universitätbonn

Infrastructure Provisioning for Human Penetration Testing

René Julian Neff

Master Thesis

12. September 2016

1st Examiner: Prof. Dr. Michael Meier
2nd Examiner: Jun. Prof. Dr. Ing. Delphine Reinhardt
Supervisor: Dipl. Inf. Matthias Wübbeling

Institute for Informatik IV
Work Group IT-Security
Rheinische Friedrich-Wilhelms-Universität Bonn

Danksagung

An dieser Stelle möchte ich mich bei all denjenigen bedanken, die mich während der Anfertigung dieser Masterarbeit unterstützt und motiviert haben.

Zuerst gebührt mein Dank Herrn Prof. Dr. Michael Meier, der meine Masterarbeit betreut und begutachtet hat. Für die Möglichkeit mich ins aktive Projektgeschehen einzubringen sowie für seine hilfreichen Anregungen bei der Erstellung dieser Arbeit möchte ich mich herzlich bedanken.

Ebenso danke ich mich bei Frau Jun.-Prof. Dr.-Ing. D. Reinhardt für die Bereitschaft, die Aufgabe des Zweitbetreuers zu übernehmen.

Weiterhin danke ich Dipl.-Inform. Matthias Wübbeling, der mich während dieser Arbeit und meinen Studien bestens unterstützte und mir immer ein offenes Ohr schenkte. Danken möchte ich außerdem allen Mitarbeitern der Arbeitsgruppe IT-Sicherheit, mit denen ich zusammen seit meiner Bachelorzeit an neuen Herrausforderungen wachsen konnte.

Ebenfalls möchte ich mich bei meinen Kommilitonen Thomas Maqua und Jörg Stucke für die zahlreichen interessanten Debatten und die erfolgreiche Teamarbeit in den Vorlesungen des Masterstudiums bedanken.

Ein besonderer Dank gilt meiner Freundin Anuschka Clasen, die mit Liebe, viel Geduld und starkem Rückhalt immer für mich da war.

Für die immer gebotene Unterstützung zum Studium und bei all meinen Entscheidungen und sowie das in mich gesetzte Vertrauen möchte ich mich insbesondere bei meinen Eltern Ulrike Neff und Helmut Meis bedanken.

Selbstständigkeitserklärung

Hiermit erkläre ich, die vorliegende Masterarbeit selbstständig nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Statement of Authorship

I hereby confirm that the work presented in this master thesis has been performed and interpreted solely by myself except where explicitly identified to the contrary. I declare that I have used no other sources and aids other than those indicated. This work has not been submitted elsewhere in any other form for the fulfillment of any other degree or qualification.

Bonn, 12.09.2016

René Julian Neff

Abstract

In contrast to classic penetration tests, *Human Penetration Testing (HPT)* probes the user of an IT system, not the machines. *Artifacts*, presented to a test subject, are used to challenge a reaction which can be used to evaluate the subject's *IT Security Awareness* level. With the *IT-Security Awareness Penetration Testing Environment (ITS.APE)* a tool for performing a HPT is provided.

Within this thesis an extension, improving framework's task of generating and managing of required infrastructure for artifact presentation, is designed and developed. Manual configuration of routing is automated; a routing system is conceptualized and implemented. The implemented extension supports generic infrastructure types, with no need for sourcecode manipulation. Three concrete types are designed; a fully configured webserver and e-mail sending infrastructure are provided. All designs and implementations are presented thoroughly, while an evaluation shows the created software's performance.

Contents

1 Introduction

Modern industrial production, daily business and personal lives are heavily influenced by the fast improvement of communication technology. Within industrial production cycles, automated processing is widely used; machines, robots and industrial factories make use of interconnected control, monitoring and information gathering systems. Additionally, many critical infrastructure related tasks are handled via complex interconnected computer systems, including oil and gas pipeline systems, refineries, water distribution systems, communication service providers, hospitals, nuclear power plants and train signaling systems.[58] E-Commerce and around-the-clock online services provide benefits to customers and new opportunities to start-ups.

Attacks against such systems are on the rise, as interconnection through the Internet provides a world-wide available weak point.[9] Penetration Testing is used to find open vulnerabilities as well as to test a company's IT systems against already known attacks. A resulting report allows system administrators to apply necessary patches or other instruments to mitigate the risk from a technical point of view. Such *classic penetration testing* lacks the ability to test another typical *component* within a company's IT environment: the human user. A typical test environment is depicted in Figure 1.

While special cyber-security trainings are used to provide a basic knowledge about IT security, the user's *IT security awareness* is hard to determine [9, 44]. Evaluation is often done via written or multiple-choice tests and fails to test a deeper understanding of IT security in real life situations. Users lacking IT security awareness are especially prone to *social-engineering attacks*, including widely used *phishing attacks*.

The *IT-Security Awareness Penetration Testing (ITS.APT)* project aims to provide a methodology to perform close to real life testing of user's IT security awareness. Within the project, the *IT-Security Awareness Penetration Environment (ITS.APE)* framework is designed and presented.

With this framework, multiple testing situations can be generated for groups of subjects. These tests require certain infrastructure to present test elements to a specific subject and observe the subject's reaction to such a test element. Setting up, configuration and managing such infrastructure is a tedious task.[44]

This master thesis presents an extension to the already existing ITS.APE framework to simplify the work of the penetration tester while performing a *Human Penetration Test (HPT)*. The following contributions are made:

1. An extension design based on software design patterns and conform to ten identified requirements. Including designs of three concrete infrastructure services used for HPT.
2. The design of a redirecting system, used to ensure test-conform exposure of subjects to test elements.
3. Implementation of an extension capable of managing HPT infrastructure with support of legacy script-based infrastructure management, configurable black- and whitelisting of managed infrastructure types. The extension uses a simple configuration, which requires only four parameter stated in a text-based configuration file to support a new managed infrastructure type. It is capable of generating generic managed infrastructure based on containers and consolidating of commissioned demand with compatible running containers.
4. An implementation of an IP and name-based redirect system capable of routing specific subjects to designated infrastructure based on a provided schedule.
5. An evaluation confirming compliance of the implemented extension to the formulated requirements by RSpec unit tests and performed test cases of provided infrastructure.

To provide a structured overview of motivation, necessary background information, design and implementation as well as the process of evaluation of stated contributions and an outlook on future work this thesis is structured into six chapters.

Chapter 1 **Introduction** presents the basic idea of a Human Penetration Test. The reader is introduced to the IT-Security Awareness Penetration (ITS.APT) Project and its realization of such a test. A description of related work closes this chapter.

The following **Background** chapter provides necessary information needed for the understanding of this thesis. *Services* within the context of infrastructure demands of a HPT are explained. Network protocols relevant for such services in a business environment are identified and technical details are presented. A section about the concept of *virtualization* and the concrete difference between *containers* and fully virtualized machines follows. Next the *IT-Security Awareness Penetration Testing Environment (ITS.APE)* is introduced. The framework's idea, its models and modules as well as used technology are presented. The workflow to realize a HPT concludes this chapter.

The **Design** chapter presents the idea of the *Infrastructure Management Extension (IME)*. A *requirement analysis* identifies and formulates ten demands to the extensions design and implementation. A decision for a requirement conform *virtualization technique* is presented. Said technique is incorporated into the design of a class-based module for automated infrastructure generation and managing. In context of the developed design three distinct services relevant for human penetration testing are identified. The chapter closes with the presentation of a designed redirecting system.

The fourth chapter presents the **Implementation** of the designed extension. Deviating from the original design a more generic implementation and its type-based concept are introduced. The software's configuration option are described and explained. Implementation of realized modules and models their properties and methods are presented. Required modifications to legacy framework components are disclosed and extended functionality is explained. The realized implementation is finalized by a presentation of two realized managed infrastructure compositions, including all necessary components to be usable with a HPT.

An **Evaluation** chapter shows compliance of the designed and implemented software to all of the formulated requirements. Used testing methods and their implementation for testing the IME are presented and justified. A description of a testing environment for the implemented managed infrastructure types and performed test cases conclude this chapter.

This master thesis closes with a summary of the contributed work and an outlook on possible future enhancements.

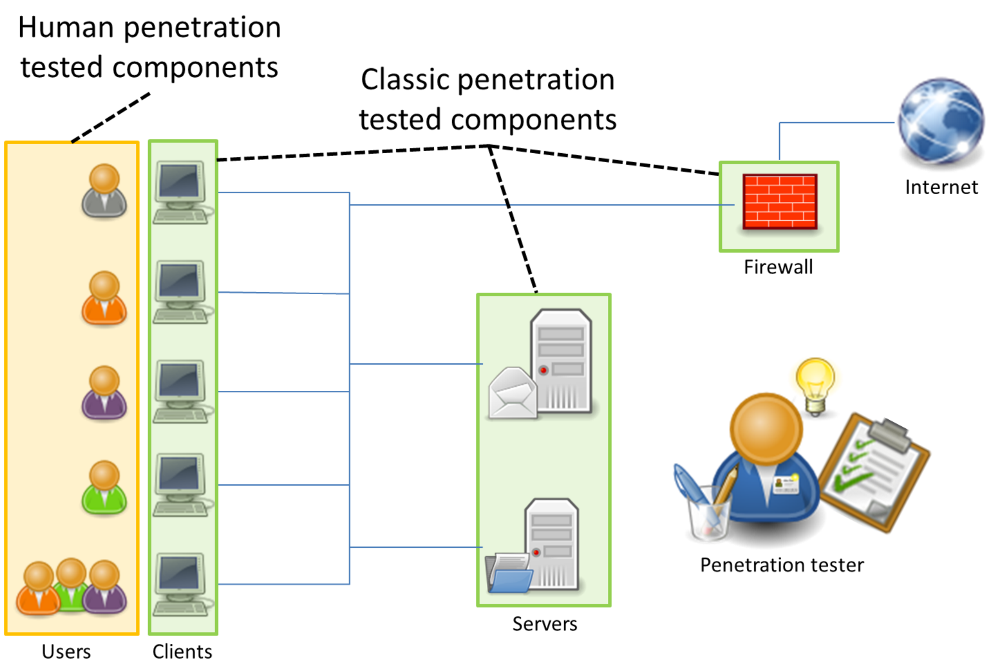

Human penetration tested components
Classic penetration tested components
Internet
Firewall
Servers
Penetration tester
Users
Clients

Figure 1: *Penetration testing environment with highlighted parts of classic and human penetration testing. According to [44]*

1.1 Human Penetration Testing (HPT)

Human penetration testing (HPT) describes a new concept to enhance a company's IT security. While technical assessment of IT infrastructure is a well-known concept that can be performed in an automated fashion, acquiring an evaluation of users' *IT security awareness* is not.[41]

IT security awareness is defined as follows:

> 'IT security awareness is situation awareness limited to elements directly or indirectly related to IT security.'[41]

Within a situation, a subject comes into contact with different kinds of elements as part of its (IT) surrounding. Elements can be part of either of two groups. First there are typical elements generated by legitimate procedures or techniques to protect IT systems. These elements are called *natural elements*. For example a legitimate authentication prompt which expects an username and a password. In contrast, elements that are part of an attack, directly or indirectly, are called *artifacts*. This includes all observable objects, items, events or effects as far as they are artificially constructed. A suitable example is an artificially introduced authentication prompt with the objective of gaining knowledge of the subject's credentials.

A subject is able to recognize such an attempt of cyber theft, if it is aware of the specific incident situation on all three levels of *situational awareness*. For level 1 of situational awareness, a subject must *'perceive the status, attributes, and dynamics of relevant elements in the environment'*[52]. Based on the perceived situation from level 1, level 2 requires a subject to *'form a holistic picture of the environment, comprehending the significance of objects and events'*[52]. Finally level 3 of situational awareness allows the subject to *'project the future actions of the elements in the environment [...] achieved through knowledge of the status and dynamics of the elements and comprehension of the situation [...]'*[52].[41, 67]

In context of human penetration testing, situational awareness of level 1 is given if the subject is able to distinguish between elements that are relevant to IT security in general. Further, level 2 situational awareness for IT security is present if the subject can differentiate natural elements from artifacts. Projecting future relevance of these artifacts relates to situational awareness level 3, as it requires knowledge about a systems' security mechanisms and possible attack strategies.[41]

Attacks on IT systems can leave behind some recognizable residuals. These leftover elements can be used to identify already known techniques and parts of cyber attacks or to identify new ways of compromise. While this highly technical investigations are performed by security community members, some of these residuals are recognizable by typical IT system users as well. Therefore, artifacts are separated into two groups: *1st-order artifacts* and *2nd-order artifacts*. 1st-order artifacts are inserted by the penetration tester into the IT infrastructure and are persistent against a variability of this infrastructure. One or more multiple 1st-order artifacts are then part of a perceivable stimulus to the subject.[41]

The subject's reaction to perceived 2nd-order artifacts is used to evaluate the subject's IT security awareness level. Consequently, it must be assured that the artifacts were perceived by the subject in the first place. Additionally, capturing the subject's reaction towards this artifacts is required for later evaluation.

1.2 IT-Security Awareness Penetration (ITS.APT) Project

Derived from this background the IT-security awareness penetration testing (ITS.APT) project aims to extend classic penetration testing by including the IT security awareness of the participating users. Therefore, a scaling of IT security awareness besides suitable methods for its measurement are designed. As classic IT security evaluations, such as questionnaires and interviews, are biased, ITS.APT aims to provide practical concept for measuring a subject's IT security awareness. This requires specific, structured and cost-efficient methods.[41]

With the *IT-Security Awareness Penetration Testing Environment (ITS.APE)* a penetration tester is provided a tool to present predefined artifacts to participating users, called subjects.[41] Given a suitable selection of artifacts the subject's level of IT security awareness can be determined.[53]

The basic idea of an ITS.APT penetration test follows the idea of pre-post design. More precisely, a pre-test is performed and subjects are exposed to artifacts. This pre-test provides a baseline of the subjects' current IT security awareness level. Followed by an educational program, for example in class training, the subjects are taught details about IT security, for example how to recognize possible attacks and how to react in such situations. A post-test is conducted to capture immediate results of such training, while later follow-up tests can be used to show long-term improvement.[44]

The following describes how a human penetration test can be performed using ITS.APE. A detailed description of the ITS.APE framework's models and modules is given in section 2.4.

Provided with a selection of artifacts, a penetration tester has to prepare a dataset with necessary subject details, such as subject's name, surname, e-mail address, title and other data depending on the chosen recipe. A classification of artifacts and details about suitable compilations of artifacts are provided in [53]. Given selected artifacts and subject details, a matching between each subject and artifact is created. It is ensured that every subject is scheduled for exposure to each kind of artifact class, as only this will test all aspects of the subject's IT security awareness [44, 53].

A prepared schedule is including time periods in which a specific artifact is presented to the subject. Additionally, this schedule ensures that no subject is exposed to more than one artifact at any given time. This restriction is applied to avoid biasing and overstraining the subject.[44]

Furthermore, some artifacts are enriched with subject-specific information. Such details can be used to provide different forms of artifacts. For example, a spear phishing mail artifact may include proper naming of the addressed subject.[44]

Generation of artifacts and deployment are used to provide selected artifacts to the subject's IT interaction point, most often a PC workstation which is used by the subject. Artifacts are designed to allow the ITS.APE framework to recognize if and when a subject is actually exposed to the specific artifact. A subject's reaction to a perceived artifact is the most important part of the testing process, recording is key for a human penetration test. All observations, collected throughout all deployed artifacts and subjects, are aggregated to generate a statistical evaluation for a final report.[44]

As different kind of artifact types exist, different kind of infrastructure is needed to generate corresponding 1st-order artifacts that cause desired 2nd-order artifacts to a subject. Moreover, this infrastructure must support various observation mechanisms, for example logging a subject's interaction with artifacts.[44] A compilation of artifacts is needed to test a subject's IT security awareness properly [53]. For each test, artifact infrastructure is needed, it must be provided by the penetration tester. Configuration of each prepared infrastructure is needed in order to be compatible with the company's IT environment and its specific artifact. If the human penetration test is completed and all relevant test data is collected, set up infrastructure must be torn down again.[44] Infrastructure needed for HPT with two artifacts is depicted in Figure 2.

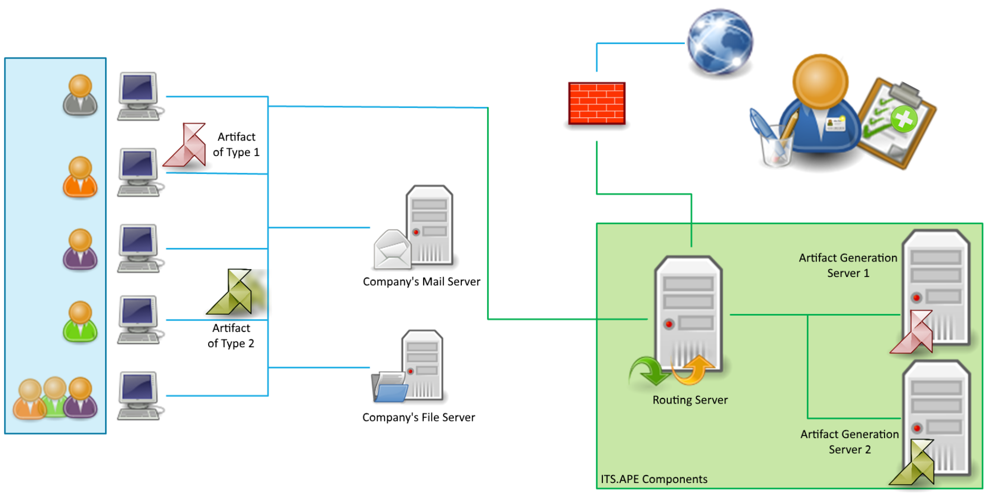


Figure 2: *A Human Penetration testing setup within a company's IT environment including highlighted infrastructure used for presenting two types of artifacts to subjects. According to [44]*

Performing such a human penetration test might require changes to the company's network system or the subject's workstation. Introduced changes should not create obstacles to any user's daily work. This policy might only be infringed for participating subjects confronted by a concrete artifact, therefore possible obstruction is limited to their work alone. It is essential to bear in mind that if the penetration test is finished and all needed data is collected, the original network state needs to be established again.[44]

The process of setting up, configuring, instantiating and tearing down of the described infrastructure can be improved. As stated above, 1st-order artifacts are platform-agnostic, as such a required set of infrastructures can be identified to provide the required deployment services. Some artifacts require a common infrastructural service for their deployment, as such these infrastructural requirements can be pooled. For known artifacts, suitable configuration parameters can be tailored to match this prepared infrastructure. For example, generic phishing mails and personalized spear phishing mail artifacts can be deployed via the same e-mail transport server.

To be able to perform a HPT, with the current framework, every single artifact has to have its own infrastructure, even if identical infrastructure demands exist. This forces a penetration tester to perform repetitious and inconvenient tasks. In addition, every single instantiated infrastructure requires configuration parameters to allow for seamless interaction with the company's IT environment. In summary, test deployment is a costly and cumbersome undertaking. These repetitious but complex tasks provide an error-prone system that makes regular human penetration tests costly and difficult to perform.

1.3 Related Work

Publicly available tools or frameworks that relate to the ITS.APE human penetration testing framework are spare. The *Social-Engineer Toolkit (SET)* provides a combination of penetration tests aimed at targeting different aspects of Social-Engineering [72]. It is an open-source tool developed in Python by TrustedSec Inc. and its supporting community [21]. The toolkit supports multiple attacks such as: Spear-Phishing Attacks, Website Attacks, Mass Mailer Attacks, Wireless Access Point Attacks and others. Furthermore, it can generate malicious QR-Codes, infectious media and payloads as well as counterpart listeners. SET is designed to leverage the Metasploit framework for certain scenarios.[73, 54]

SET's mailer function can be used to send generated attack payloads that connects back to the penetration tester's machine. Additionally, it can be used to send in a mass mailing phishing attack that holds no payload but a link to lure the recipient to a certain website.[73]

Different malicious websites can be generated by SET, websites holding Java Applets, performing a Tabnabbing attack or other similar attacks. A method to clone legitimate websites, used for these attack scenarios is provided. For websites used to authenticate users, a special credential harvesting attack can be set up. Entered credentials are collected in a text document.[73]

SET uses its own python web server and the *sendmail* command to propagate generated websites and e-mails. Where necessary, a penetration tester alternatively can set up an Apache web server and configure SET to deploy generated websites via its server.[73]

SET does not support complex coordinated test scheduling, nor does it provide an evaluation or report function. A penetration tester can *automate* different functions by calling SET with a small text file that holds menu point numbers and necessary parameters associated to the desired attack method. Further automation is not possible.[73]

2 Background

This chapter presents necessary background information for the understanding of this master thesis. At first the idea of *services*, which are used in context of the ITS.APT project are explained in detail. Further, different network communication protocols, used by the mentioned services, are described. Another section focuses on virtualization. It compiles virtualization basics while highlighting differences to the later used virtualization technique. As this thesis proposes an extension, a description of the already existing framework is given.

2.1 Service Environment

IT infrastructure is involved in many aspects of a typical day-to-day office job. Database systems, file storage systems, user communication systems such as mail servers and others are part of this infrastructure environment. All these systems aid users in their daily job routine, they are monitored, patched and expected to be available when needed.

These systems provide a *Service* to users in form of functionality. Their use can be controlled via policies, based for example on the user's identity.[8]

The *Organization for the Advancement of Structured Information Standards (OASIS)* defines service as:

> '[...] mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description.'[8]

As for the ITS.APT project, human penetration tests are conducted in the Universitätsklinikum Schleswig-Holstein (UKSH) [44]. With a workforce of over 13,000 people and about 80 medical centers and departments it is the major employer in the state of Schleswig-Holstein [75]. As part of their IT infrastructure network, several services are present: directory services, websites, e-mail, malware protection systems, network-attached file storage, printing and VoIP telecommunication [45]. An authorized user can directly interact with those systems while other systems are only provided as part of the underlying network architecture. Underlying network services are for example name resolution, access control and services that dynamically distribute network configuration parameters. While both are not exhaustive lists, several of these services can be used to present artifacts to subjects participating in human penetration tests [53].

Each of these infrastructure services can be realized by different kinds of software, depending on the specific requirements and other factors. A web server has different features, such as TLS or PHP support. Different products and projects compete for this job. There are three major products present on the Internet: *Apache HTTP Server Project*, Microsoft's *Internet Information Services (IIS)* and the *NGINX* web server.[60] Furthermore, other web server projects aim to provide lightweight web servers, which have a low memory footprint and light CPU load and focus on providing static content. Examples for static web servers are *lighttpd* and *darkhttpd*.[49, 12] Consequently, there is no such thing as a *single* web server product to chose from.

This example can be transferred to different information technologies used within a typical office environment. A common determination is the information technologies' supported protocol and therefore the provided *service* of a network infrastructure component.

2.2 Protocols

This section starts with a presentation of necessary protocols used within the implementation of this master thesis. As presented, services build upon protocols to provide a way for exchanging data. A protocol is a set of rules and regulations that defines how data is transmitted between two or more entities [71].

For network communication protocols, a special conceptual model exists which provides a standard for characterization of protocols. The *Open Systems Interconnection model (OSI model)* focuses on telecommunication and computing system protocols. It excludes underlying internal structure and technology. Seven layers are described by the model, these can be separated into two sets: The media layers (1. to 3. layer) and the host layers (4. to 7. layer) [71]. The media layers focus on the connection itself, the communication between two distinct endpoints and the running of a multi-node network. In contrast, the host layers ensure reliable end-to-end connections, host-to-host communication, concrete representations of data and finally provide transferred data to applications. Host and media layer as well as their associated sets of layers, including their purpose, are depicted in Figure 3.[76]

Besides the general approach provided by the OSI model, other models exist. The *TCP/IP model* provides a concrete description of layers and protocols which are used on the Internet. In contrast to the OSI model there are only four layers in the TCP/IP model: link, internet, transport and application. [7] The link layer in the TCP/IP model is a combination of layer 1 and 2 as defined within the OSI model. General host-to-network communication is part of this first layer and can be realized via various technologies, examples are: ARPANET, SATNET, wireless connections and LAN. Further, there is no session and presentation layer as in the OSI model, these layers are omitted as they were thought none essential.[71] Figure 3 matches the TCP/IP layers to similar OSI model layers.

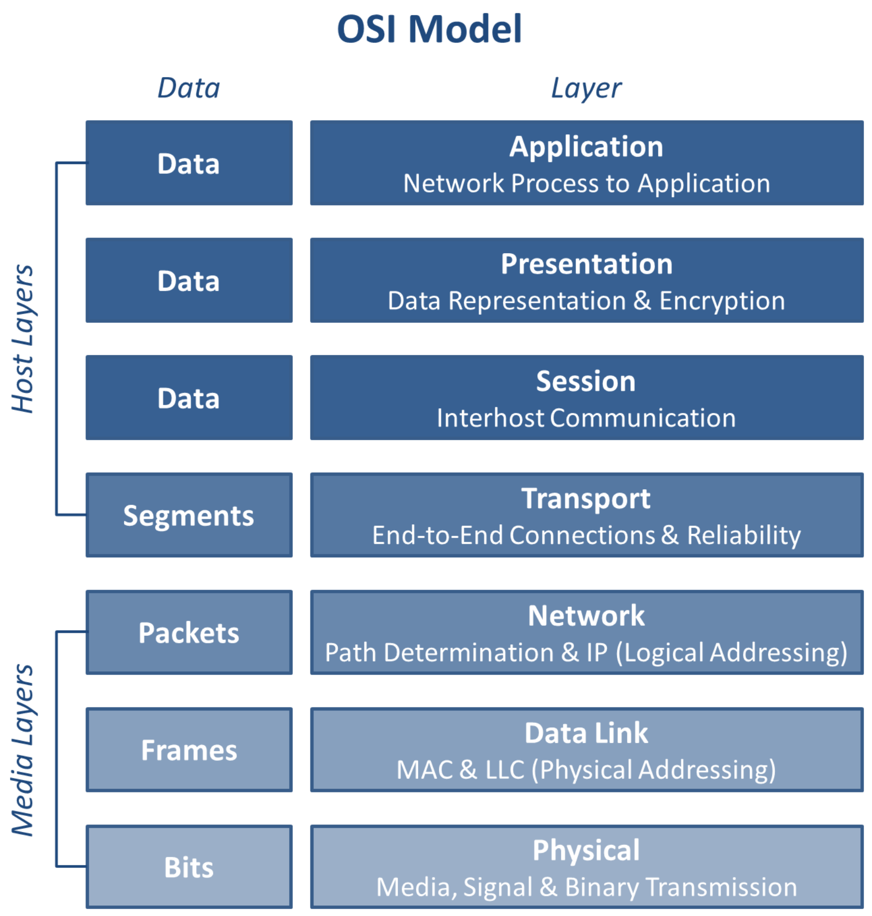

OSI Model
Data
Layer
Host Layers
Media Layers
Data
Application
Network Process to Application
Data
Presentation
Data Representation & Encryption
Data
Session
Interhost Communication
Segments
Transport
End-to-End Connections & Reliability
Packets
Network
Path Determination & IP (Logical Addressing)
Frames
Data Link
MAC & LLC (Physical Addressing)
Bits
Physical
Media, Signal & Binary Transmission

Figure 3: *Open Systems Interconnection model (OSI model) protocol stack, as defined in [76, 71]*

2.2.1 Internet Protocol version 4 & 6 (IPv4 & IPv6)

The *Internet Protocol version 4 (IPv4)* is part of the fourth layer of the OSI model. As defined in RFC 791 it is a connectionless protocol, therefore no apriori information is exchanged and no information about ongoing, future or past communication is stored or available [7]. IP is part of the *Internet Protocol Suite* as are the *User Datagram Protocol (UDP)* and the *Transmission Control Protocol (TCP)*. As defined by the TCP/IP model, the combinations of TCP/IP and UDP/IP are used heavily on the Internet [6].

