process deciphered binary samples in memory and no longer have to deal with unpackers or loaders, and thus handle polymorphic and metamorphic software with relative ease.

By tracking a binary using a series of system calls, more and more pieces of malware can be revealed and evaluated. This results in an outcome very much desired by malware researchers; the longer malware operates, the more information can be extracted from malware with less overhead for dealing with obfuscation [13][36][38].

Cryptographic key material and parameters used for establishing and maintaining a secure tunnel between peers would be extremely useful in protocol emulation. These values can be extracted or even replaced within the guest's memory. While replacing such data automatically at runtime may be

challenging, it would allow emulation of an encrypted peer using valid and trusted key material.

While the service provisioning element raises several technical issues regarding protocol identification, connection proxying, transparent redirection, payload modification as well as detection, observation and interception of malicious (C&C-) traffic, most of these issues can be addressed based on existing work. We referred to related research in this area and focused on the issue of handling unknown traffic patterns, such as C&C protocols.

To this end, we presented a proof of concept implementation leveraging the output of ScriptGen for use within the service provisioning component of our presented approach. Using this proof of concept, we evaluated the feasibility of using ScriptGen for generating service emulation scripts intended to spawn an emulated C&C service. We setup a minimal botnet using customized malware in order to generate the corresponding C&C traffic. Out of this recorded traffic we derived FSMs, which were then used to generate a service emulation script incorporating the abstract protocol behavior.

REFERENCES

[1] Martin Brunner, Christian M. Fuchs, and Sascha Todt. Awesome - automated web emulation for secure operation of a malware-analysis environment. In *Proceedings of the Sixth International Conference on Emerging Security Information, Systems and Technologies (SECURWARE 2012)*, pages 68–71, Rome, Italy, August 2012. International Academy, Research, and Industry Association (IARIA), XPS. ISBN: 978-1-61208-209-7. Best Paper Award.

[2] M. Apel, J. Biskup, U. Flegel, and M. Meier. Early warning system on a national level - project amsel. In *Proceedings of the European Workshop on Internet Early Warning and Network Intelligence (EWNI 2010)*, January 2010.

[3] M. Bailey, E. Cooke, D. Watson, F. Jahanian, and N. Provos. A hybrid honeypot architecture for scalable network monitoring. 2006.

[4] M. Brunner, M. Epah, H. Hofinger, C. Roblee, P. Schoo, and S. Todt. The fraunhofer aisec malware analysis laboratory - establishing a secured, honeynet-based cyber threat analysis and research environment. Technical report, Fraunhofer AISEC, September 2010.

[5] Martin Brunner. Integrated honeypot based malware collection and analysis. Master's thesis, 2012.

[6] BSI. Die lage der it-sicherheit in deutschland 2011. Bundesamt fuer Sicherheit in der Informationstechnik, May 2011.

[7] David M. Buches. Fast system call hooking on x86-64 bit windows xp platforms, April 2010.

[8] J. Caballero, P. Poosankam, C. Kreibich, and D. Song. Dispatcher: enabling active botnet infiltration using automatic protocol reverse-engineering. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, pages 621–634, New York, NY, USA, 2009. ACM.

[9] Juan Caballero, Chris Grier, Christian Kreibich, and Vern Paxson. Measuring pay-per-install: the commoditization of malware distribution. In *Proceedings of the 20th USENIX conference on Security*, SEC'11, pages 13–13, Berkeley, CA, USA, 2011. USENIX Association.

[10] D. Cavalca and E. Goldoni. Hive: an open infrastructure for malware collection and analysis. In *proceedings of the 1st workshop on open source software for computer and network forensics*, 2008.

[11] Jay Chen, John McCullough, and Alex C. Snoeren. Universal honeyfarm containment. Technical Report CS2007-0902, New York University and University of California, San Diego, 9500 Gilman Dr., La Jolla, CA 92093, USA, September 2007.

[12] W. Cui, V. Paxson, Nicholas C. Weaver, and Y H. Katz. Protocol-independent adaptive replay of application dialog. In *In The 13th Annual Network and Distributed System Security Symposium (NDSS*, 2006.

[13] A. Dinaburg, P. Royal, M. Sharif, and W. Lee. Ether: malware analysis via hardware virtualization extensions. In *CCS '08: Proceedings of the 15th ACM conference on Computer and communications security*, pages 51–62, New York, NY, USA, 2008. ACM.

[14] M. Dornseif, T. Holz, and C.N. Klein. Nosebreak - attacking honeynets. In *Information Assurance Workshop, 2004. Proceedings from the Fifth Annual IEEE SMC*, june 2004.

[15] M. Engelberth, F. Freiling, J. Göbel, C. Gorecki, T. Holz, R. Hund, P. Trinius, and C. Willems. The inmas approach. In *1st European Workshop on Internet Early Warning and Network Intelligence (EWNI)*, 2010.

[16] P. Ferrie. Attacks on virtual machine emulators. In *AVAR Conference, Auckland*. Symantec Advanced Threat Research, December 2006.

[17] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff. A sense of self for unix processes. In *Security and Privacy, 1996. Proceedings., 1996 IEEE Symposium on*, pages 120–128, 1996.

[18] Jason Franklin, Adrian Perrig, Vern Paxson, and Stefan Savage. An inquiry into the nature and causes of the wealth of internet miscreants. In *Proceedings of the 14th ACM conference on Computer and communications security*, CCS '07, pages 375–388, New York, NY, USA, 2007. ACM.

[19] Christian M. Fuchs. Deployment of binary level protocol identification for malware analysis and collection environments. Bacherlor's thesis, Upper Austria University of Applied Sciences Hagenberg, May 2011.

[20] Christian M. Fuchs. Designing a secure malware collection and analysis environment for industrial use. Bacherlor's thesis, Upper Austria University of Applied Sciences, Bacherlor's degree programme Secure Information Systems in Hagenberg, January 2011.

[21] Tal Garfinkel and Mendel Rosenblum. A Virtual Machine Introspection Based Architecture for Intrusion Detection. In *In Proc. Network and Distributed Systems Security Symposium*, pages 191–206, 2003.

[22] Christian Gorecki. Trumanbox - improving malware analysis by simulating the internet. RWTH Aachen, Department of Computer Science, 2007. Diploma Thesis.

[23] Guofei Gu, Junjie Zhang, and Wenke Lee. Botsniffer: Detecting botnet command and control channels in network traffic. In *NDSS*. The Internet Society, 2008.

[24] P. Gutmann. The commercial malware industry. In *DEFCON 15*, 2007.

[25] Steven A. Hofmeyr, Stephanie Forrest, and Anil Somayaji. Intrusion detection using sequences of system calls. *Journal of Computer Security*, 6:151–180, 1998.

[26] Thorsten Holz, Markus Engelberth, and Felix Freiling. Learning more about the underground economy: A case-study of keyloggers and dropzones. Technical report, University of Mannheim, Laboratory for Dependable System, 2008.

[27] Honeynet Project. Know your enemy: Sebek, November 2003. last visited: 2011-03-11.

[28] X. Jiang and D. Xu. Collapsar: a vm-based architecture for network attack detention center. In *Proceedings of the 13th conference on USENIX Security Symposium - Volume 13*, SSYM'04, Berkeley, CA, USA, 2004. USENIX Association.

[29] Xuxian Jiang and Xinyuan Wang. "out-of-the-box" monitoring of vm-based high-interaction honeypots. In *Proceedings of the 10th international conference on Recent advances in intrusion detection*, RAID'07, pages 198–218, Berlin, Heidelberg, 2007. Springer-Verlag.

[30] Tobias Klein. *Buffer Overflows und Format-String-Schwachstellen: Funktionsweisen, Exploits und Gegenmassnahmen*. Dpunkt.Verlag GmbH, 2004. ISBN 9783898641920.

[31] Clemens Kolbitsch, Thorsten Holz, Christopher Kruegel, and Engin Kirda. Inspector gadget: Automated extraction of proprietary gadgets from malware binaries. *Security and Privacy, IEEE Symposium on*, 0:29–44, 2010.

[32] C. Kreibich, N. Weaver, C. Kanich, W. Cui, and V. Paxson. Gq: practical containment for measuring modern malware systems. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, IMC '11, pages 397–412, New York, NY, USA, 2011. ACM.

[33] C. Leita, K. Mermoud, and M. Dacier. Scriptgen: an automated script generation tool for honeyd. In *Proceedings of the 21st Annual Computer Security Applications Conference (ACSAC)*, Washington, DC, USA, 2005. IEEE.

[34] M. Lindorfer, C. Kolbitsch, and P. Milani Comparetti. Detecting environment-sensitive malware. In *Recent Advances in Intrusion Detection (RAID) Symposium*, 2011.

[35] P. Milani Comparetti, G. Wondracek, C. Kruegel, and E. Kirda. Prospex: Protocol specification extraction. In *Proceedings of the 30th IEEE Symposium on Security and Privacy*, pages 110–125, Washington, DC, USA, 2009.

[36] Anh M. Nguyen, Nabil Schear, HeeDong Jung, Apeksha Godiyal, Samuel T. King, and Hai D. Nguyen. Mavmm: Lightweight and purpose built vmm for malware analysis. In *Proceedings of the 2009 Annual Computer Security Applications Conference*, ACSAC '09, pages 441–450, Washington, DC, USA, 2009. IEEE Computer Society.

[37] G. Ollmann. Behind today's crimeware installation lifecycle: How advanced malware morphs to remain stealthy and persistent. Whitepaper, Damballa, 2011.

[38] J. Pfoh, C. Schneider, and C. Eckert. Nitro: Hardware-based system call tracing for virtual machines. In *Advances in Information and Computer Security*, volume 7038 of *Lecture Notes in Computer Science*. Springer, November 2011.

[39] Daniel Plohmann, Elmar Gerhards-Padilla, and Felix Leder. Botnets: Detection, measurement, disinfection & defence. European Network and Information Security Agency (ENISA), 2011.

[40] M. Polychronakis, P. Mavrommatis, and N. Provos. Ghost turns zombie: exploring the life cycle of web-based malware. In *Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*, Berkeley, CA, USA, 2008. USENIX Association.

[41] Georgios Portokalidis, Asia Slowinska, and Herbert Bos. Argos: an emulator for fingerprinting zero-day attacks for advertised honeypots with automatic signature generation. In *Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006*, EuroSys '06, pages 15–27, 2006.

[42] Mila Dalla Preda, Mihai Christodorescu, Somesh Jha, and Saumya Debray. A semantics-based approach to malware detection. *SIGPLAN Not.*, 42:377–388, January 2007.

[43] Niels Provos. Improving host security with system call policies. In *SSYM'03: Proceedings of the 12th conference on USENIX Security Symposium*, page 18, Berkeley, CA, USA, 2003. USENIX Association.

[44] Niels Provos. A virtual honeypot framework. In *Proceedings of the 13th USENIX Security Symposium*, 2004.

[45] Konrad Rieck, Guido Schwenk, Tobias Limmer, Thorsten Holz, and Pavel Laskov. Botzilla: detecting the "phoning home" of malicious software. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 1978–1984, New York, NY, USA, 2010. ACM.

[46] J. Rutkowska. Red pill... or how to detect vmmusing (almost) one cpu instruction, 2004. http://invisiblethings.org.

[47] Hovav Shacham. The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86. In *In Proceedings of CCS 2007*. ACM Press, 2007.

[48] D. Song, D. Brumley, H. Yin, J. Caballero, I. Jager, M. Gyung Kang, Z. Liang, J. Newsome, P. Poosankam, and P. Saxena. Bitblaze: A new approach to computer security via binary analysis. In *Proceedings of the 4th International Conference on Information Systems Security.*, 2008.

[49] M. Vrable, J. Ma, J. Chen, D. Moore, E. Vandekieft, A. C. Snoeren, G. M. Voelker, and S. Savage. Scalability, fidelity, and containment in the potemkin virtual honeyfarm. In *Proceedings of the 20th ACM symposium on Operating systems principles*, SOSP '05, New York, NY, USA, 2005. ACM.

[50] Carsten Willems, Thorsten Holz, and Felix Freiling. Toward automated dynamic malware analysis using cwsandbox. *IEEE Security and Privacy*, 5:32–39, March 2007.

[51] J. Zhuge, T. Holz, X. Han, C. Song, and W. Zou. Collecting autonomous spreading malware using high-interaction honeypots. In *Proceedings of the 9th international conference on Information and communications security*, ICICS'07, Berlin, Heidelberg, 2007. Springer-Verlag.

OSGiLarva: a Monitoring Framework Supporting OSGi's Dynamicity

Yufang Dan*§, Nicolas Stouls*, Christian Colombo†, and Stéphane Frénot‡
*Université de Lyon, INSA-Lyon, CITI-INRIA F-69621, Villeurbanne, France – Email: first.second@insa-lyon.fr
†Department of Computer Science, University of Malta – Email: first.second@um.edu.mt
‡Université de Lyon, INRIA, INSA-Lyon, CITI-INRIA, F-69621, Villeurbanne, France – Email: first.second@insa-lyon.fr
§College of Computer Science Chongqing University, Chongqing, China

***Abstract*—Service-Oriented Architecture is an approach where software systems are designed in terms of a composition of services. OSGi is a Service-Oriented Framework dedicated to 24/7 Java systems. In this Service-Oriented Programming approach, software is composed of services that may dynamically appear or disappear. In such a case, classical monitoring approaches with statically injected monitors into services cannot be used. In this paper, we describe ongoing work proposing a dynamic monitoring approach dedicated to local SOA systems, focusing particularly on OSGi. Firstly, we define two key properties of loosely coupled monitoring systems: *dynamicity resilience* and *comprehensiveness*. Next, we propose the OSGiLarva tool, which is a preliminary implementation targeted at the OSGi framework. Finally, we present some quantitative results showing that a dynamic monitor based on dynamic proxies and another based on aspect-oriented programming have equivalent performances.**

***Keywords*-Monitoring, Dynamic SOA, OSGi, Larva, LogOs.**

I. Introduction

This article is an extended version of [1], which has been published in IARIA Conferences 2012.

The service-oriented architecture (SOA) is one of the current approaches to develop well structured software supporting agility. It is focused on loosely coupled client-server interaction enabling the client to request server functionality through a repository that exposes appropriate interfaces. Subsequently, the client is bound to the service and is allowed to invoke methods as long as the interface types match. Among SOA approaches, we distinguish between web services and other more local approaches such as OSGi [2] and .NET [3]. The main difference is that in the case of web services one would not typically be able to have the full view of the system, i.e., one can either monitor the client or the server but not both. On the other hand, in the case of local approaches one can reason about the full picture by also taking into consideration the OSGi framework events such as registration of services, service requests by different clients, etc.

In this work, we focus on OSGi, usually used in 24/7 systems, where the system is not restarted when a service appears or disappears. This framework is targeted to embedded systems such as cars, ADSL boxes, or network systems. In such systems, web services cannot be used either due to the lack of connectivity, network limited bandwidth, or for efficiency reasons. In the following, we focus on the OSGi framework, but the same principles can be applied to other local SOA systems, such as .NET.

In dynamic SOA, each service invocation must be considered as a complete context switch since potentially new services may appear and others disappear at runtime. From a dynamic SOA point of view, binding a client to a service is a matter of interface matching, but, neither the client nor the service has a guarantee that the other part behaves as expected. So, after interface matching, continuously ensuring the client's authenticity and the validity of the activities carried out are important for critical systems. For instance, each time a client makes a request to a server, a formally specified constraint can be checked to ensure that the client is authorized to perform that call.

Existing runtime monitoring tools such as JavaMOP [4] or Larva [5] weave interception calls using aspect-oriented programming techniques. This approach works fine in non-dynamic SOA since client-server bindings are usually generated upon the first invocation and preserved throughout the entire client life cycle. On the other hand, in dynamic SOA, due to runtime dynamic changes in the underlying service implementation, the monitoring state woven into the service implementation gets reset.

Our proposal is to bring a dynamic approach to runtime monitoring systems by inserting monitors at the point of client-server binding rather than "statically" at compile-time or loading-time. This means that both the service bindings and the behavioral monitoring bindings are dynamic and loosely coupled, thus supporting service substitution. This approach would preserve behavioral monitoring states across different service versions and check that both versions are behaviorally compatible.

Another major concern in a highly dynamic context, where the implementation of an interface may be substituted, is to ensure that no implementation, or part thereof, can bypass the monitoring framework. Note that if this could happen, the monitor would not be able to detect any malicious code which might be executed. Moreover, what can be concluded about a system's observation if some events could have been missed? Our aim is to enable the monitoring system to be fully active, even if the service provider ignores it.

In this context, we conjecture that a dynamic runtime monitor must have two significant traits: *dynamicity resilience* and *comprehensiveness*. The former refers to the preservation of the behavior flow: in case the monitored service is substituted, the monitor and its state should be transfered; meaning that the property cannot be hard-linked to the code. The latter characteristic means that we cannot allow services to restrict what is observable by the monitor: if we want to check a property, we need to ensure that all the relevant events are monitored. Note that we are not assuming that every service behaves as expected, but only that if an authorized service is to be checked for a particular property, then no event of the service behavior can bypass the monitor observations. For this reason, the architecture relies on a generic event-interception mechanism and a dynamic, loosely coupled, wiring mechanism for automaton verification. The verification logic of the automaton is then handled by an adaptation of the existing monitoring tool Larva [5].

Finally, the introduction of dynamicity to the monitor also increases the scope of properties we are able to address. Thus, we introduce some dynamic primitives in the property description language in order to make it possible to describe behavioral properties, where the registration/un-registration of a service is an expressible event. Furthermore, we also adapt the life cycle of properties, since, under different circumstances, the monitor state might need to be preserved or reset when the underlying service is substituted.

Section II is a case study showing some requirements of this work. Section III presents some runtime verification approaches and proposes a classification of them, showing the gap we propose to fill. It also discusses the trade-off between the observation scope and the expressible properties. Section IV expresses the architectural model for a dynamic runtime verification tool and introduces our OSGi reference implementation. Section V describes our modifications of the Larva specification language in order to consider dynamicity. Section VI illustrates the OSGiLarva tool by some quantitative results. Finally, Section VII shows our initial conclusions and Section VIII our future works.

II. Case Study

In order to ease the understanding of our contribution, this section introduces an example of a dynamically monitored system in line with our proposition.

Let us consider an embedded client on a mobile device based on a dynamic SOA platform, which needs to communicate with a distant system according to a particular protocol (Fig. 1). Let two services S_1 and S_2 provide an identical interface to access the distant system through different media: S_1 using a WiFi connection, and S_2 using a 3G connection. With such a configuration, we can consider that each time the WiFi connection goes down, the system unregisters S_1, effectively switching the client onto S_2, and vice-versa.

Moreover, we consider that the use of the distant system requires that the client is authenticated with the service and that some system actions have to execute atomically. Such requirements correspond to any typical secured system supporting concurrent access by transactions.

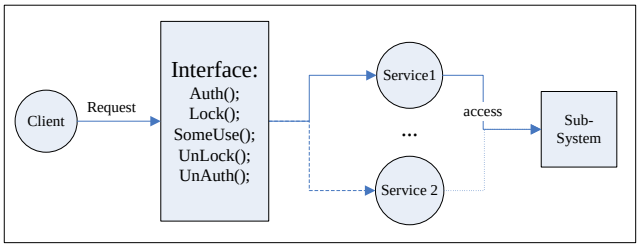


Figure 1. Dynamic SOA system supporting service substitution

In such an example, the possibility of service substitution is crucial. We then propose, in Fig. 2, an example of an execution scenario that has to be supported by the system. In this scenario, the service S_1 is substituted by S_2 during the atomic part of the run.

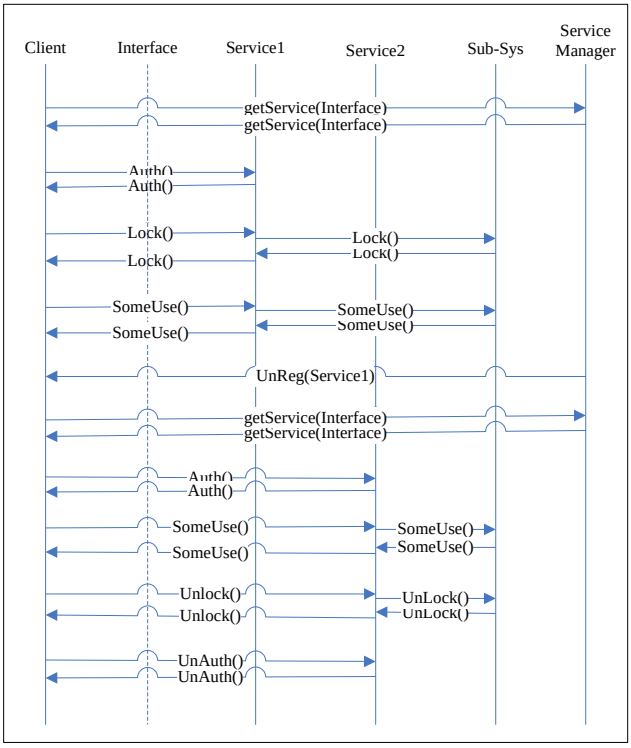
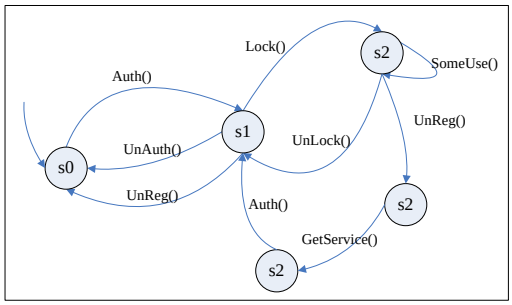


Figure 2. Example of scenario supported by example in Fig. 1

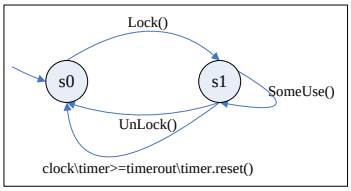
In another part, we can describe the correct use of the system in some property and check it by monitoring at

runtime. For instance, the two following properties express the expected behavior, described earlier: (i) the client is locally authenticated on the service before using it, and (ii) the concrete use of the sub-system requires that the client opens the lock and closes it after use. In this example, one would like to ensure that the execution described in Fig. 2 is correct with respect to these properties. Such verification and the description of the property itself are the main contributions of this article.

These properties can be described by a couple of automatons (Fig. 3), but with a different interpretation of each. The local authentication automaton's (Fig. 3.A) internal state should be maintained in case of service substitution and should be instantiated for each distinct client using the system. In the following, we will call such properties as *Instance-Properties* as they are instantiated on a per object basis; in this case a client. On the contrary, the management of the atomic use of the sub-system (Fig. 3.B) needs to be centralized and shared by all clients. Even if a service is removed and substituted, we would want to keep the current state of the sub-system in memory. In the following, we call such properties *Class-Properties* because its lifetime spans throughout the system's life cycle and is not bound to a particular entity.