IPv4 is used in networks that focus on packet-switching. In such a network data is grouped into bundles of data, called *packets*. These packets are sent over a shared medium in which multiple communications can take place. Each packet is separated into two parts: header and payload. While the payload of a packet which originates from a lower layer contains again header and payload needed by layers above. This encapsulation of data is a main principle on the Internet [7]. A packet from the Internet layer, as defined by the TCP/IP model, is used by network routing technology to deliver each packet to its final host. Arrived at its destination, a packet is processed and the

Internet layer header is stripped, leaving the payload part. This payload part is then a transport layer packet, as defined by the TCP/IP model, and can be further processed. Only then, if the transport layer packet is stripped from its header, the actual data is finally made available for application software. Figure 4 shows the encapsulation of data through the various layers.

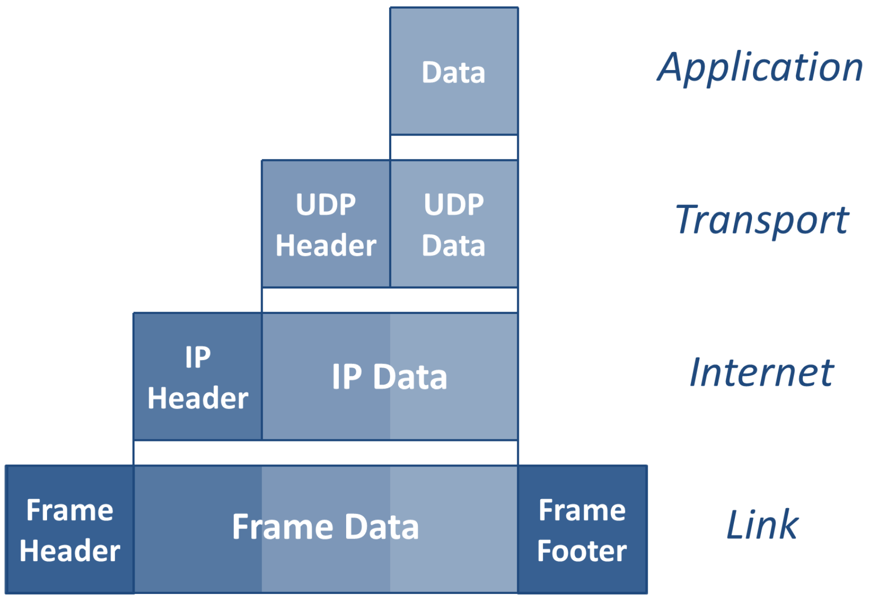


Figure 4: *Encapsulation of application data descending through the layers, as described in [7]*

IPv4 is used for communication on the Internet, its origin goes back to the ARPANET in 1983. A final version is defined in IETF publication RFC 791 in end of 1981 [7]. It is the fourth version of the network protocol and provides a standard method for communication between many different kinds of technologies. With the beginning of the nineties, the need for a larger address pool of devices connected to the Internet is discussed. Since 2011 no more IPv4 addresses can be allocated by the *Internet Assigned Numbers Authority (IANA)* [37]. The exhaustion of addresses used in IPv4 had occurred, the growing number of devices joining the Internet had depleted the address space of $4,294,967,296$ (2^{32}) addresses.

The huge success of the Internet and its IPv4 led to the development of a successor supporting more address allocations. From 1992 up to 1996 the new *Internet Protocol version 6 (IPv6)* is developed by the Internet Engineering Task Force (IETF). While not compatible with IPv4 this new protocol supports a vastly larger address space. An IPv6 address is defined by 128 bits, while an IPv4 address has only 32 bits. The IPv6 address space therefore has 2^{128} or approximately $3.4 * 10^{38}$ addresses.[39, 13] An IPv4 and IPv6 address are depicted in Figure 5 respectively Figure 6.

In addition to different ways of addressing network components, other features are introduced with this new protocol. For example a way of multicasting, previously not available with IPv4, and stateless address auto-configuration (SLAAC) are added. Alongside other improvements, these two allow for abstaining formerly additionally needed protocols. The provision of these advantages is in contrast to the fact, that IPv4 and IPv6 networks are not able to communicate with each other directly. IPv6 does create a parallel independent network, besides an already existing IPv4 network.[13] To exchange data between these two networks translating gateways are required [3]. A different approach can be used, if IPv4 traffic is tunneled through an IPv6 network [17].

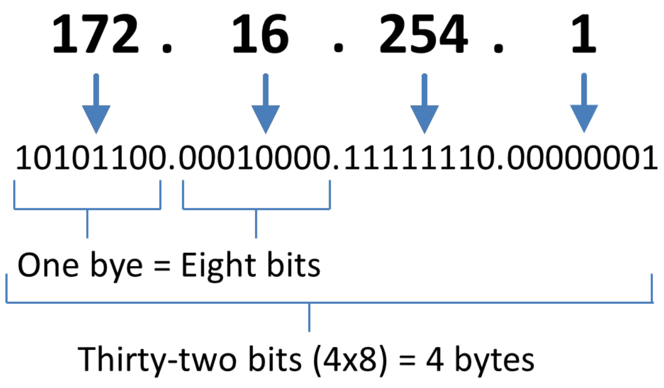


Figure 5: *An IPv4 address in dotted-decimal notation, as defined in [38]*

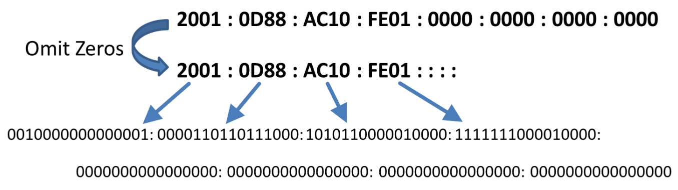


Figure 6: *An IPv6 address in hexadecimal notation, as defined in [13]*

While IPv6 is needed, as IPv4 addresses are becoming rare, its usage on the Internet is rather low. As one of the major companies, Google publishes information about the availability of IPv6 among their users worldwide. In May 2016 around 10% of these users are connected via IPv6 [23]. Other companies report about 40% of IPv6 connectivity in Belgium and about 25% availability in Switzerland, Germany or Greece [2].

2.2.2 Domain Name System (DNS)

As IPv4 and IPv6 addresses are hard to remember and a full address table of all valid host names and their corresponding IP mapping is not practicable, an hierarchical model is used to allow for a distributed system to assign names to computers or resources connected to the Internet or to a private network [47]. The *Domain Name System (DNS)* provides a translation service between domain names and a numerical or hexadecimal representation used by the IP protocols.[40, 64] As it provides a classic directory service, it is part of the application layer (layer 4) in the TCP/IP model.

The communication protocol defined for DNS requests allows to request mappings between a certain domain name and its associated records. Different kinds of record types exist and can be requested; for example holds the A record the IPv4 address stored to this domain while the AAAA record holds the IPv6 address [40, 64]. If no record for a domain is found in the local database, for example if the record's time-to-live (TTL) has expired, a request to a root server is initiated. The root server responds with an associated top-level domain name server which can further resolve the authoritative name server that holds the actual name record. Recursion of this process will finally lead to the requested record, if it exists.[40]

A resolving task like this is handled by a *resolver*; it can either handle requests recursively or iteratively. In both cases a request is looked up in the name server's local database. If such a record is present, it is used to satisfy the request. Only if a record cannot be found, these two kinds of requests perform differently. With a recursive request, each requested name server initiates further requests to the next name server which is needed for resolving the domain request. While, with an iterative request, a name server responds with the address of the next name server. Which is then requested and further resolving is performed by the initiating resolver.[40]

Defined by the standard, DNS request communication has to take place via port 53. A single UDP packet is sent from the client to the name server which responds with a single UDP packet as well. Only if the response message size is exceeding 512 bytes, a TCP connection can be used. In this case the UDP request is replied to by a truncated answer by the name server. This answer must have the TC flag set to indicate this circumstance. A new domain name request over TCP can be initiated if the truncated record does not hold all needed information.[40, 74]

2.2.3 Server Message Block (SMB) / Common Internet File System (CIFS)

To exchange application data between two hosts, additional protocols are needed. IP and DNS support identification and exchange between two or more hosts. As already stated, file exchange through an IP network connection is used by the UKSH. The *Server Message Block (SMB) Protocol* is used for this task. It is part of the application layer in both the OSI as well as the TCP/IP model. Different versions of this network file sharing protocol exist, while no clear standard documentation or definition is available through IETF.[11]

The development started in the early 80's by Dr. B. Feigenbaum at IBM. Back then it is given the name *BAF* (after his initials). The protocol is renamed to *SMB* and further improved by Microsoft, Intel and 3Com. Again, it is given a new name, *CIFS*, when an even further improved version is presented by Microsoft in 1996.[11] Multiple attempts to form a proper standard were made by different groups, but ultimately failed.[48, 35] By today, there still exists no authoritative protocol specification to check for correctness of any given implementation.[11]

Microsoft is supporting its implementation of *SMB*, still called *CIFS*, and provides a detailed description of its *'dialect' of SMB* as it is used by the Microsoft NT LAN Manager (NTLM) [56]. Microsoft products such as Windows NT Server 4.0, Windows XP or Windows 2000 use CIFS [56]. To improve speed of data exchange, a reduction in commands and a new way of linking them to each other, were implemented by Microsoft. The CIFS successor is again renamed and is called *SMB 2.0*. SMB 2.0 is used in Microsoft's Windows Vista, Windows Server 2008 and Windows 7. With SMB 3.0 the protocol supports multichannel transmissions and an end-to-end encryption is added. SMB 3.0 is used in Microsoft's Windows 8, Windows 8.1, Windows Server 2012 and Windows 10.[57]

As SMB/CIFS is part of the application layer, it relies on lower-level protocols for transport. The transport layer protocol in use, is most often *NetBIOS over TCP/IP (NBT)*. Alternatively, SMB can be run directly through a TCP connection, while NBT is used for backwards compatibility.[55] In case of a direct TCP connection port 445 is used, NTB connections use TCP port 139 [57].

2.2.4 Simple Mail Transfer Protocol (SMTP)

The *Simple Mail Transfer Protocol (SMTP)* is an application layer protocol which is used for Internet electronic mail transport. It is designed as a mail transport and delivery protocol that only relies on an ordered data stream channel. Mainly SMTP over TCP is used, but other transport protocols, such as NCP in the ARPANET, can be used too. SMTP features *mail relaying*, which allows for transporting mails across multiple networks. Therefore, a mail message can be transmitted through some intermediate relay or gateway hosts on its way from sender to ultimate recipient.

A basic transmission of a typical message via the SMTP protocol is depicted in Figure 7. From a user's host a *mail user agent (MUA)* is used to transfer electronic mail to a *mail submission agent (MSA)*, using SMTP on TCP port 587 [29]. While submitting a message is not defined to be done via TCP port 25, it is supported by many mailbox providers.[29] At the beginning, a message is forwarded from the MSA to a *mail transfer agent (MTA)*, these two agents can be bundled as a single piece of software. If agents are distributed among multiple machines SMTP is used to transfer the electronic messages between them. The MTA is responsible for locating the target host, which can be within the same local network or part of the Internet. If it is the latter, DNS is used to look up the *mail exchange (MX) record* for the associated domain provided as recipient. This record holds the target host, which is a MTA, too. The originating MTA can connect to this *mail exchange server (MXS)* as an SMTP client and transfer its electronic message. The transfer can occur directly between two MTAs or through multiple hops passing intermediate systems. Finally, the recipient's MTA delivers the message to a *mail delivery agent (MDA)*.[32] Retrieving a message from an MDA can be done using specialized protocols such as *Post Office Protocol version 3 (POP3)* or *Internet Message Access Protocol (IMAP)* [30, 25].

All mentioned SMTP connections are, by definition, handled via TCP port 25. Legacy systems that use *Transport Layer Security (TLS)* to transfer messages use TCP port 465.

These connections are called *Secure Simple Mail Transfer Protocol (SSMTP)* [18]. Modern systems use *Opportunistic TLS*, called *STARTTSL*, to secure plain text communication such as SMTP, POP3 and IMAP [31]. STARTTLS is designed to upgrade an existing connection. This allows for retaining the originally assigned TCP port 25, instead of providing a separate port used for encrypted connections only.[28]

SMTP is just a transport protocol and does not define the message content. Consequently, only the message's envelope and its parameters are defined; these are: the envelope sender, but no header or message content. The message itself, header and body, is defined by a different standard and is referred to as the Internet Message Format.[34]

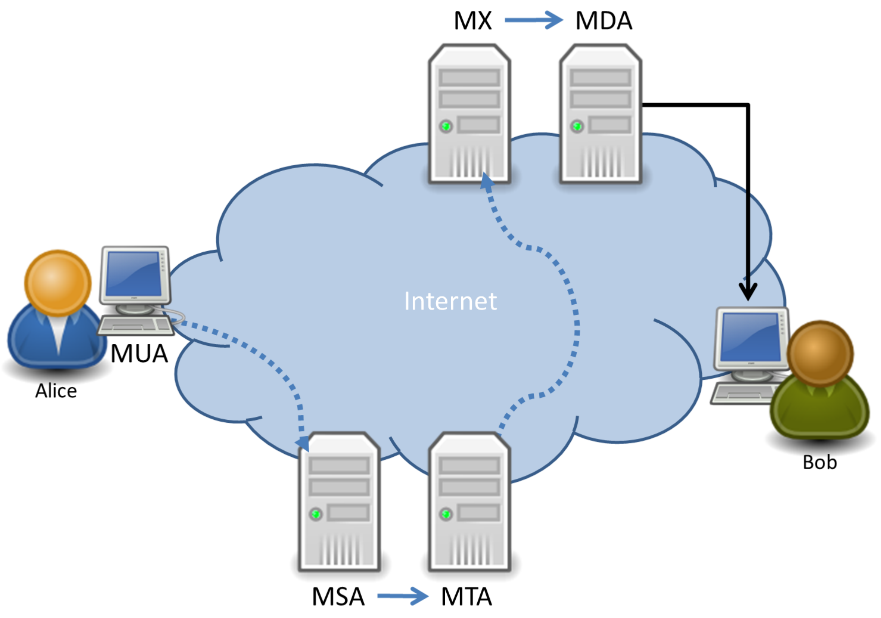


Figure 7: *Basic SMTP communication, including several SMTP connections to deliver an user's message, as defined in [29].*

2.2.5 Hypertext Transfer Protocol (HTTP)

The *World Wide Web (WWW)* is a worldwide network for exchanging data via various protocols. It was first used by the European Organization for Nuclear Research (French: Organisation européenne pour la recherche nucléaire). The CERN is an European research organization that operates the largest particle physics laboratory in the world. In 1989 the WWW is invented to share scientific research of the CERN with scientists from all over the world. Its main purpose is to enable exchange of reports, drafts, sketches, pictures and other documents in an efficient and persistent way.[71]

Berners-Lee presented his proposal of a network of linking documents, including a first prototype for a text-based browser in 1991. A connection between two documents via a *Link* is called *Hypertext* and is presented by Vannevar Bush back in 1945. Berners-Lee used this already known concept and combined it with the idea of the Internet.

He presented a system to identify resources (e.g. documents, pictures, ...) via *Uniform Resource Identifier (URI)* in a general way. Additionally, special URIs are available that define the means of acting upon or obtaining the representation of a resource: *Uniform Resource Locator (URL)*. The documents used in his proposal and his idea of the Web are written in a standardized way: the publishing language *HyperText Markup Language (HTML)*. They may include other resources, these are identified via *Multipurpose Internet Mail Extensions (MIME)* types, also used for e-mail attachments. Different MIME-types allow for bundling different kind of content within a single file, MIME-tags provide suitable markers for separation of this content for example by the web browser. Furthermore, to provide a standardized way of communication, a protocol is defined: *the Hypertext Transfer Protocol (HTTP)*.[71]

The above mentioned development makes it obvious that *the World Wide Web* is different from *the Internet*. As the *Web* only defines an information network of hyperlinked documents and other resources identified via URIs. This network *facilitates* the services provided by the Internet, which is done by its *Web-client (Browser)* and *Web-server*. They communicate via *HTTP*.[71] HTTP communication, by definition, is realized via TCP/IP on TCP port 80 [63]. Besides TCP/IP, DNS is used to locate the associated Web-server that is specified in the *host* part of an URI [71].

In May 2015 a successor of the HTTP 1.1 protocol is presented: HTTP/2. The new protocol is developed by Google and focuses on improving latency of website requests. While the protocol aims for compatibility to HTTP 1.1, it supports new features: compressed HTTP headers, pipelined requests and multiplexing multiple requests over a single connection.[61] It is supported by all major browsers in June 2016 [10]. In October 2015 about 31% of Alexa top 100 websites supported HTTP/2. Varvello et al. find an 80% decrease in page load time of their measurements of Alexas top 1 million websites.[19]

2.2.6 Protocols over SSL/TLS (HTTPS, SMTPS)

Traffic sent via TCP is not secured against outside attackers. Additionally, a user can not validate authenticity of a website without additional information.[63] To protect users' privacy and integrity of exchanged data via TCP, an additional protocol is used. Secure Sockets Layer (SSL) and Transport Layer Security (TLS) are used to secure a connection between two communication partners. They are both part of the application layer within the *TCP/IP model*.[71] SSL and TLS provide a sub-layer within the actual communication layer as they encapsulate traffic from typical application layer protocols.[26]

To secure traffic, HTTP and other application protocols can use SSL/TLS. HTTP is then called HTTPS. Other protocols are for example FTP, then called FTPS, or SMTP, which is then called SMTPS [27, 28]. For HTTP traffic secured via SSL/TCP, TCP port 443 is standardized. In addition URLs of a secured website begin with "https://" [26]. As early implementations of SSL/TLS are insecure, new TLS versions are developed [4]. As of this writing, TLS protocol version 1.2 is the newest standard, since April 2014 TLS protocol version 1.3 is under development and new drafts are released on a regular basis [69, 68, 33].

SSL/TLS communication uses different cryptographic principles to secure a connection. Symmetric cryptography is used to encrypt actual application data, these symmetric keys are generated for each connection. The identity of a communication partner can be authenticated via *public-key cryptography*. Standard browsers are equipped with around 100 public keys signed by trusted certificate authorities. To provide integrity of a connection, *message authentication codes (MACs)* are used to detect alteration and loss of data.[71] Said newest version of TLS supports *forward secrecy*, which prevents decryption of messages sent in the past by future encryption keys [69].

Each communication therefore has to exchange or agree upon certain cryptographic parameters. This cryptographic information is sent, in addition to more general information, such as supported compression algorithms, in a single TLS *ClientHello* message. The SSL/TLS client sends a cipher suite list that contains supported cryptographic algorithms in order of its preference (favorite choice first). A cipher suite defines a key exchange algorithm, an encryption algorithm used for data transfer, a MAC algorithm and a pseudo-random function. The server will select a cipher suite from this list or, if no acceptable choices are available, it will return a handshake failure alert and close the connection.[26] This is an example for a 1st order artifact.

Different websites may have different website certificates used to establish a secure connection. As the actual HTTP GET request, including the requested domain, is only transmitted after a secure connection could be established, a server can not decide on which website host and therefore which certificate is requested. *Server Name Indication (SNI)* is an extension that sends the requested host name along the TLS ClientHello message, which lets the server choose the correct certificate. Without SNI, every server's IP interface could only be used for a single secure website.

2.3 Virtualization

With the beginning of mainframe computers back in 1960 to 1970, virtualization is used to share limited resources between multiple users or application. Before this technological improvement, only a single application could be run at a time. Besides computing power, applications are sometimes run on dedicated machines to avoid conflicts with other applications.[5]

With separation between running instances of the same or of different applications, the user is provided with additional security against attacks on one of the applications. In a case like that only the attacked virtual machine if said application is affected, while this is no guaranteed security because this additional layer can be mitigated. It still provides an additional obstacle for an attacker.[46]

Different kinds of definitions can be found, Amit Singh's definition tries do give a rather broad idea of *virtualization*:

> "'Virtualization is a framework or methodology of dividing the resources of a computer into multiple execution environments, by applying one or more concepts or technologies just as hardware and software partitioning,

time-sharing, partial or complete machine simulation, emulation, quality of service, and many others."'[5]

Classic Virtualization

A classical virtualization system uses a software called *hypervisor* that orchestrates control flow between one or multiple virtual machines. It handles all communications between the host operating system, the so-called *host OS*, and each virtual machine. *Type 1* hypervisors are applications running on top of an OS present at the host system. In contrast, *type 2* hypervisors operate directly on the system's hardware. Each machine has its own separate operation system, called *guest OS*.[24]

The hypervisor tries to emulate basic underlying hardware to the guest OS. Therefore the guest system can be run as if it would be running on the bare system's hardware machine. A virtual system's guest OS can be different from the host OS or can be the same, while restricted by the underlaying hardware. For example compatible hardware systems such as x86 can be virtualized on an x86-64 host machine as the x86 instruction set uses a subset of the x86-64 instruction set. Known implementations of this are *AMD64* and *Intel 64*. Applications running on a virtual machine OS have their own libraries and resources. Even if the same libraries or resources are used they have are placed separately into each machine. A typical virtualization stack is depicted in Figure 8.[24]

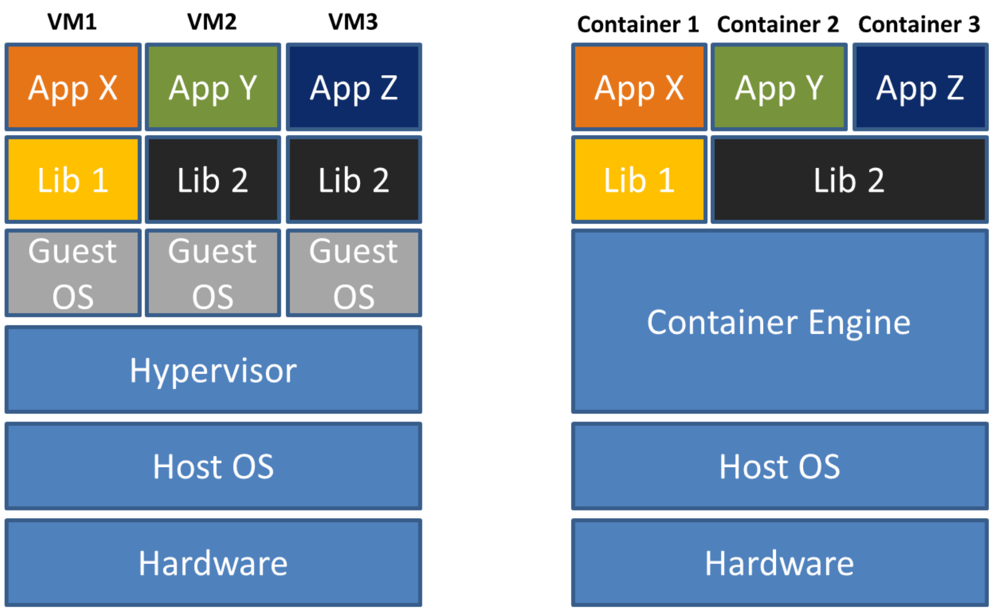


Figure 8: *Basic virtualization stack running three seperate virtual machines on a single host machine is compared to a container virtualization stack running three seperate containers on a single host. Two applications require the same library.*

Virtualization with Containers

This concept provides rather strict separation through fully virtualized machines. Different to this hard separation, other virtualization technologies omit a separate guest OS. *Solaris Zones*, limited to Solaris OS, provides such application separation [66]. For Linux, *OpenVZ* is presented in 2005, but required a patched host system kernel [65]. The same holds for *Linux VServer* [51]. By a combination of several technologies, like CGroups, kernel namespaces and chroot, *Linux Containers (LXC)* provided a container solution from 2008 onwards. LXC offers an interface for the Linux kernel containment features, it allows for creating and managing whole virtualized systems or application containers.[50] Docker Inc. present's their *Docker Engine* with a focus on providing an operating environment for applications, rather than systems. Their engine is capable of running so-called *Docker Containers*, these are portable encapsulation of applications and their needed dependencies. In contrast to full virtualized machines they share resources with the host's OS, therefore hardware emulation of the hypervisor can be omitted.[14]

A *container engine* provides necessary translation and interconnection between the host OS and other containers, it acts similar to a hypervisor used by a full virtualized machine. Almost direct interaction with the host system's resources allow containers to be more efficient, as such they can be started and stopped within in seconds. Furthermore, applications executed in a container, run only with little overhead compared to applications operating on the host system directly.[14]

As most resources are already provided by the underlying host OS, only application specific dependencies must be made available to the container. This allows for lightweight and therefore portable application containers. Additionally, such a controlled environment, provided by a container, can be defined to hold necessary parameters needed for complex applications, this lowers configuration complexity for users.[14]

A container setup with three containers is depicted in Figure 8, too. Two of these application containers share the same libraries instead of having their separate copies.

2.4 IT-Security Awareness Penetration Testing Environment (ITS.APE)

The following will present the software framework implemented to support the ITS.APT project. A short description of the basic functionality of this framework is followed by a more detailed run through the different modules and models used within the framework. Used tools and technologies are presented, as this section closes with an exemplary description of the different stages a penetration test will pass during its execution. The internal frameworks' concept documentation is used as the primary source for this section [42].

2.4.1 ITS.APE framework goals

To achieve the idea of measuring the security awareness level, the *IT-Security Awareness Penetration Testing Environment (ITS.APE)* is used. This framework allows evaluating a user's security awareness level within its normal working environment. ITS.APE supports and handles most of the penetration tester's tasks, as it can be used for test preparation, actual test running as well as the generation of presentable test results.[42]

As described in section 1.2, *artifacts* are mixed into a subject's daily routine work, while its reactions are observed. Generation and deployment of artifacts should be done in a way which will not reveal the artifacts artificial nature. Therefore, a subject may not be able to distinguish between an artifact provided by ITS.APE and an artifact provided by actual malicious sources. Overall, subjects' interactions with the framework must not provide any unintended obstacles for their daily work. In direct interaction with framework parts, as presented, artifacts are one mean in which a subject can come into contact with the framework. Indirect contact with the framework can happen if a subject reports intentionally introduced anomalies to the companies' IT helpdesk. The helpdesk supporter is therefore able to report the subject's observation via the reporting feature of the framework. Again, this is to be kept simple even if the supporter is not a direct subject himself but is reacting on behalf of a subject. Reactions to artifacts presented by ITS.APE must be observed, as they provide the necessary feedback for later evaluation.[42]

The system is operated by a penetration tester whose duty is the provision of artifacts which should be presented. Depending on the artifact's type, different *services* or *combined services* are required to present an artifact to a subject.[53] All necessary information about an artifact is bundled in a so-called *Recipe*, which will be explained in detail in section 2.4.3. Provided with a *Recipe*, the framework is able to generate personalized artifacts. While the *Recipe* provides the basic template, the specific subject data is required too. Consequently, the penetration tester has to select the subjects who are tested and must provide required subject specific information. Setting-up an ITS.APT penetration test shall be possible within a short time frame; only a small set of configuration parameters is required.[42] These parameters depend on the chosen type of artifact [53]. The penetration tester's interaction with configuration files and command line outside of the ITS.APE framework shall be avoided.[42]

The framework is used within a given infrastructure of a company, this testing environment allows for testing the user's security awareness level with a low bias. Not all required services might be available in an examined company. Furthermore, services already present within a company's infrastructure might not be operated by a framework compatible software. It is possible, that this infrastructure is not as configurable or up-to-date as necessary. Finally, due to policies within a company, some services are not accessible for necessary changes in configuration. For example, these configurations could be the installation of outdated certificates to test subjects for their reactions towards affiliated browser warnings. Moreover, only a limited group of users within a company's network are participating in an ITS.APT penetration test. Therefore only these subjects must be affected by the ITS.APE introduced artifacts. Hence, the framework must provide infrastructure on its own, which provides the necessary and compatible set of configuration parameters stated within the penetration tester's recipe.[42]

2.4.2 Modules

The following paragraphs decribe all modules present in the ITS.APE framework. A short motivation for each module is followed by a description of the module's features.

APE

Different components are orchestrated, additional control flow and user interaction needs to be managed. The *APE module* allows for user input and will present necessary prompts to the penetration tester. Furthermore, all control flow starts and returns to this module. Subsequent modules return to the *APE* module when their actions are finished.

The module is run as a daemon and interacts with all other modules as it is the main process. Hence, initializing is done from this module. Through a socket connection, the module is capable of interacting with the user via an user interface. It provides several commands via this interface to start, stop and control ITS.APT penetration tests.[43]

Artifact Generator

As explained, the ITS.APE framework uses *Recipes*, datasets containing technical information of a human penetration test needed to set up a complete testing environment. For example e-mail templates used to create a specific e-mail phishing campaign are part of those datasets.[53]

Some kinds of artifacts need modification or adaption before a single artifact can be presented to a subject. For example a simple phishing mail might only use a single generic recipient e-mail address. In contrast, a more sophisticated phishing mail uses the subject's e-mail address and a valid response e-mail address. Enriching artifacts with subject specific details, such as the subject name or correct e-mail address are therefore needed. Such details are used to create more precise artifacts, which can be part of more sophisticated ITS.APE penetration tests. Besides subject specific details, other parameters could be part of this adaption process as well. Typical examples are the response e-mail address or phishing websites. To create different kinds of phishing website artifacts, a penetration tester can for example specify a more realistic domain name in contrast to a simple random string domain name.[43]

Deriving enriched artifacts from the *Recipe's* template, by using additional information, is done by the *Artifact Generator* module. Generated artifacts are temporarily stored on the machine running the ITS.APE framework daemon.[43]

Infrastructure Generator

As stated earlier, the framework has to leverage different kinds of infrastructure to provide required services for selected *Recipes*.[53] Different approaches can be used to achieve this goal:

1. In one possible scenario the company already has compatible infrastructure software within its network and the possibility to adapt its configuration.

   This arises the need for a suitable instruction for manually configuring this infrastructure; a written instruction is required. This kind of infrastructure is not easily manageable, as all configuration and changes to already leveraged infrastructure must be handled manually. These changes are time consuming as they are performed by personnel from within the company and not the penetration tester himself.[43]

   Infrastructure used as described within this scenario is called: *External Infrastructure*.
2. Within a different scenario, no compatible infrastructure software inside the company's network exists or can be altered due to other factors. This scenario requires that infrastructure is set up including a suitable configuration.

   Required infrastructure is therefore set up manually by the penetration tester and necessary configuration is entered as provided by the framework's instructions. Besides the additional task of setting up needed infrastructure, written instruction are required too. This kind of infrastructure is under full control of the penetration tester. It can be made accessible and changes can happen automatically if necessary access is configured. Nevertheless, setting up, maintaining and later tearing down this infrastructure is a tedious task, as manual work is necessary for every new recipe and its infrastructural needs.[43]

   Infrastructure used as described within this scenario is called: *Internal-Manually-Setup Infrastructure*.

The *Infrastructure Generator Module* is responsible for this task. It uses the information stored within the selected *Recipe*, to generate instructions which are presented to the penetration tester. All necessary information required for the infrastructure is available, this includes at least: IP address, port and credentials.[43]

With this limited information, the framework itself cannot distinguish whether provided infrastructure is set up by the penetration tester or has already been part of the company's network. Such infrastructure is indifferent from a technical standpoint. A differentiation is only relevant on an organizational level, hence it determines whether the penetration tester has to manually provide infrastructure or can leverage it when already existing.

Deployment & Delivery Manager

As stated before, generated artifacts are created and stored on the machine running the ITS.APE framework daemon. The framework is designed to use services provided by machine independent infrastructure to present artifacts.

Artifacts, which are ready for presentation, need to be transferred to the corresponding infrastructure machine. The *Deployment Manager* module is designated to fulfill this task.[42]

As prepared artifacts are deployed to the infrastructure by the *Deployment Manager*, the *Delivery Manager* module is designated to invoke necessary actions in order to deliver a single artifact from the infrastructure to the user. As an example for an e-mail the execution of a *send* function with parameters could be named; this could be the subject's e-mail address and the prepared phishing mail.[43]

Scheduler

Typically an ITS.APT penetration test can be separated into multiple parts. These are: *pre-evaluation*, *training* and a *follow-up evaluation*, referring to 1.2. Furthermore, preparing an ITS.APT penetration test involves many factors and parameters. Depending on a chosen artifact, infrastructure required to be set up and/or configured. A selection of an *Recipe* and necessary artifacts template's parameters have to occur. A group of test subjects must be assembled while other test specific actions have to be undertaken. Besides these technical tasks, organizational obstacles require to be taken care of. Therefore an important part is subject test timing. Presenting artifacts while the subject is on vacation or working in shifts might result in a state in which the subject is avoided any contact to the specific artifact.[42]

To tackle such timing issues, the *Scheduler* module is used.[42] Provided with parameters such as a penetration test runtime and all test specific parameters, it generates a *test schedule*. Besides a test's runtime, certain constraints are taken into account when a schedule is generated. For example, there shall be no double exposure of two artifacts or more to a single subject at any given moment. It is possible to provide a set of office hours in which artifacts are presented. Furthermore, a schedule is automatically adapted if it was paused during its execution. Therefore running tests will be postponed to meet the predefined test runtime. According to this schedule the *Delivery Manager* is used to deliver artifacts. Additionally, the *Scheduler's* logic allows for determining, when a certain test is finished; as no more artifacts are commissioned for delivery. This information is used to initiate function to tear down used infrastructure.[43]

Track Collector

As already stated, a crucial task within the ITS.APT project is tracking user reactions towards presented artifacts. For example, reactions can be observed by checking access log files for entries which are associated with artifacts only made available to a single subject. A simple example might be a logged access to a specific website through a link only provided to a specific subject via a phishing e-mail.[42]

Recipes can leverage different services to present artifacts. This necessitates the need for different infrastructure. As presented in the *Infrastructure Generator* paragraph, different kinds of infrastructure exist. Especially *External Infrastructure* is not easily accessible, as all configuration and changes to already leveraged infrastructure are handled manually and are time consuming because of organizational obstacles.

Collecting user reactions, especially from *External Infrastructure*, therefore needs to happen in an automated fashion. This feature is provided by the *Track Collector* module and its supporting external component, the so-called *Satellite*.[43]

The module uses predefined filter rules upon log files, which allow for a transformation of general log file entries to ITS.APE framework conform representation of an observed action, so-called *Tracks*. A *Track* therefore is an observation of a specific reaction of a subject to an artifact. Necessary filter rules are provided with each recipe and are called *Track filter*.[43]

To eliminate the need for manual log file retrieving, a local component is used which is called *Satellite* and is running on used infrastructure machines. It can connect to the ITS.APE framework through a secure connection and observes a single log file. By this setup, *Tracks* can be created automatically while confidentially and integrity are ensured.[43]

Result Generator

Observed reactions to artifacts are scored with different values, indicating the subject's security awareness level. Ranking of a reaction towards an artifact is given by predetermined scoring defined by the penetration tester. Training for example can have an effect on the subject's awareness. Conducting an equivalent penetration test as a follow-up might show an improvement in the subject's security awareness.[44]

Subject specific results, performance information of conducted penetration tests are collected. As part of the ITS.APT project, penetration tests are reevaluated as an ongoing task to improve future testing.[42]

Generating reports is done with the *Result Generator* module. These reports provide graphical representation of statistical information and results, which can later be used in trainings and penetration test reports.[43]

Different levels of detail are possible. On the one hand information about subject's performance towards one specific artifact can be evaluated to allow for inter-artifact comparison. On the other hand, to evaluate subjects, two kinds of reports are possible. *Status reports* allow for evaluating the progress of an ongoing test season. A *final report* compares two test seasons and allows for observing the effect of intermediate training.[43]

Miscellaneous Modules

Modules necessary for the direct ITS.APT project purpose, additional modules help to provide consistent access to information within the implemented framework.

Database access is realized through the ActiveRecord *Database Connector* module and is used heavily by the ITS.APE models which are described in the next section.[43]

For debugging and error logging purposes throughout all parts of the framework, a *logging module* is provided. Different levels of log entries are available: error, warning, info, debug.[43]

The *Configuration Manager* module allows for an easy access to the framework configuration information. By design, several information such as subject data, recipes and others are not stored in the database or are accessible via the module class. This module allows for a generic and easy access to this data throughout the whole framework.[43]

Future extensions shall make use of the provided logging capability and may use presented modules as needed.

2.4.3 Models

The ITS.APE framework uses models to provide consistent instances of datasets used by multiple modules. This section presents important models of the ITS.APE framework and their use within the framework is explained while some specifics are highlighted. All modules except for the *Subject* module are Active Record modules and therefore stored in the framework database.[43]

Recipe

A *Recipe* contains all data needed to create a specific ITS.APT penetration test season. Every *Recipe* must have a name and a duration, defining a penetration test's runtime. Furthermore, a description of necessary infrastructure known as *Infrastructure Elements* is given too. Regular expressions are used within *Track Filter* instances and allow for later use within the *Track Collector*. A parameters' hash map allows for defining test specific configuration options as for example placeholder identifiers. These parameters are used in conjunction with an *Artifact Generation's Script* used by the *Artifact Generator*, to create subject specific artifacts. Finally, a *Recipe* has to provide scripts which are used to handle and deliver these artifacts. For transferring or providing access to prepared artifacts, for final presentation to a subject, a so-called *Deploy Script* is used. *Arm- & Disarm-Scripts* are used for activating and deactivating artifact presentation to a subject.[43]

Infrastructure Element

An ITS.APT penetration test can use different kinds of services as well as infrastructure to present artifacts to a subject. An *Infrastructure Element* describes a required service, for example a web server, to present a website artifact. All kind of infrastructure used within the framework needs detailed setup descriptions. Accordingly, written guides are part of this dataset. In addition to setup descriptions, it provides appropriate tear down instructions. As an alternative to a written guide on how to set up needed infrastructure, a penetration tester can provide scripts which are executed to set up and tear down infrastructure. Either way, while running a set up service, it requires monitoring, a dedicated *status script* is used by the framework to check whether the infrastructure is providing correct artifacts. This status script is supplied by the penetration tester as well.[43]

Subject

For each test participant a *Subject* instance is created. While each *Subject* has to have a name, surname, e-mail address and an username, there can be additional subject specific information. This additional information can be defined by the penetration tester and used within scripts if necessary. For example a subject's gender can be stored too; this will allow for a correct form of address, for example in a phishing mail. As such information is highly personal, database duplication must be avoided. This design decision respects the German laws governing data protection and data security. The *Subject* class therefore is capable of accessing the subject's information from a single comma-separated value's (CSV) file, provided by the penetration tester. In this document the class model is denoted by *Subject*, while a human test subject is referred to as subject.[43]

Test Specific Models

A penetration tester can create multiple *Test Programs* within a single test environment. While for example the administration department is tested for its security awareness in context with phishing mails, manipulated website certificate warnings are presented to the sales department.[43]

Each *Test Program* has multiple *Test Series*. A *Test Series* is created to store selected subjects for a specific *Test Program*. This data model is used to provide a persistent access to subject information for all modules.[43]

Again, each *Test Series* has multiple *Test Seasons*. A *Test Season* is created to store the penetration tester's selected *Recipe* for a specific *Test Season*.[43] The *Artifact Generator* module can use additional parameters which are stored in the *Test Season's* dataset to further adapt a created artifact. After the generation is done by the *Scheduler*, each *Test Season* holds all generated *Test Episodes* which have been created for the specific *Recipe*.[43]

By creating a cross product between *Subjects* and *Test Seasons*, *Test Episodes* are created by the *Scheduler* module. Each *Test Episode* has exactly one *Schedule* instance containing information about start and end date. Furthermore, each *Test Episode* instance stores a status, whether this episode is prepared, running, paused or already completed.[43]

A *Schedule* instance stores the start and end date of an ITS.APT penetration test instance. The duration between the start and end date is given by the *Recipe's* parameters. A typical example is a phishing website which is available for just one week.[43]

Track Filter

The *Track Filter* model is used to store regular expressions used for later extraction of generated log entries generated on infrastructure that is part of this human penetration test. Such log entries are used to determine the action a subject took during its penetration test. Additionally, each action is rated with a score to allow for later evaluation. Suitable regular expression as well as the ranking of each action and its associated score are provided by the penetration tester.[43]

A processed log entry is stored in an instance of a *Track Entry*. This model allows for storing the concrete observed action, the subject's id and the associated score of the chosen action. Each *Track Entry* has a timestamp, preserving when the subject showed this reaction.[43]

When a subject interacts directly with the framework, for example when visiting a phishing website as part of a human penetration test, this interaction can be associated to the subject's accessing device. Each of these devices is stored in an instance of a *Computer* model. This model stores the computer associated IP address and its host name. As multiple reactions can be observed, for example a phishing mail can be read and a phishing website can be visited, each *Computer* instance can be linked to multiple *Track Entries*.[43]

2.4.4 Technology

Several well established technologies are used to implement the ITS.APE framework, this section presents them.

To allow for use in combination with well-established technical penetration testing frameworks, such as the Metasploit Framework, ITS.APE is written in the Ruby programming language.[44, 43]

It uses object-relational mapping (ORM) to store most of its data in a provided database. This feature is realized by the ActiveRecord library available for Ruby.[43] The ActiveRecord library is the default 'model' part of the Ruby on Rails framework which is a well-established web-application model-view-controller framework. The library itself is available as a stand-alone ORM package for other Ruby applications and benefits from improvements made by many contributors. [70]

Within the ITS.APE framework an SQLite database is used during development. For later production purposes a PostgreSQL database should be used.[43]

Furthermore, a high test coverage percentage (approximately 90%) is required for all framework source code; documented and test-driven software provides a foundation for further development. Extensions such as the one developed within this thesis context, have to comply with these goals too.[43]

2.4.5 Human Penetration Test Workflow

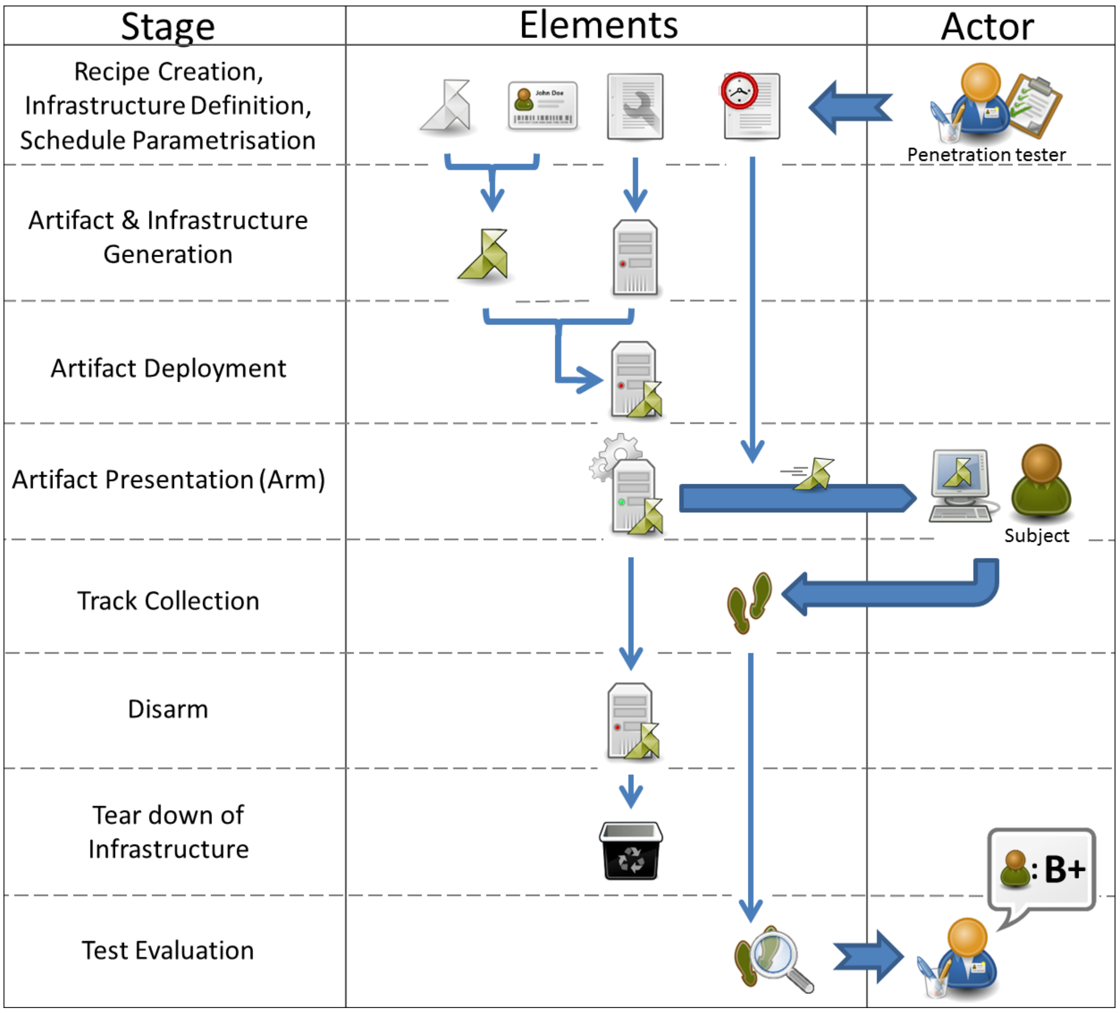

Stage
Elements
Actor
Recipe Creation, Infrastructure Definition, Schedule Parametrisation
John Doe
Penetration tester
Artifact & Infrastructure Generation
Artifact Deployment
Artifact Presentation (Arm)
Subject
Track Collection
Disarm
Tear down of Infrastructure
Test Evaluation
: B+

Figure 9: *A single lifecycle of a* ***Human Penetration Test****. Showing process flow between different stages. Designed according to [43].*

Figure 9 shows a full *Human Penetration Test's* including its several stages:
First, the penetration tester provides items and configuration for this test instance:

1. An artifact template, included in a *Recipe*, which can be personalized.
2. Subject information, including subject specific information to create personalized artifacts.
3. Information about infrastructure used to present these artifacts in the form of *Infrastructure Elements*.
4. Parameters used to create a test schedule.

With this given information the ITS.APE framework creates subject specific artifacts. The provided infrastructure information is used to interact with framework external infrastructure via scripts. The *generate* script is executed during this stage.

Followed by this, the artifacts are deployed to the infrastructure. Therefore, the *deploy* script is executed.

According to a the specified parameters a test schedule is created by the framework. This schedule is used to control artifact presentation to its specific subject. A subjects test is activated by executing the *arm* script.

The Observation of a subject's reaction to an artifact is required for later test evaluation. During the subjects test period, *Tracks* are recorded, they store the subject's reaction to the presented artifact.

The disarm stage concludes a test period. A single subject's test period is finalized by executing the *disarm* script. If all test periods are finished the infrastructure can be torn down, therefore the *destroy* script is executed.

The framework provides a reporting functionality based on the observed reaction. Recorded *Tracks* are evaluated according to the scoring provided with the artifact template. A subject or subject group specific rating can be published.

Infrastructure Interaction

Most of the presented stages involve actions coupled to infrastructure for this HPT. This infrastructure is:

1. set up,
2. deployed with artifacts,
3. used to to present artifacts,
4. required to support recording of *Tracks*,
5. finally torn down.

It supports report of status information, as program flow can only continue if certain operations are finished. For example artifact deployment requires finished generation of its infrastructure.

The communication diagram, Figure 10, focuses on module communication involving the *Infrastructure Generator*.

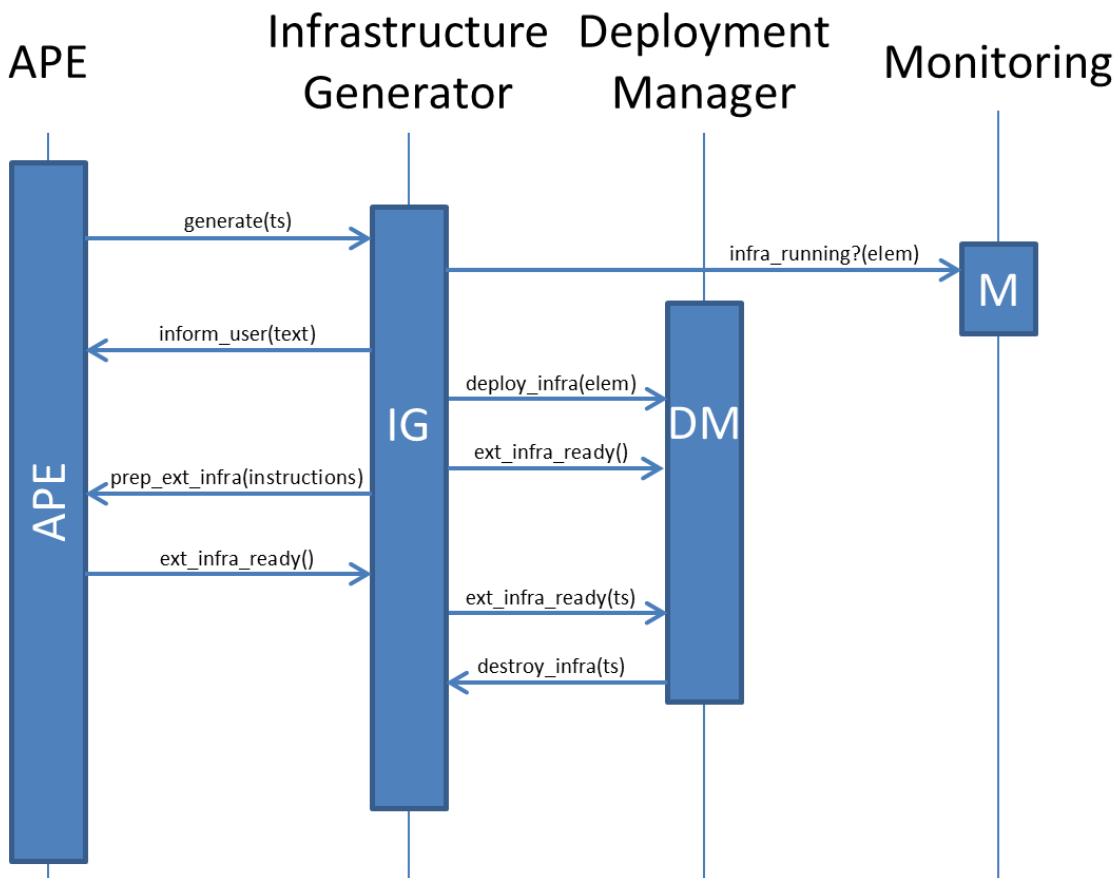


Figure 10: *Communication diagram focusing on the* ***Infrastructure Generator*** *module communication.*[43]

3 Design

The ITS.APE framework provides a tool for conducting *Human Penetration Tests (HPTs)*. HPTs have several stages which require interaction between the penetration tester and the framework. Providing suitable test artifacts, to quantify a subject's IT security awareness level, is an initial task.