A. Client-side: instance property



B. Interface-side: class property

Figure 3. Example of a property associated to example in Fig. 1

In summary, our proposition is to provide a monitoring framework, which is able to monitor such properties by listening to method calls and OSGi framework events in a dynamic, resilient, and comprehensive manner.

III. Related Works

The contributions of this article include a monitoring approach for dynamic SOA and the expressiveness of its associated description language. In this section, we discuss separately related works about each of these two parts of our contribution.

A. Resilience to Dynamicity and Monitoring Comprehensiveness

We propose to classify existing runtime verification approaches according to the monitor configuration with respect to the monitored service. Property may be: manually written inside the code (Hard-Coding), automatically injected inside the code (Soft-Coding) and kept out of the code (Agnostic-Coding). For each of these families, we will discuss two points:

- resilience to dynamicity
- monitoring comprehensiveness

1) Hard-coding: In this category, where properties are manually added at source time, we can cite all annotation techniques, like JML [6] and Spec# [7]. In both cases, the monitor is not *resilient to dynamic* code loading. If the monitored system is substituted, then its monitor is also substituted, since it is inlined. However, this approach is interesting in terms of *comprehensiveness*, since we can observe anything in the program. A limitation of this approach is the dispersion of the monitor throughout the code, requiring significant intervention to write the property or to check that its description is correct.

2) Soft-Coding: In this category, where properties are injected at compilation time, or load-time, we can cite Enforcement monitor [8], Larva [5] and JavaMOP [4]. These tools use a standalone description of a property and inject the synthesized monitor inside the code by AspectJ technology.

Advantages of Soft-Coding approach are then the same as in the previous case, but specifying the monitor is easier, since the description of the property is centralized. However, these approaches from Enforcement monitor [8], Larva [5] or JavaMOP [4] are only partially *resilient to dynamicity*; at best, the tool may inject the property at first-time binding, but once injected, the property is hard-coded within the service for the whole execution of the class. Indeed, while it is technically possible to use AspectJ to support dynamic class loading and unloading in OSGi, then the monitored bundle must declare the import of the AspectJ library inside its Manifest file — an operation which is not really transparent to the service. Note that this restriction does not exists in Equinoxe implementation of OSGi (Eclipse), but it is because some choices would have been done in the configuration of the framework, requiring to restart the whole framework each time a new service is installed. Furthermore, if monitors need to be started or stopped at runtime it cannot be done directly through AspectJ without restarting the service — something which is undesirable in 24/7 services.

3) Agnostic-Coding: In this third category, where the monitor is kept out of the code, we include any trace analyzes approach, such as intrusion detection systems [9] or logging systems [10]. The main advantage of the approach is the loose linking between the property and the monitored system. Hence, if a package is substituted, the monitor can observe it inside the logs and the monitored properties are still the same for the whole system. Moreover, the description of the property is located into a single location, which facilitates property management.

However, such Agnostic-Coding systems can be bypassed, e.g., [9] and [10] can only observe what services accept to push. If a package provides a service without writing sufficient logs, then the monitor does not have sufficient information to check a particular property [11].

4) Monitoring of Web Services: There are a number of works (e.g., [12], [13]) that support the monitoring of web services. These provide both dynamicity resilience and comprehensiveness (although these are not explicitly identified as such) by listening to events from a web service composition engine. Furthermore, they also enable properties to be defined both as class properties and instance properties. However, to the best of our knowledge, no similar monitoring techniques have been proposed for the OSGi framework. Moreover, the context is not the same, since in a web service context, we can easily distinguish between callers by their IP address and port number, but it is impossible to know who is the caller, or which class or software is making the call. This can be a considerable restriction in the expression of security policies.

B. Property description

This part discusses the property description language and focuses mainly on the scope of the property, mainly induced by the location of its associated monitor. Indeed, since we are not in a 1-1 system, we could have many clients using many services at the same time. In such a case, the location of the monitor can change the point of view of the property and hence its expressiveness. Each property can be defined with at least three points of view (eg. Fig. 4): (i) client point of view, (ii) service implementation point of view and (iii) interface point of view.

The proposition made in this work consists in considering all properties as a composition of two parts: a part that handles the client's point of view and a part that handles the interface point of view. In this section, we discuss each of these three possibilities to justify our proposition.

1) Property Described from Service Side Point of View: If the designer describes a property with this point of view, shown in Fig. 5, he/she considers the use of a single service [14]. It is easy to consider some behavioral dependence in some parallel uses by multiple clients. However, since we are considering automaton-based properties, it is not obvious how to distinguish between clients within the

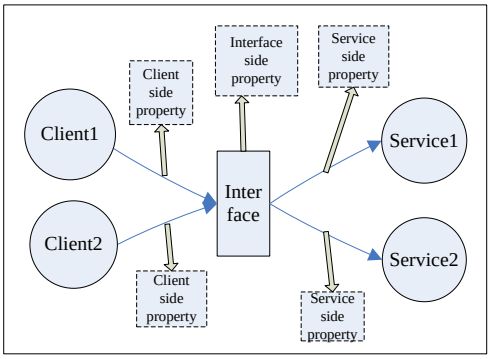


Figure 4. Possible point of view for properties

property. Moreover, it is complex to consider the use of multiple implementations of an interface simultaneously, with potentially some communication between them.

For the dynamical part, it is not intuitive to describe and use the fact that a new implementation of the same service interface has been loaded on the platform. Moreover, it seems to be complex to share property memory between implementations of the same interface. Hence, if a service is substituted, there is no means of keeping its property in memory, with its internal state, and to map it on another implementation designated to continue the started work.

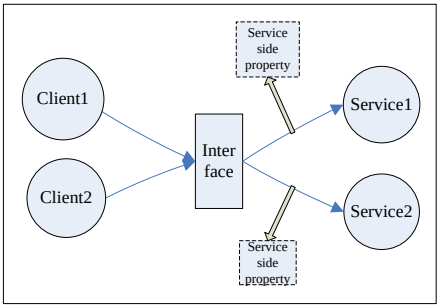


Figure 5. Property description: service implementation point of view

Advantages:

- Simplicity to describe behaviors of each service implementation without the need to make the link with other possible implementations.
- In case of stateful services, with a different memory address space for each implementation, it is very easy to describe the system.

Disadvantages:

- Complexity to describe shared memory between services.
- Impossibility to describe a generic behavior for each client, since we cannot distinguish between clients.

2) Property Described from Service Interface Point of View: In this point of view, we consider what can be done through a service interface. It is easy to describe the global use of any implementation of this interface by any client, but not to make distinction between clients or between used implementations.

By its nature, such a property is not directly associated to a service and thus describes a property shared by all implementations. Note that it is easy to consider the loading or unloading of a service implementation, even if it is a substitution, willing to keep the current state of the property.

Since our property description language is automaton-based, the only manner to consider parallel use of many clients is to make some composition between the property and itself. However, such technique leads to a combinatorial explosion of the automaton size. Moreover, it limits the maximum number of clients and services, since we need to have this information to make the composition.

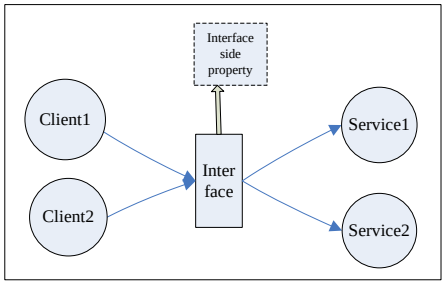


Figure 6. Property description: service interface point of view

Advantages :

- Easy to make a description of the authorized uses, with a global point of view
- Easy to consider loading/unloading of implementations
- Possibility to share a single property state between service implementations

Disadvantages :

- Risk of the shared property description size explosion if we want to describe the concurrent behaviour of several clients.
- Impossibility to describe a generic behavior for each client, since we cannot distinguish between clients

3) Property Described from Client Point of View: This third possibility considers that each client has its own instance of the property (Fig. 7). Hence, it is easy to describe the correct use of a service from one client point of view and to consider as many parallel uses as we want, without any combinatorial explosion.

Moreover, it is easy to describe the use of multiple services by a single client and the behavioral dependence in case of concurrent use of services.

In case of substitution of a service, this approach can be resilient, since the property is attached to the client.

However, in case of the simultaneous use of a single service by several clients, if there is some interactions between these usages, it is more complex to describe it.

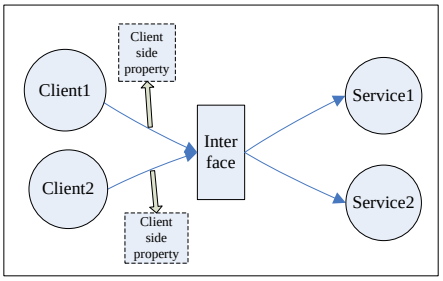


Figure 7. Property description: client point of view

Advantages :

- Easy to make a description of a particular client authorized usages
- Easy to consider loading/unloading of implementations
- Possibility to share a single property state between several service implementations
- No risk of size explosion of the shared property, since it cannot be described

Disadvantages :

- Complexity of describing global behavior including several clients

In this paper, we propose to consider properties as a combination of two kind of properties, associated to two point of views: client and interface. These two parts are respectively called **Instance-Property** and **Class-Property** and are more detailed in section V-B. We propose not to consider the first case (i.e., service point of view), since in typical use of OSGi, if multiple services implement a single interface, the framework favours the use of the same implementation by all clients. Moreover, from our experience we conjecture that properties are typically client side, since an interface property cannot consider the concurrent use of services by many clients without a state explosion. Finally, to have the possibility to add a centralized property, interface properties can be useful to express some shared constraints such as locking/unlocking systems.

In the following, we present the first part of our contribution: the architecture.

IV. Dynamic-SOA Monitoring Architecture

In the first part of this section, we describe an abstract architecture of a monitoring system supporting specific features of dynamic SOA systems, and we discuss its characteristics. In the second part, we propose a concrete implementation of this architecture under OSGi: OSGiLarva.

A. Proposition of a Generic Architecture

Our proposition consists of dynamically inserting a monitoring proxy in front of each service, and executing monitors in some autonomous services (Fig. 8). When a service usage event occurs, a notification is sent to each associated monitor, which checks the event against its property.

An interesting advantage of using a dynamic proxy over AspectJ, is that we can start or stop the monitoring of a property without restarting the service. Indeed, since the proxy is bound upon a service request, this can be handled easily, while AspectJ aspects are bound at class load-time, requiring to restart the service.

Since services are treated as black boxes from the running environment's point of view, such an architecture is designed to consider only properties of their external interface. This corresponds to properties expressing the normal authorized use of a service. However, since we are considering dynamic systems, we also want to consider dedicated framework events, such as unregistration of a service or getting a new service. In this approach, we will then focus on behavioral properties.

Since several clients can be running simultaneously within the framework, the scope of properties should not be restricted to the use of a single client. We consider the possibility of adding a monitor in front of several client. By considering both the monitoring of Instance-Properties and Class-Properties, we enable the possibility of simultaneously checking both local as well as global properties on the system.

In order to enable properties expressed in terms of method call events and framework events (requests, registration, unregistration, etc.), we need to capture both kinds of events — the ones between the client and the service, and events from the service registration system. To inject a monitor between a service and a client using it, we adapt the framework in order to make this invisible both to the client and the service. Two interesting characteristics of this approach are that it does not change the binary signature of the service and that neither the service, nor the client, are aware of a potentially running monitor. By adding another proxy in front of the service management system of the framework, we are notified of requests for getting service references.

Fig. 8 describes the abstract architecture. In the following, we delve deeper into our two main principles.

Resilience to Dynamicity: Since the monitoring system is externalized in an autonomous service, monitors are separated from the code. When changes occur in the framework, the observation mechanism and its properties remain unaffected.

Comprehensive Monitoring: One of the main concepts of dynamic SOA is to have a framework which allows dynamic loading and unloading of loosely coupled services. Since the framework is in charge of providing an implementation to each service request, the framework can add a proxy between the client and the service to observe their communications. This observation is comprehensive and no communication can bypass this proxy, since neither the client nor the service know each other directly.

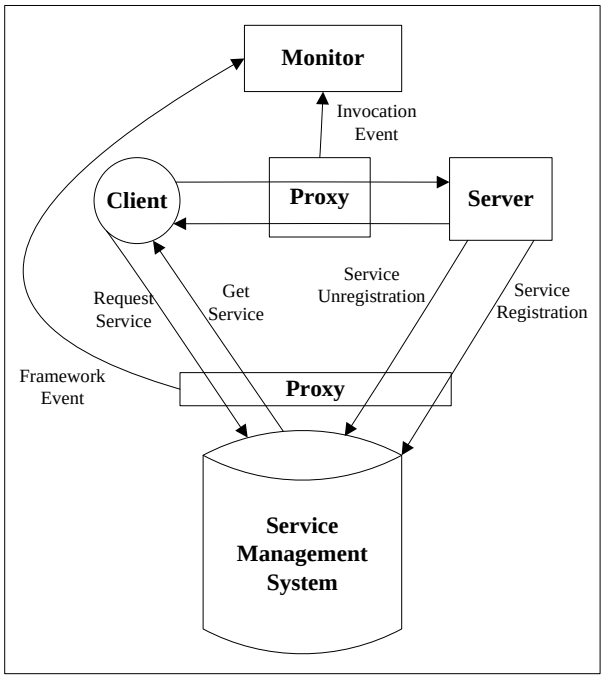


Figure 8. Proposed abstract architecture for monitoring system

B. OSGiLarva — A monitoring tool for OSGi

OSGiLarva (Fig. 9), is an implementation of the described abstract architecture in the context of the OSGi framework. In our tool, we use Java mechanisms in order to generate a proxy between each client and service. This proxy is dynamically generated from a framework proxy, hooked onto the OSGi framework, and listens to all framework events such as the introduction of a new service or the requesting of a service by a client.

This implementation integrates two existing tools: Larva [5] and LogOs [15]. LogOs is a special logging tool based on the OSGi framework, developed at the CITI Lab during the LISE project [16]. We will use it as a hooking mechanism to observe services' interactions. Larva is a compiler which generates a verification monitor that may be injected into Java code. We use an adaptation of Larva to enable property verification on events reported by LogOs.

We describe the monitor implementation with three key parts: we first present our adaptation of LogOs to intercept service interactions; next, we give some details about our modifications of Larva; finally, we describe how the registration process of a service under OSGi will take into account an existing property monitor to insert it between the service consumer and the service itself.

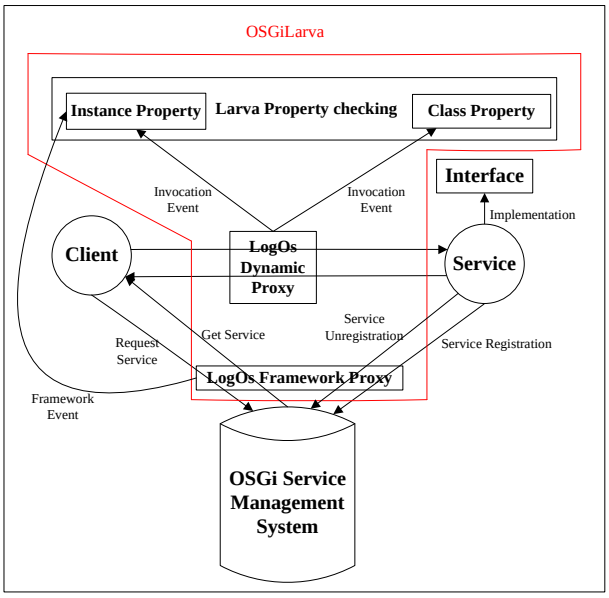


Figure 9. OSGiLarva implementation

1) LogOs – a Hook to Intercept Services' Interactions: LogOs is a transparent logging toolkit for the service activity inside the OSGi architecture. As soon as the LogOs bundle is started, each service registration is observed by the system. Thanks to the OSGi hooking mechanism, a LogOs proxy is generated between the service and its consumer. Hence, every method call, including parameters and returned values, are automatically intercepted.

For each event captured by a LogOs proxy, a corresponding LogOs event-description is forged and propagated to LogOs. In our adaptation, LogOs proxy forwards them to the associated monitors.

We have extended LogOs annotations to enable the user to declare whether an interface is to be monitored or not. If an annotation is present, the monitoring class is loaded when a service implementation is registered.

Moreover, LogOs integrates a mechanism to observe services registration, which is originally used to generate service proxy at load-time. This information is sent to the Larva monitor.

2) Larva — a Monitoring Tool: Larva is a tool that injects monitoring code in a Java program to check a property described in a Larva script file. Upon compiling a script, the Larva Compiler generates two main outputs: (i) a Java class coding the property and (ii) an aspect which links the monitoring code with the source code. An *aspect* is defined to statically inject some calls to the monitor inside the Java software by using the AspectJ compiler. The Java code translating the property is called each time an expected event occurs.

We adapted Larva to OSGiLarva by removing the part associated with the injection of aspects. In order to replace this part by a call from LogOs, we make the generated Java code from the properties implement an interface provided by LogOs. In order to consider dynamic events in described properties, we introduced some new primitives in the property description language (Section V) corresponding to event descriptions generated by the latest version of LogOs.

3) Registration of a Service Providing Specification: We propose to enable the declaration of properties to be monitored to be included as part of OSGi bundles, as shown in Fig. 10. Indeed, an OSGi bundle is an archive providing three elements: a collection of *interfaces*, a collection of *services implementations*, and *bootstrap code*, which is called when loading or unloading the bundle. Thanks to the OSGi architecture, service interfaces, service implementations and bundles may have different life cycles depending on the deployment scheme, since interfaces may be deployed with a bundle other than the one containing the service implementation.

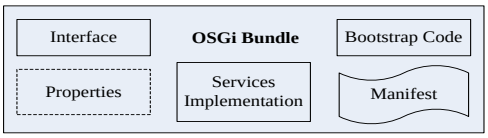


Figure 10. Structure of an OSGi bundle providing properties

As such, we propose to keep the same philosophy when providing properties. We consider that they can be either provided by the same bundle as implementation or by another one. Since interfaces are typing specifications of services and OSGiLarva Class-Properties are behavioral specification of services, it makes sense to map the life cycle of class properties to that of interfaces. On the other hand, the Instance-Properties life cycle describes the behavior of a single service interaction and thus it makes sense to map its life cycle to the client-service connection life cycle.

In next section, we introduce the property description language, which is an adaptation of the one used in Larva.

V. Introducing Dynamicity in Property Description

The OSGiLarva description language is originally based on the Larva property description language, but we adapt it in order to support more dynamicity. This adaptation is done through two extensions. The first one is the introduction of framework-event primitives in the language. The second one expresses a property as a composition of Class-Properties and Instance-Properties. In this section, we introduce these modifications.

A. Adding Dynamic Primitives

Larva uses as input a property description language based on automatons, extended by timers, variables and actions.

In the property itself, the user can define the set of symbols used in the automaton. These symbols are events which, in the original version of Larva are defined in terms of method names. We thus propose to add some new primitives in the event definition in order to support framework-event.

A monitor is started when a monitored service is registered in the framework. From this moment, each event related to this service is propagated to this monitor. Since we are in a dynamic framework, dynamic events can occur, e.g., the loading of a second implementation of the same interface, or the un-registration of an existing service. We propose to introduce the three following primitives:

- **REGISTER**: this event occurs when a new service implementation is registered on the framework. It means that a client can now get this service reference at any time. If another implementation is already registered, it shares the same Class-Property.
- **GETSERVICE**: this event occurs when a client is asking for a service, by calling the framework `getService` method. It can lead to two situations: client gets a service or the client does not get any service. If a client could not get a service from the server, it means that there is no registered service corresponding to the client request. For this reason, we introduce the **NOGETSERVICE** event to handle this case.
- **UNREGISTER**: this event occurs when a bundle is unregistered from the framework. It means that, the `stop` method of the bundle has been executed. Used resources are then considered as released. However, if any reference to an instance of the code provided by this bundle still exist, they are now called *Stall references* — meaning that if a client was using this service it has to consider this code as perhaps no longer safe or functional.

In order to generate and provide these events to Larva-monitors, LogOs needs to register some listeners on the framework.

Event **GETSERVICE** is obtained by using an OSGi `FindHook` instance, registered in the OSGi framework. When registered, such object is called each time a service is obtained. Originally, this mechanism was defined in order to make a filter on services obtained as a result of `getService` call. Indeed, the `getService` method accepts as an input a description of the expected service and returns an array of corresponding service implementations among the available ones. The `FindHook` mechanism has been introduced in order to allow service filtering (i.e., to hide some services). Note that LogOs also uses this mechanism to ensure that, if a service is monitored, every calls to this service are necessarily done through a proxy, and never directly.

REGISTER and **UNREGISTER** events are obtained by registering an OSGi `EventHook` with the service management system. An object implementing the `EventHook` class and registered in the framework is called each time the service management system observes a modification, such as new incoming service, a service un-registration, or a service property modification.

In each of these cases, an event descriptor is forged by LogOs and sent to the Larva monitor. On its end, Larva treats such events like all other events. Hence, the event descriptor is compared to the list of events the monitor is listening to, and, if the property is expecting this kind of event, it triggers upon it.

B. Property Description Language

Since our contribution is based on the Larva description language [17], chosen for its closeness to our requirements, we mainly orient our proposition according to Larva. In Larva, properties are described by automatons, where a single script file can contain several automatons. Moreover, Larva provides in its language the possibility of defining parametrized automatons which can be instantiated using event parameters, through the **FOREACH** keyword. We exploit this characteristics in order to use properties composed by two parts (Instance-Property and Class-Property):

- **Instance-Property:** If a property is defined as an Instance-Property, then each time a new client accesses the interface, a new instance of the property is generated and added inside the monitor. When the client terminates, the associated instance of the property can also be removed. Hence, while such properties are still resilient to service implementations' dynamicity, they are intentionally not resilient to clients' dynamicity.
- **Class-Property:** This case corresponds to a centralized property, meaning that several clients using a particular interface will share the same Class-Property. Such property is more resilient to dynamicity since a Class-Property can be kept in memory until the associated interface is unloaded. As such it is not associated to a particular user's interaction or a particular service implementation, and can thus be used, for instance, to express some centralized locking/unlocking mechanisms. However, if several implementations are used concurrently, then they would probably need to be synchronized.

In the following, we present the main principles of the Larva property description language together with small modifications done in the context of OSGiLarva.

1) Existing Larva Property Description Language: A Larva property description file can contain several automatons. The file is structured in terms of contexts. The global context can contain several properties and each of them can introduce a new context. A context is defined by variables and listened events. Each inner context can access the global variables.