The framework supports the penetration tester in his job, for example generation of subject specific artifacts based on provided templates and subject information is automated. Other preparations are laborious and can be repetitive: To artifacts present artifacts to a subject, infrastructure is required, for example a webserver. The framework provides a system to leverage script-based instructions for infrastructure generation and management.

This kind of generated infrastructure is artifact- and HPT-specific, in addition each penetration test instantiates its own infrastructure. It is unaware of already existing infrastructure, for example used by a similar HPT. Often artifact presentation involves redirecting of subject's traffic. With the current implementation this routing has to be established manually.

This master thesis provides a system for automated generation and consolidation of artifact infrastructure. It improves to the former described process of infrastructure generation and managing. Additionally, the penetration tester is no longer required to perform manual configuration to establish necessary routing of subjects. IP- and name-based routing of subject traffic is automated and requires no more manual interaction. This chapter presents the design of an extension used with the ITS.APE framework to achieve this.

3.1 Requirements Analysis

The following tasks have bee identified to achieve the former described challenges and conform to the goals set:

1. **Select Services (R1):** Identify typical IT services used within a company. Given these services derive infrastructure types and systems which are used to provide them. A justified selection of five services is required.
2. **Customizability (R2):** Having a set of services from the previous requirement, a suitable technology for providing necessary them must be selected. Furthermore, it is required to compile dependencies of the selected technology; providing a selec-

tion of software for said technologies. Different flavors of configurations (**R2.1**) of a service are possible too, they have to be easily selectable (**R2.2**). As the ITS.APE framework tests for IT security awareness, especially multiple configurations in regard to a service's confidentiality (**R2.3**) have to be possible.

3. **Suitable Technology (R3):** Furthermore, it is required to identify a suitable environment for rapid deployment (**R3.1**) of the selected software. A selected solution has to respect low resource (**R3.2**) and timeliness (**R3.3**) requirements. To prove the applicability of the identified deployment environment, it is required to set up and configure identified services using suitable software.
4. **Support Satellites (R4):** As observation of subject reactions is key for evaluating the subject's IT security awareness, actions performed shall be tracked by the ITS.APE satellite system. Infrastructure developed within this thesis support the use of said tracking system.
5. **Garbage Collection (R5):** To respect the limited resource requirements (see **R3.2**) all instantiated infrastructure has to provide a suitable clean up functionality (**R5.1**). Consequently, infrastructure that is no longer needed (**R5.2**), for example if all associated HPT are complete, shall be torn down.
6. **Compatibility (R6):** Software designed and implemented to interact with the ITS.APE framework must support already existing infrastructure deployment methods, such as script-based infrastructure, refer to 2.4.2.
7. **Respect Restrictions (R7):** Company policies or other regulations can prohibit framework generated infrastructure at all or allow only specific types of infrastructure. Additionally, because the desired software is integrated into a company's network, only provided IP addresses must be used. This prevents IP address conflicts caused by generated infrastructure. These restrictions must be take into account when infrastructure is generated.
8. **Redirecting System (R8):** Obvious use cases such as redirecting websites requests for artifact presentation shall be automated. This redirection has to respond to provided domains (**R8.1**) and IP addresses (**R8.2**). Therefore a special redirecting system has to be designed and implemented. It is required to chose suitable redirection technology as well as realizing software. Configuration of said software is required too. To meet the stated requirement for easy configuration (see **R2.2**) this system must be compatible to the ITS.APE scheduler (**R8.3**). Additional protocols might be added if results from **R2** present a specific use case. Only users participating of an IT security awareness penetration test shall be affected by traffic manipulation, the redirection system must only affect specified domains and IP addresses (**R8.4**).
9. **Infrastructure Status (R9):** Current recipes are provided with a monitoring script capable of checking whether artifacts are deployed correctly. While the way of generating infrastructure should be improved, change to the concept of artifact testing is not. Generated infrastructure shall only provide basic status information such as: *running, paused, not available*.
10. **Infrastructure White Box Testing (R10):** To determine the overall extensions functionality suitable test methods should be chosen and appropriate cases must

be designed. This white box testing method provides necessary evaluation of the realized system.

3.2 Virtualization Technique

Standard virtualization techniques, cf. section 2.3, enforce duplication of shared resources. Such duplicated libraries and applications result in higher resource requirements for standard virtualization. Additionally, each new virtual machine has its own full guest operating system that is started when a virtual machine is instantiated. Updating a virtual machine requires the same work as updating a standard operating system and with each update special care is required in terms of compatibility issues regarding installed applications. Benefits such as stronger separation between guest and host system are not needed in the ITS.APE framework. A penetration test requires only known software are executed on such a virtual machine setup. Already existing security measures, such as firewalls, separate ITS.APE components from outside systems and the Internet.

In contrast a container virtualization does not require a full guest operating system. With trading of strong separation between host and guest system, a container can leverage existing host operating systems functions and libraries. Nevertheless applications can bring their own libraries separate from the host system's libraries as part of the container. Docker provides fast deployment and allows for easy generation of *images* that can be used to instantiate containers. Via an API the Docker Engine can be managed and images, containers and their configuration can be customized [16]. *Docker-api* is an Ruby gem and it provides an object-oriented interface to the Docker Remote API [22]. This gem supports a broad spectrum of requests ranging from basic status information as the container's name or its IP address to managing functionality as container start, stop and command execution. These qualities meet the stated requirement of **Suitable Technology (R3)**, as Docker containers can be started in a rapid fashion while they are resource-efficient; libraries and other components can be used by multiple containers at the same time.

Furthermore, different basic images upon which a user can run its desired application already exist and are updated regularly [15]. Maintainers of these images take care of possible update issues regarding software bundled within such a container. The *ITS.APE Infrastructure Manager Extension* uses a basic container image for each of its specialized infrastructure containers. As described, Docker Containers can be easily adapted to the users needs. This meets the **Configurable (R2.1)** requirement.

3.3 ITS.APE Infrastructure Generator Extension

The current *Infrastructure Generator* of the ITS.APE framework supports only script-based generation or usage of infrastructure. Thus, every recipe, used to bundle information for creating one or multiple HPT artifacts, has to provide all necessary *Generate Scripts* for creating and setting up required infrastructure. Such scripts are cumbersome to handle while they allow for a high degree of freedom for the penetration tester. The script-based approach separates the framework from its used infrastructure. For example, already existing company infrastructure, infrastructure provided and set up by the penetration tester or generated from within the generation script itself can be accessed via such scripts. Therefore, infrastructure is transparent to the framework.

This freedom comes with the price of inhomogeneous generation instructions, no clear managing and maintaining interface and cumbersome testing. It is only possible to test generated infrastructure by executing a created recipe within the framework. This requires creation of a full testing scenario, including: a *Test Program* and at least one *Test Series* that requires a group file with subject data. Additionally, at least one *Test Season* has to be created that refers to a recipe including to be tested infrastructure elements. For this whole setup a *Schedule* is generated and actual presentation of artifacts is simulated. Only then log file entries handled by the satellite system are generated. The generated schedule is executed fully as tearing down generated infrastructure is only initiated if all *Test Episodes* are carried out. The whole testing process requires additional or manual interaction to react to presented artifacts, while a complete artifact live cycle is required to test only a single infrastructure instance.

Furthermore, direct interaction with instantiated infrastructure is only possible via scripts specially adapted to suit the generated infrastructure. Consequently, changes in a generation script might require changes in all subsequent managing scripts too. As stated in the requirements, **Infrastructure White Box Testing (R10)** shall be possible to lessen the burden of testing HPT infrastructure.

As implemented in the current ITS.APE framework, no explicit managing functionality for created or used infrastructure is present. There exist only *Status Scripts* as part of every *Infrastructure Element* defined in a recipe. These status scripts are used to execute checks on the current infrastructure status and rate them *running* or *not running*. To provide an improvement to the current system, status indicators have to be provided. This is formulated in requirement **Infrastructure Status (R9)**.

The *Infrastructure Manager Extension (IME)* module replaces the legacy *Infrastructure Generator (IG)*, it inherits all functionality from its parent class. IME provided infrastructure services are just a basic set of services and are not suited to cover special or extravagant HPT scenarios. Company environments can have strict policies that might not allow for bringing in new infrastructure. As script-based generation of infrastructure has advantages in these cases, IME still supports recipes and infrastructure generation based on scripts. IME is required to refer to managed containers upon request; artifact deployment operations or track collection are required to interact with artifact infrastructure. The combination of supporting a backward compatible interface, script-based generation and a way to address managed containers fulfill the requirement **Compatibility (R6)**.

The referring functionality needs requires information about configuration parameters, container status and all artifact processes using this specific infrastructure. Thereby the request for **Infrastructure Status (R9)** information is fulfilled.

IME provides a basic set of infrastructure types that can be used for HPT. IME supported infrastructure types are chosen according to IT services present in UKSH and protocols used by these services. The selection of infrastructure types is based on the requirement **Select Services (R1)**. Only types actively used in the UKSH IT environment are taken into consideration. From these infrastructure types two are used in the example recipe that is part of the ITS.APE framework: Mail sender and basic webserver. These two are realized as managed infrastructure to demonstrate the provided improvement and show compatibility to already existing artifacts. All selected infrastructure types are described in detail in the subsection 3.4, which provides justifications for their selection.

A penetration tester must be able to control which kind of infrastructure is created by the framework. As explained in section 2.4.2 concerning *External Infrastructure* and *Internal-manually-setup Infrastructure*, different kinds of a company's environments or policies request such a control mechanism. This control can be exercised by black- and whitelisting of infrastructure types; each time an infrastructure element is generated its type must be checked against these listings and handled accordingly. IME needs to manage all of its created infrastructure elements. Script-based generated infrastructure elements are compatible to existing framework parts and are not managed by IME. At time of creation of a requested infrastructure element IME needs to determine if requested infrastructure element type is a supported type and if configuration (black-/whitelisting) allows its creation. If not compliant to the configuration or if the specific type is not supported, an error must be raised.

Furthermore, generated infrastructure must respect IP address ranges provided by the penetration tester. All generated containers must not use IP addresses exceeding the provided range. These two regulations, black- and whitelisting and IP ranges, must be acknowledged to fulfill the requirement of **Respect Restrictions (R7)**.

A *Factory Method* concept pattern is depicted in Figure 11. Class instantiation is deferred to subclasses. An interface to create an object is defined, while the subclass can decide which class to instantiate.[20]

Followed by the initial checks a suitable container for this infrastructure type can be created. Container creation is handled by the *Container Factory* module introduced with this extension. This module realizes a *Factory Method* concept: Creation of an infrastructure type specific container is delegated to a specific helper class capable of fulfilling this demand.

Depending on the requested infrastructure type a specific *Container* instance is instantiated. This variability provides a suitable way to extend the IME by multiple specialized containers. New container types can be added by just creating a specialized container model and by adding this type to the factories selector.

All *Specific Containers* are subclass of an abstract *Container* class. Each *Container* supports methods for creation, start, stop and destruction; each subclass is required to implemented them. The container superclass defines different parameters for each managed container: A container name and the unique container id as generated by Docker Engine, an identifier of the Docker image used to instantiate this container

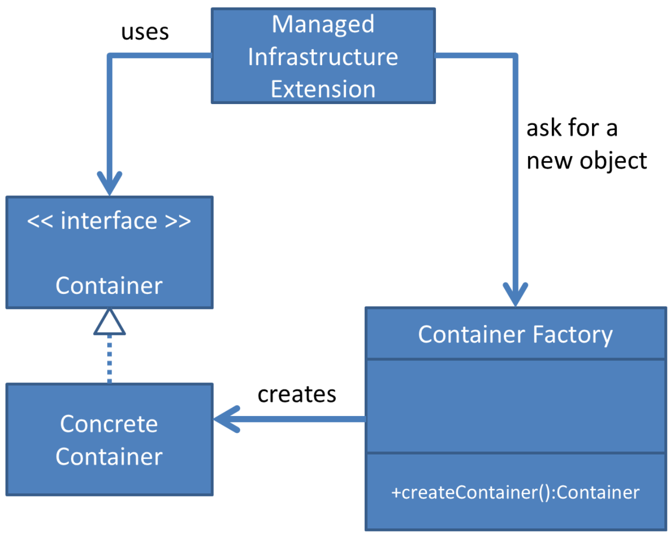


Figure 11: ***Factory Method*** *concept pattern to defer class initiation to a subclass. Figure designed according to [20].*

as well as the container's IP address. For bookkeeping purposes each container is equipped with a list of artifact processes that use its provided service. Nevertheless, each infrastructure type specific subclass can provide additional suitable methods and hold needed parameters in additional variables. Figure 12 shows a class diagram including the parent class used for each specific infrastructure container.

The combination of generation through a factory model as well as the use of a container model for specialized containers fulfills the requirement for **Customizability (R2)**.

Some infrastructure containers can be used by multiple artifact delivery operations. For example, a generic e-mail sending container can be used by all delivery processes that do not require special configuration of this e-mail sending container. Thus, if for example a special sending domain has is configured, a dedicated e-mail sender container must be instantiated. Further explanation, whether consolidation is possible for a specific container class is given in the following container specific sections. Creating a new instance of a container should only happen if no supporting container already exists or a certain limit of artifact processes linked to a single container is reached. Consequently, tearing down the specific container should happen if no more linked artifact processes are known to the IME's bookkeeping; it is triggered by the deletion of the last linked process. New artifact processes are added to the bookkeeping, while expired and thereby redundant processes are deleted. Bookkeeping and garbage collection are designed to provide a system that fulfills the requirement for low resource use and timeliness interaction as requested by **Suitable Technology (R3)**.

Testing of the IME module can be done via unit tests similar to module tests already used in the ITS.APE framework. These tests must cover stated restrictions to IP addresses and allowed infrastructure types. The containers status indication as well as available functions must be tested. This fulfills the requirement for **Infrastructure White Box**

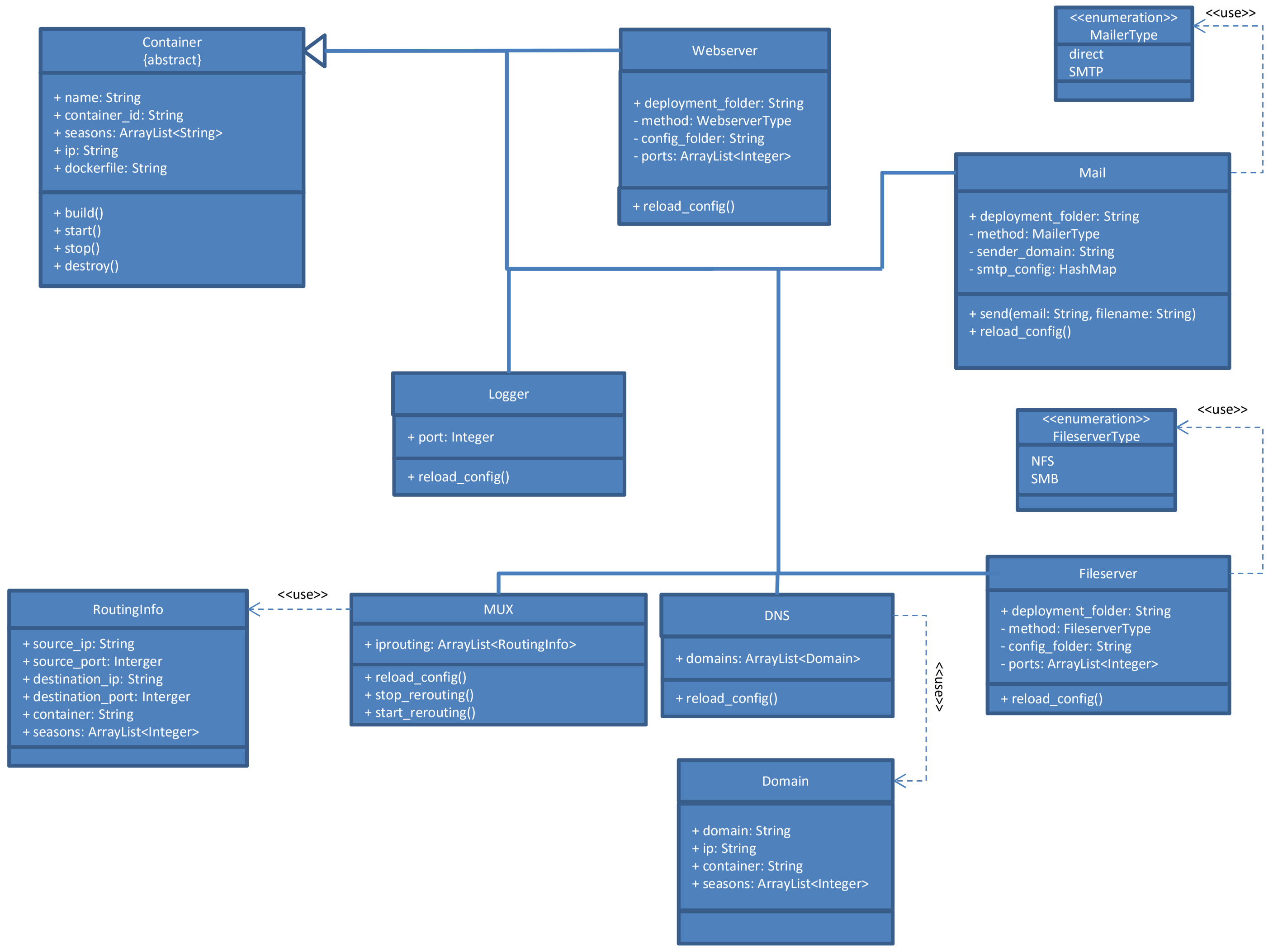


Figure 12: *Class diagram showing* ***Container*** *super class and its container specific children.*

Testing (R10) regarding the IME. Further necessary tests for each of the infrastructure containers are described in the following sections regarding each created container type.

3.4 Design of Provided Services

The *Container* class provides a basic container model that can be used to derive specialized containers. These containers can meet demands specific to their use case while compatible to all functions familiar with the parent *Container* class.

The selected services are all used at the UKSH and are common services of an office environment. A more detailed justification is given within each separate section here after. Thereby the requirement of **Selected Services (R1)** is fulfilled.

This section presents designed services and their associated specialized containers. Each specialized container is supported by a short discussion to justify its selection over all possible containers. Additionally, each subsection presents the container setup and how the parent container class is adapted. Different configuration options of each container are elaborate, as these fulfill the requirement for **Customizability (R2)**. At the end of each section an evaluation method and consolidation options and limits are presented. Evaluation of each container must be provided to satisfy the requirement for **Infrastructure White Box Testing (R10)**. Consolidation is further used to further reduce needed resources as multiple HPTs can use a single container. This goes in hand with the requirement for **low resource (R3.2)** use.

3.4.1 Mail Service

E-mail is a substantial communication method used in daily office routine. As discussed in the presentation of services (cf. Chapter 2.1) it is used at the UKSH. The ITS.APE framework provides a single recipe designed to create a phishing mail campaign. This recipe uses the script-based approach to create infrastructure, it uses this infrastructure to send a phishing mail to a test subject. The subject is lured onto a website designed to steal login credentials. For easy creation of such a regularly used HPT, e-mail sending is chosen as one of the realized services provided by IME.

Section 2.2.4 presented the necessary background information about e-mail communication services from the perspective of an e-mail sender. As described a phishing campaign consists of two parts: e-mail delivery and a webpage, therefore e-mail retrieving is left out for this scenario.

An e-mail service container class has to extend its parent container class by several parameters: sender domain name, deployment resources folder, e-mail delivery method and its correspondent optional SMTP server configuration.

Every e-mail message has to provide a sender e-mail address (*From:*), this field is part of the e-mail message itself and can be chosen at will. As described in section 2.2.4 an e-mail is sent from a MSA, the sending host's domain is recognizable by the receiving server. The sender provided from the e-mail message header (*From:*) is displayed in e-mail clients (e.g. Microsoft Outlook). This sender can differ from the sending host domain; both can be inspected if the plain e-mail message received including all its headers is checked. The created e-mail container has to support configuration of this sending host domain. It is stored with each e-mail container instance.

Created e-mail messages are generated by the *Artifact Generator*, whereas the *Deployment Manager* is used to provide these artifacts to the infrastructure. The e-mail container class provides a location for this deployment. The provided folder's location is stored with each instance. This separates artifacts and allows for identical naming of artifact files.

Two kinds of delivery methods are supported by the e-mail container. First, the e-mail container provides a direct sending service, as such it will act as a full service agent combining MSA and MTA. Thereby, no additional configuration is necessary as the e-mail container handles all sending functionality. Second, the e-mail container provides a MSA which will only forward the e-mail message to a configured MTA. Necessary configuration including domain, port, username and password have to be provided and are stored with each e-mail container object. Each object shall only support one type of delivery method.

The e-mail container provides a function to deliver a single e-mail to a single recipient. This function needs the recipient's e-mail address and a location of the prepared e-mail message. The message is expected in the *Internet Message Format* including at least *From:*, *To:* and *Subject:* as defined in [34]. A bulk sending method is not required as the ITS.APE framework only supports single artifact presentation.

As the e-mail container can be configured to leverage an MTA, certain configuration can be required. This configuration is stored in the infrastructure element as part of the recipe.

Consolidating an *e-mail service* container is possible but limited by some factors. A limiting factor is given by the *sender domain name* as this is fixed at the start of the container. Only infrastructure elements with no specific or the same *sender domain name* setting are suited to use the same *e-mail service* container. Additionally, if the container is configured to act as MSA, only infrastructure elements with the same configuration shall use this container.

Each sent e-mail has to generate a log entry which can be evaluated by the Satellite system of the ITS.APE framework. This information can be used for evaluation purpose of the associated HPT and satisfies the requirement for **Support Satellites (R4)** for the *e-mail service* container.

Evaluating a described *Mail Service*, is done by sending an e-mail to a known account. Received e-mails on this account are checked and the send e-mail is expected to be delivered in a short time frame. Received e-mail candidates are check to identify if the e-mail's *sender*, *subject* and *content* matches the send message and specified sender. This can be done by stripping the added communication header of the received e-mail and using the UNIX*diff utility* to display differences. The e-mails body has to match, while the received e-mails header must include the correct subject and sender address.

3.4.2 File Sharing Service

File sharing is commonly used in a cooperate environment. Through network shares documents and other files can be shared seamlessly and are reachable within the user's operating system. The UKSH uses Microsoft Windows as their main client operating system. File sharing in the UKSH network is realized via the Windows supported SMB protocol. Microsoft Windows is the most common client operating system [59]. Additionally it supports SMB within all of its versions and without the need for extra software, cf. section 2.2.3. The SMB equivalent for UNIX systems is NFS, which is disregarded to focus on the UKSH use case. SMB and CIFS are explained in the background chapter 2.2.3.

In general, mounted file shares' content can be easily modified, while server side modifications are directly presented to the subject via its file browser. This suites such file shares for a HPT as no changes at the subject's computer system have to be performed. Nevertheless a subject can be presented different kind of artifacts such as potentially harmful executables or ransomware specific files (e.g. *.locky*) [36].

The *File Sharing Service* container extends the parent *Container* class by necessary configuration options. An instance has to store its artifact deployment folder location. This location is used by the *Deployment Manager* to place artifacts generated by the *Artifact Generator*, which can be presented to the subject.

Similar to the deployment folder each instance stores a configuration folder location. At this location predefined configuration files are expected. These files store access parameters for files and folders present in the deployment folder location. As this access configuration is directly related to the selected HPT it can be stored as part of the HPT recipe. If no special configuration is provided full access to the deployment folder is granted to a single ITS.APE account. This basic configuration of the *File Sharing Service* provides a simple resource location accessible within the company's network. This resource location can be used if a HPT requires executable or files available to network clients. An example would be an ITS.APE executable, used to present artifacts, that is provisioned to a subject's computer. Provisioning tools can be pointed to the *File Sharing Service* container location as a resource of this executable.

Future implementation might support additional protocols. Therefore, each *File Sharing Service* stores its specific file sharing protocol in an instance variable.

Consolidating a *File Sharing Service* container is possible but some restrictions have to be taken into account. If a basic configuration is used multiple HPTs can use a single *File Sharing Service* container. HPT-specific assets are placed in subdirectories within the deployment folder to avoid possible naming collisions. No consolidation is possible if a HPT specific configuration is chosen, as two handmade configurations are likely to cause collisions. For example identical usernames used within two HPT the system cannot have different password or separate access regulations.

Logging of access to a file share is based on IP address of the requesting client. This distinct logging is required, indicating single file access as well as folder listings. Respective log files are made available to the satellite system. This satisfies the requirement for **Support Satellites (R4)** for the *File Sharing Service* container.

Evaluating a *File Sharing Service* as described can be done by connecting to said service, storing a file and requesting said file again. If the sent file matches the recieved file the *File Sharing Service* is working as intended.

3.4.3 Website Service

Besides information resources, for example *Wikipedia* or similar company internal systems, productively used applications are realized via a website-based user interface. Web standards, that ensure compatibility with different operating systems via web browsers, pave the way for their success. Additionally, changes to such an application are easily made, as only server side changes are necessary. These changes are directly reflected at the users interface or the applications workflow. There is no need for client side updates, installation procedures or manual interaction.

Besides less cumbersome updates, such applications can be reached world-wide if made accessible from the Internet. While world-wide available, they are prune to attacks from outside. Authentication is necessary and often realized via a username and password authentication. Therefore, phishing attacks are often designed to steal such user credentials. Within the ITS.APE framework a phishing campaign directed to test subjects for their IT security awareness level towards these attacks is provided. These attacks require a webserver to present a mocked phishing website to the tested subjects.

Section 2.2.5 provides necessary background information about HTTP that is used to communicate between webserver and browser.

The webserver container has to extend the parent container class by a location that holds the website data. This deployment folder location is stored with each webserver container instance. Furthermore, a configuration folder location is part of each instance. This folder should be used to provide configuration assets such as certificates used for TLS communication (cf. 2.2.6). Additionally, a server configuration can be adapted to fit a specific HPT, it is part of this HPT's recipe. If no specific configuration is given within the configuration folder a basic configuration provided by the IME is used.

Modern websites make use of dynamic content generation based on parameters provided via the website request. Commonly used are script languages such as PHP and Ruby, it is required to support them.

In general for HTTP communication port 80 is used, a IP based request via port 80 allows only a single website available through a webserver container. The requirement **low resource (R3.2)** use raises the need for consolidation of realized services, if possible. To allow consolidation, additional ports besides port 80 can be used to provide websites. Thereby a single container can provide multiple websites.

As presented in section 2.2.6, websites can be accessed through a secure channel provided by TLS. Webserver supporting such a secure connection are configured differently, as the described basic webserver. They require additional assets, for example certificates and keys; these have to be provided by the penetration tester.

Without compromising the requested URL via *Server Name Indication*, no two HTTPS connections can be realized using the same port [62]. Therefore, again multiple HTTPS requiring recipes can only be run within the same container if multiple ports are used. Each container therefore holds a list of used ports, type of connection (HTTP or HTTPS) and their associated HPT.

Each website access has to generate a log entry which can be evaluated by the satellite system of the ITS.APE framework. This information can be used for evaluation purpose of the associated HPT and satisfies the requirement for **Support Satellites (R4)** for the *Website Service* container.

A webserver container can be evaluated by checking the webserver's status code answer to a request. In addition the downloaded website can be matched against the provided one. This can be realized by comparing the checksum of the original and the received website, it is expected to be identical.

3.4.4 Redirect System

Redirection allows for exact artifact delivery, presentation and monitoring of participating subjects and is required by requirement **Redirecting System (R8)**. The ITS.APE framework is integrated into the company's network; as such it can interact with all connected devices and users. The requirements state: only users participating of an IT security awareness penetration test shall be affected by traffic manipulation **R8.4**. Figure 2 shows the integration of the ITS.APE framework into a company's network infrastructure.

All presented services are based on the Internet Protocol (IP), this allows for a uniform handling of redirection. A subject's IP address is provided by the *Subject Tracker* module of the framework and is necessary as stated in requirement **R8.2** Redirecting can be done by targeting only traffic originating from a subject's associated IP address. All other traffic is thereby not affected. As described in section 2.2, specific services, such as website requests, are handled via defined standard ports. This allows for a service-oriented redirecting of requests.

Artifact delivery is initiated by the *scheduler* module and realized by the *Delivery Manager* module of the framework. The *Delivery Manager* is used to perform final steps to present artifacts to subjects. Redirecting a subject's traffic to present a prepared phishing website is one possible action. A modification of the *Delivery Manager* to inform the internally managed redirecting system of necessary adaption fulfills the requirement of **R8.3**. This notification has to include enough information to identify the test specific internal infrastructure. Combined with the provided subject's IP address, this allows identifying traffic that must be redirected for artifact presentation. Therefore, requirement **R8.4** can be fulfilled.

Besides IP, DNS provides convenient naming of systems and can be used to create IT security relevant artifacts and is required by **R8.1** [53]. Subject's DNS requests can be identified by the described IP and port based redirecting system. Such requests can be answered if a subject's test episode is scheduled and a DNS specific artifact is part of this test. Such an artifact has to provide a domain name; this domain name is prepared to be redirected. If a subject's test episode is due the IP redirecting system has to redirect the subject's DNS requests to the internal DNS service. Only the domain specified by the artifact is redirected to a designated test infrastructure, all other domain requests are forwarded to the company's DNS service. Such a system fulfills the requirement for specific domain redirection as described by **R8.1** and **R8.4**.

Logging of redirected requests can be used to determine if a subject has reacted to a presented artifact. For example it can be observed if a subject requested a phishing website. Nevertheless, all redirected requests are forwarded to an infrastructure serving artifacts. Each of these services, for example a webserver or a file sharing service, have to fulfill the requirement **Support Satellite (R4)** to support the *ITS.APE Satellite System*. No additional information, relevant for the evaluation of a HPT, is acquired by logging of requests at the redirecting system. The redirect system has to provide a fast and reliable service for all users and may only forward subject's request at the time of testing. Therefore, no logging capability is designed within the redirecting system.

The presented redirection system can be evaluated by using several dedicated machines. These have to act as participating subject machine, not-participating user machine and a machine realizing the presented domain and IP redirecting infrastructure systems. A mocked infrastructure can be used to simulate a managed infrastructure used for artifact presentation. It is required to shown that all desired requests originating from the participating subject machine are redirected. HPT relevant requests must be redirected to the designated infrastructure used for artifact presentation. Furthermore, it has to be shown that only these requests are redirected. A designated not-participating user machine can be used to test if a request for configured domain is answered correctly. This user machine must receive a different IP address, as only participating subject machines should receive the artifact infrastructure machines IP address.

4 Implementation of ITS.APE Infrastructure Management Extension (IME)

From the presented design the *Infrastructure Management Extension (IME)* is implemented. It orchestrates generation, configuration and live-time handling of Docker containers to provide infrastructure to HPTs.

Generation requests based on *Infrastructure Elements* as provided by *Recipes* are used to determine suitable container images and configuration. A bookkeeping functionality within the IME allows observing each container status and controlling it during a human penetration test. To save resources a designed naming scheme for infrastructure requirement descriptions is used. This allows to unite multiple compatible *Infrastructure Elements* in a single container.

To provide an easy and seamless integration of extensions for the ITS.APE framework, a replacement of original modules with extension modules is used. This replacement is suited best at an early stage during the framework start up process. Therefore, such extended modules are instantiated at the *APE* modules initialization.

The developed extension consists of adaption of the mentioned ITS.APE modules and a complete new module. This module is the main part of the IME as it orchestrates all container interaction. In addition to necessary changes to existing framework models, new supporting models are defined as well. These allow for routing subject's traffic to created infrastructure to present desired artifacts.

This chapter opens with the presentation of the new container type-based management and configuration options provided by the IME. Follow by a detailed explanation of created models, modules and adaptions to legacy framework components. It closes with a description of a provided webserver and *Mail Service* infrastructure supported by the IME.

4.1 Type-based Naming Scheme

In section 3.3 a strict service oriented and class-based design is presented. During the implementation phase limitations of this design became apparent. Determined container classes are relevant but limit a penetration tester to prepare infrastructure-automated HPT including only these container classes. Different and new container classes could be added by providing new class models and including these models into the IME module.

Several changes in the existing codebase and a sophisticated knowledge of the provided extension code would be necessary. Therefore, the class-based container management is replaced by a more general container idea. In particular, no more changes to existing extension code are necessary to use any container class.

A *Human Penetration Test* can be performed using different services, as presented in section 2.1. These services are available in the UKSH, but future tests might require different infrastructure. A class-based system requires changes to the IME source code to add new infrastructure classes. A more general approach with no need for source code manipulation is based on a configuration-based system. Such a system allows for an easy adaption to new infrastructure requirements by changes just in a text based configuration file.

Within the requirements analysis, cf. 3.1, the requirement for low resource consumption (**R3.2**) is identified. Furthermore, similar infrastructure requests should be handled by a single infrastructure service if possible. Variable configurations cannot be used to determine similar but only identical infrastructure description. To overcome this limitation an additional parameter is introduced. This parameter is a descriptive *type* of an infrastructure requirement. It includes an infrastructure's base type, desired software to realize the service and certain add-ons. Service requests can be evaluated against such an infrastructure's *type* to determine if they can be consolidated. Additionally, this *type* is used to select a matching predefined container image. Both processes are described in detail in the following sections. An example for provided infrastructure *type* is given in Figure 13.

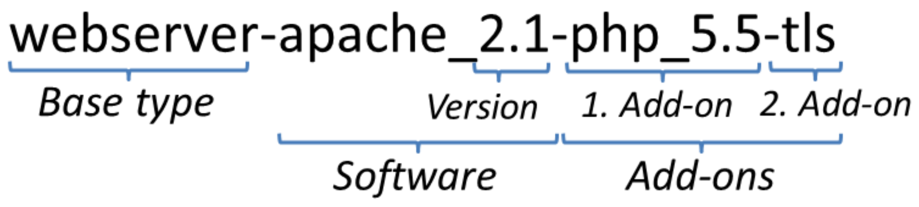
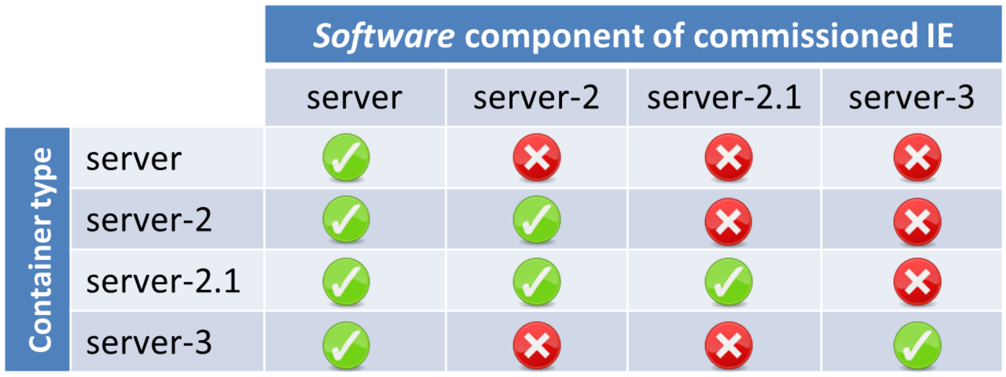


Figure 13: *Showing an infrastructure* ***type*** *conform to the presented type-based naming scheme. It describes an infrastructure's base type, a realizing software and possible add-ons.*

The base type describes a required service, as identified in section 2.1. These services are identified in the UKSH and might be extended by additional services for future test. A typical example for a base type is *webserver*. Only infrastructure requests with an identical base type can be consolidated and serviced by a single managed infrastructure. Identical infrastructure base types are capable of presenting the same kind of artifacts, for example a website. A base type describes the basic concept of artifact presentation and must be provided.

A more precise description of an artifact presentation can be given by providing a concrete *software* used. As an example, a basic type *fileserver* might be extended by this additional parameter to specify *Samba* as desired software. In this particular example the selected server software specifies the protocol used for this kind of service. For a webserver this might only present a preference for a certain software, for example *Apache* or *Ngnix*. Therefore a penetration tester might provide a *software* but this parameter is optional. Such software products are available in certain versions that might not be compatible to each other. To reflect this within the *software* parameter, a software's version can be specified. A version number is added to the software name string, for example *apache-2.2*. For consolidation purposes the existing containers are checked to provide at least the stated version. This version check is realized via a longest match in all *software* strings of the same basic type. A match is returned if the software version of the newly requested service is a sub-string of an existing service. Thereby it is ensured that only software within the same release version or newer is consolidated. The *software* parameter is optional and might be provided if needed, it can be further specified by a version.

As presented in section 2.1, infrastructure services can have a set of different features such as TLS or PHP support. A single feature or a set of features can be described by the *add-on* parameter. For example a dynamic webserver is desired; to support this *PHP-5.6* is added as a add-on parameter. This parameter is checked subsequent to the basic type and *software* parameter evaluation. Therefore only services matching basic type and *software* are evaluated. Similar to the *software* parameter a version can be added to each *add-on* parameter. The described sub-string matching is performed alike for each parameter. Only if all specified *add-ons* can be matched with a selected service, this service is regarded as valid candidate. Consequently, services with at least the requested *add-ons* are a full match. Nevertheless, they might provide additional *add-ons* which their originating request required. The *add-on* parameter is optional, it can consist of a single or multiple add-ons; each parameter can be further specified with a version.



		***Software* component of commissioned IE**			
		server	server-2	server-2.1	server-3
Container type	server	✓	✗	✗	✗
	server-2	✓	✓	✗	✗
	server-2.1	✓	✓	✓	✗
	server-3	✓	✗	✗	✓

Table 1: *Table showing results of a consolidation analysis based on the* ***type*** *parameter of newly commissioned* ***Infrastructure Elements*** *and already existing managed infrastructure* ***containers****.*

Table 1 shows the result of such a matching. A commissioned *Infrastructure Element* is matched to existing managed infrastructure. The example is reduced to show only the *software* component of IE and containers. It is assumed that their base type is identical and no *add-on* parameters are provided.

There can be four matching groups, the *software* component can be

1. identical,
2. define a simple similar version,
3. a similiar but more specific version or
4. a different version.

Depending on the provided IE a consolidation between an existing container and the commissioned can be done.

The machting of add-ons is done in a similar fashion. For matching add-ons, each requested add-on of the IE's type is checked against its counterpart of an existing container. This is only done if said counterpart exists, otherwise the current container is removed from the list of possible candidates.

4.2 Configuration of the Infrastructure Management Extension

This section introduces the configuration of the IME and its containers. As required, provided extensions have to be compatible to the existing framework **R6**. Furthermore, they have to support different configurations which must be easily selectable, too (**R2** & **R7**).

To provide a configurable control, whether IME should be used, the ITS.APE configuration is extended. This configuration is stored in a human readable JSON file which can be edited via a simple text editor. An infrastructure category includes an IME subcategory used to exclusively configure the *Infrastructure Management Extension*. In this subcategory the penetration tester can define infrastructure types to be handled by the IME. This listing represents a whitelisting of infrastructure types that IME is allowed to generate and manage. Each time the infrastructure generation process is initiated by the APE module, a commissioned *Infrastructure Element's* type is checked. If its type matches a whitelisted type, it is marked as an internally managed *Infrastructure Element*. A non-script-based generation is only initiated if the *Infrastructure Element's* type matches to at least one whitelisted type.

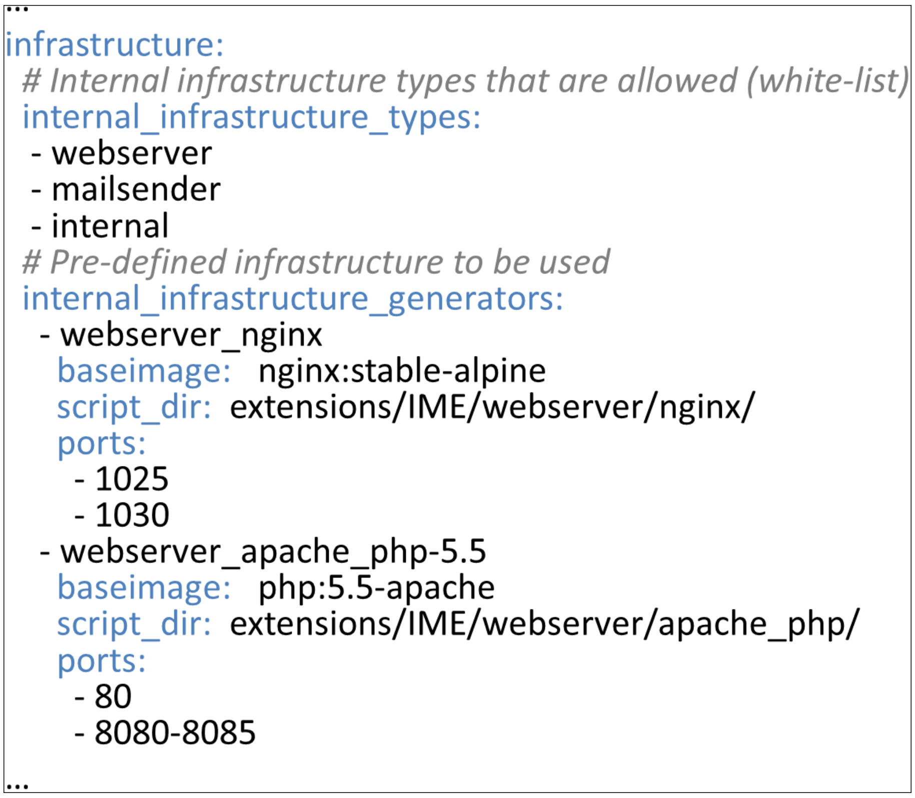
Furthermore, the configuration file *Infrastructure* category is extended by an *External Infrastructure* category. This listing represents a blacklisting of infrastructure types that must not be generated by IME. The penetration tester can use this blacklisting configuration to ensure compliance to the company's policies. Again, before each infrastructure generation IME checks whether an *Infrastructure Element's* type generation is prohibited. If so, the *Infrastructure Element's External Infrastructure* instructions have are presented to the penetration tester (refer to section 2.4.2). These instructions are part of the *Generate Script* that is executed accordingly.

The realized implementation of containers used by IME require a configuration of *predefined containers*. These container descriptions are stored within the framework configuration file, more precise within the *Infrastructure* category under the descriptor *Internal Infrastructure Generators*. Each *generator* consists of a *name*, chosen according to the later presented naming-scheme, the Docker *image*, a *script path* and a list of ports. The folder provided in the *script path* variable has to include all infrastructure-specific scripts. Especially relevant are *arm*, *disarm*, *deploy*, *undeploy* scripts as these are used to control interaction between the container and generated artifacts.

Figure 14 shows an valid configuration for the IME. It is an excerpt of the full ITS.APE configuration file focusing only on the relevant sections used by the IME's models and modules. The configured whitelisting three allows for three managed *base types*: *mailsender*, *webserver* and *interal*. There are two predefined infrastructure containers which can be used to provide managed infrastructure. Each container has additional parameters set, these specify the container's *baseimage*, location of the infrastructure specific scripts and ports designated for this container's service. This configuration uses two ways to state such ports:

1. State a single port or
2. define a port range.

The IME tries to use all stated ports within the port range including start and end port. Unavailable ports, due to missing authorization, and already used ports are recognized by the IME and not used when creating a requested managed infrastructure.



```
...
infrastructure:
  # Internal infrastructure types that are allowed (white-list)
  internal_infrastructure_types:
   - webserver
   - mailsender
   - internal
  # Pre-defined infrastructure to be used
  internal_infrastructure_generators:
    - webserver_nginx
      baseimage:   nginx:stable-alpine
      script_dir:  extensions/IME/webserver/nginx/
      ports:
        - 1025
        - 1030
    - webserver_apache_php-5.5
      baseimage:   php:5.5-apache
      script_dir:  extensions/IME/webserver/apache_php/
      ports:
        - 80
        - 8080-8085
...
```

Figure 14: *An excerpt of the ITS.APE configuration. Showing the IME relevant settings including with whitelisted infrastructure types and two predefined managed infrastructures with their parameters.*

4.3 Internal Infrastructure Manager Module (IIM)

The *Infrastructure Management Extension* relies heavily on bookkeeping and decision making based on the commissioned *Infrastructure Element*. This section describes the *Internal Infrastructure Manager* module which manages the internally managed infrastructure. As mentioned, a naming scheme for infrastructure types is used to determine whether multiple *Infrastructure Element* requests can be consolidated and served by a single container. This new naming scheme is presented in section 4.1.

Different to the legacy *Infrastructure Generator* module, the IIM module does not execute scripts to generate infrastructure but provides a management utility for *ITSAPE Containers*. This newly created container model is presented in section 4.4.

To provide said management functionality a single IIM instance is initialized during the starting process of the framework. This instance is created in the *APE* module, the main class of the framework. Further management is handled by this single instance, as this instance holds information about all managed infrastructure. Said IIM instance requires instantiation with a *Config Manager* module object and a *Subject Tracker* module object. The *Config Manager* provides access to necessary information defined in *Recipies*. For routing purposes it is necessary to gain knowledge over a subject's IP address, this functionality is provided by the *Subject Tracker*.

4.3.1 Properties

For bookkeeping two hash maps and a single array are used. The single array stores all configured predefined containers; they are provided via the ITS.APE configuration. One hash map is used to store generated infrastructure containers. Each key within this hash map represents a type of container as described in section 4.1. With each key an array of type specific containers is stored and can be accessed from different model methods. The described hash map is depiced in Figure 15.

The second hash map is used to store active routing information. Artifacts are brought out onto generated infrastructure and made accessible to subjects. This presentation of artifacts is precise, as only participating subjects must be able to access artifacts. Additionally, access must only be possible if a subject's personal HPT is scheduled. The presented IIM supports multiple HPT serviced by a single managed infrastructure. Routing is further used to ensure presentation of personalized artifacts to their respective subjects.

At initialization of the IIM instance all predefined containers are loaded by the `load_predefined_containers()` method. The configuration provided by the *Config Manager* is used to create these prepared containers. Each container is created with its container image name, configured scripts and ports. Container images can be available in the local image repository or are pulled from Docker Hub automatically.

The preparation allows checking if the defined container image is available and prepares a container object for later use. An error is raised during this process if neither source can provide the requested image. Thereby, a missing container image cannot cause failure during a running HPT.

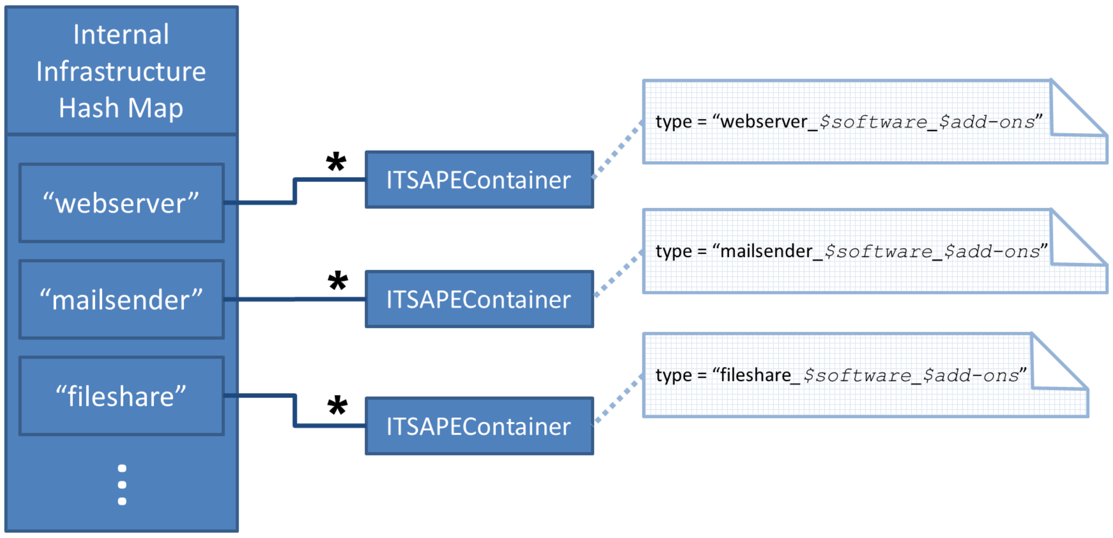

Internal Infrastructure Hash Map
"webserver"
"mailsender"
"fileshare"
*
ITSAPEContainer
type = "webserver_$software_$add-ons"
type = "mailsender_$software_$add-ons"
type = "fileshare_$software_$add-ons"

Figure 15: *Internal Infrastructure Hash Map as used by the IME. Each key is used to reference containers of a specific base type.*

4.3.2 Methods

The IIM provides methods to create, destroy, arm and disarm its internally managed infrastructure. Besides *Human Penetration Test* specific infrastructure it controls special internal infrastructure used for redirecting. The following presents the different methods and their provided function.

Generate Infrastructure

Each HPT requires certain infrastructure to perform the testing. If said infrastructure requirements are compatible with the provided IME configuration, such an infrastructure can be managed by IME. The `generate(elem)` method of the IIM requires an *Infrastructure Element* provided as parameter upon calling it. All IEs from a single *Recipe* are identical, multiple HPTs can use a single *Recipe*. Consequently, identical IEs can be commissioned for generation. While their provided service is identical, each of these *Infrastructure Elements* must only present artifacts to its associated subjects. Furthermore, each IE provides *traces* via generated log entries or similar methods to allow for later evaluation purposes. Making each IE instance used for a specific *Recipe* is therefore essential. Therefore, a provided IE is given a unique *Managed ID*:

```
elem.managed_id = SecureRandom.uuid.
```

Each *Infrastructure Element* has a *type*. This parameter is used during several decision processes within the IIM. At start up, when no containers are running, this parameter is used to check against predefined containers provided by the IME configuration. The `matching_predefined_containers` method is used. This matching is performed according to the naming scheme introduced with this extension and is explained in detail in section 4.1. The matching results in an array of suitable containers. Additional parameters might be respected in a future version of the IME as described in chapter 6. A suitable predefined container is chosen and its *build* method is called providing the IE.

Henceforth commissioned containers are not build if consolidation with an existing container is possible. This addresses especially the requirement for low resource **(R3.2)** consumption. For new commissioned *Infrastructure Elements*, all running containers are checked to decide whether or not an already running container is suited for consolidation. This is realized via the `check_for_matching_container(containers, elem)` method, it is provided with all running containers and the commissioned *Infrastructure Element*. Again this check is based on the introduced naming scheme.

As last action following a container build up or consolidation, the newly managed infrastructure is added to the bookkeeping hash map using its specific type:

```
@internal_infrastructure_hashmap[container_type] « chosen_container.
```

Additionally, the *managed* parameter of the now serviced IE is changed to *true*.

Arm Infrastructure

As depicted in Figure 9 and described in section 2.4.2, artifacts must be presented to their specific subject. IIM provides an *arm* method to start artifact presentation for a provided *Test Episode* and its respective *Test Season*: `arm(episode, season)`. Given this information all *Subject* instances and the corresponding containers and are determined. Via the *Subject* instance the framework *Subject Tracker* can provide the subject's IP address. The IP address is used for routing the subject's requests to the test infrastructure. Additionally, external ports are taken from the IE parameters and internal ports are taken from the respective container's bookkeeping. The internal port is chosen by the *ITS Container* model during its generation process, this process is explained in section 4.4.2. With this information a *Routing Information* object is created; it requires all previously stated information:

```
RoutingInfo.new(recv_ip, recv_port, src_ip, src_port).
```

This object handles creation of routing rules to redirect a scheduled subject to the HPT infrastructure.