A **FOREACH** structure allows a property to be instantiated for each different value of an element, considered as an identifier.

Channels can be used by automatons to communicate together. These channel-generated events are broadcasted to the current context and below. So, if two inner contexts need to communicate, they can do it through channels.

A generic structure of a Larva property file is given in Fig. 11. It shows a file containing two properties: a global one and an instantiated one.

```
GLOBAL{
   VARIABLES{ ... }
   EVENTS{ ... }
   PROPERTY P1 {
      STATES{...}
      TRANSITIONS{ ... }
   }
   FOREACH (Object u ){
      VARIABLES{ ... }
      EVENTS{
          %% Property designer needs to
          %% express how to retrieve the
          %% identifier:
          someEvent(User u1) = {
             u1.someMethod(); where u=u1;
          }
          ...
       }
       PROPERTY P2 {
          STATES{...}
          TRANSITIONS{ ... }
       }
   }
}
```

Figure 11. Generic larva property file with two properties of two types

2) OSGiLarva Properties: One way of introducing a new context is to use a **FOREACH** clause. This clause is a quantification on an object. Hence, for each instance of a given class, Larva generates a new instance of the inner property. We propose to adapt this structure to our needs, by introducing a new clause: **FOREACHCLIENT**.

In classical Larva, in order to distinguish between users, Larva uses the information given by the caller such as a Session ID passed as a parameter. Hence, Larva only has the same information as the service implementation to check a property. We propose to improve on this by introducing this construct based on the address of the caller. As an example, such a clause could make it possible to check that there is no IDsession spoofing.

A property described in the **FOREACHCLIENT** context will be re-instantiated for each loaded client. It will be the instance-part of the property. Conversely, the class part of the property is instantiated only once and is then shared by all clients. We will then express it in the **GLOBAL** clause. It can communicate with all "instance-part" instances of the property. Fig. 12 shows the global syntax of a global property, composed by an Instance-Property and a Class-Property.

A very important difference between the **FOREACH** and **FOREACHCLIENT** clauses is that the first one is based on values computed inside the **EVENTS** clause from observed parameters, while the second one is based on values provided directly by LogOs observation, without any interpretation of parameters.

Moreover, since **FOREACHCLIENT** is an extension of the **FOREACH** clause, then we keep all language characteristics of the latter.

```
GLOBAL{
    VARIABLES{ ... }
    EVENTS{ ... }
    PROPERTY P1 {
        %% Class property (same as Larva)
        STATES{...}
        TRANSITIONS{ ... }
    }
    %% Introduction of this new keyword
    FOREACHCLIENT(Long pid,String s){
        %% Instance property.
        %% Parameters are:
        %% - pid: client identifier
        %% - s:   name of the client
        %%        (for logs)
        VARIABLES{ ... }
        %% EVENTS clause do not need to
        %% provide method to compute the
        %% identifier. It is intricated
        %% inside the language.
        EVENTS{
            %% Just an event description
            someEvent()=frameworkEvent();
            anotherEvent()=someMethod();
            ...
        }
        PROPERTY P2 {
            STATES{...}
            TRANSITIONS{ ... }
        }
    }
}
```

Figure 12. Introducing the **FOREACHCLIENT** keyword

VI. Evaluation

In this section, we present some benches of OSGiLarva. There are mainly two implementations used for executing OSGi services: Apache Felix and Eclipse Equinox. In our benches, we use the current Apache Felix which is an open source implementation of the OSGi Release 4 core framework specification, on the top of the Java 1.6.0-06 Virtual Machine. The machine used for these tests runs on an Intel Pentium M at 1.4GHz CPU with 640MB of RAM

and running under Gentoo 4.2.3 with 2.6.22-gentoo-r8 kernel version.

In the following, we are using two examples: one without dynamicity and another with dynamicity. Indeed, since we will make efficiency comparisons against Larva, which does not support dynamicity, we then need to have a static example. This example is just a loop making some calls to a function provided by a service. On the other hand, the dynamic example is very close to the one described in Section II, but with a loop on the client side. This loop specifies the concrete actions from the client and contains a call to a service, followed by an unregistration of the service, a get service to have a second service, a second call, and finally a new registration of the unregistered service. In our benches, we modify the amount of loop iterations to study the variation of the time cost in the long run and its variation due to JIT compilation.

We made three kinds of tests to study performances of OSGiLarva: a comparison between the execution time of OSGiLarva and Larva, a comparison between the execution time of OSGiLarva and OSGi, and a comparison between the execution time of OSGiLarva and a Class-Property-only in OSGiLarva. Indeed, we hypothesized that the identification of the client (and hence the Instance-Property) is a bottleneck, but benches show that it is not so costly.

Here is the definition of some keywords appearing in this section:

- Larva: the time cost from the example with the original Larva system.
- OSGiLarva: the time cost from the example with the OSGiLarva tool.
- WithoutOSGiLarva: the time cost from the example running under OSGi, but without any monitoring system.
- OSGiLarvawithoutPID: the time cost from the example with a weaker version of OSGiLarva where we removed the generation of a caller Id from the system.

Finally, for each test, we made two curve charts. The "Time cost comparison" curve chart shows amount of loop iterations on the horizontal axis, and time cost in milliseconds on the vertical axis. The "Cost ratio" curve chart shows amount of loop iterations on the horizontal axis, and change ratio of time cost in percentage points on the vertical axis. The cost ratio is calculated by the time cost of the example with the monitor divided by the time cost of the example without the monitor.

A. Monitoring cost by Using a Proxy (OSGiLarva VS Larva)

The goal of this test is to evaluate the performance of OSGiLarva (with a proxy) and to compare with the one of the Larva tool (with AspectJ) on the same functions example. Since Larva does not support OSGi dynamicity, we made the comparison on a example without loading of services. In this kind of comparison, we just use the two tools to monitor the normal events from the communication of client using services.

Fig. 13 is a comparison of the time cost in the execution of a static example with Larva and OSGiLarva monitors. We can observe that both curves are very close. Hence, OSGiLarva does not add too much cost by its proxy approach.

In order to be more precise, in Fig. 14 we plot the curves of the cost ratio between Larva and OSGiLarva time cost results. The change ratio of time cost is lower than 1%. This change ratio is from the proxy in OSGiLarva. Thanks to this proxy, OSGiLarva can make the behavioral monitoring bindings dynamic and loosely coupled. The pre-condition of this test is that the monitored service is never replaced by another one. If the monitored service is replaced during runtime, Larva will not be able to detect any of its events. But OSGiLarva can continue to monitor it.

Since these two technologies are not using the same Virtual Machine, the JIT is also not the same. We think that this difference is the explanation for the behaviour observed in the first run, which is stable and always faster on OSGiLarva. This difference is probably also the explanation for diminution of the overhead when the loop is longer.

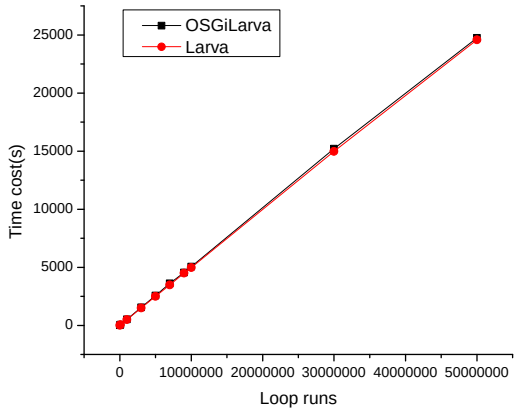


Figure 13. Comparing time cost of a static example with OSGiLarva and Larva

B. OSGiLarva Efficiency (OSGi VS OSGiLarva)

This test runs the dynamic example described as a running example in this article, but with a loop inside the client. We then run it with and without OSGiLarva in an OSGi environment. It aims to evaluate the raw impact of OSGiLarva on service invocation and service events from the framework. The property events includes normal events and framework events.

From Fig. 15, we know that the performance impact of OSGiLarva is stable at around 23% on this example.

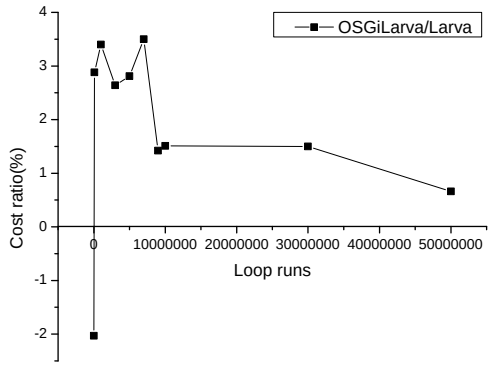


Figure 14. Comparing cost ratio of a static example with OSGiLarva and Larva

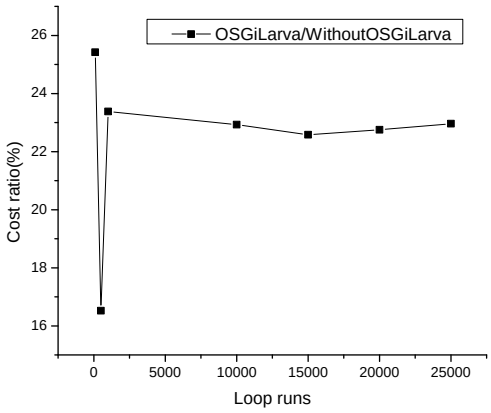


Figure 16. Comparing cost ratio of the case study example with and without OSGiLarva (simple method in service side)

For every monitored service invocation and framework events, OSGiLarva performs its indirection work: it verifies the actions from the original system and computes the current client id, and finally it outputs the monitored traces to the developer or the user at real-time. The cost ratio almost becomes a horizontal line shown in Fig. 16, except for the two first points at about loop 100 runs and 500 runs. We presume that it is the initialization of the JIT which is causing this anomaly.

It is important to note that this 23% overhead is a metric including the call of methods events and the framework events. The biggest part of this overhead is associated to the cost of generating a new proxy and placing it in front of newly requested service.

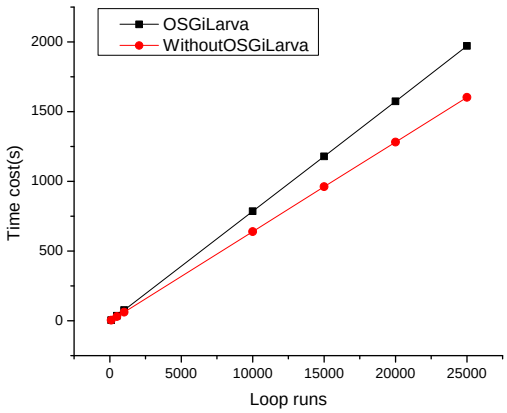


Figure 15. Comparing time cost of the case study example with and without OSGiLarva (simple method in service side)

C. Overhead Associated to Getting the Caller Id

In order to associate each communication to the right client in Instance-Properties, we compute a caller Id. However, we get it through the SecurityManager which is a non-internal way of finding the caller class and caller Id. As such, one would expect extra time costs because of the SecurityManager, warranting further investigation.

Thus, the following test is just for knowing the performance impact from compute current caller Id during runtime. We then compare the cost of the Case Study with and without the Instance-Property and then, with or without getting the caller Id.

From Figs. 17 and 18, we observe that the time cost of the two kind of monitoring are very closed. The impact cost is lower than 5%.

Indeed, in such a simple test example, the body of the called methods are very small. Hence, the most of the time cost is from invocation itself. So, if the service method is a more complex and real one, the time cost for getting caller id and caller name will far less than 5%.

Moreover, even at 5% time cost, we conjecture that it is an acceptable price to pay for obtaining the crucial information for identifying which client is currently using a particular service.

VII. Conclusions

In the highly dynamic environment of the SOA, where software can be replaced on the fly at runtime, the challenges for ensuring correct behavior increase as the software has to be checked at runtime. In this context, we have identified two properties, that we consider are required to make a dynamic monitor for dynamic SOA systems: (i) *resilience to dynamicity*, i.e., the monitor is able to maintain state even if the service implementation is substituted at runtime, and

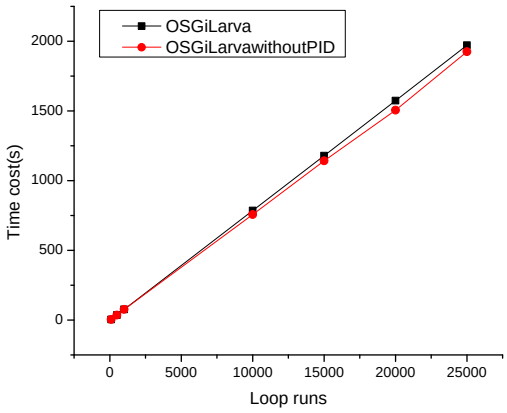


Figure 17. Comparing time cost of the case study example with OSGiLarva but with or without client Id

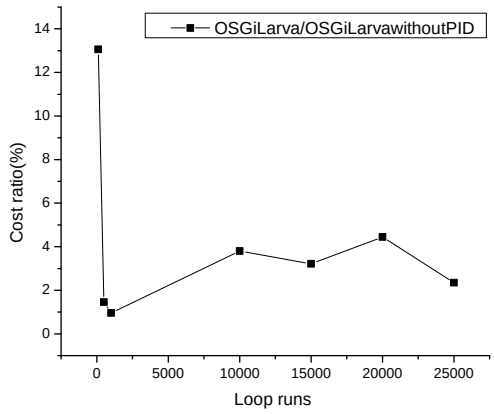


Figure 18. Comparing cost ratio of the case study example with OSGiLarva but with or without Client Id

(ii) *comprehensiveness*, i.e., that no implementation of the service can bypass the monitor's observation.

We have instantiated the approach in the context of the OSGi framework through a preliminary implementation, OSGiLarva, which integrates an adaptation of two existing tools: Larva and LogOs. Similar to Larva, OSGiLarva accepts the Larva property description language as input, hence inheriting all its features, including its expressiveness and its readability for non-expert users. Furthermore, it enables the description of both class properties and instance properties. This feature has been instrumental for OSGiLarva to monitor both properties which span the whole duration of the interface life cycle, and the individual client's point of view of the service, possibly spanning over different implementations of the service requests. We have also extended the Larva event description language, in order to consider not only calls or return of method calls, but also OSGi framework events such as the registration of a service or its request by a client; this has been achieved by introducing reserved event names which are usable transparently as if using standard method calls.

As observed in section VI, our approach is not so inefficient when compared to injection-based monitoring tools like Larva. While our approach is based on an OSGi hook observing all occurring events instead of aspect-oriented programming, the extra cost is small: tending to less than 1% increase in overheads. Since, this approach is crucial for dynamicity resilience, the cost incurred seems to be a reasonable.

An interesting element of this approach is its non-intrusive aspect. Indeed, in contrast to the aspect-oriented approach, we keep the original byte-code unchanged. This property can be useful if we want to switch off a monitor or be able to check the binary signature of the code as an authentication credential [18].

Finally, the notion of comprehensiveness also has a number of benefits since anybody with some privileged access to the platform (user, developer, or service) can define a behavioral property and ask the system to check if services respect it. This can be done for many reasons, such as: debugging deployment, privacy concerns, or to learn about typical usage patterns of a service.

VIII. Future Works

The current implementation of OSGiLarva is not complete with respect to our requirements. For instance, we have some works to do on the deployment step, in order to make it more autonomous. Each Larva property file is associated to a single interface. In the future, we aim to enable the framework to associate one file to possibly several interfaces. Moreover, in a next version of the tool, we could make some propositions to reduce the OSGiLarva time cost. For instance, we could make OSGiLarva asynchronous, by exporting monitors to separate threads, or we can limit monitoring to only occur within a fixed period of time: if the property is respected during one week by a given consumer, we can consider that it will still respect it afterwards. In OSGiLarva, the removal of a monitor is straightforward since it is non-intrusive. Similarly, one can consider sampling: monitoring only a random distribution of users, relying on the probability that the error would still occur in the sample.

References

[1] Y. Dan, N. Stouls, S. Frénot, and C. Colombo, "A Monitoring Approach for Dynamic Service-Oriented Architecture Systems," in *The Fourth International Conferences on Advanced Service Computing*, Nice, France, 2012. [Online]. Available: http://hal.inria.fr/hal-00695830

[2] Open Service Gateway Initiative (OSGi), http://www.osgi.org/ [retrieved: June, 2012].

[3] T. Thai and H. Lam, *.Net Framework Essentials*. O'Reilly Media, Incorporated, 2003.

[4] P. O. Meredith, D. Jin, D. Griffith, F. Chen, and G. Roşu, "An Overview of the MOP Runtime Verification Framework," *International Journal on Software Techniques for Technology Transfer*, 2011.

[5] C. Colombo, G. J. Pace, and G. Schneider, "Larva - safer monitoring of real-time java programs," in *SEFM*, 2009.

[6] G. T. Leavens, K. R. M. Leino, E. Poll, C. Ruby, and B. Jacobs, "JML: notations and tools supporting detailed design in Java," in *OOPSLA 2000 COMPANION*. ACM, 2000, pp. 105–106.

[7] M. Barnett, R. DeLine, M. Fähndrich, B. Jacobs, K. R. M. Leino, W. Schulte, and H. Venter, "The Spec# Programming System: Challenges and Directions," in *VSTTE*, ser. LNCS, vol. 4171. Springer, 2005, pp. 144–152.

[8] T. Jeron, H. Marchand, A. Rollet, Y. Falcone, and O. N. Timo, "Runtime Enforcement of Timed Properties," in *3rd international conference on Runtime Verification (RV)*, Septembre 2012.

[9] C. Simache, M. Kaaniche, and A. Saidane, "Event log based dependability analysis of windows nt and 2k systems," in *International Symposium on Dependable Computing*, 2002, pp. 311–315.

[10] S. Axelsson, U. Lindqvist, and U. Gustafson, "An approach to UNIX security logging," in *21st National Information Systems Security Conference*, 1998, pp. 62–75.

[11] H. R. M. Nezhad, R. Saint-Paul, F. Casati, and B. Benatallah, "Event correlation for process discovery from web service interaction logs," *VLDB J.*, vol. 20, no. 3, pp. 417–444, 2011.

[12] L. Baresi, D. Bianculli, C. Ghezzi, S. Guinea, and P. Spoletini, "Validation of web service compositions," *IET Software*, vol. 1, no. 6, pp. 219–232, 2007.

[13] F. Barbon, P. Traverso, M. Pistore, and M. Trainotti, "Run-time monitoring of instances and classes of web service compositions," in *Proceedings of the IEEE International Conference on Web Services*, ser. ICWS '06. IEEE Computer Society, 2006, pp. 63–71.

[14] Y.-C. Wu and H. C. Jiau, "A monitoring mechanism to support agility in service-based application evolution," *SIGSOFT Softw. Eng. Notes*, vol. 37, no. 5, pp. 1–10, Sep. 2012. [Online]. Available: http://doi.acm.org/10.1145/2347696.2347714

[15] S. Frénot and J. Ponge, "LogOS: an Automatic Logging Framework for Service-Oriented Architectures," in *38th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, Izmir, Turquie, Sep. 2012. [Online]. Available: http://hal.inria.fr/hal-00709534

[16] D. Le Métayer, M. Maarek, E. Mazza, M.-L. Potet, S. Frénot, V. Viet Triem Tong, N. Craipeau, R. Hardouin, C. Alleaune, V.-L. Benabou, D. Beras, C. Bidan, G. Goessler, J. Le Clainche, L. Mé, and S. Steer, "Liability in Software Engineering Overview of the LISE Approach and Illustration on a Case Study," in *ICSE'10*. ACM/IEEE, 2010, p. 135.

[17] C. Colombo, G. J. Pace, and G. Schneider, "Dynamic event-based runtime monitoring of real-time and contextual properties," in *FMICS*, ser. Lecture Notes in Computer Science, D. D. Cofer and A. Fantechi, Eds., vol. 5596. Springer, 2008, pp. 135–149.

[18] P. England, "Practical Techniques for Operating System Attestation," in *1st international conference on Trusted Computing and Trust in Information Technologies (Trust'08)*. Springer-Verlag, 2008, pp. 1–13.

Ensembles of Decision Trees for Network Intrusion Detection Systems

Alexandre Balon-Perin
abalonpe@ulb.ac.be

Ecole Polytechnique
Université libre de Bruxelles
Brussels, Belgium

Björn Gambäck
gamback@idi.ntnu.no

Department of Computer and Information Science
Norwegian University of Science and Technology
Trondheim, Norway

***Abstract*—The paper discusses intrusion detection systems built using ensemble approaches, i.e., by combining several machine learning algorithms. The main idea is to exploit the strengths of each algorithm of the ensemble to obtain a robust classifier. Network attacks can be divided into four classes: probe, remote to local, denial of service, and user to root. Each module of the ensemble designed in this work is itself an ensemble created by using bagging of decision trees and is specialized on the detection of one class of attacks. Experiments highlighted the efficiency of the approach and showed that increased accuracy can be obtained when each class of attacks is treated as a separate problem and handled by specialized algorithms. In all experiments, the ensemble was able to decrease the number of false positives and false negatives. However, some limitations of the used dataset (KDD99) were observed. In particular, the distribution of examples of remote to local attacks between the training set and test set made it difficult to evaluate the ensemble for this class of attacks. Furthermore, the algorithms need to be trained with specific feature subsets selected according to their relevance to the class of attacks being detected.**

***Keywords*-intrusion detection, ensemble approaches, bagging, decision trees, support vector machines.**

I. Introduction

Intrusion detection systems (IDSs) are monitoring devices that have been added to the wall of security in order to prevent malicious activity on a system. Here we will focus on network intrusion detection systems mainly because they can detect the widest range of attacks compared to other types of IDSs. In particular the paper discusses machine learning based mechanisms that can enable the network IDS to detect modified versions of previously seen attacks and completely new types of attacks [1].

Network IDSs analyse traffic to detect on-going and incoming attacks on a network. Additionally, they must provide concise but sound reports of attacks in order to facilitate the prevention of future intrusions and to inform the network administrators that the system has been compromised. Current commercial IDSs mainly use a database of rules (*signatures*), to try to detect attacks on a network or on a host computer. This detection method is presently the most accurate, but also the easiest to evade for experienced malicious users, because variants of known attacks (with slightly different signatures) are considered harmless by the IDS and can pass through without warning. New attacks and attacks exploiting zero-day vulnerabilities can also slip through the security net if their signatures are unknown to the IDS. A *zero-day vulnerability* is a software weakness unknown by the system developers, which potentially could allow an attacker to compromise the system. 'Zero-day' refers to the first day, day zero, that the vulnerability was observed.