Followed by this preparation the respective *ITSAPE Container* is started via its *start* method. Subsequent the *Routing Information* is used to establish necessary routing by calling its *arm* method. In a final step the IIM's *Routing Information* hash map is updated to include the now effective routing:

```
@routing_rules[episode.id] = routing_infos.
```

Disarm Infrastructure

Depending on the schedule artifact, presentation can be stopped; this is done by *disarming* the said artifacts. The IIM provides a designated *disarm* method to fulfill the complement of the presented *arm* method. Provided with a *Test Episode* and its respective *Test Season*, necessary information is provided to determine the corresponding container and the redirected subject. The associated *Routing Information* object can be determined by the provided *Test Episode* id. This allows to remove the created and active routing rule via a the *disarm* method provided by the *Routing Information* class. An additional *Subject Tracker* request is not performed, as it might provide a more resent response in conflict to the established routing rule. As a last step stored the hash map of *Routing Information* is updated to reflect the removal of the no longer active rule:

```
@routing_rules.delete(episode.id).
```

Destroy Infrastructure

In general, if a HPT is finished, set up infrastructure must be torn down to respect the requirement for low resource **(R3.2)** consumption. As multiple IE, originating from different *Human Penetration Tests*, can be consolidated and serviced by a single container, management is necessary. Calling the *destroy* method of the IIM providing an IE from said finished HPT results in a forwarded *destroy* call to the IE servicing container. To perform said forwarding, a lookup for the servicing container is performed, utilizing the *managed id* stored in the IE. If any other IE is still using provided service the container is not torn down. Consequently, only the bookkeeping hash map is updated, removing the IE to container association:

```
@internal_infrastructure_hashmap[type].delete(container).
```

Type Matching Based on Naming Scheme

The IIM provides methods to perform name matching according to the naming scheme. These evaluation methods are similar for predefined and existing containers, both require an array of *Infrastructure Elements* and a single *Infrastructure Element* as input.

Providing existing containers and checking against a requested infrastructure service results in a list of compatible containers this check starts with the `check_for_matching_container(infrastructure_array, elem)` method. The initial list of containers is trimmed down to only include containers with the same basic type, follows by identical software. The software's version is checked to match according to the beginning substring rule explained in section 4.1. This second check is performed by the `software_check(container_software, ie_software)`. Resulting containers are further checked to provide requested add-ons and add-on versions via the `addon_check(container_addons, ie_addons)` method.

If no corresponding container is found a new container instance is spawned. This uses predefined containers provided via the configuration and a single *Infrastructure*

Element. To select a suited container from all available predefined containers the single *Infrastructure Element's* type is checked against each predefined container's type. The process is performed similar to checking for existing containers.

4.4 ITSAPE Container Model

This section describes the use of Docker containers to provide managed infrastructure by the IME. The implemented IME uses the Docker API which provides needed features to configure, start, stop and manage Docker containers. A Ruby Gem is used to instantiate Docker container objects [22].

While Docker container specific information can be requested from such an object, it is necessary to store ITS.APE relevant information too. The *ITSAPEContainer* model is therefore used to represent a container instance to the IME. This model is similar to the designed *Container* model presented in Figure 12.

In contrast to the idea of separate class-based containers, the actual implementation uses a single basic container model to realize different container types. The model stores all container related information as well as the actual Docker Gem container object.

4.4.1 Properties

An instantiated *ITSAPEContainer* stores different information used during its lifetime. The Docker Gem container object can only manipulated by *ITSAPEContainer* internal functions. This is done to have a single point of interaction with outside functionality provided by Docker.

An exchange of the used Docker Gem therefore can easily be done by modifying only the *ITSAPEContainer* module. Furthermore, adaptions to changes introduced by an updated Docker Gem can be realized in an easy manner, too.

For convenience, other Docker management tools besides the framework can be used to observe Docker containers. Therefore, each container instance stores its Docker container id. This allows identification of a used container via the Docker CLI and mentioned tools.

As explained in section 3.2, a Docker container is based on a Docker image generated by a Dockerfile or pulled from Docker Hub. The specific container image name is stored with each *ITSAPEContainer* instance and used to generate the actual Docker container. As explained in section 4.2, each predefined container stated in the framework configuration provides information about its Docker image. Given this configuration IME matches an *Infrastructure Element's* type to the Docker image used to provide requested infrastructure. The used Docker image name is stored within the *ITSAPEContainer* data model.

All managed *Infrastructure Elements* are stored within their respective *ITSAPEContainer* instance. Information provided by an IE allow for requests to legacy framework parts.

Possible request include:

1. *Subjects*, served by this specific IE,
2. *Test Episodes*, scheduled to deliver *Artifacts* via this IE and
3. *Test Seasons*, holding the *Recipe* including the IE's template itself.

To support assignment of multiple *Infrastructure Elements* to a single Docker container instance, several preliminaries have to be fulfilled. The configuration provides a list of ports, these can be include single ports or a range of ports that are used for the specified type of *ITSAPEContainer*. This information is stored with each container object and used when an *Infrastructure Element* is assigned to this container. With each assignment a port from the provided ports is chosen and assigned to serve the newly added IE. A list of already assigned ports is updated and used to ensure individual allocation of each available port. Additionally, it is checked on a file system basis whether the chosen port is available or already assigned to a process. If no freely available port could from the defined ones can be chosen an descriptive error is risen.

Multiple IE can require the same port reachable for the subjects. For example, a HPT can have two webservers which serve different websites to a subject. Each webserver requires accessibility from the outside via port 80, as this is the standard HTTP port. The redirecting system of the IME is used to support this. The consolidation functionality of the IME and *ITSAPEContainer* is used to provide two webservers by a single Docker container. Each webserver is assigned its own internal port chosen from the predefined port list. Thereby, each *Infrastructure Element* can be addressed specifically by the redirecting part of the IME.

A Docker containers IP address is stored with the associated *ITSAPEContainer* instance. Again, this is used by the redirecting system of the IME. Additionally, each container instance provides information about its realized *Infrastructure Element* type and all information provided by the predefined container description.

4.4.2 Methods

The *ITSAPEContainer* model provides different function used by the IME. New container instances can be created via the *build* method, while this might not be necessary as consolidation of multiple IE is supported. Basic functions as *start* and *stop* allow to start and stop the underlying Docker container. A *destroy* function is called during the tear down process at the end of a HPT, which deletes the associated Docker container. As required by **Infrastructure Status (R9)**, status information is updated for each of these steps.

Generate or Consolidate One or More Containers

Three functions are designed to handle the initialization, building and consolidation process. During initialization, predefined containers are prepared for use as manageable infrastructure. A check ensures that the configured container's *base image* is available:

```
image_exists = Docker::Image.exist?(@baseimage).
```

Unless it is present within the local container image repository the IME tries to fetch if from Docker Hub. If no image can be found or pulled from Docker Hub an error is risen.

Before the IME calls the *build* function of the *ITSAPEContainer* it is decided whether an already in-use container can fulfill the requested container requirements. The type-based naming scheme is used for this. This process results in a list of suited containers, a single one is chosen. Chapter 6 describes possible future option to provide a more sophisticated selection.

If an existing container is capable of fulfilling the requested infrastructure the *consolidate* function of the chosen container is called: `consolidate(elem)`. This consolidation process selects a port from the mentioned predefined ports via the `select_port()` method. It is checked whether this port is free to use by the `port_open?(port)` method. This check is based on the *lsof* command provided with said port:

```
!system("lsof -i:#{port}", out:  '/dev/null').
```

If no resources are associated to a port and necessary access rights are provided the stated command returns immediately; the selected port is added to the in-use port list. If the selected port is in use or can not be required the presented process is repeated for the next port until a free port is found. If no free port is available an error is raised.

If no running container is found that suits the requested infrastructure a predefined container is used. The *ITSAPE Container* model provides a `build(elem)` method suited to fulfill this task. These methods creates the actual Docker container via die Docker API Gem using the predefined Docker image name:

```
  @container = Docker::Container.create(
'Image' => @baseimage,
'HostConfig' => {'Privileged' => true, 'NetworkMode' =>'host' }
).
```

The resulting Docker container id is stored as a property of the current *ITSAPE Container* instance. It is used for referencing the Docker container in subsequent interaction.

Docker container can be operated in *Host-mode* which allows for a direct communication between an outward facing interface and a Docker container. Each network interface is made available to a Docker container in a pure and unaltered form. This connection mode is chosen to skirt the need for extra routing configuration as a result of an additional container network. The hosts first available public IP address is chosen and stored within the *ITSAPE Container* instance:

```
Socket.ip_address_list[1].ip_address.
```

As this parameter might vary between two HPT it should be provided in form of a *Test Season* parameter added in a future release of the legacy framework and *Recipe* model. Additional parameters within a *Test Season* could be used to further specify a *Recipe* used. The current framework status does not support such parameters, therefore a best effort selection is made. As with the `consolidate(elem)` method the serviced IE gets assigned a single port from the available ports. The selected port is added to the in-use port list and the infrastructure specific *generate* script is executed.

Start a Container

Each container can be started calling the *ITSAPE Container* instance providing the respective IE. The Docker container is started, via `start(elem)`, and the associated infrastructure *arm* script is executed: `execute_script(@arm_script, environment)`.

Necessary information are provided via environment variables. The used interface's IP address and the IE internal port are used to configure software of an infrastructure container as a webserver. The temporary artifact location, containing generated artifacts based on participating subject information, is used as resource for artifact presentation. The IE unique id is used to allow for identification of artifacts from different *Human Penetration Tests*.

Deploy Artifacts to a Container

For artifact deployment the framework uses the legacy *Deployment Manager* module to execute the *Recipe's deploy* script. The script's purpose is to provide prepared artifacts to infrastructure used for the respective HPT. This approach is part of the framework's lifecycle if the *Infrastructure Management Extension* is used. The framework is provided with a *Recipe* based *deploy* script that is capable of relocating prepared artifacts. Additionally, each *Infrastructure Element* provides a *deploy* script as well. The latter is able to place prepared artifacts into the correct location inside the infrastructure container.

Both *deploy* scripts rely on environment variables set by the framework and providing information about the prepared artifacts and infrastructure in question. Beginning with the *Recipe's* script, the scripts are executed subsequently.

Remove Artifacts from a Container

A *Human Penetration Test* can be stopped at any time, the *ITSAPE Container* model is required to support this. Therefore, the model provides an `undeploy(elem)` method called with the respective IE. As other IE might be serviced by the associated container as well, only presentation of artifacts from the specific *Test Season* must be stopped.

Additionally, as presented in Figure 9 and section 2.4, artifacts can be scheduled to be active during a certain timeframe. At at the end of such a scheduled timeframe these artifacts have to be removed from the infrastructure. The presented *undeploy* method is used to achieve this.

If the container is in *running* state, the container specific *undeploy* script is executed. All information to remove created configurations and associated artifacts of the respective IE is provided via environment variables. Especially the IE unique id is used to allow for identification of artifacts and configuration files.

As a final step it is checked whether the undeployed IE has been the last IE associated to this container. If this is the case, no other IE is serviced by this container, therefore the Docker container is stopped. This addresses especially the requirement for low resource (**R3.2**) consumption.

Tearing Down a Container

As final stage in a HPT lifecycle, created infrastructure is torn down. This is done via the `destroy(elem)` method of an *ITSAPE Container*. Provided with an IE, this method removes said IE from the container's internal bookkeeping. This can result in an empty ledger and no other IE associated to this container. Given this case, the actual Docker container is deleted.

4.5 Extended Legacy Framework Modules & Models

Changes to framework modules are performed to enable the integration of the IEM. New functionality, to provide a separate processing for managed infrastructure, is added to the *Infrastructure Generator*. Followed by infrastructure generation, artifact generation and deployment are performed by the *Artifact Generator* and *Deployment Manager*. This process can be left unchanged as necessary operations are handled by the *Recipe's* deployment script. Said script has only be adopted to subsequent call an infrastructure specific deployment script. The process of *arming* an infrastructure is performed by the *Delivery Manager*. The IME needs this information to enable necessary routing for artifacts presentation. This section describes necessary modifications to the legacy framework modules and models.

4.5.1 Modifications to the APE Module

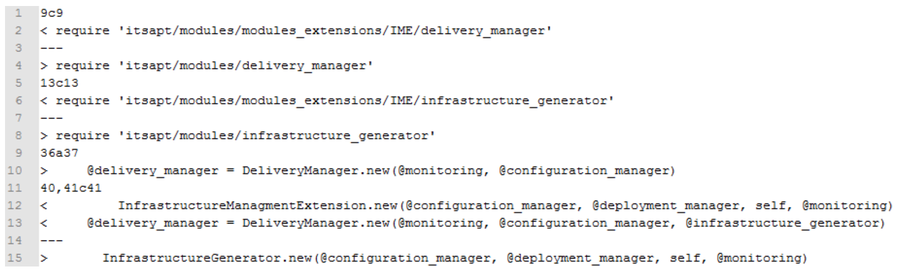
All modules are *required* in the framework's main module: the *APE* module. This module is edited to load the *extended Delivery Manager* and *extended Infrastructure Generator* module. Both *require* paths are changed to load said extended versions. These modules are subclasses of the legacy framework modules and thereby still support script-based generation and manipulation of infrastructure. Differences between these legacy modules and extended ones are shown in the next sections.

Instantiation of the *Infrastructure Manager* and *Delivery Manager* are adapted. The extended *Delivery Manager* requires an *Infrastructure Manager* instance. To instantiate the *extended Infrastructure Manager* class its new name is used.

The installation process of the IME therefore is simple:

1. The IME provides three folders which have to be symbolically linked into the framework installation.
2. Four lines of code must be edited in the *APE* module to install the IME into an existing framework setup.

3. A valid configuration of predefined infrastructure including corresponding infrastructure-specific scripts must be provided. Examples are implemented and presented in section 4.7.



```
9c9
< require 'itsapt/modules/modules_extensions/IME/delivery_manager'
---
> require 'itsapt/modules/delivery_manager'
13c13
< require 'itsapt/modules/modules_extensions/IME/infrastructure_generator'
---
> require 'itsapt/modules/infrastructure_generator'
36a37
>     @delivery_manager = DeliveryManager.new(@monitoring, @configuration_manager)
40,41c41
<         InfrastructureManagmentExtension.new(@configuration_manager, @deployment_manager, self, @monitoring)
<     @delivery_manager = DeliveryManager.new(@monitoring, @configuration_manager, @infrastructure_generator)
---
>       InfrastructureGenerator.new(@configuration_manager, @deployment_manager, self, @monitoring)
```

Figure 16: *Output provided by **diff utility** between the* modified APE module *and its legacy framework implementation.*

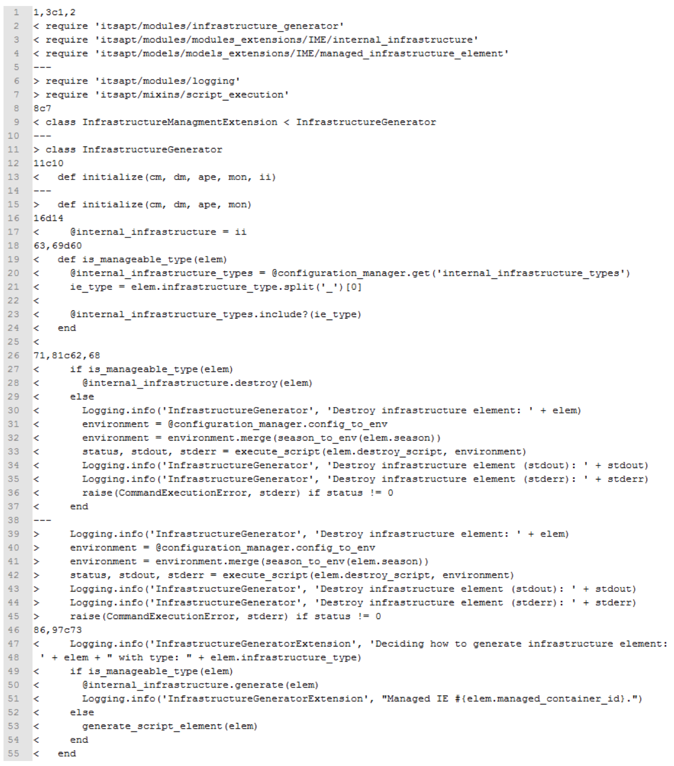
4.5.2 Modifications to Infrastructure Generator (IG)

The *Infrastructure Generator* module is loaded by the *APE* module. The legacy module is adapted to allow the use of the IME. Using the legacy module code basis ensures compatibility to script-based infrastructure generation.

The *extended Infrastructure Generator (eIG)* realizes the selection of manageable infrastructure. Each commissioned *Infrastructure Element* is checked for its infrastructure type first. Based on the provided configuration, it is decided whether a *Infrastructure Element* is processed by script-based processing or be the IME. If a commissioned type matches a manageable type, subsequent processing of this *Infrastructure Element* is handled by the IME. Otherwise, legacy script-based processing is performed.

At the end of a HPT, infrastructure must be torn down. The legacy *Infrastructure Generator* uses a destroy script, provided by the penetration tester, to stop and shutdown created infrastructure. Scrip-based infrastructure is still handled via said script. Manageable *Infrastructure Elements* are forwarded and their tear down process is handled by IME.

Changes between the legacy *Infrastructure Generator* and its modified version are shown by the *diff utility* in Figure 17. The *Internal Infrastructure Manager* module is made known to the eIG module and added as `ii` to the *Infrastructure Generator* initialization signature: `initialize(cm, dm, ape, mon, ii)`. The legacy IG is *require*, as eIG is based on the legacy IG. Further, the managed IE with its extended initialization signature is *required*. A class variable is added providing access to the IIM. A method to check whether a provided IE is a manageable IE is added: `is_manageable_type(elem)`. *Generate* and *destroy* are extended to call their IIM counterparts if said check returns returns positive.



```
1,3c1,2
< require 'itsapt/modules/infrastructure_generator'
< require 'itsapt/modules/modules_extensions/IME/internal_infrastructure'
< require 'itsapt/models/models_extensions/IME/managed_infrastructure_element'
---
> require 'itsapt/modules/logging'
> require 'itsapt/mixins/script_execution'
8c7
< class InfrastructureManagmentExtension < InfrastructureGenerator
---
> class InfrastructureGenerator
11c10
<   def initialize(cm, dm, ape, mon, ii)
---
>   def initialize(cm, dm, ape, mon)
16d14
<     @internal_infrastructure = ii
63,69d60
<   def is_manageable_type(elem)
<     @internal_infrastructure_types = @configuration_manager.get('internal_infrastructure_types')
<     ie_type = elem.infrastructure_type.split('_')[0]
<
<     @internal_infrastructure_types.include?(ie_type)
<   end
<
71,81c62,68
<     if is_manageable_type(elem)
<       @internal_infrastructure.destroy(elem)
<     else
<       Logging.info('InfrastructureGenerator', 'Destroy infrastructure element: ' + elem)
<       environment = @configuration_manager.config_to_env
<       environment = environment.merge(season_to_env(elem.season))
<       status, stdout, stderr = execute_script(elem.destroy_script, environment)
<       Logging.info('InfrastructureGenerator', 'Destroy infrastructure element (stdout): ' + stdout)
<       Logging.info('InfrastructureGenerator', 'Destroy infrastructure element (stderr): ' + stderr)
<       raise(CommandExecutionError, stderr) if status != 0
<     end
---
>     Logging.info('InfrastructureGenerator', 'Destroy infrastructure element: ' + elem)
>     environment = @configuration_manager.config_to_env
>     environment = environment.merge(season_to_env(elem.season))
>     status, stdout, stderr = execute_script(elem.destroy_script, environment)
>     Logging.info('InfrastructureGenerator', 'Destroy infrastructure element (stdout): ' + stdout)
>     Logging.info('InfrastructureGenerator', 'Destroy infrastructure element (stderr): ' + stderr)
>     raise(CommandExecutionError, stderr) if status != 0
86,97c73
<     Logging.info('InfrastructureGeneratorExtension', 'Deciding how to generate infrastructure element:
 ' + elem + " with type: " + elem.infrastructure_type)
<     if is_manageable_type(elem)
<       @internal_infrastructure.generate(elem)
<       Logging.info('InfrastructureGeneratorExtension', "Managed IE #{elem.managed_container_id}.")
<     else
<       generate_script_element(elem)
<     end
<   end
```

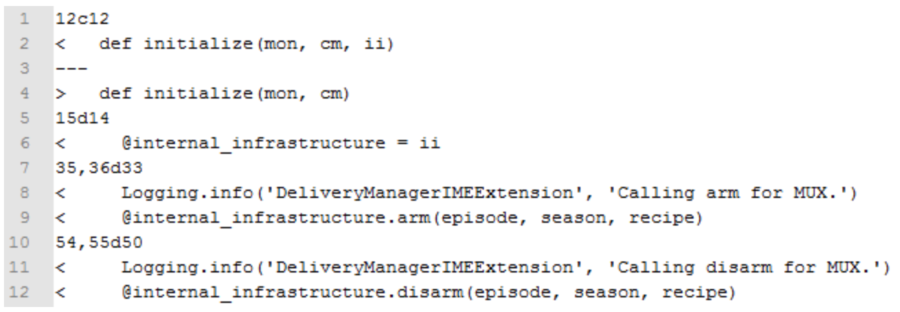
Figure 17: *Output provided by* ***diff utility*** *between the* extended Infrastructure Generator (eIG) *and its legacy framework implementation.*

4.5.3 Modifications to Delivery Manager (DM)

The *Delivery Manager* is used to call *arm* and *disarm* scripts, they are executed by script-based infrastructure. Arm and disarm methods are provided with a *Test Episode* instance which provides not enough information to decide whether or not managed infrastructure is involved. Therefore, all arm and disarm events are forwarded to the IME. Within the IME further analysis of involved components is performed. Only within the bookkeeping of the IME the relation between a created container and its season is available. This relation is used to determine all containers which must be started and which routing operations are required.

To support script-based infrastructure as well, the described interaction with the IME is performed additionally to the legacy execution of *arm* and *disarm* scripts. For managed infrastructure these script might just return without performing any action.

Changes between the legacy *Delivery Manager* and its modified version are shown by the *diff utility* in Figure 18. The *Internal Infrastructure Manager* `ii` is added to the *Delivery Managers* initialization signature: `initialize(mon, cm, ii)`. A class variable is added providing access to the IIM. The legacy *arm* and *disarm* methods are extended by a single line of code respectively. Performing an arm/disarm call providing *Test Episode*, *Test Season* and *Recipe* to the corresponding IIM methods.



```
12c12
<   def initialize(mon, cm, ii)
---
>   def initialize(mon, cm)
15d14
<     @internal_infrastructure = ii
35,36d33
<     Logging.info('DeliveryManagerIMEExtension', 'Calling arm for MUX.')
<     @internal_infrastructure.arm(episode, season, recipe)
54,55d50
<     Logging.info('DeliveryManagerIMEExtension', 'Calling disarm for MUX.')
<     @internal_infrastructure.disarm(episode, season, recipe)
```

Figure 18: *Output provided by* ***diff utility*** *between the* extended Delivery Manager *and its legacy framework implementation.*

4.5.4 Modifications to Infrastructure Element (IE)

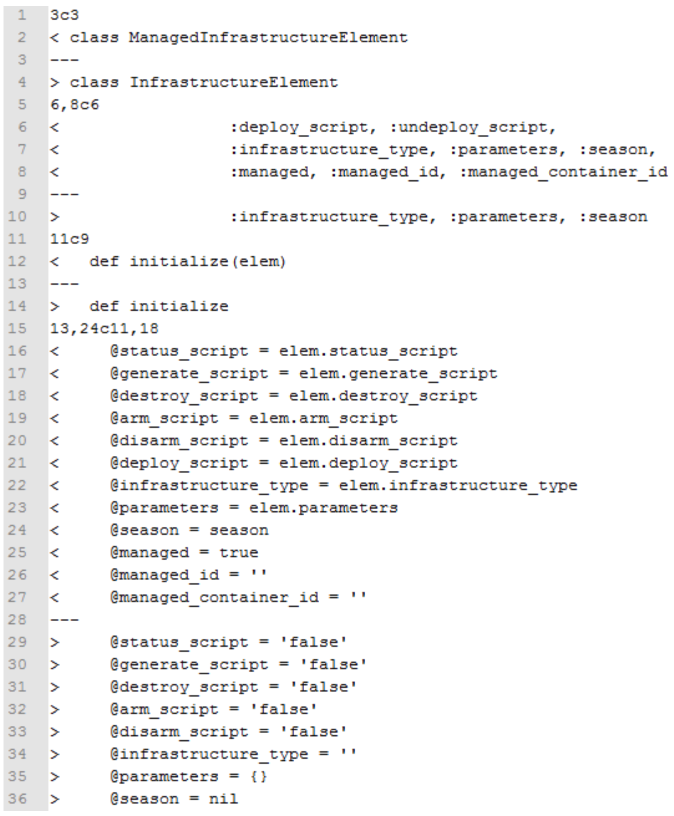
Infrastructure Elements are provided with a *Recipe* and are dynamically created by the *Config Manager*. Every *Test Season* holds a recipe id to identify a configured *Recipe*. Only when this *Recipe's* information is requested an actual *Infrastructure Element* is created. Within the current framework no persistent *Infrastructure Element* handling exists. Script-based infrastructure is controlled via generic scripts, they facilitate environment variables to specify their operations. *Infrastructure Elements* are created on the fly and are needed to provide said scripts and static information used by these scripts.

For example, if a HPT is set up, infrastructure is generated. The *Config Manager* uses the defined recipe id to identify the correct *Recipe* and an *Infrastructure Element* instance is generated. Infrastructure is created based on ports defined within an *Infrastructure Element's* properties using the IE's *generate* script. After the generation process this *Infrastructure Element* instance are no longer used and abandoned.

While performing an *arm* operation *Infrastructure Elements* are required again as they provide necessary *arm* and *disarm* scripts. Consequently, new *Infrastructure Element* instances are created to provided these scripts. As a result, changes to IE instances used within the generation process are not present within the IE instances used in the arm and disarm process.

With managed infrastructure this concept is no longer viable. Managed infrastructure is created at one point and further managed during its operation time. A clear relation between an *Infrastructure Element*, requested by a specific *Test Season* of a HPT, and its managed container is needed. The *Infrastructure Element* model is therefore extended to include additional managed infrastructure relevant information.

A *managed* parameter indicates whether this IE is managed by IME. The *managed id* is used to provide a unique identifier, for a specific IE. It is needed as *Recipes* are intended for multiple use and identical IE can be commissioned for infrastructure generation. If a container is matched to an IE, the container's id is stored in a *container id* property. Managed IE are stored within the IME to enable consisted management.



```
3c3
< class ManagedInfrastructureElement
---
> class InfrastructureElement
6,8c6
<                   :deploy_script, :undeploy_script,
<                   :infrastructure_type, :parameters, :season,
<                   :managed, :managed_id, :managed_container_id
---
>                   :infrastructure_type, :parameters, :season
11c9
<   def initialize(elem)
---
>   def initialize
13,24c11,18
<     @status_script = elem.status_script
<     @generate_script = elem.generate_script
<     @destroy_script = elem.destroy_script
<     @arm_script = elem.arm_script
<     @disarm_script = elem.disarm_script
<     @deploy_script = elem.deploy_script
<     @infrastructure_type = elem.infrastructure_type
<     @parameters = elem.parameters
<     @season = season
<     @managed = true
<     @managed_id = ''
<     @managed_container_id = ''
---
>     @status_script = 'false'
>     @generate_script = 'false'
>     @destroy_script = 'false'
>     @arm_script = 'false'
>     @disarm_script = 'false'
>     @infrastructure_type = ''
>     @parameters = {}
>     @season = nil
```

Figure 19: *Output provided by **diff utility** between the* managed Infrastructure Element *and its legacy framework implementation.*

Changes between the legacy IE and managed IE model are shown by the *diff utility* in Figure 19. New properties and their accessors are added. A new initialize signature (`initialize(elem)`), extended to expect a legacy IE, allows for a direct initiation of a managed IE.

Based on the stored season id and type, the IME can correlate a provided IE from an arm operation to an IE stored during infrastructure generation. Thereby, the correct container can be identified and necessary routing operations can be performed.

4.6 Routing via Redirect System

With generated infrastructure personalized artifacts can be presented to a subject. This presentation is performed according to scheduled time slots, these slots may respect working hours and are configured by the penetration tester. The framework provides *arm* and *disarm* methods to start and stop artifact presentation. Script-based infrastructure is entirely controlled via scripts, which are used to prepare and enable all necessary operations needed to present an artifact to a subject.

For managed infrastructure this task is handled differently. As a single infrastructure can be used for multiple artifacts, special attention must be provided to ensure subject specific presentation. Routing of IP packages is used to redirect subject requests to a designated artifact infrastructure bearing the correct prepared artifacts. Domain names can be used to create special artifacts designed to test a subjects security awareness in regard to web and mail addresses [53]. The following presents the implementation of the IP routing and domain name system of the IME.

For routing purposes a designated Docker container is created with its network configured as *host mode*. This network mode allows the container to directly access the host's network stack. To be able to manipulate routing via iptables the container is started with the parameter *privileged=true*. A container setup is chosen to have an encapsulated environment, all routing operations are executed in this designated Docker container.

4.6.1 IP

The IME uses iptables for redirecting IP traffic, the routing functionality is realized through the *Routing Info* model. The model requires information about the container's IP address and the artifact specific port. Further, the subject's IP address and port of the request is needed to only redirect relevant traffic. If not configured otherwise, TCP is regarded as default transport protocol. It is the protocol used by all presented artifact services, refer to section 2.1. Optionally, a domain name can be provided, too. The handling of a the Domain Name System is explained in the next section.

```
# all dns to company's DNS
  iptables -t nat -I PREROUTING 1 --protocol udp --destination-port 53 \
             --jump DNAT --to-destination $COMPANYDNS
  iptables -t nat -I POSTROUTING -j MASQUERADE

# redirect only subject to internal dns
  iptables -t nat -I PREROUTING 1 -s $SUBJECTIP -p udp --dport 53 -j ACCEPT

# redirect subject http to docker port
  iptables -t nat -I PREROUTING 1 -s $SUBJECTIP -p tcp \
             --dport $EXTERNALPORT -j REDIRECT --to-port $INTERNALPORT
```

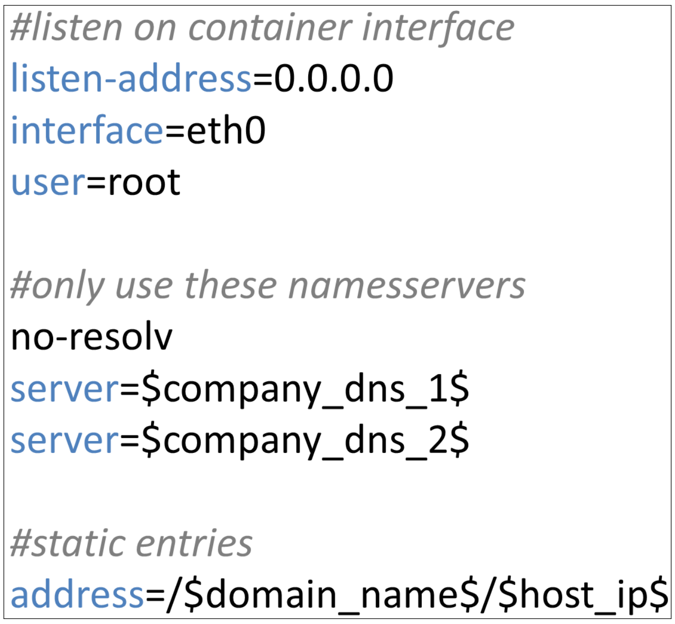
Figure 20: *Iptables command to enable subject-specific routing. These commands are stored and executed during an arm operation by the* ***Routing Info*** *model.*

At generation of a managed IE, it is checked whether routing is required. Sending e-mail messages for example does not require traffic redirection, while a website artifact does. If an IE has port parameters set these are taken as an indicator to provide redirecting of subject requests to said IE. Ports specified by an IE are regarded as *external* ports. Requests, originating from the subject's IP address to an external port, are redirected if a subject's test is scheduled for presentation. These requests are forwarded to the respective container serving this test's IE. The IE specific *internal* port is chosen based on the provided container configuration and explained in detail in section 4.4.2.

Equipped with these information a iptables rule is created. Two commands to create and delete this rule are stored within the *Routing Info* object. Via the *arm* and *disarm* methods the respective command is executed and routing but into effect. Figure 20 shows commands used to produce these *arm* rules. First all DNS traffic is routed to the company's DNS and its origin is masqueraded. Scheduled subject's DNS requests are accepted to be handled by the local ITS.APE DNS server provided with the routing system. Additionally, subject request's for a specific service are redirected from their external port, for example port 80 for HTTP requests, to the associated Docker container internal port. *Disarm* rules are the respective counterpart of the added rules, therefore executed with an `-D` flag. If all tests are finished the client's DNS server can be changed back to their original configuration.

4.6.2 DNS

The described *Routing Info* model can be provided with a domain name. This domain name is configured by the penetration tester as a parameter within an *Infrastructure Element* as part of a *Recipe*. The routing setup expects all subject clients to use the ITS.APE host machine for domain name resolution (DNS).



```
#listen on container interface
listen-address=0.0.0.0
interface=eth0
user=root

#only use these namesservers
no-resolv
server=$company_dns_1$
server=$company_dns_2$

#static entries
address=/$domain_name$/$host_ip$
```

Figure 21: *Dnsmasq configuration template used by the routing system.*

The redirecting system's container provides a dnsmasq server to support DNS requests. Only if a subject's test is scheduled and armed a subject's DNS requests are handled by the redirect system. Additional iptables rules, to ensure this DNS request routing, are created and stored along the artifact specific routing rules, they are depicted in Figure

20. Only requests for domains specified within armed IEs are answered by the dnsmasq server while all others are forwarded to a configured default DNS server. This could be the standard DNS server used within the evaluated company.

The answer of a request send by a subject for an armed domain is answered with the IME host IP address. This ensures that sub-sequent traffic designated for this domain name is send to the IME and can be routed by iptables.

The dnsmasq's configuration used by the redirecting system is depiced in Figure 21. Static entries for provided domain names (`$domain_name$`) resolve for the framework's host IP address (`$host_ip$`). Every other request is forwarded to the company's DNS servers (`$company_dns_1$` & `$company_dns_2$`).

4.7 Generalized Service Infrastructure

Infrastructure Elements describe a specific need of service to present artifacts. This can be a webserver, a mail sending service or other services as presented in section 2.1. As required by **Suitable Technology (R3)**, a provided extension to manage infrastructure needs to demonstrate its functionality by using its implemented system. Therefore two of the mentioned services are provided with this implementation. The mail sender service and webserver service are chosen, as these are the same services as provided by the script-based *Testrecipe*.

While not provided with this thesis, realizing a *file sharing service* could be done using a SMB/CIFS Docker image available from Docker Hub. For example, to publicly share a folder using SMB protocol the *jenserat/samba-publicshare* can be used. Some adaptations to the image's `smb.conf` are required; the designated internal ports and corresponding folder have to be added to the configuration, this can be done with the infrastructure specific deploy script. Artifacts are added via this script, too. Routing of SMB traffic can be done via the implemented routing system.

The IME requires manageable infrastructure to provide infrastructure specific script which are executed at a certain phase of a HPT. This section introduces the concept of said scripts. And closes with the description of the provided two services.

4.7.1 Infrastructure Specific Scripts

With the script-based solution each *Infrastructure Element* has a script for each lifecycle state of an artifact. An artifact lifecycle is depicted in Figure 9. At total there are six script to allow for artifact and infrastructure manipulation:

1. A *generate* script to generate infrastructure for artifact delivery,
2. a *deploy* script to provide artifact to this infrastructure,
3. an *arm* script to start artifact presentation,
4. a *disarm* script to stop artifact presentation,
5. an *undeploy* script to remove artifacts from infrastructure and
6. finally a *destroy* script to tear down created infrastructure.

Each of these scripts is created to suit exactly one special *Recipe*. They are specifically designed for infrastructure created via the *generate* script and the artifacts generated. As penetration testers are free do design their *Recipes* and scripts, interoperability between infrastructure and scripts from different *Recipes* can not be assumed. Further, the framework is not aware of already existing infrastructure even if two HPTs use the same *Recipe*.

With managed infrastructure it is a set goal to reuse already available infrastructure if possible. This reuse has two aspects, on the one hand infrastructure can be used by two different HPTs in general. For example two HPTs using different *Recipes* but both require a webserver. On the other hand a single managed infrastructure can serve two HPTs using the same *Recipe* but are set up to test different sets of subjects. With the script-based solution such scenarios could not be handled at all or would require cumbersome manipulation to existing scripts.

The IME is designed to support the general idea of the artifact lifecycle including *Recipes* and scripts. This is achieved by having each infrastructure container providing infrastructure specific scripts similar to the presented artifact specific scripts. At total four scripts must be provided with each container:

1. A *deploy* script executed in a subsequent call by the artifact deploy script. It handles file transfer and adaptions of artifacts, for example adding an artifact's season to its file name.
2. An *arm* script used to execute certain final operations to deliver an artifact. For example using *sendmail* to send an e-mail message to a subject.
3. A *disarm* script used to stop presentation of an artifact. For example disable a website.
4. An *undeploy* script used to remove a specific artifact from a container. It can be used if a subject does no longer want to participate in a HPT.

A *generate* script is obsolete as managed infrastructure is handled by the IME. Similar, a *destroy* script is is no longer needed as tear down of containers is managed too. Each predefined container has to provide these infrastructure specific scripts to comply with IME requirements.

4.7.2 Mail Service

To provide a e-mail sending service, as described in section 2.1, a suitable Docker image is used. A Dockerfile is provided to create the image, which is added to the local Docker repository. To respect the requirement for low resource (**R3.2**) consumption, a base image designed to have a small footprint is chosen: *phusionbaseimage* [1]. *Sendmail* is chosen as delivery software for e-mails. It provides an easy way to deliver e-mail messages via the command line.

To send messages via sendmail a host name is required within the sendmail machine. This is taken care of by a script designated to set a host name. The sendmail configuration must be rerun to recognize this change. Said script and the sendmail configuration program are run during creating of the Docker container. Additionally, the *ITS.APE*

Satellite is copied to the container including all required keys and certificates to establish a secure connection to the framework's *Track Collector*. Based on the infrastructure's base type, its software and the chosen host name, the infrastructure type is set to: *mailsender_sendmail_its.ape*

The created predefined container does have a fixed host name as specified in the mentioned script. The selected host name is be used as add-on parameter within the container's type. If a penetration tester wants to have a specific host name set for e-mails of a HPT, he can specify the host name in the script and modify the container's type accordingly. A *Recipe* with an IE using its specified container's type as *Mail Service* lets the IME select the correct predefined container.

The *Mail Service* container uses the infrastructure specific scripts to prepare for mail delivery. Via the container's *deployment* script, generated artifacts are placed in a designated folder. Each e-mail message required in a single text file containing at least a *FROM:* line who's content can be set at will. The actual message follows there after. Additionally, The *deployment* script renames each message file to include the subject's unique *subject id* and the specific test season the subject is part of. These information are needed to support that a subject can be part of two HPTs at the same time, while each HPT provides different message artifacts.

To send an e-mail, the infrastructure specific *arm* script is used, it expects all commissioned e-mails in said designated folder. The IME's *arm* method executes the infrastructure specific *arm* script. It expects several parameters provided via environment variables: an e-mail address, the armed season's id and the subject id. The later items are used to identify a specific message file from the available artifact files. By calling sendmail with the e-mail address and the selected message an e-mail is finally send. Each sent e-mail generates a log entry in a log file which is provided to the ITS.APE satellite.

A special *disarm* script is not needed as the artifact is no longer under the service's control after being send. An *undeploy* script is provided, which requires calling with a subject's id and a season. Using these two parameters a specific message file can be identified and removed from the container.

4.7.3 Website Service

To provide websites an Apache 2.2 webserver image equipped with PHP 5.6 is chosen. Apache and PHP are commonly used to provide dynamic websites, further the *Testrecipe* uses a website build with this setup in mind. During creation of said image the *ITS.APE Satellite* is copied to the container including all required keys and certificates to establish a secure connection to the framework's *Track Collector*.

The infrastructure specific *deploy* script copies generated artifact files to the webserver's document folder. Additionally, it creates a season specific configuration file in the Apache *sites-available* folder. The configuration's file name has to include the season id for identification. This configuration file defines a *Virtual Host* bound to the container's IP address an the port designated to this specific IE chosen by the IME. The configuration includes the season specific document folder it is as the virtual host's *Document Root*. Required parameters such as IP address, port and season id are provided via environment variables.

The container's *arm* script enables created configuration via executing the Apache specific *a2ensite* tool. The tool is provided with the configuration file's name which includes the season id. Said id is provided via environment variables. At last the Apache service is forced to reload its configuration.

To disable a season's armed website the *disarm* script is used. It uses the *a2dissite* tool to remove a season's specific configuration. Again the season's id is provided via environment variables. A forced reload of the Apache service is used to effectuate the reduced configuration.

An *undeploy* script can be used to remove all created configuration and artifact files of a season. It requires the specific season id provided via environment variables.

The Apache webserver is configured to log each website access to a log file which is provided to the ITS.APE satellite.

5 Evaluation

The implemented *Infrastructure Management Extension (IME)* is designed to simplify the use of the ITS.APE framework to perform *Human Penetration Tests*. The following chapter evaluates the implemented system and its components.

The system is evaluated against the identified requirements listed in section 3.1. Modules' and models' functionalities are tested via unit tests while the implemented redirecting system and provided website and *Mail Service* containers are tested within a small testing environment.

5.1 Conform with formulated Requirements

A requirements analysis is performed to identify necessary and important features a managed infrastructure system has to have, cf. 3.1. In the following each identified requirement is discussed and it is shown how the design and implemented software system realizes these demands.

The ITS.APE framework is used in the UKSH network; therefore an overview of the used software components is used to identify relevant IT services, this acknowledges requirement **Select Services (R1)**. Five service items are identified and three can be used directly to present artifacts to a subject. Two are needed to support an automated presentation. These items are:

1. Website Service,
2. Mail Service,
3. File Sharing Service.

Additionally, a redirecting system based on:

4. IP and
5. host names.

For each of the identified services a design of the service, its functionality and possible configurations are given. If possible and relevant for the UKSH test environment, a configuration to enhance a service's confidentiality is provided. This fulfills the requirement for **Customizability (R2)** for these services.

Besides the designed services, the implemented extension provides a configurable generic interface to add new services. This interface uses a text based configuration editable with a simple text editor. A presented naming scheme is used to support any kind of service as long as necessary predefined infrastructure and corresponding *deploy*, *arm*, *disarm* and *undeploy* scripts are provided. Thereby a highly reusable system is given. Providing a new service for use with the IME can be done with a small set of parameters set in the ITS.APE configuration file. The implemented software can be extended by any number of possible services without the need to change the extension's or ITS.APE framework's source code. This exceeds the set goal of the requirement for **Customizability (R2)** which required five distinct services.

The presented extension uses Docker containers to provide managed infrastructure used for HPTs. Containers in general are capable of providing required functionality as do full virtualized machines. In contrast to full virtualized machines their required resources in terms of file space and processing power are small as they can use already existing libraries and system functions. Moreover, generation and launching of a container can be done much faster than creating a full virtualized machine. Configuration of container images is done via a text based *Dockerfile*, which provides an easy way to define a reusable image. With the use of the Docker Engine a broad collection of existing service images are usable through the Docker Hub. The selection of Docker and a container-based solution fully acknowledges the requirement for **Suitable Technology (R3)**. Again, with the available image base provided by Docker Hub a large spectrum of possible infrastructure services can be realized with small effort. The provided *Website Service* and *Mail Service* and their infrastructure specific scripts and configurations can be used as reference.

By including the ITS.APE satellite into the Dockerfile of service infrastructure, the requirement to **Support Satellites (R4)** is fulfilled. Appointed to the correct log file the satellite system provides necessary tracks to the ITS.APE framework. Thereby evaluation of a subject's actions towards an artifact can be done.

By design containers provide a low resource footprint. Additionally the IME tries to match compatible infrastructure demands to be served by a single infrastructure container. Whenever an infrastructure demand is no longer valid the framework's infrastructure *destroy* function is called. With legacy script-based infrastructure the tear down process is performed by scripts, these could include instructions to manually modify network settings such as DNS entries or firewall rules. This process is automated with the use of IME and relies on the internal bookkeeping to check whether any other service demand exists. For the service demand in question the created iptable rules and DNS entries are removed by the IME's routing system. Unless there is any existing demand from other HPTs, used infrastructure is torn down and the associated container is deleted. This fulfills the requirement for **Garbage Collection (R5)** completely.

The IME is designed to respect the legacy script-based generation of infrastructure. By adapting the framework's *Infrastructure Generator* and *Delivery Manager* modules necessary program flow control is obtained to integrate this extension. Keeping the already existing functionality of said modules, script-based infrastructure generation is maintained. The modules' unit tests are extended to test the added introduced functionality needed for managed infrastructure. The achieved *passed* status for these extended unit tests proves the successful implementation of required **Compatibility (R6)** to existing infrastructure deployment methods. While full compatibility is achieved, the IME supports black- and whitelisting of infrastructure types. Blacklisted types of infrastructure requests are never handled via the IME, but a script-based approach is chosen. Whitelisted types of infrastructure might be managed by IME if matching configuration of predefined infrastructure is provided. Each commissioned *Infrastructure Element* is checked to comply with these list-based restrictions.

The IME hosts all containers on the framework's host machine. No additional IP addresses are used other than IP addresses assigned to the host's machines network interfaces. This fulfills the requirement to **Respect Restrictions (R7)**.

The IME's redirecting system uses iptables and dnsmasq to redirect the subject's requests to the specific artifact infrastructure. Configuration is provided via the *Recipe's Infrastructure Element* and its parameters. Requests to specified ports and host names are redirected for participating and scheduled subjects. The framework's *Subject Tracker* module is used to determine a subject's IP address. No manual interaction is necessary to activate the redirecting process of managed infrastructure. This improves script-based infrastructure which can present the penetration tester instructions to manually activate redirecting of scheduled subjects. The IME's redirecting system thereby fulfills the requirement set up by **Redirecting System (R8)**. A small test environment as described in 3.4.4 is set up to manually test the functionality of the implemented redirect system. The setup of this environment and testing procedure is described in section 5.7.

The main module of the IME, the *Internal Infrastructure Manager*, provides two methods to gain knowledge about the managed infrastructure status. First, Information about an *Infrastructure Element* can be requested. This method returns the IE's type, managed status, its unique managed id, its associated container id and the Docker container status. Second, a method which expects an IE's type and Docker container id. This method returns the Docker container running state. These two methods provide the requested status information as described in requirement **Infrastructure Status (R9)**. They even exceed the requested status information and allow for a more detailed report about requested managed infrastructure.

The implemented modules and models are tested via *RSpec* tests. All legacy modules and models are provided with unit tests; these tests are extended to cover made extensions. The designed naming scheme is tested heavily to show generic compatibility to possible infrastructure types. This fulfills the requirement for **White Box Testing (R10)**. The following sections present the performed unit testing including said naming scheme.

5.2 Type-based Naming Scheme

Consolidation is based on type-based naming scheme performed by the IME. These tests are designed to check whether correct matching for compatible infrastructure types is achieved. Further, tests ensure that incompatible infrastructure types produce no match. They are part of the IME modules unit test and confirm a correct implementation of the described naming scheme.

A test scenario in which a commissioned IE is matched to an existing container is created. First, a *base type* matching is tested, only identical base types are consolidated. Second, an identical base type is assumed and matching based on the *software* parameter is performed. For this matching each combination shown in Table 1 is tested.

At last, it is assumed that a container with matching base type and software parameter is present. For a single add-on a similar version testing is performed. To test correct behavior when multiple add-ons are provided additional test cases are necessary. The placing of an add-on within the add-on parameter should not matter. Therefore, these tests use multiple add-ons with all possible placement combinations in said parameter. Each add-on can provide a specified version which must be accounted to the correct add-on. This is tested by using version-specific and version-agnostic add-ons throughout this test.

The implemented IEM passes all described tests. While these tests consist of numerable test cases, it can not cover every possible combination of base type, software and add-on parameter. A functional code coverage of the responsible matching methods is achieved, this proofs the generic concept of selecting consolidatable containers is implemented in the correct way.

5.3 Configuration

Providing an easy and usable configuration is a requirement identified in section 3.1. All models and modules introduced by IME rely on a configuration provided via specific entries in the ITS.APE configuration file. Thereby all necessary parameters can be adjusted with a simple text editor and within a single file. The configuration provides an interface for defining generic predefined managed infrastructure containers.

Testing all possible configurations is not possible due to the generic fashion of supported configurations. A similar configuration as shown in Figure 14 is used as a mocked configuration for all unit tests. All models and modules introduced by IME are tested using this configuration. Every of the performed unit tests are passed, thereby said configuration is shown to be functional.

5.4 Internal Infrastructure Module

The IIM orchestrates all managed infrastructure; its bookkeeping functionality is the core for consolidating and controlling managed infrastructure. The provided unit test's checks of functions and processes of the implemented module are explained in the following.

With initializing the IIM instance, predefined containers are loaded. Therefore a configured container type is requested and expected to be generated by the IIM. To test correct behavior of a missing configuration, generation of an unconfigured IE is expected to cause an error.

For generation of a commissioned IE, it is tested whether a container can be build and started. If a requested image is not available from the local repository, it is pulled from Docker Hub. This is tested with the *hello-world* container provided by Docker and requires an Internet connection.

Created infrastructure is torn down at the end of a HPT. Therefore the provided *destroy* function is tested by creating and subsequently tearing down a container. The test is passed, if said container is no longer listed as available container by the Docker Engine.

The IIM's *arm* method starts a created container for artifact delivery. Again a container is generated and it is checked if after calling the *arm* method said container is in *running* state. As a counterpart the *disarm* method stops a created container. Similar a started running container is generated and expected in a *stopped* state after calling the *disarm* method.

In addition to starting a managed infrastructure, arming it requires certain routing operations. Namely, correct *Routing Info* instances for the provided *Test Episode* and *Test Season*are expected. Testing the *armmux* and *disarmmux* methods of the IIM expects said preparations to be performed correctly.

To fulfill the requirement **Infrastructure Status (R9)** a *get_info* method is provided by the IIM. To test this method a new managed infrastructure is created. Its status after creation, arming and disarming is checked and expected to provide correct information about the managed infrastructure status. With tearing down this managed infrastructure the associated IE is expected to reflect its legacy *unmanaged* state.

Testing full artifact delivery would have to test functionality of a provided predefined container and its scripts. This kind of integration test is not performed within this module's unit test.

5.5 ITSAPE Container Model

The *ITSAPE Container* model represents the IME interpretation of an infrastructure container. It performs direct communication between the IME and the Docker API.

Each of the IIM module and its tests facilitates ITSAPE containers. Conform implementation of the container's function is expected. A unit test of the *ITSAPE Container* model ensures the model's compliance.

The test expects correct initialization of a container. The provided preparation of predefined container images is expected to provide a container with correct properties. Generation and destruction of the associated Docker container is expected for the *build(elem)* and `destroy(elem)` methods. `Start(elem)` is expected to produce a running Docker container. `Consolidate` and `disarm` methods are expected to produce script execution of the container specific scripts. The `status` method has to provide the associated Docker container status.

5.6 Extended Legacy Framework Modules & Models

Each of the extended framework's modules and models are provided with a *RSpec* unit test. These tests are extended to maintain test coverage of their legacy functionality, for example script-based infrastructure generation. Thereby compliance to legacy processes as well as support for managed infrastructure generation can be shown.

For the *Extended Delivery Manager*, its initialization test is adapted to reflect its extended signature: `DeliveryManager.new(@monitoring, @configuration_manager, @infrastructure_manager)`. *Arm* and *disarm* methods are expected to produce callbacks to the IIM module.

Similar the initialization test of the *Extended Infrastructure Generator* is adapted to correspond to its changed signature:
`InfrastructureManagmentExtension.new(@configuration_manager, @deployment_manager, @ape, @monitoring, @internal_infrastructure)`.