In order for an intrusion detection system to be able to detect previously unseen attacks or variants of known attacks, there is a need for mechanisms allowing the IDS to learn by itself to identify new attack types. However, the problem is further complicated by the extreme requirements of robustness of the IDS. It must be able to detect all previously seen and unseen attacks without failure, it must never let an attack pass through unnoticed, and it must never deliver unwanted warnings when the traffic is in fact legitimate. Sommer and Paxson (2010) give a summary of the main challenges that machine learning has to overcome to be useful for intrusion detection [2].

Despite these constraints and challenges, several attempts have been made to build automatically adaptable intrusion detection systems using various machine learning algorithms. So far though, the machine learning classifiers trigger too many false alarms to be useful in practice. Part of the problem is the lack of labelled datasets to train the classifiers on. The only freely available labelled dataset is the KDD99 dataset [3] described below (Section III). To address these problems, new machine learning paradigms have been introduced in the field of intrusion detection, and in general the machine learning community has in recent years paid more attention to ensemble approaches, that is, to combinations of several machine learning algorithms.

Network attacks can be divided into four classes: probe, remote to local, denial of service, and user to root. Most previous machine learning-based solutions include a single algorithm in charge of detecting all classes of attacks. Instead, in this work, one module of an ensemble is specialised on the detection of attacks belonging to one particular class. The main idea is to exploit the strengths of each algorithm of the ensemble to obtain a robust classifier. Ensembles are

particularly efficient in cases like this, when a problem can be segmented into parts, so that each module of the ensemble is assigned to one particular subproblem. The modules in turn include one or more algorithms cooperating.

Furthermore, each class of attacks is characterized by very specific properties, observable through the values of certain features on instances in the dataset belonging to a specific class of attacks. However, even though feature selection is often applied in IDSs using machine learning techniques, often only one set of features is selected for *all* classes of attacks. In this work, one set of features is selected for *each* class of attacks according to their relevance to the corresponding class. The corresponding algorithm(s) is then fed with the appropriate set of features. The system can, in theory, reach a very high accuracy with a small cost, and the ensemble processing can potentially be parallelized using a multicore architecture. In the best scenario, each algorithm could run on a different core of the processor allowing the IDS to attain extremely high performance.

The experiments performed in this paper are in direct continuity of the work done by Mukkamala *et al.* [4]–[6], which identified the key features relevant to each of the four classes of attacks. The objectives of our experiments were multiple. In particular, to answer the following questions:

- Can ensemble approaches improve intrusion detection accuracy even when using the simplest algorithms without fine-tuning?
- Are the results of Mukkamala *et al.* (2005) [4] concerning the features selected by the three algorithms support vector machines (SVM), linear genetic programming (LGP), and multivariate adaptive regression splines (MARS), for each class of attacks, correct?
- Are the false positive (FP) and false negative (FN) rates close enough to zero for the IDS to be efficient?

The rest of the paper is laid out as follows: First, Section II introduces the machine learning methods utilized in the paper, in particular discussing ensembles and feature selection. Section III then discusses the data set used in the experiments, while Section IV gives an overview of the state-of-the-art and related work, in particular focusing on previous efforts in applying ensemble-based methods to intrusion detection. The core of the paper is Section V that details two rounds of experiments carried out, on feature selection for ensembles resp. on feeding an ensemble of machine learning algorithms with the most successful sets of features identified. Section VI then discusses the results of the experiments at length and points to ways in which the present work could be extended. Finally, Section VII sums up the previous discussion.

II. Ensemble-based Intrusion Detection

Machine learning algorithms operate in two main steps. In the first, the algorithm uses a training dataset to build a model of the data. In the second step, the model is applied to new examples. Usually, a test set is used to assess the performance of the algorithm. The model differs greatly depending on the type of algorithm used. In the case of regression, the algorithm must find the function that fits the data as well as possible. In the case of classification, the algorithm must find decision boundaries that separate the data as well as possible according to the number of desired classes. In both cases, a cost function is used to evaluate how good the model fits the data. The goal of the machine learning algorithm is to find the model that minimizes the cost function.

A. Supervised and Unsupervised Learning

In general, machine learning algorithms can be divided into two major classes depending on their learning technique: supervised and unsupervised. Supervised learning implies to obtain a training dataset in which every entry is labelled with class the example belongs to, while unsupervised learning algorithms do not need the dataset to be labelled. This is the most obvious disadvantage of supervised learning: obtaining data is cheap whereas obtaining labels for the data is very expensive in terms of both time and money because one or more experts must go through millions of examples and assign them a label. Apart from this main drawback, supervised learning also has some advantages. The first one is the ease of use and interpretation of the results. Indeed, the output of the classifier belongs to one of the classes defined by the labels of the dataset. The second advantage of supervised learning is its accuracy to classify similar examples. However, this accuracy drops significantly when the new examples are not so similar to the ones in the training set [7].

The most popular technique of unsupervised learning is clustering, where the algorithm exploits the similarity of the examples in order to form clusters or groups of instances. Examples belonging to the same cluster are assumed to have similar properties and belong to the same class. In contrast to supervised learning, disadvantages of unsupervised learning include manual choice of the number of cluster that the algorithm must form, lower accuracy of the prediction, and that the meaning of each cluster must be interpreted to understand the output. However, unsupervised learning is more robust to large variations. This is a very important advantage when applied to the problem of intrusion detection, since it means that unsupervised learning is able to generalize to new types of attacks much better than supervised learning. In particular, this property could be quite beneficial when trying to detect zero-day vulnerabilities.

B. Ensembles

The ensemble method is a way to build different types of approaches to solving the same problem: the outputs of several algorithms used as predictors for a particular problem are combined to improve the accuracy of the overall system. Ensemble approaches were introduced for the first time in the late 80s. In 1990, Hansen and Salamon showed that

the combination of several artificial neural networks can drastically improve the accuracy of the predictions [8]. The same year, Schapire showed theoretically that if weak learners (i.e., classifiers able to correctly classify only a small fraction of the examples in a dataset) are combined, it is possible to obtain an arbitrary high accuracy [9].

The difficulty of ensemble approaches lays in the choice of the algorithms constituting the ensemble and the decision function that combines the results of the different algorithms. Often, the more algorithms the better, but it is important to take into account the computational expense added by each new algorithm. The decision function is often a majority vote that is both simple and efficient, but alternatives should be analysed to obtain an optimal combination. Another advantage of ensemble approaches is their modular structure, unlike hybrid constructions that are engineered with algorithms having non-interchangeable positions. Consequently, the ensemble designer can easily replace one or more algorithms with a more accurate one.

Bagging and boosting are the two main techniques used to combine the algorithms in an ensemble. In an ensemble using the *boosting* technique, the algorithms are used sequentially. The first algorithm analyses all the examples in the dataset and assigns weights to each of them. The examples with a higher value for the weight are the ones that were classified wrongly by the algorithm. Then, the next algorithm receives as input the dataset as well as the weights for all examples in the dataset. The weights allow the algorithm to focus on the examples that were the most difficult to classify. These weights are updated according to the results of the second algorithm and the process moves to the third algorithm. This sequence continues until the last algorithm of the ensemble has processed the data. The advantage of this technique is that the most difficult examples can be classified correctly without adding too much computational overload. The use of weights, which are continuously updated, reduces the processing time as the data goes down the chain of algorithms.

In an ensemble using the *bagging* technique, all algorithms of the ensemble are used in parallel. In this case, each algorithm builds a different model of the data and the outputs of all predictors are combined to obtain the final output of the ensemble. In order to build different models, either each algorithm of the ensemble, or the data fed to each algorithm, or both, can be different. Since all algorithms perform in parallel, each of them can be executed on a different processor to speed up the computation. This is an important advantage over the boosting technique because nowadays multicore processors are very common even on personal computers. With this kind of architecture, the ensemble does not significantly increase the processing time compared to a single algorithm because the only additional time needed is used for the decision function that combines the outputs of all algorithms.

C. Feature Selection

Feature selection is a very efficient way to reduce the dimensionality of a problem. Redundant and irrelevant variables are removed from the data before being fed to the machine learning algorithm used as a classifier. Feature selection is a preprocessing step that commonly is independent of the choice of the learning algorithm. It can be used in order to improve the computational speed with minimum reduction of accuracy. Other advantages include noise reduction and robustness against over-fitting since it introduces bias but drastically reduces the variance. Generally, automatic selection of features works much better than manual selection because the algorithm is able to find correlations between the features that are not always obvious even for a human expert. Feature selection is an important preprocessing step of a machine learning algorithm that should not be overlooked. In particular, it should always be applied when the problem has a high dimensionality, as is the case for intrusion detection, since there is no point in feeding an algorithm with features that are irrelevant or add an insignificant amount of new information.

The main feature selection algorithms are minimum redundancy maximum relevance (mRMR) and principal component analysis (PCA). The former selects the subset of variables most relevant to the problem. The variables are ranked according to the information that they contain. This quantity of information is calculated by using the concept of entropy from information theory. The latter, PCA transforms the set of variables into a new smaller set of features. In both cases, the goal is to extract as much information as possible from as few features as possible. While PCA has been extensively used for the problem of intrusion detection, particularly on the KDD99 dataset, surprisingly, mRMR seems not to have been used much or at all [10].

III. The KDD99 Dataset

As observed in the introduction, part of the problem of automatically creating good intrusion detection systems is the lack of labelled datasets to train on. The only one freely available is the KDD Cup 99 dataset, which was used for the first time in the 3rd International Knowledge Discovery and Data Mining Tools Competition in 1999. It is an adaptation of the DARPA98 dataset [11] created in 1998 by the (then) Defense Advanced Research Projects Agency (DARPA) Intrusion Detection Evaluation Group (now the Cyber Systems and Technology Group of MIT Lincoln Laboratory). The DARPA98 set includes seven weeks of data (captured in the form of a tcpdump) from traffic passing through a network engineered for the purpose, i.e., the traffic was generated in a simulated and controlled environment.

A few alternative datasets exist, but are limited by either not being generally accessible to the research community or by not being annotated. The UNB ISCX Intrusion Detection Evaluation DataSet [12] from the Information Security Centre

of eXcellence at University of New Brunswick contains more realistic/real network traffic data than the KDD99 dataset (i.e., mainly normal traffic, with just some intrusion attempts). However, it seems unfortunately currently not to be available to other researchers. Publically available datasets from, for example, the Internet Traffic Archive (http://ita.ee.lbl.gov) and University of New Mexico (http://www.cs.unm.edu/~immsec) contain a lot of data, but unannotated. Tavallaee *et al.* (2010) give an overview of most currently available datasets [13].

A. Types of Attacks

All the examples in the KDD99 dataset are separated into five classes: `Normal`, `Probe`, `R2L` (remote to local), `DoS` (denial of service), and `U2R` (user to root). The class `Normal` of course denotes normal (legitimate) network traffic. The other four classes denote different types of attacks (intrusion attempts) and are further described in turn below.

`Probe` attacks are scouting missions used to gather information about the targeted network or a specific machine on a network: attackers scan a network to find vulnerabilities and to create a map of the network, often as the first step of one of other types of attacks. Hence it is crucial to detect this type of attacks. However, it is difficult to differentiate attacks from regular actions, since probing or scanning typically abuse perfectly legitimate features used by network administrators to check on machines in a network. The most common program to scan a network is 'nmap', which can be used to look for active machines and active ports on a machine, or to discover the type and version of the server and the operating system. Other probes such as 'saint' and 'satan' are specialised in discovering vulnerabilities in the targeted system.

In `R2L` attacks, external attackers start a session on a computer outside of the targeted network and then manage to exploit some vulnerability in a system in order to get local user access on a computer in the network. In order to do this, the attackers must have the ability to send network packets to the victim host. Many remote to local attacks (e.g., 'warezmaster', 'warezclient', 'imap', 'named', and 'sendmail') exploit bugs or weaknesses in different Internet protocols such as FTP, DNS, and SMTP. Other `R2L` attacks exploit system misconfigurations (e.g., 'dictionary', 'ftp_write', 'guest', and 'xsnoop').

`DoS` attacks aim either to overload a system so that it cannot process all requests, or to directly deny legitimate users access to a system or network resource, such as network bandwidth, computer memory or computing power. An attacker can abuse a legitimate feature of a network protocol by, for example, sending replies to protocol queriers faster than the destination of the query in order to falsify the information contained in the network tables of the victim. Some of these attacks are 'mailbomb', 'neptune', 'smurf', and 'ARP poisoning'. Others such as 'teardrop' and 'ping of death' ('pod') exploit implementation bugs of the TCP/IP protocol, while attacks like 'apache2', 'back', and 'syslogd' target a specific program running on the victim host.

In the type of `DoS` attacks focusing on resource exhaustion, the attacker typically sends a huge amount of queries in a short amount of time to the targeted victim. If the victim is a server, resource exhaustion occurs when the server receives more queries than it can process: in a 'udpstorm' (also called 'UDP Port DoS' attack or 'UDP packet storm'), an attacker forges a packet with a spoofed source address of a host running an "echo" or "chargen" process and sends it to another hosts running a similar process. The receiving host replies with an echo packet to the spoofed source, which replies with another echo packet, etc., creating a loop leading to resource exhaustion or performance degradation [14].

A variant of `DoS` used extensively by hackers is distributed denial of service (DDoS) [15], [16]. A DDoS is performed in two main steps. First, an attacker gains control over a (often huge) number of computers, called slaves or zombies, by exploiting unpatched vulnerabilities found in the target systems. Then the attacker orders all slaves to query a designated machine (usually a server) at the same time.

In `U2R` attacks, access to a normal user account (with restricted rights) is used as a starting point to gain root user permissions and take over a system, e.g., by exploiting some vulnerability in the system. There are several different types of user to root attacks, with the most common being 'buffer overflow' [17] that aims to corrupt a program with high privileges (i.e., root) in order to take control of the host computer running the vulnerable program. The attacker uses a buffer with non-existent or poorly performed boundary checking to launch a root shell and then corrupts the stack pointer to point to the attacker's own malicious code. Other `U2R` attacks such as 'loadmodule' or 'perl' take advantage of the way some programs sanitize their environment. Others still (e.g., 'ps') exploit poor management of temporary files.

Table I
DISTRIBUTION OF INSTANCES IN THE KDD99 DATASETS

Class	Training set	Test set
`Normal`	972,781	60,593
`Probe`	41,102	4,166
`R2L`	1,126	16,347
`DoS`	3,883,370	229,853
`U2R`	52	70
Total	4,898,431	311,029

B. Training and Test Sets

The KDD99 dataset is divided into a training set and a test set. Table I shows the distribution of instances of the KDD99 training and test sets over the different classes of attacks. The various types of attacks belonging to each of these classes are further detailed in Table II.

Table II
TYPES OF ATTACKS IN THE KDD99 DATASETS

Class	Attack	Training set	Test set
Probe	satan	15,892	1,633
	ipsweep	12,481	306
	portsweep	10,413	354
	nmap	2,316	84
	mscan	0	1,053
	saint	0	736
	Total	41,102	4,166
R2L	warezclient	1020	0
	guess_passwd	53	4,367
	warezmaster	20	1,602
	imap	12	1
	ftp_write	8	3
	multihop	7	18
	phf	4	2
	spy	2	0
	snmpgetattack	0	7,741
	snmpguess	0	2,406
	httptunnel	0	158
	named	0	17
	sendmail	0	17
	xlock	0	9
	xsnoop	0	4
	worm	0	2
	Total	1,126	16,347
DoS	smurf	2,807,886	164,091
	neptune	1,072,017	58,001
	back	2,203	1,098
	teardrop	979	12
	pod	264	87
	land	21	9
	mailbomb	0	5,000
	apache2	0	794
	processtable	0	759
	udpstorm	0	2
	Total	3,883,370	229,853
U2R	buffer_overflow	30	22
	rootkit	10	13
	loadmodule	9	2
	perl	3	2
	ps	0	16
	xterm	0	13
	sqlattack	0	2
	Total	52	70
Normal		972,781	60,593
Total		4,898,431	311,029

Each entry in the sets is represented by a label and 41 features such as `duration`, `src_bytes`, and `dst_bytes`. Of the features, 38 are numerical and thus only three non-numerical: `protocol_type`, `service`, and `flag`. For the non-numerical features, there are three protocol types (`TCP`, `UDP`, and `ICMP`), 70 different services, and 11 possible flags. The non-numerical variables are normally transformed into numerical ones to ensure that all the machine learning algorithms are able to process their values.

The KDD99 training set contains $4,898,431$ entries and is highly unbalanced. Whereas the `DoS` class contains $3,883,370$ instances, the classes `U2R` and `R2L` are represented by only 52 and $1,126$ instances, respectively. With such a small number of examples to train on, it can be expected that it will be difficult for the classifiers to predict the correct classes of unseen examples.

The test set is composed of $311,029$ entries with a distribution of the examples over the different classes similar to that in the training set. However, the number of examples belonging to the class `R2L` is more than ten times higher than in the training set, so that in order to perform well on the test set, the predictor must acquire a very high power of generalisation with $1,126$ training examples. Most importantly, the number of unseen attacks added in the test set is huge: for the classes `U2R`, `R2L` and `Probe`, it is respectively $44.29\,\%$, $63.34\,\%$ and $42.94\,\%$. Furthermore, the attacks 'spy' and 'warezclient' belonging to the class `R2L` are not represented in the test set. In particular, 'warezclient' attacks count for more than $90\,\%$ of the `R2L` training set.

Notably, two entries in the test set erroneously have a `service` value of `ICMP`, as also previously reported [18]. Those faulty entries were removed from the test set before carrying out the experiments reported in Section V.

The major criticisms of the KDD99 dataset include the unbalanced distribution of the data, that the redundant records can introduce a bias in the learning phase because of their frequency, that the dataset includes old attacks which have been mostly mitigated, and that the data were captured from a controlled environment somewhat different from what is observed in the wild. The first two issues can be addressed by sampling appropriate sets of examples for each class of attacks. However, the distribution of `R2L` attacks in the training set and the test set is a problem that is difficult to overcome.

Nevertheless, the KDD99 dataset is far from useless. Firstly, if an IDS using machine learning does not perform well on old attack provided that the data are well sampled, why would it on newer ones? Furthermore, most of the research in the field of machine learning applied to intrusion detection uses the KDD99 dataset, making it a vector of comparison between different approaches. The controlled nature of the environment in which the data were captured is probably the most problematic. For example, the high number of attacks in comparison to normal traffic does not reflect the reality of a network in which almost all traffic is normal. Again, appropriate sampling is required. In addition, the IDS should at least be accurate on data produced by a simulated environment before being tested on a real network where the traffic pattern is probably less predictable.

IV. Related Work

Intrusion detection systems have been around since the 80s. In 1980, Anderson introduced the concept of host-based intrusion detection [19]. Seven years later, Denning laid the foundations of intrusion detection system development [20]. Network-based intrusion detection systems were introduced in 1990 [21]. In the late 90s, researchers in artificial intelligence started to investigate applying machine learning algorithms to improve intrusion detection.

An intrusion detection system should be able to autonomously recognize malicious actions in order to defend itself against variants of previously seen attacks and against attacks exploiting zero-day vulnerabilities. Misuse-based IDSs can only detect attacks whose signatures are available in their signature database. Signatures of attacks are very specific, and a slight variation of the attack can make it unnoticeable for the IDS. That is why learning mechanisms must be implemented to detect and prevent these attacks without having to wait for an update of the signature database or a patch for the vulnerable system. Still, machine learning algorithms are designed to recognize examples similar to those available in the training set used to build the model of the data. Consequently, an IDS using machine learning would have a hard time detecting attacks which patterns are totally different from the data previously seen. In other words, even though machine learning is a suitable candidate to detect variants of known attacks, detecting completely new types of attacks might be out of reach for these kinds of algorithms.

For a summary of most research involving machine learning applied to IDSs until 2007, see Wu & Banzhaf (2010) who cover a range of techniques, including fuzzy sets, soft computing, and bio-inspired methods such as artificial neural networks, evolutionary computing, artificial immune systems, and swarm intelligence; comparing the performance of the algorithms on the KDD99 test set and showing that all algorithms perform poorly on the `U2R` and `R2L` classes [22]. The best results reported are by genetic programming with transformation functions for `R2L` and `Probe` and by linear genetic programming (LGP) for `DoS` and `U2R` (with $80.22\,\%$, $97.29\,\%$, $99.70\,\%$ and $76.30\,\%$ accuracy, respectively). However, since ensemble-based methods are fairly new in being applied to intrusion detection, the description of them in the review is somewhat limited. The first works on the topic date from 2003 and many papers were written in 2004 and 2005; recently (from 2010 onwards), there has been a renewed interest of ensembles in this field.

Abrahams *et al.* have performed several types of ensemble-based experiments, all on a subset of the DARPA98 dataset composed of $11,982$ randomly selected instances from the original dataset with a number of data for each class proportional to the size of the class, except for the smallest class which was included entirely. This data was then divided into a training set of $5,092$ and a test set of $6,890$ instances.

First, in [23], an ensemble composed of different types of artificial neural networks (ANN), support vector machines (SVM) with radial basis function kernel, and multivariate adaptive regression splines (MARS) combined using bagging techniques was compared to the results obtained by each algorithm executed separately. SVM used alone outperformed the other single algorithms, but was totally outperformed by the ensemble. This ensemble surprisingly obtained a $100\,\%$ accuracy on the test set for the `R2L` class. However, the researchers warn that some of these results might not be statistically significant because of the unbalanced dataset.

Second, in [24], [25], the combination of classification and regression trees (CART) and Bayesian networks (BN) in an ensemble using bagging techniques was explored, as well as the performance of the two algorithms when executed alone. Feature selection was applied to speed up the processing: the performance on the set of 41 features was compared to a set of 12 selected by BN, 17 selected by CART and 19 features selected by another study. BN performed worse with a smaller set of features except on the `Normal` class. However, when using the set of 19 features, BN and CART complemented each other to increase the IDS accuracy for all classes. The final ensemble was composed of three CART to detect `Normal`, `Probe` and `U2R` examples, respectively; one ensemble of one CART and one BN to detect `R2L` examples; and one ensemble of one CART and one BN to detect `DoS` examples — with each classifier trained on its resp. reduced set of features; an approach quite similar to the one used in the present paper.

This was then extended by adding a hybrid model composed of SVM and decision trees (DT) to the ensemble [26], [27]. In the new model, the data was first sent to the DT that generated a tree to fit the features and values of each example in the dataset. The tree was then sent to the SVM to produce the final output. A single DT was in charge of detecting `U2R` attacks, a single SVM in charge of detecting `DoS` attacks, the hybrid model in charge of `Normal` instances, and the same ensemble as above in charge of `Probe` and `R2L` attacks. However, the hybrid model did not seem to help much.

Third, in [28], [29], fuzzy rule-based classifiers, linear genetic programming (LGP), DT, SVM, and an ensemble were evaluated using feature selection to reduce the number of variables of the dataset to 12 on a subset of the DARPA 1998 dataset, selected in the same way as in the work mentioned previously. The fuzzy rule-based classifier outperformed the other methods when trained on all 41 features, with the second set of rules scoring 100 % accuracy for all classes of attacks; while LGP seemed more appropriate when using a smaller feature set, except for the U2R and Normal classes. The ensemble was composed of one DT in charge of the Normal instances, one LGP each for Probe, R2L and DoS, and one fuzzy set of rules for U2R. The results obtained with the ensemble were very encouraging with accuracy $> 99\,\%$ for all classes (on the subset data).

Finally, the results of several machine learning algorithms were compared in [30]. In particular, the performance of linear genetic programming (LGP), adaptive neural fuzzy inference system (ANFIS), and random forest (RF) were analysed, and an ensemble was created by combining the LGP, ANFIS, and RF algorithms. The ensemble outperformed the single algorithms, but its exact configuration is not described in the paper.

Folino *et al.* [31], [32], instead used the entire KDD99 dataset and examined the performance of a system composed of several genetic programming ensembles distributed on the network based on the island model. Each ensemble was trained on a different dataset for a number of rounds. Once the ensemble had been trained for one round, it was exchanged with the other islands through the distributed environment. The advantage of a distributed system, as pointed out by Folino *et al.*, is the increase in privacy and security in comparison to a central IDS that has to collect audit data from different nodes on the system. The system showed average performance for the Normal, Probe and DoS classes, but very low for the U2R and R2L classes. However, very few papers study distributed environment for intrusion detection even though this might be a very good idea.

Bahri *et al.* [33] introduced Greedy-Boost, a noise resistant adaptation of the AdaBoost boosting technique [34]. The Greedy-Boost classifier contrasts with AdaBoost by being a linear combination of models and by updating the distribution of weights according to the initial distribution instead of the previous one. Greedy-Boost's performance in terms of precision and recall on the KDD99 dataset was extraordinary good. In particular, the precision of the most difficult class (R2L) was much higher than what is usually observed. However, it is not clear from the paper if the model was evaluated on the test set, the training set, or a modified version of one of the sets.

Peng Zhang *et al.* [35] evaluated the robustness of an ensemble when confronted with "noisy" data sets, that is, data sets containing incorrectly labelled instances. In order to tolerate label imprecision and errors, they used an aggregate ensemble of SVM, DT, and logistic regression. An aggregate ensemble builds several classifiers over a range of data sets using different learning algorithms. The aggregate approach was shown to outperform both a horizontal ensemble framework in which classifiers were built over different data sets with only one learning algorithm for each set, and a vertical ensemble framework in which several classifiers were built over the data sets using different learning algorithms that were then combined into an ensemble.

The key conclusion from all these works is that ensemble approaches generally outperform approaches in which only one algorithm is used. An ensemble is a very efficient way to compensate for the low accuracy of a set of weak learners. Moreover, feature selection should provide specific subsets to train algorithms specialised in the detection of one particular class of attacks.

Mukkamala *et al.* [4]–[6] identified the five most important features for each class of attacks, as shown in Table III. The features were selected using support vector machines, linear genetic programming, and multivariate adaptive regression splines, with in total 16 distinct features selected for SVM,

Table III
MOST RELEVANT FEATURES FOR EACH ATTACK CLASS IN THE KDD99 DATASET ACCORDING TO MUKKAMALA *et al.* (2005) [4]

SVM features	LGP features	MARS features
	Probe	
src_bytes	srv_diff_host_rate	src_bytes
dst_host_srv_count	rerror_rate	dst_host_srv_count
count	dst_host_diff_srv_rate	dst_host_diff_srv_rate
protocol_type	logged_in	dst_host_same_srv_rate
srv_count	service	srv_count
	U2R	
src_bytes	root_shell	dst_host_srv_count
duration	dst_host_srv_serror_rate	duration
protocol_type	num_file_creations	count
logged_in	serror_rate	srv_count
flag	dst_host_same_src_port_rate	dst_host_count
	R2L	
srv_count	is_guest_login	srv_count
service	num_file_access	service
duration	dst_bytes	dst_host_srv_count
count	num_failed_logins	count
dst_host_count	logged_in	logged_in
	DoS	
count	count	count
srv_count	num_compromised	srv_count
dst_host_srv_serror_rate	wrong_fragments	dst_host_srv_diff_host_rate
serror_rate	land	src_bytes
dst_host_same_src_port_rate	logged_in	dst_bytes
	Normal	
dst_bytes	dst_bytes	dst_bytes
dst_host_count	src_bytes	src_bytes
logged_in	dst_host_rerror_rate	logged_in
dst_host_same_srv_rate	num_compromised	service
flag	hot	hot

21 for LGP and 13 for MARS. Features that are selected by at least two different algorithms for the same class of attacks are highlighted because they should most definitely be in the subset of features used to detect that specific class of attack.

Surprisingly, neither `protocol_type` nor `service` was selected by the three algorithms for the `DoS` class in the experiments by Mukkamala *et al.* (2005) [4]. In contrast, Kayacik *et al.* (2007) [36] concluded that those two features were the most significant ones for the denial of service class of attacks, even though the experiments by Kayacik *et al.* were conducted on hierarchical self-organizing maps (SOM).

V. Experiments

The problem of intrusion detection can be divided into five distinct subproblems, one for each class of instances (`Normal` and the four types of attacks: `Probe`, `U2R`, `R2L`, and `DoS`). Here each problem will be handled by one or more algorithms of an ensemble, allowing each subproblem to be treated separately in the experiments and to join the solutions to the subproblems into a general solution for the problem of intrusion detection.

A dedicated training dataset for each attack subproblem was built by sampling a number of instances from that class of attacks and the same number from the class `Normal` in order to have a balanced dataset with $50\,\%$ anomalous and $50\,\%$ normal examples (no algorithm was explicitly designed to detect normal traffic). A balanced dataset is necessary to avoid the problem of skewed classes where the accuracy of the predictor can be made artificially high by increasing the number of instances from one of the classes.

For the classes of attacks with few examples, `R2L` and `U2R`, the entire set was selected (i.e., 52 instances of `U2R` and $1,126$ of `R2L`). For the `Probe` class, $10,000$ instances were selected randomly. This number was chosen to have a significant sample with as many different examples as possible without affecting the training time too much. The `DoS` training set contains $3,883,370$ instances, with 'neptune' and 'smurf' attacks counting for the majority (with $1,072,017$ and $2,807,886$ instances, respectively). The other types of attacks have much smaller number of examples, e.g., the type of `DoS` called 'land' is represented only 21 times. For this reason, samples of $5,000$ examples each were selected randomly from the 'neptune' and 'smurf' sets. All examples of the other types of `DoS` attacks were included for a total of $13,467$ `DoS` instances. For all four classes, the same number of `Normal` instances was selected from the normal dataset leading to a total training set size of 104 examples for `U2R`, $2,252$ for `R2L`, $20,000$ for `Probe`, and $26,934$ for `DoS`.

In order to investigate the applicability of ensemble-based approaches to intrusion detection, two sets of experiments were carried out. The first step of the experiments was to assess the sets of features selected in [4], that is, the key features relevant to each of the four classes of attacks. Then in a second round of experiments, those sets were fed to an ensemble of machine learning algorithms. All models were evaluated by 10-fold cross-validation.

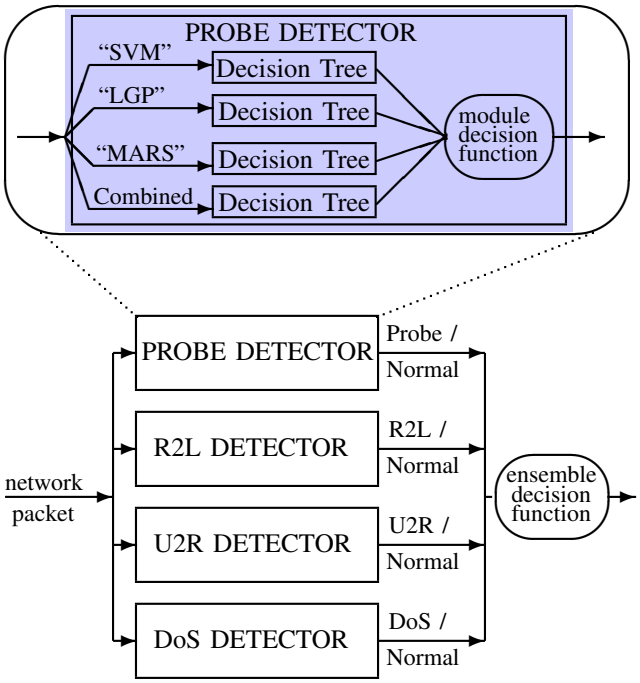


Figure 1. Overview of the ensemble model

A. Experimental Setup

Figure 1 gives a schematic overview of the ensemble model used in this work. First, the network packet being analysed is sent to four different detector modules, one each for `Probe`, `R2L`, `U2R`, and `DoS`. Each module executes a preprocessing step to extract a number of features from the packet; the set of features varies depending on the module (as further described in Section V-B). The extracted features are then dispatched to different decision trees that have been previously trained with the same features on the training set, as shown at the top of the figure for the `Probe` detector. Each decision tree is a binary classifier which outputs 0 if the packet is considered normal traffic and 1 if the packet is classified as anomalous. A vector of dimension n containing the output of n classifiers is then fed to the module decision function. In the figure $n = 4$ (one each for the features selected by SVM, LGP and MARS plus the set of those features combined), but it could be any number of algorithms.

Finally, a vector of dimension 4 containing the output of each detector module is fed to the ensemble decision function that combines the results and outputs a value describing if the packet is considered normal or anomalous, and if anomalous from which class of attacks. The easiest situations are when all modules, or all modules except one, output `Normal`. In the former case, the system classifies the packet as normal. In the latter, the system classifies the packet

as anomalous and is able to unambiguously identify the class of attack concerned. If more than one module classify the packet as anomalous, it will be more difficult for the network administrator to understand which class of attack the anomalous packet belongs to.

The resulting model is an ensemble of ensembles with feature selection applied independently for each module. However, in this work, we will not be concerned with the decision functions for each module. Instead, we will evaluate the intersection of the sets of false positives and false negatives produced by the four algorithms in each module. This will give us the optimal performance that each module could achieve. The most important advantages of this model are the possibility to execute the algorithms in parallel and the modularity allowing the exchange of any algorithm of the ensemble without any modification of the rest. The complete ensemble model is as shown in Figure 2 (on the next page).

B. Feature Selection Assessment with Decision Trees

In the first experiment, several classifiers were trained with different number of features. The goal of the experiment was not to find the best algorithm possible and fine-tune it, but rather to conclude on how well an algorithm performs with a smaller set of features. In this case, it is only natural to use exactly the same setting for the algorithms and to compare the performance based only on the sets of features. Five decision trees were trained with different sets of features. Only the training set was used for this experiment. The results obtained represent the performance of the algorithms on the cross-validation set which is extracted from the training set. The model assessment experiment (described below, in Section V-C) evaluated performance on the test set.

The first classifier was trained with all 41 features in the dataset. The next three were trained with five features selected in [4] for each class of attacks by the three algorithms support vector machines (SVM), linear genetic programming (LGP) and multivariate adaptive regression splines (MARS). These features are listed in Table III. The last classifier was trained on a "combined" set of features: the union of the feature sets selected by the three algorithms from which redundant features have been removed. The number of features in the "combined" set is 11 for `Probe`, 14 for `U2R`, 11 for `R2L` and 12 for `DoS`, as can be seen in Figure 2. These additional sets help bringing down the number of false positives and false negatives, as we will see in the results of the experiments. Note that there is no extra cost from the extraction of these features from the original network packets since they have to be extracted for the other algorithms anyway.

The results obtained in terms of accuracy are shown in Table IV and can be compared to those obtained with 41 features by Peddabachigari *et al.* (2007) [27] using the same decision tree. For the class `Probe`, the accuracy of the classifier trained with all 41 features is exactly the same as reported in [27]: 99.86 %. The classifiers trained with sets of five features are not far behind the one trained with all 41 features. The reduced feature sets seem to be a good choice when the algorithms are trained using decision trees. However, the classifier fed with the five features selected by LGP performs slightly worse than the others and could be replaced by a more accurate algorithm.

Table IV
ACCURACY OF THE FEATURE SELECTION ASSESSMENT

Decision Tree Classifier	`Probe`	`U2R`	`R2L`	`DoS`
41 features	99.86	93.00	99.02	99.95
5 SVM features	99.82	96.00	98.58	93.35
5 LGP features	99.32	90.00	97.38	98.69
5 MARS features	99.75	97.00	98.04	99.86
11–14 combined features	99.90	96.00	98.93	99.95
Peddabachigari *et al.* (2007) [27]	99.86	68.00	84.19	96.83

The results for `U2R` are worse than for `Probe`, but this was expected: each false positive and false negative has a larger impact on the general accuracy due to the small number of examples. The results (93–97 %) are much better than the 68 % accuracy obtained by Peddabachigari *et al.* on `U2R`. However, the classifier trained on features selected by LGP again performed poorly. Interestingly, the algorithms trained on the features selected by SVM and MARS outperformed the one trained on all features. This is probably since 41 features are too many to generalize from given the small number of examples; recall that the training set for `U2R` only held 52 instances (cmf. Table I).

The results for `R2L` are similar to those obtained for `Probe`, even though the number of instances in the dataset is much smaller. The results (97–99 %) are also much better than Peddabachigari *et al.* who obtained 84 % accuracy on this class. This experiment clarifies that classifying `Probe` attacks and `R2L` attacks are two very distinct problems, even if they are both intrusions, which is why they should be treated separately. Again, the selected features seem to be a good choice even if a small drop of accuracy can be observed compared to `Probe`. The classifier trained on the features selected by MARS has a high rate of false positives and the one trained on features selected by LGP has the lowest accuracy, but also a lower false positive rate, which implies a higher false negative rate.

`DoS` also shows better results than Peddabachigari *et al.* who obtained 96.83 % accuracy. The classifier trained on features selected by SVM obtained the worst score, whereas features selected by MARS gave the best accuracy (99.86 %) after the set of all features and the combined feature set that both reached 99.95 %. This is important, since there is a set of five features that can perform almost as well as the full feature set even on larger number of training examples.

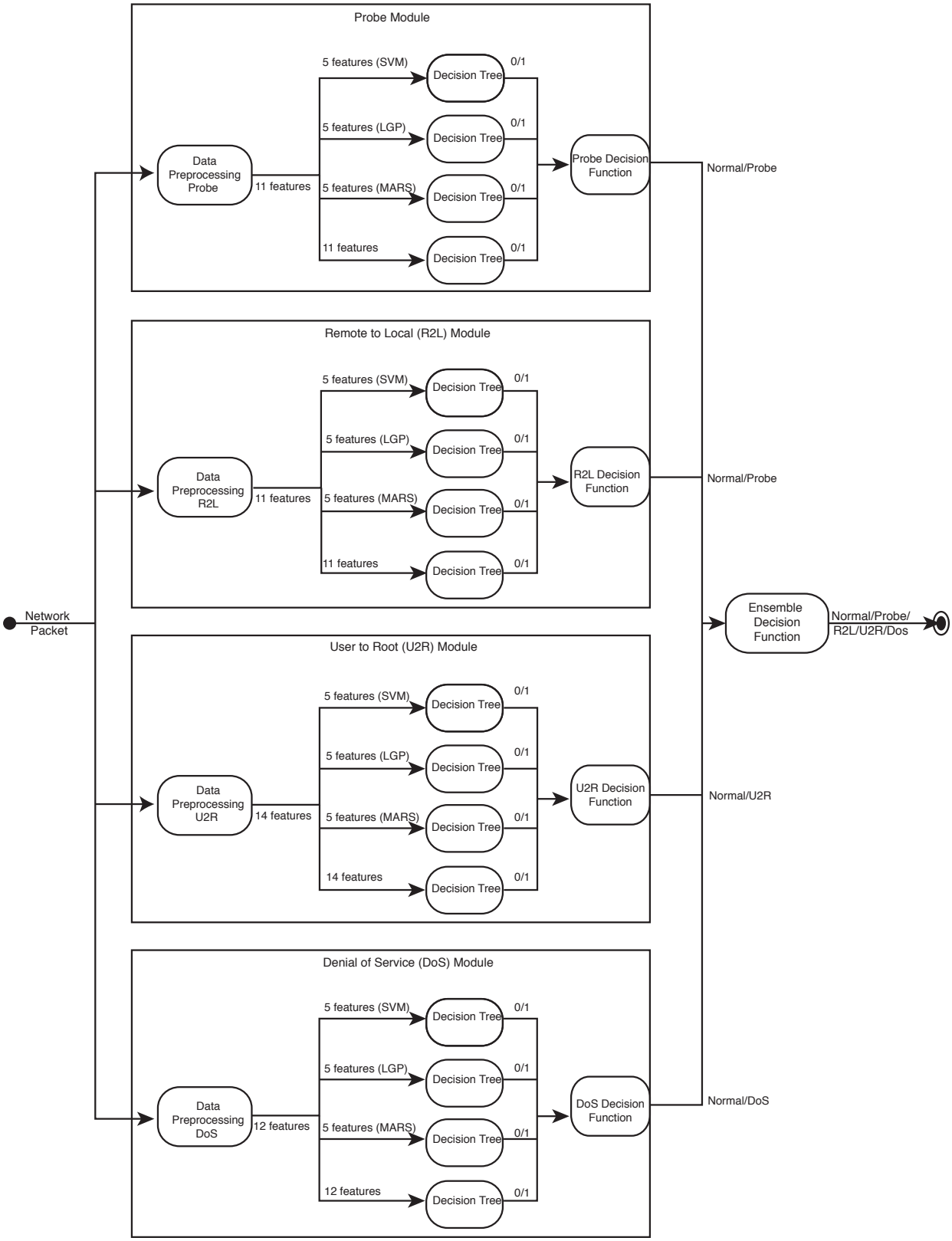


Figure 2. The complete ensemble. Each algorithm is a binary classifier outputting 0 if the packet is considered normal traffic and 1 if anomalous.

Table V
FEATURE SELECTION ASSESSMENT: FALSE POSITIVES

Decision Tree Classifier	Probe	U2R	R2L	DoS
41 features	12.0	4.0	17.0	6.0
ENSEMBLE$_{max}$	0.7	0.3	6.6	0.0

Table VI
FEATURE SELECTION ASSESSMENT: FALSE NEGATIVES

Decision Tree Classifier	Probe	U2R	R2L	DoS
41 features	17.0	3.0	10.0	8.0
ENSEMBLE$_{max}$	3.0	0.3	0.5	1.6

The overall numbers of false positives (FP) and false negatives (FN) drop significantly when using more than one algorithm, as Tables V and VI show. For the FP and FN analysis, we call ENSEMBLE$_{max}$ the number of examples wrongly classified by all three algorithms trained on sets of five features and the one trained on the "combined" feature set. This is the maximum an ensemble composed of the four algorithms could achieve if the combination of their individual results was optimal; here calculated by taking the intersection of the set of examples misclassified by each algorithm. The experiment was run ten times for each attack class to ensure accuracy of the results and to find the attack types in each class that ENSEMBLE$_{max}$ misclassified most. Hence the values displayed in the table are average values over the ten validation sets of the 10-fold cross-validation.

All types of Probe attacks appear at least once as a false negative, however, 'satan' and 'portsweep' seem to be the most difficult attacks to detect. When comparing the problematic instances of 'satan', 'portsweep' and 'ipsweep' with true instances of the same attack types, it seems that src_bytes is the feature that gives the classifiers most trouble. In fact, for probe attacks, src_bytes should be very small, although not equal to zero (since there is always a number of bytes contained in the header); when an example of these attacks has a high value for src_bytes, it goes undetected. This is a big problem since an attacker could easily fill the packets of the attack with random bytes to evade the IDS. It could seem like a good idea to get rid of this feature; however, src_bytes is very important to detect Probe attacks: the only classifier that performs poorly is the one trained on the features selected by LGP, a feature set that does not include src_bytes.

For the U2R class, in general either one false positive or one false negative appears in each test run. The false positive can be explained by the small number of examples in the dataset, only 52 Normal examples are present. The false negative is always a 'rootkit' attack that is wrongly classified as normal traffic, but it is not always the same instance, indicating that some information is missing for the decision tree to classify 'rootkit' attacks. These can be any kind of malware such as worm, Trojan or virus with the ability to hide its presence and actions to the users and processes of a computer; this is called a stealth attack. The diversity found in malware probably has a huge impact on the problem. Moreover, as shown in Table II, there are only 10 'rootkit' attacks in the dataset, increasing the difficulty. Examining the values of these examples for the 14 features of the combined algorithm revealed that almost all 10 instances have very different values for those features. The ENSEMBLE$_{max}$ performs perfectly in most cases, but it is difficult to conclude anything with such a small dataset: One false positive or false negative out of ten instances of the cross-validation set is quite a bad score.

The combination of all algorithms helps to bring down the number of false positives and false negatives also for R2L, but these numbers are again too high for a real-world application. There are eight different types of R2L attacks represented in the training set. After running the experiments ten times, only three types of these attacks trigger false negatives for the ENSEMBLE$_{max}$: 'spy', 'imap', and 'phf'. There is not much documentation about 'spy' attacks, which are not even represented in the test set. However, the signatures of 'imap' and 'phf' are described in [37]. Detection of these attacks requires very specific features. In the case of a 'phf' attack, the IDS according to Kendall (2007) "must monitor http requests watching for invocations of the phf command with arguments that specify commands to be run" [38]. None of the 41 features in the KDD99 dataset gives any information about a specific command being run on the system. It would be impractical to do so for each specific command vulnerable to an attack. However, this could be the reason why the machine learning algorithms are incapable of detecting these kinds of attacks with certainty. Without meaningful information, the algorithms are powerless in building a proper model.

There are two ways to solve this problem, either new features could be added to the dataset or an IDS using signatures of attacks should perform the detection for these particular types of attacks. In the former case, the new features should not be too specific to ensure that new attacks could also be identified. In the second case, the IDS loses its ability to detect similar attacks but its accuracy increases. To detect an 'imap' attack, an IDS should be "programmed to monitor network traffic for oversized Imap authentication strings" [38]. This seems more within reach of our IDS, since service and src_bytes are both represented in the feature set.

ENSEMBLE$_{max}$ was highly successful on the DoS class, returning zero false positives. Table VI shows that the number of false negatives is reduced as well. Three types of attacks trigger false negatives: 'smurf', 'neptune', and 'back'. The first two rarely appear in the list; however, the third seems

Table VII
ACCURACY OF THE MODEL ASSESSMENT

Decision Tree Classifier	Probe	U2R	R2L	DoS
41 features	93.09	90.00	50.00	79.34
5 SVM features	77.63	40.00	50.00	87.70
5 LGP features	87.48	83.57	61.03	76.10
5 MARS features	84.04	85.00	50.00	82.20
11–14 combined features	79.97	94.29	50.00	85.36

Table VIII
MODEL ASSESSMENT: FALSE POSITIVES

Decision Tree Classifier	Probe	U2R	R2L	DoS
41 features	86.0	3.0	0.0	69.0
ENSEMBLE$_{\max}$	11.4	1.6	1.0	16.6

Table IX
MODEL ASSESSMENT: FALSE NEGATIVES

Decision Tree Classifier	Probe	U2R	R2L	DoS
41 features	490	11	16,347	7,268
ENSEMBLE$_{\max}$	524	1	7,779	688

to be the most difficult type to handle. This is not a surprise, since to detect a 'back' the IDS must look for a big number of frontslashes ("/") in the request URL [37]. There are no features in the dataset taking this particularity into account. Consequently, the model has to rely on other features to make up for the lack of information, leading to an imperfect result. Nevertheless, as expected, ENSEMBLE$_{\max}$ brings robustness to the accuracy of the IDS.

C. Model Assessment Experiment

In the second round of experiments, several classifiers were trained with different number of features on examples from the training set. Decision tree was again the algorithm used as classifier. The goal of the experiment was to evaluate the model used in the previous experiment on the test set after training on the same number of examples as selected for the training set for each class in the first experiment. As discussed in Section III, the test set is composed of many examples of unseen attacks (attacks that are not represented in the training set). The experiment aimed to assess if the ensemble was capable of generalizing to new types of attacks belonging to the same classes as the ones previously seen.

In most cases, the accuracy of all algorithms degraded drastically in comparison to the first experiment as shown in Table VII, where the values represent one run of the program. In particular, the set of features selected by SVM obtains the worst results, and does not seem to generalize well to new types of attacks. The set selected by LGP managed to keep a respectable accuracy on the Probe class, while all classifiers except SVM showed results very similar to those in the feature selection experiments on U2R, with the "combined" set of features being the best one, outperforming even the algorithm trained with all 41 features in the same way that was observed in the feature selection experiment.

Particularly bad results could be expected for R2L because of the poor distribution of attacks in the training set, and Table VII confirms this: the accuracy of all algorithms is equal or close to the 50 % guessing baseline. Most of the attacks are 'warezclient' (1020 out of 1126 in total for the R2L training set) leaving only 106 instances of all other attack types (seven different types) to train on — and 'warezclient' is not even represented in the test set. There is no chance that the models built would perform well on new attacks (or even on old) with this limited training set. In addition, the results for DoS were much worse than in the first experiment, with the set of features selected by LGP obtaining by far the worst results. Nevertheless, all other algorithms performed better than the one trained with all features.

As Tables VIII and IX show, the ENSEMBLE$_{\max}$ is able to handle part of the new attacks, but does not recognize them as easily as the old ones, and the number of false negatives is very high for most classes. For Probe, the most surprising fact is that the attack 'ipsweep' seems to go undetected almost all the time. This result is unusual because 'ipsweep' was available in the training set and did not cause any trouble in the previous experiment. One reason for this could be if the examples of 'ipsweep' from the test set were very different from the ones in the training set. However, after examining the training set carefully, typical values for the features of an 'ipsweep' attack were observed, and it appears that the values of 'ipsweep' in the test set are in the same range as those in the training set. Overall, the results are not bad. First, almost all old attacks are perfectly detected, especially 'portsweep' and 'satan' which triggered false negatives in the first experiment are now absent from the attacks triggering false negatives. The new attacks are detected most of the time, but the number of false negatives is still too high to be useful in a real-world application. Finally, solving the problem of 'ipsweep' would substantially bring down the number of false negatives.

For U2R, ENSEMBLE$_{\max}$ brings down the average number of false positives to 1.6 and the average number of false negatives to 1.0, respectively, over five runs of the program. As expected, a 'rootkit' attack sometimes goes undetected, just as in the previous experiment. 'ps' also occasionally appears as a false negative. The most surprising result comes from undetected 'buffer overflow', which did not occur in the feature selection experiment. However, 'xterm' and 'sqlattack' are detected all the time, which is good because it means that ENSEMBLE$_{\max}$ generalizes well for the U2R class.

Table X
FEATURE SELECTION ASSESSMENT WITH SVM CLASSIFIER

SVM Classifier	Probe	U2R	R2L	DoS
41 features	99.61	88.00	97.60	99.83
5 SVM features	97.46	67.00	75.11	89.55
5 LGP features	96.91	82.00	65.02	98.83
5 MARS features	86.92	95.00	92.53	97.74
11–14 combined features	99.28	90.00	98.13	99.90

(a) Accuracy

SVM Classifier	Probe	U2R	R2L	DoS
41 features	19.9	15.9	6.7	0.2
ENSEMBLE$_{\max}$	51.9	0.0	2.6	2.2

(b) False positives

SVM Classifier	Probe	U2R	R2L	DoS
41 features	66.4	0.5	50.4	43.6
ENSEMBLE$_{\max}$	18.5	0.1	6.8	9.4

(c) False negatives

Table XI
MODEL ASSESSMENT WITH SVM CLASSIFIER

SVM Classifier	Probe	U2R	R2L	DoS
41 features	50.00	50.00	50.00	50.00
5 SVM features	55.45	50.00	50.15	76.42
5 LGP features	64.45	50.00	51.65	77.23
5 MARS features	84.82	72.86	50.00	76.78
11–14 combined features	51.90	50.00	49.97	72.27

(a) Accuracy

SVM Classifier	Probe	U2R	R2L	DoS
41 features	0.0	70.0	0.0	0.0
ENSEMBLE$_{\max}$	0.0	1.0	0.0	173.4

(b) False positives

SVM Classifier	Probe	U2R	R2L	DoS
41 features	4166.0	0.0	16,347.0	17,761.0
ENSEMBLE$_{\max}$	871.8	0.8	4380.8	715.0

(c) False negatives

The number of false negatives for the R2L class explodes. Old and new types of attacks are similarly misclassified. The only conclusion that can be drawn is that the R2L training set contains too few examples of each type of attack to be of any help.

For DoS, the majority of the false negatives are due to new attacks. Of the old attack types, 'pod' is the only one that regularly triggers a few false negatives, for each run of the program. Other old attacks, such as 'smurf' and 'neptune', sometimes trigger false negatives, but this happens extremely rarely. New attacks are more problematic, with 'mailbomb', 'apache2', 'processtable', and 'udpstorm' recurrently triggering false negatives, even if most of these attacks are detected in general. Even though its generalization power is limited, ENSEMBLE$_{\max}$ performed quite well overall on unseen DoS attacks and helped bring down both false positives and false negatives. This is quite an improvement, but again not enough for a real-world application.

D. Ensembles with Support Vector Machine Classifiers

As displayed in Figure 1, the main algorithms used as classifiers in the ensemble were decision trees (DT). Attempts were also made to use support vector machines (SVM) with a Gaussian radial basis function kernel (which is one of the most powerful machine learning algorithms currently available). However, the results were not very encouraging, as displayed in Table X.

Utilizing support vector machines with Gaussian radial basis functions as the main ensemble classification algorithm for the model assessment experiment yielded even worse results, as shown in Table XI, where the accuracy for many classes and classifiers is around the $50\,\%$ guessing baseline.

One reason why the SVMs do not give accurate results might be that many data points corresponding to different attacks are located among the data points corresponding to normal traffic in the dimensional space. Hence, the SVMs are not able to find hyperplanes that clearly separate the examples in the dataset as attacks or normal traffic. On the other hand, decision trees are able to identify key features that appear at the top of the tree and are in this way able to accurately separate the examples.

Intuitively, it is reasonable to assume that, in this case, using SVMs as multiclass classifiers (one class for each type of attack) instead of as binary classifiers (attack or normal) would give more accurate results, since there should be even more similarities between instances belonging to the same attack type within an attack class ('smurf', 'neptune', etc.) than instances belonging to the same class (DoS, R2L, etc.).

Experiments were also carried out on SVMs using different kernel functions (linear, quadratic and polynomial of order 3). The maximum number of iterations then had to be increased to $100,000$ in order to have the algorithm converge all the time. These experiments were run on the test set only, but several times for each kernel.

Surprisingly, quite good results were obtained on the test set with the linear kernel (except for the R2L class, but that was to be expected). The results were non-conclusive, but sometimes even better than with decision trees. The quadratic kernel also gave some interesting results, but quite limited. Hence, an accuracy of $94\,\%$ for the LGP feature set on Probe attacks could be reached several times. However, the order 3 polynomial kernel produced terrible results across the board, and the algorithm did not converge for the polynomial kernel even after $100,000$ iterations on the feature selection experiment with cross-validation. It transpires that when adding more complex forms for the boundaries, the results are getting worse and worse. The SVMs probably overfit the data in the training phase. This would explain why the results are so bad when increasing the boundary complexity.

Furthermore, SVM was much slower than DT, roughly two orders of magnitude both for training and for classification. In particular classification time is an important criterion to take into account when building a real-world application: if the classifier is very accurate but need too much time to analyse each packet on a network, chances are that many packets will go through without being analysed, leading to poor performance of the intrusion detection system.

VI. Discussion and Future Work

The Feature Selection Assessment experiments (Section V-B) showed that the ensemble approach is indeed a very powerful paradigm that can be used to bring down the number of false positives and false negatives. The lower accuracy observed by individual algorithms is countered by the union of their results. Even with sets containing only five features, the results are very encouraging. Moreover, treating each class of attack as a different problem solved by a specialised algorithm seems to work well when compared to strategies using one algorithm to detect all classes of attacks. "Divide and conquer" and "Unity is strength" seem to be opposite views, but they are actually both applied in this work with impressive results. In general, algorithms using fewer features have slightly lower accuracy and prediction time, but much lower training time.

The results obtained by Mukkamala *et al.* [23] seem to be correct. However, the features selected by LGP give the worst result in most cases except for DoS where it is the feature set selected by SVM that performs poorly. Consequently, the sets of features selected by LGP should be reconsidered for all classes except DoS, while the set of features selected for DoS by SVM should be replaced. The number of different types of attacks that go undetected is very small and only few examples of these attacks are problematic. Most of the time, the problem lays in the lack of information contained in the dataset. Some attacks require very specific features and should probably be handled by specialized programs or signature-based IDSs. The class Probe is a bigger problem since most of the attacks belonging to this class exploit a legitimate feature used by network administrators; as a result, all types of Probe attacks trigger FN at some point, even though 'portsweep' and 'satan' are the most problematic.

A smaller feature set means that less information must be extracted from a network packet in the data preprocessing phase. Since the accuracy is not lowered too much in the best cases, this is a huge improvement that could be used in real IDSs. Moreover, the union of all algorithms using fewer features tremendously improves the accuracy: on average over ten runs of the program, only 0.7 FP and 3.0 FN were observed for Probe over $20,000$ examples, 6.6 FP and 0.5 FN for R2L over $2,252$ examples, 0.3 FP and 0.3 FN for U2R over 104 examples, and 0.0 FP and 1.6 FN for DoS over $20,000$ examples. Even though these results are much better than what could be achieved with a single algorithm, they are still quite far from being useful in a real-world application where the false positives and negatives should be < 1 for some 15 million instances in a 10Gb/s Ethernet network. Arguably, over $90\,\%$ of those instances will be normal traffic containing no attack at all, but $\text{ENSEMBLE}_{\max}$ still has to be improved to stand a chance against clever hackers.

The results described above are the best that an ensemble composed of these algorithms and sets of features could achieve. In its current state, there is no point in building an experiment to assess a real combination of the results of the individual algorithms in the $\text{ENSEMBLE}_{\max}$. Further work will have to be carried out to find the best suitable algorithms and sets of features. Nevertheless, it is interesting to see how well this $\text{ENSEMBLE}_{\max}$ can perform when predicting previously unseen attack types. That was the topic of the second round of experiments, on Model Assessment (Section V-C). Even if $\text{ENSEMBLE}_{\max}$ in general helps tremendously in bringing down the numbers of false positives and false negatives, it is still far from reaching the accuracy appropriate for a real-world application. In particular, datasets that are not carefully designed are proven to be useless in building accurate models of the attacks. This is the case with the R2L training set, which mainly contains examples of the 'warezclient' attack (which is not even represented in the test set) and very few examples of all other types of attacks. The performance of $\text{ENSEMBLE}_{\max}$ was acceptable for the classes of attacks U2R and DoS. The performance on the Probe class was also standard, even though 'ipsweep' attacks went undetected for unknown reasons. Overall, the results of this second round of experiments were not very satisfying, but once again proved the usefulness of the ensemble approach.

In the future, particular attention has to be paid to the features relevant to each attack. New features carrying meaningful information about the attacks must be designed to help the machine learning algorithms to successfully classify all types of attack. The DoS and Probe classes are mostly characterized by time-related features, whereas R2L and U2R mostly are characterized by content-related features extracted from the payload of the network packets.

VII. Conclusions

The aim of this work was to show that ensemble approaches fed with appropriate features sets can help tremendously in reducing both the number of false positives and false negatives. In particular, our work showed that the sets of relevant features are different for each class of attacks, which is why it is important to treat those classes separately. We developed our own IDS to evaluate the relevance of the sets of features selected by Mukkamala *et al.* [4]. This system is an ensemble of four ensembles of decision trees. Each of the four ensembles is in charge of detecting one class of attacks and composed of four decision trees trained on different sets of features. The first three decision trees were fed with sets of five features selected in [4]. The last decision tree was fed with the union of these three sets of five features from which the redundant features were removed.

The experiments showed that these sets were appropriate in most cases. In the first experiment, the set of features selected by linear genetic programming gave the worst results, except for the class `DoS` for which the set of features selected by SVM performed poorly. The second experiment gave less interesting results because of the inappropriate distribution of examples between the training and test sets of the KDD99 data. In particular, the ensemble could not generalize well enough to be useful for the `R2L` class because the training set mainly contains a type of attack that is not represented in the test set. In both experiments, we looked at the number of instances that were misclassified by all four algorithms in order to obtain a result from the best combination of these algorithms. Further work would be required to develop an appropriate decision function combining the results of the different algorithms. However, since the accuracy obtained here was not good enough for a real-world application, designing decision functions was unnecessary. Nevertheless, we are convinced that this work is heading in the right direction in order to overcome the limitations of current intrusion detection systems.

Finally, a thorough analysis of the examples that were misclassified by the ensemble was performed, in particular highlighting the types of attacks that were systematically misclassified by the ensemble. By looking at the signatures of these attacks, we were able to find the reasons for the classification errors. In most cases, the attacks displayed very specific features not captured by the set of variables in the dataset. These attacks should probably be handled by a specialized system or new variables should be developed to train the classifiers.

Acknowledgements

The authors would like to express their thanks to Lillian Røstad (Norwegian University of Science and Technology), Esteban Zimanyi, Olivier Markowitch, Liran Lerman (all at Université Libre de Bruxelles), and the anonymous reviewers for valuable comments on previous versions of the text.

References

[1] A. Balon-Perin, B. Gambäck, and L. Røstad, "Intrusion detection using ensembles," in *Proceedings of the 7th International Conference on Software Engineering Advances*, Lisbon, Portugal, Nov. 2012, pp. 656–663.

[2] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *Proceedings of the 2010 IEEE Symposium on Security and Privacy*. Washington, DC: IEEE Computer Society, Jun. 2010, pp. 305–316.

[3] C. Elkan, "KDD Cup 1999: Computer network intrusion detection," Webpage (last accessed: June 12, 2013), 1999, http://www.sigkdd.org/kdd-cup-1999-computer-network-intrusion-detection.

[4] S. Mukkamala, A. Sung, and A. Abraham, "Cyber security challenges: Designing efficient intrusion detection systems and antivirus tools," in *Enhancing Computer Security with Smart Technology*, V. R. Vemuri and V. S. H. Rao, Eds. Boca Raton, Florida: CRC Press, Nov. 2005, pp. 125–161.

[5] S. Mukkamala and A. H. Sung, "Identifying significant features for network forensic analysis using artificial intelligent techniques," *International Journal of Digital Evidence*, vol. 1, no. 4, pp. 1–17, 2003.

[6] A. H. Sung and S. Mukkamala, "The feature selection and intrusion detection problems," in *Proceedings of the 9th Asian Conference on Advances in Computer Science*. Chiang Mai, Thailand: Springer-Verlag, May 2004, pp. 468–482.

[7] P. Laskov, P. Düssel, C. Schäfer, and K. Rieck, "Learning intrusion detection: supervised or unsupervised?" in *Proceedings of the 13th International Conference on Image Analysis and Processing*. Cagliari, Italy: Springer, Jun. 2005, pp. 50–57.

[8] L. K. Hansen and P. Salamon, "Neural network ensembles," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 10, pp. 993–1001, Oct. 1990.

[9] R. E. Schapire, "The strength of weak learnability," *Machine Learning*, vol. 5, no. 2, pp. 197–227, Jul. 1990.

[10] J. J. Davis and A. J. Clark, "Data preprocessing for anomaly based network intrusion detection: A review," *Computers & Security*, vol. 30, no. 6-7, pp. 353–375, 2011.

[11] R. Lippmann, "1998 DARPA intrusion detection evaluation data set," Webpage (last accessed: June 12, 2013), 1998, http://www.ll.mit.edu/mission/communications/cyber/CSTcorpora/ideval/data/1998data.html.

[12] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *Computers & Security*, vol. 31, no. 3, pp. 357–374, May 2012.

[13] M. Tavallaee, N. Stakhanova, and A. A. Ghorbani, "Toward credible evaluation of anomaly-based intrusion-detection methods," *IEEE Transactions on Systems, Man, and Cybernetics — Part C: Applications and Reviews*, vol. 40, no. 5, pp. 516–524, Sep. 2010.

[14] D. M. Gregg, W. J. Blackert, D. C. Furnanage, and D. V. Heinbuch, "Denial of service (DOS) attack assessment analysis report," Johns Hopkins University, Baltimore, Maryland, Tech. Rep. AFRL-IF-RS-TR-2001-223, Oct. 2001.

[15] F. Lau and S. H. Rubin, "Distributed denial of service attacks," in *Proceedings of the 2000 IEEE International Conference on Systems, Man, and Cybernetics*. Nashville, Tennessee: IEEE, Oct. 2000, pp. 2275–2280.

[16] C. Patrikakis, M. Masikos, and O. Zouraraki, "Distributed denial of service attacks," *The Internet Protocol Journal*, vol. 7, no. 4, pp. 13–35, Dec. 2004.

[17] C. Cowan, P. Wagle, C. Pu, S. Beattie, and J. Walpole, "Buffer overflows: attacks and defenses for the vulnerability of the decade," in *Foundations of Intrusion Tolerant Systems (Organically Assured and Survivable Information Systems 2003)*, J. H. Lala, Ed. Los Alamitos, California: IEEE Computer Society, Jan. 2003, pp. 227–237.

[18] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in *Proceedings of the 2nd International Conference on Computational Intelligence for Security and Defense Applications*. Ottawa, Ontario, Canada: IEEE, Jun. 2009, pp. 53–58.

[19] J. P. Anderson, "Computer security threat monitoring and surveillance," James P. Anderson Co., Fort Washington, Pennsylvania, Tech. Rep., Apr. 1980.

[20] D. E. Denning, "An intrusion-detection model," *IEEE Transactions on Software Engineering*, vol. 13, no. 2, pp. 222–232, Feb. 1987.

[21] L. T. Heberlein, G. V. Dias, K. N. Levitt, B. Mukherjee, J. Wood, and D. Wolber, "A network security monitor," in *Proceedings of the IEEE Symposium on Security and Privacy*. Los Alamitos, California: IEEE Computer Society, May 1990, pp. 296–304.

[22] S. X. Wu and W. Banzhaf, "The use of computational intelligence in intrusion detection systems: A review," *Applied Soft Computing*, vol. 10, no. 1, pp. 1–35, Jan. 2010.

[23] S. Mukkamala, A. H. Sung, and A. Abraham, "Intrusion detection using an ensemble of intelligent paradigms," *Journal of Network and Computer Applications*, vol. 28, no. 2, pp. 167–182, Apr. 2005.

[24] S. Chebrolu, A. Abraham, and J. P. Thomas, "Hybrid feature selection for modeling intrusion detection systems," in *Proceedings of the 11th International Conference on Neural Information Processing*, ser. Lecture Notes in Computer Science, N. R. Pal, N. Kasabov, R. K. Mudi, S. Pal, and S. K. Parui, Eds., vol. 3316. Calcutta, India: Springer, Nov. 2004, pp. 1020–1025.

[25] ——, "Feature deduction and ensemble design of intrusion detection systems," *Computers & Security*, vol. 24, no. 4, pp. 295–307, Jun. 2005.

[26] A. Abraham and J. P. Thomas, "Distributed intrusion detection systems: A computational intelligence approach," in *Information Security and Ethics: Concepts, Methodologies, Tools, and Applications*, H. Nemati, Ed. Hershey, Pennsylvania: IGI Global, Sep. 2005, vol. 5, pp. 105–135.

[27] S. Peddabachigari, A. Abraham, C. Grosan, and J. P. Thomas, "Modeling intrusion detection system using hybrid intelligent systems," *Journal of Network and Computer Applications*, vol. 30, no. 1, pp. 114–132, 2007.

[28] A. Abraham and R. Jain, "Soft computing models for network intrusion detection systems," in *Classification and Clustering for Knowledge Discovery*, ser. Studies in Computational Intelligence, S. K. Halgamuge and L. Wang, Eds. Berlin Heidelberg, Germany: Springer, Oct. 2005, vol. 4, pp. 191–207.

[29] A. Abraham, R. Jain, J. P. Thomas, and S.-Y. Han, "D-SCIDS: Distributed soft computing intrusion detection system," *Journal of Network and Computer Applications*, vol. 30, no. 1, pp. 81–98, Jan. 2007.

[30] A. Zainal, M. A. Maarof, S. M. Shamsuddin, and A. Abraham, "Ensemble of one-class classifiers for network intrusion detection system," in *Proceedings of the Fourth International Conference on Information Assurance and Security*. Naples, Italy: IEEE Computer Society, Sep. 2008, pp. 180–185.

[31] G. Folino, C. Pizzuti, and G. Spezzano, "GP ensemble for distributed intrusion detection systems," in *Proceedings of the 3rd International Conference on Advances in Pattern Recognition*, Bath, England, Aug. 2005, pp. 54–62.

[32] ——, "An ensemble-based evolutionary framework for coping with distributed intrusion detection," *Genetic Programming and Evolvable Machines*, vol. 11, no. 2, pp. 131–146, Jun. 2010.

[33] E. Bahri, N. Harbi, and H. N. Huu, "Approach based ensemble methods for better and faster intrusion detection," in *Proceedings of the 4th International Conference on Computational Intelligence in Security for Information Systems*, ser. Lecture Notes in Computer Science. Torremolinos-Málaga, Spain: Springer, Jun. 2011, pp. 17–24.

[34] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and application to boosting," *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139, Aug. 1997.

[35] P. Zhang, X. Zhu, Y. Shi, L. Guo, and X. Wu, "Robust ensemble learning for mining noisy data streams," *Decision Support Systems*, vol. 50, no. 2, pp. 469–479, Jan. 2011.

[36] H. Gunes Kayacik, A. Nur Zincir-Heywood, and M. I. Heywood, "A hierarchical SOM-based intrusion detection system," *Engineering of Applied Artificial Intelligence*, vol. 20, no. 4, pp. 439–451, Jun. 2007.

[37] K. Kendall, "A database of computer attacks for the evaluation of intrusion detection systems," Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts, Jun. 1999.

[38] ——, "Intrusion detection attacks database," Webpage (last accessed: June 12, 2013), 2007, http://www.ll.mit.edu/mission/communications/cyber/CSTcorpora/ideval/docs/attackDB.html.

Towards Enhanced Usability of IT Security Mechanisms

How to Design Usable IT Security Mechanisms Using the Example of Email Encryption

Hans-Joachim Hof
Munich IT Security Research Group (MuSe)
Department of Computer Science and Mathematics
Munich University of Applied Sciences,
Lothstraße 64, 80335 Munich, Germany
email: hof@hm.edu

***Abstract*—Nowadays, advanced security mechanisms exist to protect data, systems, and networks. Most of these mechanisms are effective, and security experts can handle them to achieve a sufficient level of security for any given system. However, most of these systems have not been designed with focus on good usability for the average end user. Today, the average end user often struggles with understanding and using security mechanisms. Other security mechanisms are simply annoying for end users. As the overall security of any system is only as strong as the weakest link in this system, bad usability of IT security mechanisms may result in operating errors, resulting in insecure systems. Buying decisions of end users may be affected by the usability of security mechanisms. Hence, software providers may decide to better have no security mechanism then one with a bad usability. Usability of IT security mechanisms is one of the most underestimated properties of applications and systems. Even IT security itself is often only an afterthought. Hence, usability of security mechanisms is often the afterthought of an afterthought. This paper presents some guidelines that should help software developers to improve end user usability of security-related mechanisms, and analyzes common applications based on these guidelines. Based on these guidelines, the usability of email encryption is analyzed and an email encryption solution with increased usability is presented. The approach is based on an automated key and trust management. The compliance of the proposed email encryption solution with the presented guidelines for usable security mechanisms is evaluated.**

Keywords-usability; IT security; usable security; email encryption.

I. INTRODUCTION

This paper is an extension of the usability design guide presented in [1].

Any improvement of the overall security level of any system requires to improve the security level of all subsystems and available mechanisms as the overall security level of a system is determined by the weakest link in this system [2,3]. Howe et al. found that current software and approaches for security are not adequate for end users, because these mechanisms are missing ease of use [4]. In [2,3] end user are identified as weakest link in a company. Hence, improving the usability of security mechanisms helps to improve the overall security level of a system.

Examples of bad usability of security mechanisms are all around, some are discussed in Section IV. Bad usability of security mechanisms may slow down the adoption of a security system. This happened for example with email encryption. Today, it is very unlikely that an average user uses email encryption. Major problems for average users are key exchange and trust management, both having a very bad usability in common email encryption solutions. Figure 1 shows a completely useless error message during the generation of a key pair for email encryption in GPGMail [5], as one example of bad usability.

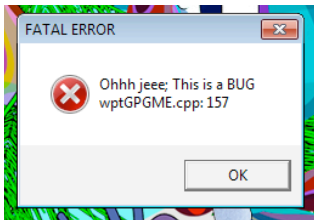


Figure 1. Error message during generation of a key pair for email encryption in GPGMail [5]

The use of email encryption in companies shows that an improved usability may lead to the adoption of the formerly despised technology. In companies, key exchange and trust management are usually not done by the users themselves, but they can rely on central infrastructures such as a central company directory with keys that are trusted by default (all employees). Such a directory ensures average users can use email encryption.

The example of email encryption shows that designing security mechanisms with good usability is worth an effort. For the ordinary software developer, i.e., non security expert, it makes sense not to implement core security mechanisms like encryption algorithms or signature algorithms. Those mechanisms are usually available in security libraries written by security experts and could be easily used by software developers. However, software developers often decide on how security mechanisms are integrated into an application. For example, when implementing an email encryption security solution like GPGMail [5], the software developer decides on the interfaces for setting up trust and importing keys. Both mechanisms are application specific, hence must be implemented by the application developers. Usually, these functionalities are exposed to the users, hence should have a

good usability. Guidelines for usability that focus on IT security mechanisms and their integration into applications may help software developers to improve the usability of IT security mechanisms in these applications. This paper presents some guidelines that should help software developers to improve end user usability of security-related mechanisms. To underline the importance of the presented guidelines, weaknesses of security mechanisms in common applications regarding usability for end users are shown in an analysis of common applications and security mechanisms on basis of the presented guidelines. Other important aspects of software security, e.g., secure coding guidelines, testing of security, and threat analysis are out of scope of this paper.

The rest of this paper is structured as follows: Section II gives an overview on related work, especially on existing guidelines for usability. Section III presents guidelines for usable IT security mechanisms. Section IV analyzes the usability of some common security mechanisms and applications on the basis of the guidelines of Section III. Section V uses email encryption as an example on how to apply the guidelines on a problem from the field. An email encryption solution with good usability is presented. Section VI evaluates the usability of the email encryption solution on the basis of the guidelines presented in Section III. Section VII concludes the paper and gives an outlook on future work.

II. Related Work

Several standards focusing on usability in general exist, e.g., EN ISO 9241 [6]. In EN ISO 9241-11, which is part of EN ISO 9241, requirements for the usability of system are described. These requirements include effectiveness, efficiency and satisfaction. EN ISO 9241-10, another part of EN ISO 9241, lists requirements for usable user dialogs. However, the rules and guidelines of EN SIO 9241 are very general and not targeted on security mechanisms. The design guidelines presented in this paper interpret the general requirements and rules of EN ISO 9241 and its parts for the special case of security mechanisms. As the guidelines presented in this paper are focused only on the topic IT security, the presented guidelines are more detailed and may be easier to follow for software developers.

Other publications like [7-11] focus on the usability of security mechanisms in special applications (e.g., email encryption), or focus on the usability of special security mechanisms (e.g., use of passwords). The guidelines presented in this paper are more general such that they are useful for the design of a wide variety of applications and security mechanisms.

Existing guidelines for usability of security mechanisms like those in [12, 13] focus very much on user interface design. The design guide presented in this paper take a slightly different approach by focusing more on the security mechanism itself. It is considered possible to change the design of a security mechanism for the sake of good usability.

Markotten shows how to integrate user-centred security engineering into different phases of the software development process [14]. However, the emphasize of Markotten's work is more on integration of usability engineering into the software development process than on a design guide.

Several works on zero-configuration IT security exist, e.g., [15-22]. While zero-configuration can significantly improve the usability of an application, a systematic approach to usability for IT security is still missing. Zero-configuration may be one building block of usable security.

To summarize, previous works either are not focused on usability of IT security at all, are focused on one special aspect of usable IT security, or are focused on user interface design. This paper presents some guidelines for software developers to help them improve the usability of security-related functionality.

III. Guidelines for Good Usability of Security Mechanisms

The guidelines presented in this section are the result of several years in teaching IT security to beginners (and seeing their difficulties) as well as industrial experience in the design of products requiring IT security mechanisms that are operated by end users. The guidelines reflect our viewpoint on usability of security mechanisms. It is not assumed that those guidelines are complete. It is important to notice that the usability of any system depends on the specific user and his experiences, knowledge and context of use, which includes the task at hand, the equipment at hand, and the physical and social environment of the user. Hence, it is hard to objectively evaluate the usability of a system. However, we hope that the following set of nine design guidelines coming from the field may be of help for software developers:

G1 Understandability, open for all users: This paper focuses on usability for end users. The average end users should be able to use the security mechanism. Otherwise, the security mechanism is not useful for the intended audience. The average user neither has a special interest in IT security nor understands IT security. It is the responsibility of the software developer to hide as many security mechanisms as possible from the user. For those security mechanisms that are exposed to the end user it is necessary to get security awareness. The process of educating people is easier if suitable metaphors are used. A good metaphor is taken from everyday life of the average user and is easy to grasp. A good metaphor is simple but powerful in its meaning. Example: an email encryption application should not use the term "encrypted email." It is better to talk about a "secret message for xy" or "email readable only by xy" where xy is the receiver of the message.

Usable security should be available for all users. It should especially not discriminate against any group of people. For example, usable security mechanisms should not exclude disabled people that use special tools to access applications (e.g., Braille reader for vision impaired people). Example of compliance with G1: if captchas are used in an application, multiple versions of the captcha should exist. Each version of the captcha should address another sense.

G2 Empowered users: Ideally, a usable security mechanism should not be used to restrict the user in what he is doing or what he wants to do. This allows end users to efficiently fulfill their tasks. Efficiency is one of the general usability requirements of EN ISO 9241 [6]. The absence of user restrictions often results in a better acceptance of securi-

ty by users. The focus of a security mechanism should be on protecting the user. Any security-motivated restriction of the user should be carefully evaluated regarding necessity for system security and adequateness. The user should at least have the impression that he is in control of the system and not the system is controlling him. Security mechanisms should interfere with the usual flow of user activities in the least possible way. Security mechanisms should allow the user to execute activities in any way he wants. Other drivers than protecting the user and the system should not be motivation for restrictions. Especially, users should not be restricted by a security mechanism for the only reason of copyright protection or other business reasons. While such security mechanisms are of great use for businesses, they constantly restrict the user, hence force him to bypass security mechanisms. As users are very imaginative in bypassing unwanted restrictions, it is very likely that a non-security-motivated restriction decreases the security level of a system. The Apple iPhone is a good example: as the phone enforces many restrictions, many user bypass the security mechanisms by using a jailbreak software to revoke those restrictions.

Another important rule is that the user should decide on trust relations. A security mechanism should not enforce trust relations given by a software vendor. The user should always have the possibility to revoke preinstalled trust relations. Trust relations should only be established in advance for the purpose of IT security. For example, having a preinstalled certificate to verify software patches is OK. Establishing trust relations out of business purposes should be avoided. Example of compliance with G2: applications should have an interface that lists preinstalled certificates. The user should have the possibility to revoke certificates and install custom certificates.

G3 No jumping through hoops: Users should only be forced to execute as little tasks as possible that exist only for IT security reasons. Otherwise, users get annoyed and refuse collaboration with IT security mechanisms. The ideal security mechanism does not interfere with user tasks at any time (also see G2) if it is not absolutely necessary to maintain the user's security.

An example on how to not design security mechanisms are captchas: the user is forced to read a nearly unreadable and meaningless combination of letters and numbers and enter it before he can execute the wanted task. Example of compliance with G3: an application that uses a challenge-response mechanism similar to hashcash [23] instead of a captcha to avoid abuse of a service by automated scripts.

G4 Efficient use of user attention and memorization capability: Users have problems memorizing data that does not belong to their social background. Hence, they tend to use all kind of optimization to reduce the amount of data they have to remember. This is why users only use few passwords for all logins where they need passwords. In [24], it is stated that an average user uses only 6.5 passwords for all his web accounts. A later survey [25] found that more than 80% of all participants of the survey reuse a set of password in different places. 73% of the participants use one password with slight modification on different accounts.

But not only does an average user use the same password more than once, he also selects easy to remember passwords as he is not good in memorizing passwords with a mix of upper and lower case letters, numbers and special characters. Hence, security mechanisms should require the user only to remember little data or no data at all. Example of compliance with G4: an application uses an existing account from another site for login, e.g., by using OpenID [26]. The user can use an existing account, hence does not have to remember another password.

Security mechanisms should only require as little interaction with the user as possible. The security mechanism should only requests the attention of the user if it is absolutely necessary. Interaction with the user should be done in the most minimalistic way. See also G1 for user interaction. Example of compliance with G4: an email encryption application that does not ask a user for each mail if he wants to encrypt the mail or not. Instead, the email application offers a configuration option to always encrypt mails. Additionally, the email composition window clearly states the current protection status and offers a possibility to override the preferences.

G5 Only informed decisions: A user only feels secure and cooperates with a system if the system does not ask too much of him. Hence, users should only have to make decisions they can decide on. If there is an important security decision to take, it must be ensured that the user has the capability to make this decision. This means that the user has enough information about the situation that requires him to make a decision, and it must be ensured that the average user is capable to make an informed decision on this issue. If it is not clear if the user can decide on an issue, the decision should be avoided. G5 is hard to achieve and requires a careful examination during the design of an application. Example of compliance with G5: an application automatically deals with unknown certificates and does not prompt a user for a decision (see Section IV.D).

G6 Security as default: Good usability requires efficiency. Hence, the user should not have to configure security when he first starts an application. Software for end users should always come preconfigured such that the software is reasonable secure and usable. All security mechanisms of a system should be delivered to the end user with a configuration that offers adequate security for the end users. If a preconfiguration is not possible, the configuration effort must be minimized for users. This requires an analysis of the security requirements of average users during software development prior to the deployment of the software to find the adequate security level for most users. Example of compliance with G6: a home wifi access point comes preconfigured with a random WiFi password.

G7 Fearless System: The security system should support a positive attitude of the user towards the security system. A user with a positive attitude towards security mechanisms is cooperative and more likely to not feel interrupted by security mechanisms. Hence, security mechanisms should protect the overall system in a way that the user neither has fear when the system is in a secure state nor feels secure when the system is not in a secure state. The security state of the sys-

tem should be visible at all times. A security mechanism should be consistent in its communication with its user. A security mechanism should not use fear to force users to obey security policies or get a wanted reaction. G7 is hard to achieve and requires a careful examination during the design of an application.

G8 Security guidance, educating reaction on user errors: Users tend to make mistakes, especially in respect to IT security. It is important that the security system hinders the user to make mistakes. However, as blocked operations can be very frustrating for users, the response of the security system must provide information why a given operation was blocked and should also offer a solution on how the user could proceed. The solution must be adapted on the situation and should keep the overall security of the system in mind. A security system should guide the user in the usage of security mechanisms. Errors should be prevented and there should be ways to "heal" errors. Example of compliance with G8: when an email encryption application fails to encrypt an email because of a missing public key of the recipient, the error message should explain how to import certificates from and how to verify certificates by comparing fingerprints of keys. To "heal" the error, the email encryption application offers to send the mail as password-protected PDF and instruct the user to call the recipient and tell him the password for the PDF.

G9 Consistency: Consistency allows users to efficiently fulfill their tasks. Security mechanisms should fit into both the application and the system context where they are used. Security mechanisms should have the look and feel the user is used to. G9 is hard to achieve and requires a careful examination during the design of an application.

IV. Analysis of the Usability of Common Security Mechanisms and Applications on Basis of the Presented Guidelines

In this section, common applications and security mechanisms are analyzed on basis of the guidelines given in Section III. The analysis identifies room for improvement in these applications and security mechanisms. It also shows some good examples for certain aspects of security usability.

A. Email Encryption using GPGMail

GPGMail is a popular open source email encryption solution for Mac home users [5]. The encryption process itself is fairly easy, usually requiring one click to enable email encryption. However, key and trust management requires significant effort. For a secure exchange of public keys, the user has to get the public key itself (e.g., from a key server or the homepage of the receiver of a message) and verify the authenticity of the key. Certificates may be in use. The authentication requires the use of another channel to communicate with the key owner (e.g., telephone or in person) and to read a number to the owner that is meaningless for the user. There is no guidance for this process. Then, the user has to change the trust of the exchanged public key. It gets more complicated when using a web of trust for trust management: for the web of trust to work, the user must decide on how trustworthy a person is to verify public keys/certificates in addition to managing direct trust into keys. The distinction between those different types of trust is very hard to understand for average users.

This application is compliant with the following guidelines:

- G2 (user decides on trust relations)
- G4 (minimal interaction)
- G7 (does not frighten user)
- G9 (usually good integration, depends on system, mail client)

This application is not compliant with the following guidelines:

- G1 (hard to understand trust management and process of key verification)
- G3 (complicated trust management)
- G5 (hard to understand trust management and process of key verification)
- G6 (not set to "encrypt all" by default)
- G8 (not much guidance with trust management)

B. Forced Updates

Keeping a system up-to-date requires a timely use of provided security patches. However, many users are quite lax in applying security patches. Hence, nowadays, more and more software providers let not the users decide on when to patch a system but automatically apply security patches as soon as available. While this relieves the user from applying patches, it does not take into consideration the situation of the user at the moment of a forced update. The update process may require downloading a large amount of data. This is a problem when the user is temporary on a low-bandwidth connection. The update process may change security or trust relevant configuration of the application, e.g., by revoking certificates or adding new certificates that are considered trustworthy by the software provider. Often, forced updates cannot be stopped by the user, hence hinder the user.

This security mechanism is compliant with the following guidelines:

- G1 (easy to understand)
- G5 (no user decisions involved)
- G6 (keeps system up-to-date)
- G7 (does not frighten user)
- G8 (no user action necessary (or possible))
- G9 (well integrated)

This security mechanism is not compliant with the following guidelines:

- G2 (user can not decide to not apply a patch, user can not decide on time to apply patch (e.g., do not patch presentation application before presentation on CENTRIC 2012))
- G3 (in some cases user has to wait until patch was applied)

- G4 (full attention of the user when waiting for process to finish)

C. Captchas

A captcha is a security mechanism avoiding that automated scripts use services. In theory, a captcha should be designed in a way that only humans can solve the given problem. Common captcha design requires users to read a distorted and meaningless combination of letters and numbers and enter it before he can use the service. Figure 2 shows a captcha that is even worse from a usability point of view. Another side effect of the use of captchas is that captchas may discriminate against disabled people (e.g., vision impaired people). Some websites offer different types of captchas (e.g., an image captcha and an audio captcha). Vision impaired people can decide to use the audio captcha.

Figure 2. Complicated captcha

This security mechanism is compliant with the following guidelines:

- G5 (no user decision needed)
- G6 (always used)
- G7 (does not frighten user)
- G8 (gives instructions on how to use it)
- G1 (if multiple captchas are used, e.g., image and audio)

This security mechanism is not compliant with the following guidelines:

- G1 (if only a single image captcha is used that discriminates against disabled people)
- G2 (does not allow users to use automation tools)
- G3 (additional task without value for the user)
- G4 (unnecessary user interaction)
- G9 (many different kinds of captchas are in use)

D. HTTPS Certificate Validation in Common Browsers

HTTPS allows for confidential and integrity protected communication on the web. For example, HTTPS is used for online banking or shopping. Nowadays HTTPS is widely used on the web. However, for a secure communication it is necessary to avoid man-in-the-middle attacks. To do so, certificates are used to authenticate the web site that one communicates with. As it is not practicable to install a certificate for each and every web site one visits, most common browsers come with preinstalled certificates of so-called Certificate Authorities (CAs). A browser accepts all certificates that have been signed by such a CA. For example, Mozilla Firefox version 14.0.1 comes with over 70 preinstalled CA certificates. The browser software developer decides on the trustworthiness of a CA (and hence on the trustworthiness of web sites), not the end user.

Figure 3 shows a typical error message of Firefox when encountering a certificate signed by an unknown CA. The text of this error message may be too complicated for average users. Above this, average users are not capable of deciding on the validity of unknown certificate anyway. As this error often occurs, the users get used to it and usually just add a security exception to the system to access the web site, bypassing the security mechanism. Adding a security exception involves multiple steps (see Figure 4 for a screenshot of the second page of the error message when clicking on "Add Exception."

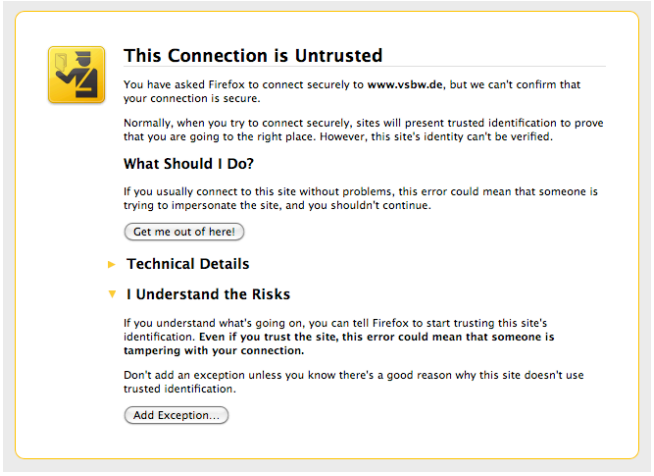


Figure 3. Typical error message of Firefox when encountering an unknown certificate

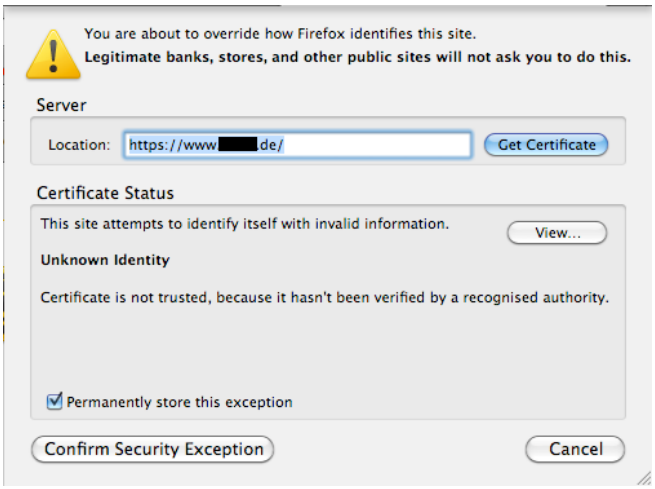


Figure 4. Second dialogue page if user clicked "Add Exception"

This security mechanism is compliant with the following guidelines:

- G6 (large number of preinstalled CAs for secure communication)

- G8 (guidance is given, however, the texts used may be not suited for average users)

This security mechanism is not compliant with the following guidelines:

- G1 (hard to understand error message given when browser encounters an unknown certificate / a certificate from an unknown CA)
- G2 (many preinstalled CA certificates, the user does not initially decide on trust relations. However, expert users can change the trust settings)
- G3 (annoying additional tasks when unknown certificate / a certificate from an unknown CA is encountered)
- G4 (error unknown certificate happens often, hence most users simply ignore the message and add a security exception)
- G5 (no informed decision possible)
- G7 (error message unknown certificate implies an ongoing attack)
- G9 (look and feel is not consistent with the rest of the browser – it changes from a website (Figure 3) to a window (Figure 4))

V. Application of the Presented Design Guide: Design of an Email Encryption Solution with Good Usability

Section IV shows usability problems of security mechanisms in common applications. This section shows for one class of application, email encryption solutions, how good usability of security mechanisms can be achieved using the guidelines presented in Section III. Section IV.A identifies the complicated key and trust management in email encryption solutions like GPGMail [5] as cause of most of the usability problems. Hence, the design of an email encryption solution presented in this paper focuses on automated key and trust management to improve the usability of email encryption.

A. Definitions

Figure 5 shows a simplified email encryption setup: A sender wants to send an email to a receiver. The sender has his own private key ($Priv_S$) that it uses for email signatures and for decryption of emails. Please note that usually two private keys are used, one for decryption and another for signature. For the sake of simplification, this distinction is omitted in this paper. In the rest of this paper, the term "email encryption solution" is used for the presented system despite the fact that the system also decrypts and signs emails.

The sender keeps a list of email addresses and associated public keys. Public keys usually have meta information, e.g., expiration date and the like. This meta information is usually stored together with the public key and the associated email addresses in a certificate. To encrypt emails, the sender uses a public key associated with the email address of the receiver.

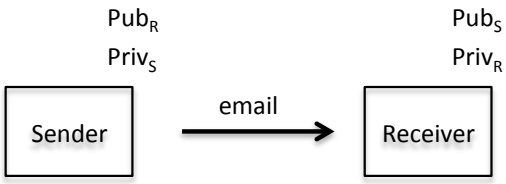


Figure 5. Keys used in email encryption

In Figure 5, the sender uses the key of the receiver, called Pub_R, to encrypt the email. When the receiver gets an email, he uses the public key of the sender (Pub_S) to verify the email signature. The receiver uses his private key ($Priv_R$) to decrypt messages encrypted with Pub_R. The simplified secure email setup described here is implemented by many email encryption solutions, e.g., GPG Mail [5].

B. Approach

The email encryption solution described in this paper offers an automated key and trust management that does not require the user to take any action. Hence, it hides the most complicated part of email encryption from the user. For expert users, a manual key and trust management is still possible. The automated key and trust management is described in the following subsections in detail.

The proposed email encryption solution offers security by default: all emails are encrypted and signed by default. The necessary keys are established by the automated key and trust management if necessary and without any interaction with the user.

The user can override the default security settings: he is offered the possibility to send emails as "public postcard" by a button when composing a mail. The term "public postcard" is used as a metaphor for an unsecured email. This metaphor comes from the experience of a user, hence is a better fit than the term "unencrypted and unsigned email." The proposed email solution is realized as a plug-in to an email client, e.g., as an extension of the well-known GPGMail plugin. Existing security functionality for email is used, e.g., public key encryption and decryption as well as symmetric key encryption and decryption. The solution presented in this paper does not suppose that the receiver of an email uses the same email encryption solution. However, if the receiver of an email uses another encryption solution or no encryption solution at all, the email handling of the receiver may be a little bit more complicated then usual.

C. Triggers for Invocation of Automated Key And Trust Management

Actions of the key and trust management are performed in the following situations:

- A user wants to send an encrypted and signed email (default) and does not have a valid public key of the receiver (Pub_R missing). An automated key exchange must take place.

- A key of a receiver will expire in the near future, hence an automated rekeying is necessary.

Automated rekeying and automated key exchange are described in the following.

D. Automated Rekeying

Automated Rekeying is invoked when the key of a receiver is about to expire. In this case, there has already been a key exchange in the past and a valid key for the receiver is still available. On the receiver side, there are two possibilities:

- The receiver has a valid key of the sender
- The receiver does not have a valid key of the sender, e.g., because the key of the sender already expired and there was no rekeying or the rekeying was not successful. This may for example be the case if the receiver does not use the same email encryption solution.

For management of keys, a list of all public keys of all past email receivers is kept. The email encryption solution regularly checks for all keys if the expiration time is near. Already expired keys are removed from the list. If the expiration time is within the time period `now+maxCheck`, a rekeying request email is sent to the owner of the associated public key. The constant `maxCheck` is a system parameter, e.g., 14 days. The rekeying request is sent by an ordinary email. It includes a certificate with the current public key of the sender and an explaining text that states something like: "Your public key with the fingerprint [fingerprint] is about to expire. Please send a new key. Please send the mail by replying to this mail and attaching a certificate with the new key." The text helps receivers that do not use the same email encryption solution to still communicate with the sender. If the receiver uses the same email encryption solution, the receiver will not see this email but a reply message will be sent. The receiver part of automated rekeying is described below. The sender waits for a reply message with a certificate holding the new public key of the receiver. The message must be signed with the old key of the receiver. If the sender does not get a reply to the rekeying request email before the expiration of the key, the key will be removed from the list. A new key exchange is necessary next time the sender sends an email to the receiver. Otherwise, it stores the received certificate and the included public key that has a validity starting in the future in the list of keys together with the current key.

If the receiver uses the email encryption solution described in this paper, each rekeying request email is deleted from the account of the user so the user never sees those requests in his emails. The following checks are performed:

- Is the email signature valid?
- Is the expiration time of the key within `now+maxCheck` (`maxCheck` is a system parameter see above)?

If the first check fails, the rekeying request is ignored to avoid denial of service attacks on public keys. If the other check fails, the key of the sender is deleted from the list. A new key exchange will be necessary in the future.

If none of the checks failed, the receiver of the key request email checks if he already has a key with a validity starting after the expiration time of the current key. If this is the case, it sends this key to the sender of the rekeying request by an encrypted and signed email that has the certificate with the new public key attached. Otherwise, the receiver of the key request email creates a new public key and associated private key with a validity starting at the expiration date of the current key and an expiration date after the starting date. The receiver creates a certificate for the public key and sends the new public key as described above. Figure 6 summarizes the control flow of the receiver on reception of a rekeying request email.

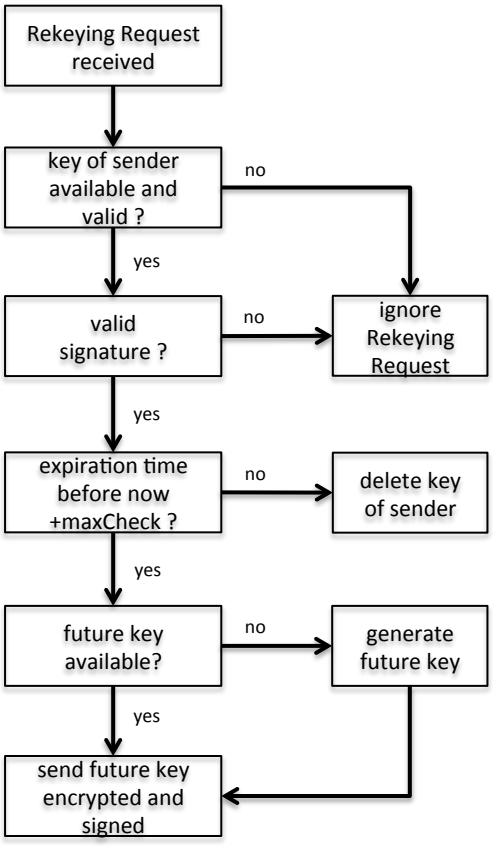


Figure 6. Control flow after receiving a rekeying request email

E. Automated Key Exchange

The automated key exchange takes place when an email is sent to a receiver and no public key is available for this receiver. The missing key may be the result of an unsuccessful rekeying (see above). In most cases, there is no key because the sender never ever has sent an email to the receiver. In both cases, it is not clear if the receiver uses the same

email encryption solution or not. To deal with receivers not using the proposed email encryption solution, all messages used during the automated key exchange are human readable and give detailed information what must be done to interact with the sender of the email. All actions can be done manual. Please note that good usability for the receiver can be achieved if both sides use the proposed solution. Two different automated key exchange implementations are described below.

1) Automated Key Exchange Using a Leap of Faith

In IT security, a "leap of faith" means that at some point in time an entity has blind trust in another entity. For the automated key exchange this means that it is expected that at the time of key exchange, there is no attacker. Attackers may be present in the future. A leap of faith approach may for example protect against adversaries that hack into email accounts e.g., by guessing weak passwords. If a key exchange took place before the hacking of the email account, the exchanged keys cannot be manipulated, as the keys are stored in the email client of the user. The hacker cannot read or manipulate any email because he only has access to the account but not to the keys and all emails are encrypted and signed. However, if the key exchange takes place and the account was already hacked, a man in the middle attack is possible. Please note that automated rekeying is not prone to this attack because all messages are encrypted and signed.

Actions of the automated key exchange at sender side:

1. Generate a random string with at least 20 chars.
2. Send the random string to the receiver in an email that states: "There will be an encrypted email for you in the near future. Please use this password to decrypt the email. This email is not protected. Hence, this is the leap of faith.
3. The sender composes the key exchange message: original email is modified as follows:
    a. The sender creates an ASCII armor for the public key. An ASCII armor is a human readable representation of a public key or certificate. GPGMail offers the possibility to export certificates using an ASCII armor
    b. The sender creates an encrypted PDF that includes the original text of the message, the public key in its ASCII armor, and some explaining text: "The sender of this message wants to exchange a public key with you. Please reply with a public key in an ASCII armor in an encrypted PDF using the same password as this PDF." As password of the encrypted PDF, the sender uses the random string from step 1. The encrypted PDF is intended for receivers that do not use any email encryption solution at all.
    c. The sender creates a message encrypted by a symmetric key encryption using a string to key function to get a symmetric key from the random string generated in step 1. The message includes the original email text, some explaining text similar to the text in step 3.b and the public key of the sender.
    d. Finally, the sender prepares an email with the following text: "This message contains an encrypted email and an encrypted PDF. A password for these files was sent to you before." The encrypted PDF and the symmetrically encrypted message are both attached to the email.
4. The receiver stores the random string from step 1 in the list of keys together with an expiration time that is `leapOfFaithPeriode` in the future (`leapOfFaithPeriode` is a system value, usually a few days).

If the receiver uses the same email encryption solution, it is triggered on reception of a key exchange message. It performs the following actions:

1. Store random string together with the sender address.
2. Remove email with random string from mail server. This ensures that an attacker does not have access to the random string if the account is hacked in the future.
3. Decrypt message using the string to key function to convert the random string to a key. Retrieve public key and store it in the list of public keys.
4. Restore the original mail and encrypt it with the own public key. Delete the received mail and replace it with the mail encrypted with the public key. This avoids that an attacker can get access to the mail if the account is hacked in the future. Also, it allows forgetting the random string.
5. Compose an email including the own public key in a certificate, encrypt it with the random string using a symmetric key encryption and send it to the sender of the original message.

On reception of a reply to its key exchange email, the sender performs the following actions:

1. Retrieve random string from list of keys. If there is no random string, the message is ignored.
2. Decrypt information
    a. If it is a PDF, open it using the random key as a password. Extract the certificate and convert the ASCII armor to a binary representation of the certificate. Store the certificate in the list of keys.
    b. If it is a symmetrically encrypted message, use the string to key function to get a symmetric key from the random string.

Decrypt the message. Store the certificate in the list of keys.

3. Remove random string from the list of keys.

2) Automated Key Exchange Using Side Channels

The idea of the automated key exchange is to use side channels for the exchange of the random key. Compared with the leap of faith approach described in the last section, the use of a side channel improves the possibility that an attacker does not have access to the side channel used.
Side channels can be harvested from the system the email client is running on. Side channels include:

- Alternative email addresses: many people use more than one email address.
- Instant messenger addresses.
- Telephone numbers for text messaging.

The first two side channels can easily be used to send a short random string. To use side channels for key exchange, the automated leap of faith using a leap of faith is modified as follows:

In step 2, the sender sends the email message with the random string not to the same email address as the encrypted email but to a selection of available side channels for a user. If email is used as side-channel, it is very likely that the receiver collects more than one email account in the same email client. Hence, the email encryption solution has access to the side channel. An automated response is possible in this case.

F. User Controlled Trust Management and Security as Default

While automated key and trust management relives users from the burden of manual key and trust management, the user now does not decide on trust relations. This is against G2. The proposed usable email encryption solution lets the user decide on general trust management rules during installation. The user is presented several scenarios, which he can state that he believes in or not. These questions are used to configure the key and trust management. For example, if a user answers yes to the first question, the "leap of faith" approach is not used for key exchange.

G. Error Handling

Error Handling has been omitted in this section for sake of clarity of the presentation. Errors may occur during the automated key exchange or during automated rekeying. By sending a message again after a certain amount of time, the proposed email encryption solution presented in this paper deals with lost emails and the like. However, there are situations where automated key exchange or automated rekeying permanently fails, including situations in which the intended receiver of an email does not want to participate in a key exchange. As the message has already been transferred in the encrypted PDF, no further action must be taken.

VI. Evaluation of the Proposed Email Encryption solution Based on the Design Guide

In this section, it is evaluated if the proposed email encryption solution follows the design guidelines presented in Section III:

G1 (Understandability, open for all users): the proposed solution is compliant with G1 as only good metaphors and scenarios are used for security related configurations.

G2 (Empowered users): the proposed solution is compliant with G2 as the user can decide on the key and trust management configuration. Also, the user can override the security settings by sending an email without protection as "public postcard".

G3 (No jumping through hoops): the proposed solution is compliant with G3 as there are no security specific actions the user must take. It should be noted that this is not true for the receiver of an email if the receiver does not use the proposed email encryption system.

G4 (Efficient use of user attention and memorization capability): no user actions are necessary and the user does not have to memory anything, hence the proposed solution is compliant with G4.

G5 (Only informed decisions): no user actions are necessary. Hence, the proposed solution is compliant with G5.

G6 (Security as default): emails are encrypted and signed by default. Hence, the proposed solution is compliant with G6.

G7 (Fearless System): no user actions are necessary. Hence, the proposed solution is compliant with G8.

G8 (Security guidance, educating reaction on user errors): no user actions are necessary. Hence, no security guidance or education reaction on user errors is necessary.

G9 (Consistency): No user actions are necessary. Hence, there are no consistency issues. It should be noted that this is not true for the receiver of an email if the receiver does not use the proposed email encryption system.

VII. Conclusion and Future Work

This paper presented guidelines for software developers to improve the usability of security-related mechanisms. The analysis of security mechanisms in common applications showed weaknesses in the usability of security-related mechanisms as well as good examples of security usability. To demonstrate the application of the guidelines, the second part of the paper improved the identified usability weaknesses of one common application: email encryption. The approach for email encryption offers automated key and trust management to improve the usability of email encryption. The evaluation showed that the proposed email encryption solution is compliant with the usability design guide presented in this paper.

Future work will include the design of usable security mechanisms for other common problems as well as a user satisfaction study on the effectiveness of the guidelines. The guidelines presented in this paper are focused on usability

for the end user. Future extensions of the design guides will focus on better usability for other groups, e.g., system administrators, testers, and developers.

REFERENCES

[1] H.-J. Hof, "User-centric IT security - how to design usable security mechanisms", Proc. Fifth International Conference on Advances in Human-oriented and Personalized Mechanisms, Technologies, and Services (CENTRIC 2012), International Academy, Research, and Industry Association (IARIA), 2012, pp. 7-12.

[2] I. Arce, "The weakest link revisited", in: Security & Privacy, vol. 1, no. 2, IEEE, 2003, pp. 72-76.

[3] T. Caldwell, "Training – the weakest link", Computer Fraud & Security, vol. 2012, no. 9, Elsevier, 2012, pp. 8-14.

[4] A. Howe, I. Ray, M. Roberts, M. Urbanska, and Z. Byrne, "The psychology of security for the home computer user", Proc. 2012 IEEE Symposium on Security and Privacy, IEEE, 2012, pp. 209-223.

[5] https://www.gpgtools.org/gpgmail/index.html, accessed 12.06.2013.

[6] ISO, "Ergonomie der Mensch-System-Interaktion", EN ISO 9241, 2006.

[7] J. Sunshine, S. Egelmann, H. Almuhimedi, N. Atri, and L. Cranor, "Crying wolf: an empirical study of SSL warning effectiveness", Proc. USENIX Security Symposium, 2009, pp. 399-416.

[8] S. Adams and M. Sasse, "Users are not the enemy", in: Communications of the ACM, vol. 42 no. 12, 1999, pp. 40-46.

[9] A. Whitten and J. Tygar, "Why Johnny can't encrypt: a usability evaluation of PGP 5.0", Proc. 8th conference on USENIX Security Symposium, Volume 8, Berkeley, CA, USA, USENIX Association, 1999, pp. 169-183.

[10] M. Zurko and R. Simon, "User-centered security", Proc. NSPW96 - 1996 Workshop on New Security Paradigms", ACM, 1996, pp. 27-33.

[11] C. Birge, "Enhancing research into usable privacy and security", Proc. 27th ACM International Conference on Design of Communications, ACM, 2009, pp. 221-225.

[12] S. Furnell, "Security usability challenges for end-users", in Social and Human Elements of Information Security: Emerging Trends and Countermeasures, Information Science Reference, 2009, pp. 196-219.

[13] J. R. C. Nurse, S. Creese, M. Goldsmith, and K. Lamberts, "Guidelines for usable cybersecurity: past and present", Proc. Cyberspace Safety and Security (CSS), 2011 Third International Workshop on, IEEE, 2011, pp. 21-26.

[14] D. G. T. Markotten, "User-centered security engineering", Proc. NordU2002 – The 4:rd Eur/Open/USENIX Conference, Helsinki, Finland, USENIX Association, 2002.

[15] M. Conrad and H.-J. Hof, "A generic, self-organizing, and distributed bootstrap service for peer-to-peer networks", Proc. New Trends in Network Architectures and Services: 2nd International Workshop on Self-Organizing Systems (IWSOS 2007), Springer, 2007, pp. 59-72.

[16] M. Bechler, H.-J. Hof, D. Kraft, F. Pählke, and L. Wolf, "A cluster-based security architecture for ad hoc networks", Proc. INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies, Hong Kong, China, März 2004, IEEE, 2004, pp. 2393 - 2403,

[17] D. Kraft, M. Bechler, H.-J. Hof, F. Paehlke, and L. Wolf, "Design and evaluation of a security architecture for ad hoc networks", International Journal of Pervasive Computing and Communication, Vol. 5, No. 4, 2009, pp. 448-475.

[18] H.-J. Hof, E. O. Blaß, and M. Zitterbart, "Secure overlay for service centric wireless sensor networks", Proc. First European Workshop on Security in Ad-Hoc and Sensor Networks (ESAS 2004) , Springer, 2005, pp. 125-138.

[19] H.-J. Hof, E. O. Blaß, T. Fuhrmann, and M. Zitterbart, "Design of a secure distributed service directory for wireless sensor networks", Proc. First European Workshop on Wireless Sensor Networks, Springer, 2004, pp. 276-290.

[20] H.-J. Hof and M. Zitterbart, "SCAN: a secure service directory for service-centric wireless sensor networks", Computer Communications, vol. 28 no. 13, Elsevier, 2005, pp. 1517-1522.

[21] N. Kuntze and R. Carsten, "On the automatic establishment of security relations for devices", Proc. IFIP/IEEE International Symposium On Integrated Network Management, 2013, pp. 1-4.

[22] H. K.-H. So, S. H. M. Kwok, E. Y. Lam, and K.-S. Lui, "Zero-configuration identity-based signcryption scheme for Smart Grid", Proc. 1st SmartGridComm", IEEE, 2010, p. 321-326.

[23] A. Back, "Hashcash – a denial of service couter-measure", Technical Report, 2002, http://www.hashcash.org/papers/hashcash.pdf, accessed 12.06.2013.

[24] D. Florencio and C. Herley, "A large-scale study of web password habits", Proc. WWW '07: 16th international conference on World Wide Web", ACM, 2007, p. 657-666.

[25] R. Shay, S. Komanduri, P. G. Kelley, P. G. Leon, M. L. Mazurek, L. Bauer, N. Christin, and L. F. Cranor, "Encountering strong password requirements: user attitudes and behaviors", Proc. Sixth Symposium on Usable Privacy and Security", ACM, 2010, pp. 1-20.

[26] OpenID Foundation, "OpenID authentication 2.0 – final", http://openid.net/specs/openid-authentication-2_0.html, accessed 12.06.2013.

www.iariajournals.org

International Journal On Advances in Intelligent Systems

ICAS, ACHI, ICCGI, UBICOMM, ADVCOMP, CENTRIC, GEOProcessing, SEMAPRO, BIOSYSCOM, BIOINFO, BIOTECHNO, FUTURE COMPUTING, SERVICE COMPUTATION, COGNITIVE, ADAPTIVE, CONTENT, PATTERNS, CLOUD COMPUTING, COMPUTATION TOOLS, ENERGY, COLLA, IMMM, INTELLI, SMART, DATA ANALYTICS

issn: 1942-2679

International Journal On Advances in Internet Technology

ICDS, ICIW, CTRQ, UBICOMM, ICSNC, AFIN, INTERNET, AP2PS, EMERGING, MOBILITY, WEB

issn: 1942-2652

International Journal On Advances in Life Sciences

eTELEMED, eKNOW, eL&mL, BIODIV, BIOENVIRONMENT, BIOGREEN, BIOSYSCOM, BIOINFO, BIOTECHNO, SOTICS, GLOBAL HEALTH

issn: 1942-2660

International Journal On Advances in Networks and Services

ICN, ICNS, ICIW, ICWMC, SENSORCOMM, MESH, CENTRIC, MMEDIA, SERVICE COMPUTATION, VEHICULAR, INNOV

issn: 1942-2644

International Journal On Advances in Security

ICQNM, SECURWARE, MESH, DEPEND, INTERNET, CYBERLAWS

issn: 1942-2636

International Journal On Advances in Software

ICSEA, ICCGI, ADVCOMP, GEOProcessing, DBKDA, INTENSIVE, VALID, SIMUL, FUTURE COMPUTING, SERVICE COMPUTATION, COGNITIVE, ADAPTIVE, CONTENT, PATTERNS, CLOUD COMPUTING, COMPUTATION TOOLS, IMMM, MOBILITY, VEHICULAR, DATA ANALYTICS

issn: 1942-2628

International Journal On Advances in Systems and Measurements

ICQNM, ICONS, ICIMP, SENSORCOMM, CENICS, VALID, SIMUL, INFOCOMP

issn: 1942-261x

International Journal On Advances in Telecommunications

AICT, ICDT, ICWMC, ICSNC, CTRQ, SPACOMM, MMEDIA, COCORA, PESARO, INNOV

issn: 1942-2601