The generation of infrastructure is tested, and callbacks to the configured white- and blacklist are expected. Further callbacks to the *Configuration Manager* requesting internal infrastructure configuration details are anticipated.

For the *Infrastructure Element* model it is tested to provide all new properties.

5.7 Evaluation of Routing via Redirect System

The redirecting system is evaluated by preparing the webserver infrastructure containers as described in 5.8.2. The ITS.APE host machine's network interface is configured to IP address 192.168.1.12. Three additional computers connected to the test machines network are used for the performed test scenario. Alice's machine is assigned the IP address 192.168.1.3, Bob's machine is assigned the IP address 192.168.1.25 and both are part of a planned IT security awareness test. Charlie represents a non-participating user, his machine is assigned the IP address 192.168.1.99. The standard DNS server's IP address of this network is: 8.8.8.8.

An IT security awareness test is planned, therefore subject and user machines are configured to use the ITS.APE host as their DNS server. All machines are expected to have access to the Internet, the ITS.APE ITS.APT public website is reachable. A test scenario of four stages is evaluated:

1. Subject and user machines are expected to be configured correct. All can access the Internet; the ITS.APT public website (`its-apt.de`) is reachable. A to be used phishing website cannot be reached via its domain: `phising-website.itsape`.
2. Alice is scheduled for an active *Human penetration test*, therefore she should be able to reach the prepared phishing website: `phising-website.itsape`. Bob and Charlie are expected to have no access to this website from their machines. All machines are expected to have access to the Internet, the ITS.APE ITS.APT public website is reachable.

3. Alice's test period is over, she is expected to have no longer access to the phishing website from her machine. Bob's test is active, he should be able to reach the prepared phishing website: `phising-website.itsape`. Charlie is expected to have no access to this website with his machines. All machines are expected to have access to the Internet, the ITS.APE ITS.APT public website is reachable.
4. Testing is finished, the prepared phishing website must not be reachable from any machine. All machines are expected to have access to the Internet, the ITS.APE ITS.APT public website is reachable.

Figure 22 to 27 show the results of accessing both websites from Alice's Bob's and Charlies systems. All machines can access the Internet and reach the ITS.APE website during all test stages. Only Alice and Bob can access the prepared website artifact from their machines. This access is limited as requested within different stages of this evaluation. Alice is only able to access the phishing website during test stage two, Bob can request the website artifact only during test stage three and Charlie is never able to reach the prepared website.

Figure 30 to 32 show the ITS.APE host iptable configuration of during each test stage. Figure 28 & 29 show the dnsmasq configuration during active during the test stages.

The iptable rules presented in section 4.6.1 are effective and allow for correct presentation of artifacts to subjects. Other users are not affected in their daily work and schedule-conform presentation is possible.

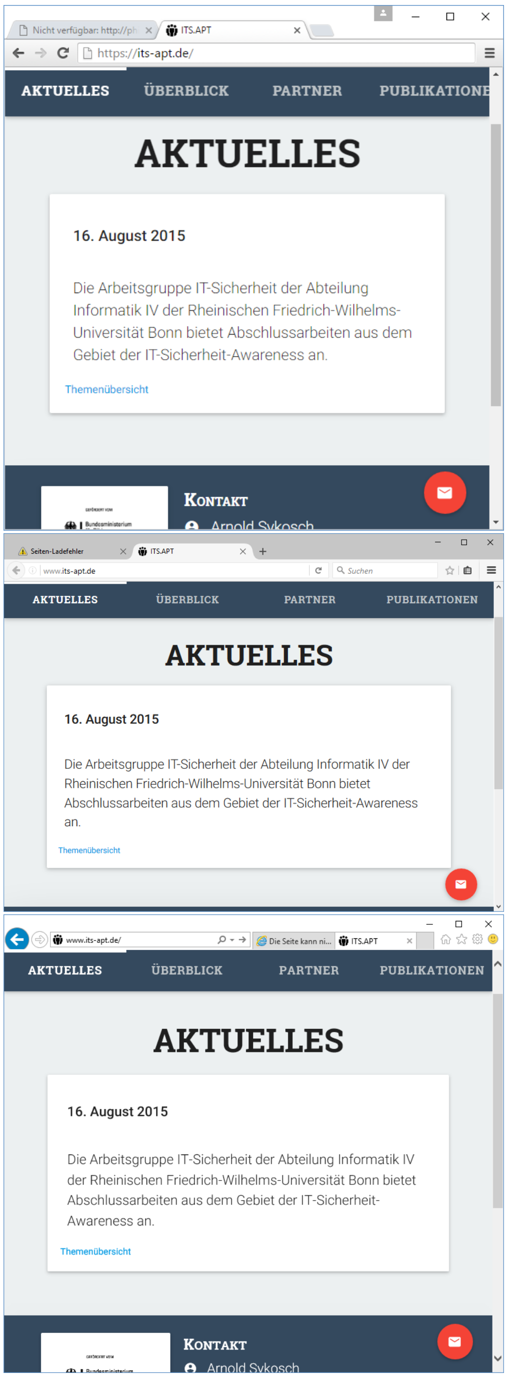
5.8 Evaluation of Generalized Service Infrastructure

Both provided infrastructure containers are set up and their specific use case in a HPT is used to evaluate their functionality. The mail container is used to send an e-mail which could be a phishing e-mail as part of an IT security awareness test. A phishing e-mail could contain a link to a prepared phishing website used to further probe a subject's reaction to a website. A typical phishing website is used to evaluate the webserver container.

5.8.1 Evaluation of Mail Service

To evaluate the e-mail sender's infrastructure the provided Dockerfile is used to build an *itsapemailer* container. A container instance of this image is started and a folder with mail files is linked. The container's sendmail program is used to send an e-mail to a valid e-mail address. Used commands are depicted in Figure 33.

The e-mail message body was received within the same minute, the mail body is depicted in Figure 34. It contains correct sender, subject and message. Sendmail used a TLS secured connection to deliver the provided message.

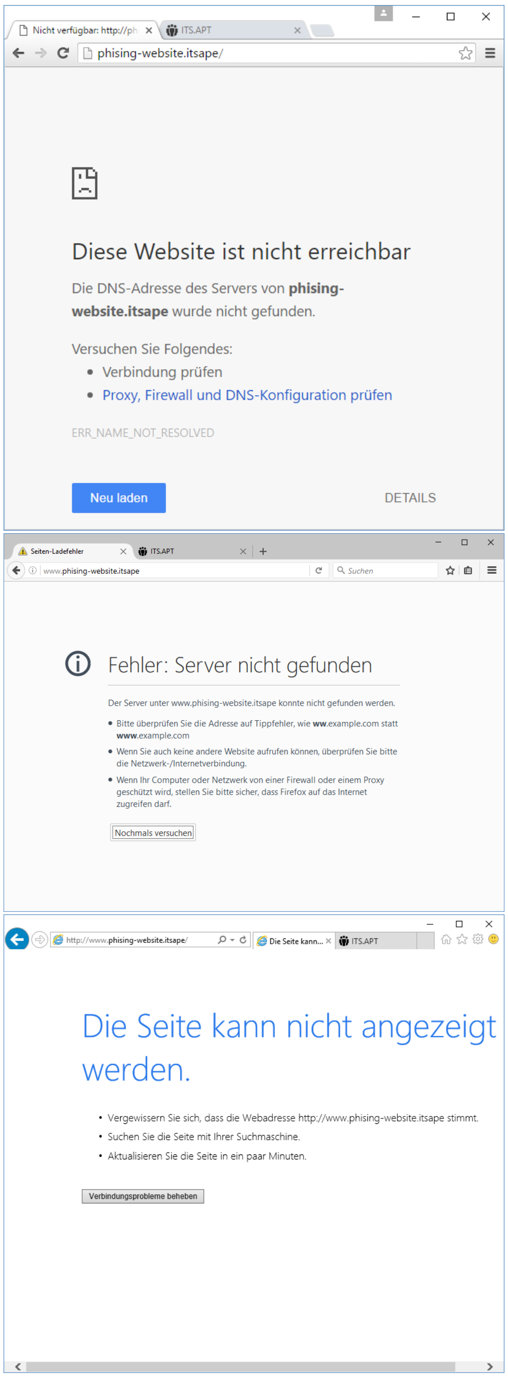


AKTUELLES

16. August 2015

Die Arbeitsgruppe IT-Sicherheit der Abteilung Informatik IV der Rheinischen Friedrich-Wilhelms-Universität Bonn bietet Abschlussarbeiten aus dem Gebiet der IT-Sicherheit-Awareness an.

Themenübersicht



Nicht verfügbar: http://ph
ITS.APT
phising-website.itsape/
Seiten-Ladefehler
www.phising-website.itsape
Suchen
http://www.phising-website.itsape/
Die Seite kann...

Diese Website ist nicht erreichbar

Die DNS-Adresse des Servers von **phising-website.itsape** wurde nicht gefunden.

Versuchen Sie Folgendes:

- Verbindung prüfen
- Proxy, Firewall und DNS-Konfiguration prüfen

ERR_NAME_NOT_RESOLVED

Neu laden DETAILS

Fehler: Server nicht gefunden

Der Server unter www.phising-website.itsape konnte nicht gefunden werden.

- Bitte überprüfen Sie die Adresse auf Tippfehler, wie **ww**.example.com statt **www**.example.com
- Wenn Sie auch keine andere Website aufrufen können, überprüfen Sie bitte die Netzwerk-/Internetverbindung.
- Wenn Ihr Computer oder Netzwerk von einer Firewall oder einem Proxy geschützt wird, stellen Sie bitte sicher, dass Firefox auf das Internet zugreifen darf.

Nochmals versuchen

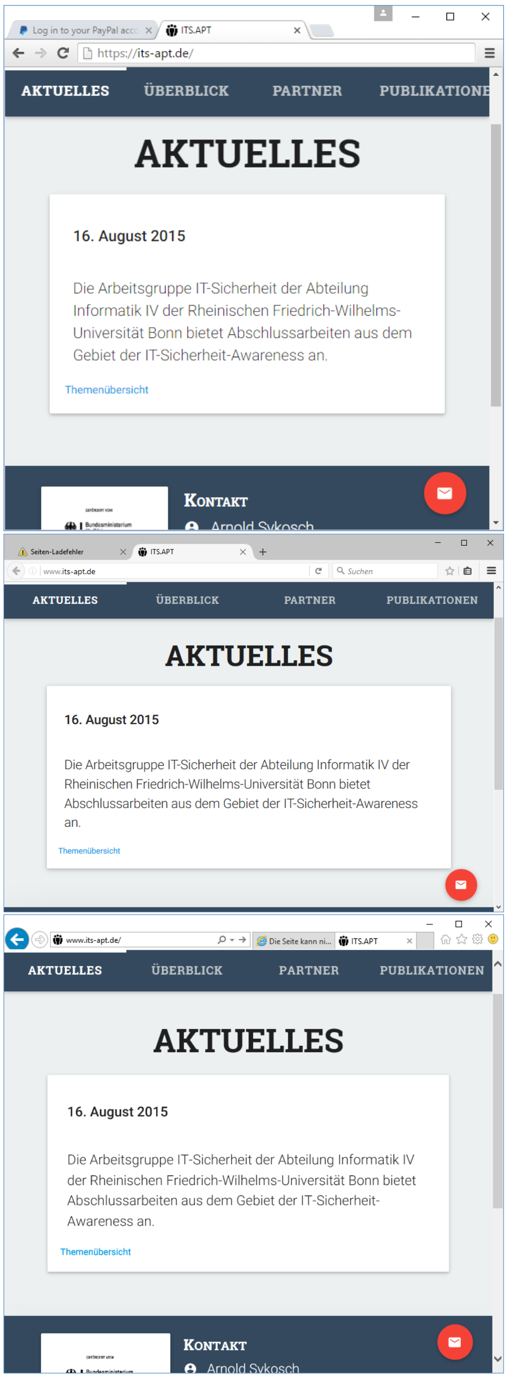
Die Seite kann nicht angezeigt werden.

- Vergewissern Sie sich, dass die Webadresse http://www.phising-website.itsape stimmt.
- Suchen Sie die Seite mit Ihrer Suchmaschine.
- Aktualisieren Sie die Seite in ein paar Minuten.

Verbindungsprobleme beheben

Figure 22: *Test Stage 1. & 4. Requesting* `its-apt.de` *from Alice's, Bob's and Charlie's machine in test stage one is successful.*

Figure 23: *Test Stage 1. & 4. Requesting* `phising-website.itsape` *from any machine in test stage two results in an error.*

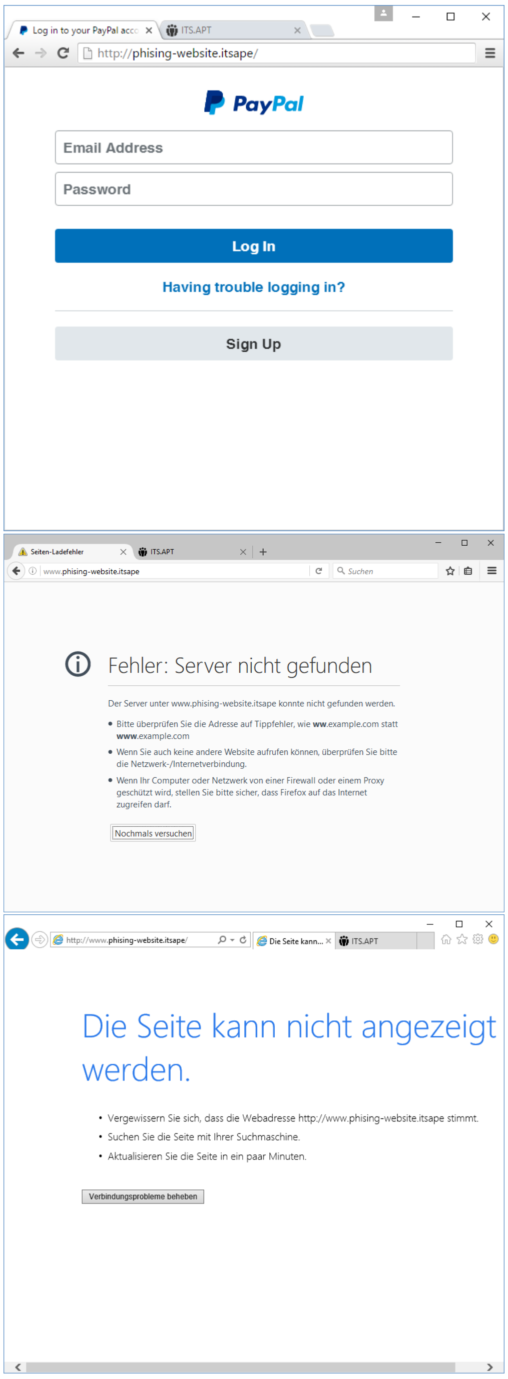


AKTUELLES

16. August 2015

Die Arbeitsgruppe IT-Sicherheit der Abteilung Informatik IV der Rheinischen Friedrich-Wilhelms-Universität Bonn bietet Abschlussarbeiten aus dem Gebiet der IT-Sicherheit-Awareness an.

Themenübersicht


Log in to your PayPal acc
ITS.APT
http://phising-website.itsape/
PayPal
Email Address
Password
Log In
Having trouble logging in?
Sign Up
Seiten-Ladefehler
www.phising-website.itsape
Suchen
Nochmals versuchen
http://www.phising-website.itsape/
Die Seite kann...
Verbindungsprobleme beheben

Fehler: Server nicht gefunden

Der Server unter www.phising-website.itsape konnte nicht gefunden werden.

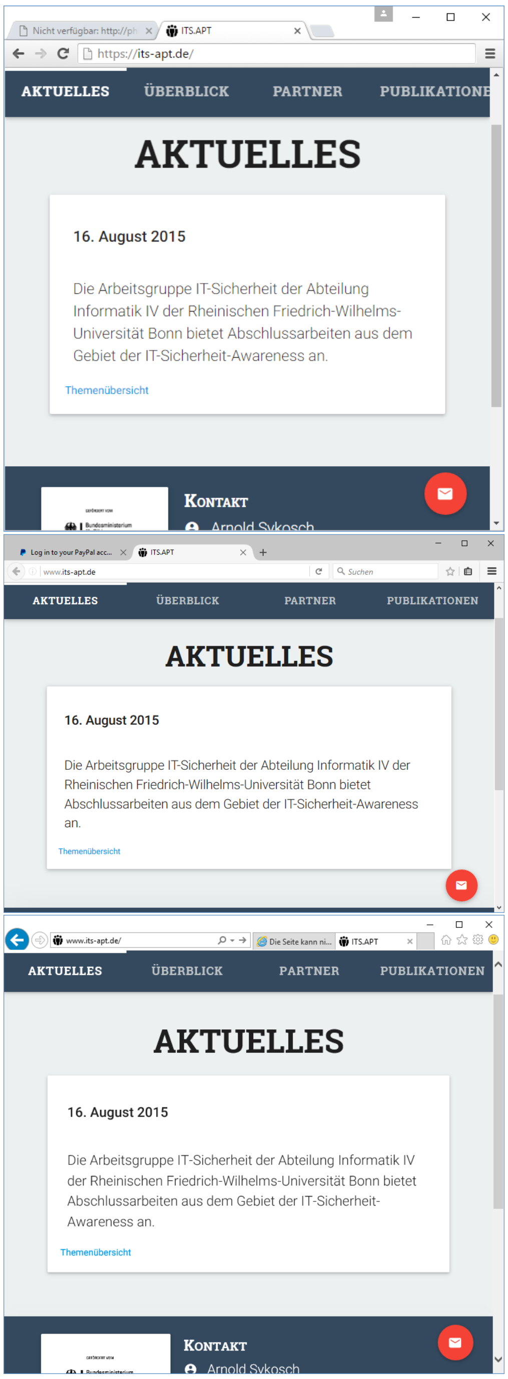
- Bitte überprüfen Sie die Adresse auf Tippfehler, wie **ww**.example.com statt **www**.example.com
- Wenn Sie auch keine andere Website aufrufen können, überprüfen Sie bitte die Netzwerk-/Internetverbindung.
- Wenn Ihr Computer oder Netzwerk von einer Firewall oder einem Proxy geschützt wird, stellen Sie bitte sicher, dass Firefox auf das Internet zugreifen darf.

Die Seite kann nicht angezeigt werden.

- Vergewissern Sie sich, dass die Webadresse http://www.phising-website.itsape stimmt.
- Suchen Sie die Seite mit Ihrer Suchmaschine.
- Aktualisieren Sie die Seite in ein paar Minuten.

Figure 24: *Test Stage 2. Requesting* `its-apt.de` *from Alice's, Bob's and Charlie's machine in test stage two is successful.*

Figure 25: *Test Stage 2. Requesting* `phising-website.itsape` *from all machines; only Alice is able to perceive a phishing website artifact designed in [53].*

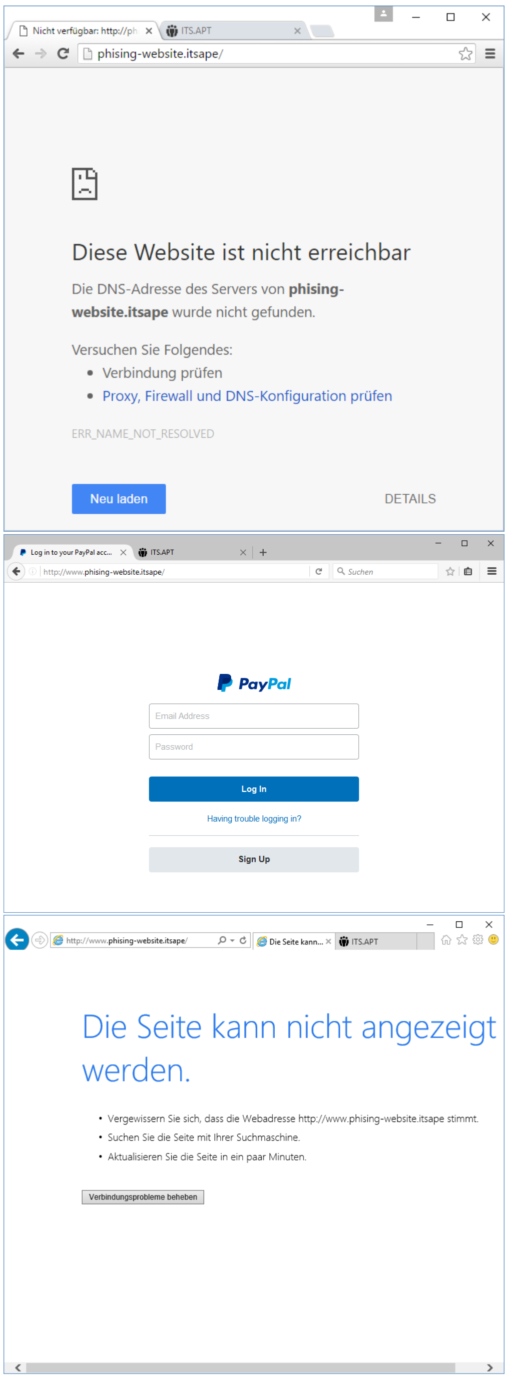


AKTUELLES

16. August 2015

Die Arbeitsgruppe IT-Sicherheit der Abteilung Informatik IV der Rheinischen Friedrich-Wilhelms-Universität Bonn bietet Abschlussarbeiten aus dem Gebiet der IT-Sicherheit-Awareness an.

Themenübersicht


Nicht verfügbar: http://ph
ITS.APT
phising-website.itsape/
Neu laden
DETAILS
Log in to your PayPal acc...
ITS.APT
http://www.phising-website.itsape/
Suchen
PayPal
Email Address
Password
Log In
Having trouble logging in?
Sign Up
http://www.phising-website.itsape/
Die Seite kann...
ITS.APT
Verbindungsprobleme beheben

Diese Website ist nicht erreichbar

Die DNS-Adresse des Servers von **phising-website.itsape** wurde nicht gefunden.

Versuchen Sie Folgendes:
- Verbindung prüfen
- Proxy, Firewall und DNS-Konfiguration prüfen

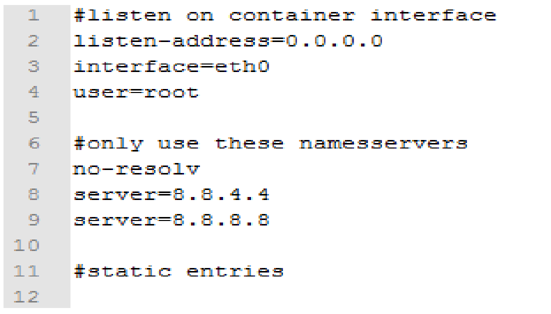
ERR_NAME_NOT_RESOLVED

Die Seite kann nicht angezeigt werden.

- Vergewissern Sie sich, dass die Webadresse http://www.phising-website.itsape stimmt.
- Suchen Sie die Seite mit Ihrer Suchmaschine.
- Aktualisieren Sie die Seite in ein paar Minuten.

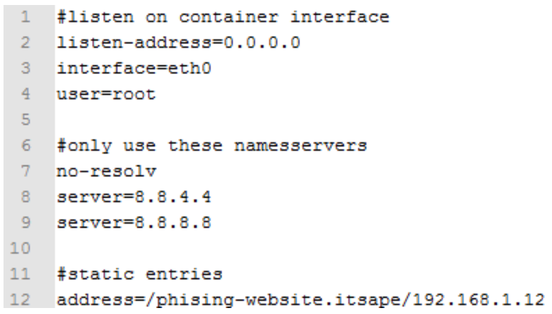
Figure 26: *Test Stage 3. Requesting* `its-apt.de` *from Alice's, Bob's and Charlie's machine in test stage three is successful.*

Figure 27: *Test Stage 3. Requesting* `phising-website.itsape` *from all but Bob's machine in test stage three results in an error. Bob is now able to perceive a phishing website artifact.*



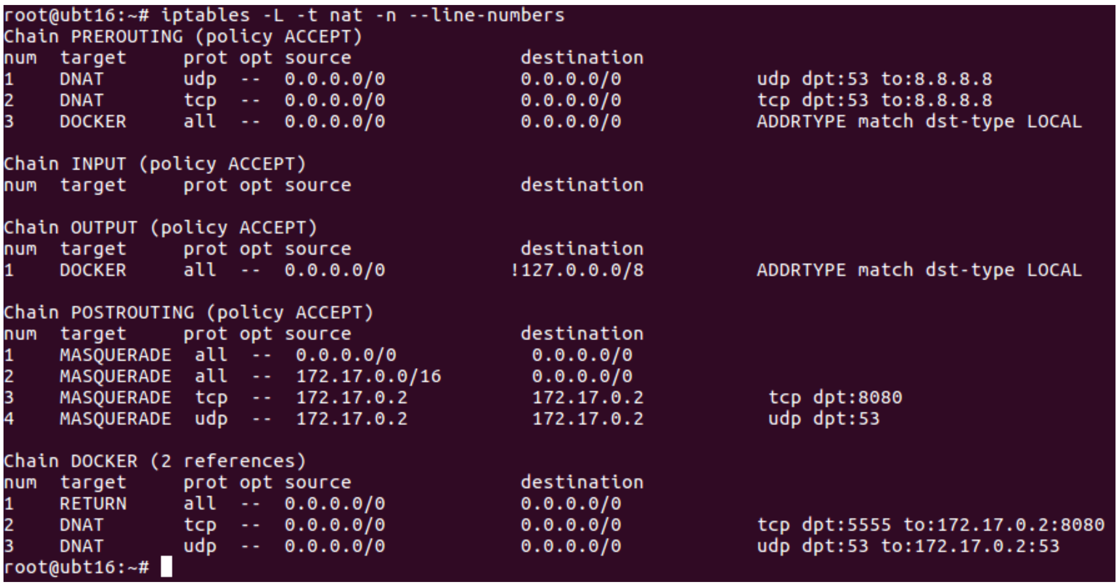
```
#listen on container interface
listen-address=0.0.0.0
interface=eth0
user=root

#only use these namesservers
no-resolv
server=8.8.4.4
server=8.8.8.8

#static entries

```

Figure 28: ***Test Stage 1. & 4.*** *Dnsmasq's configuration is not containing any static entries.*



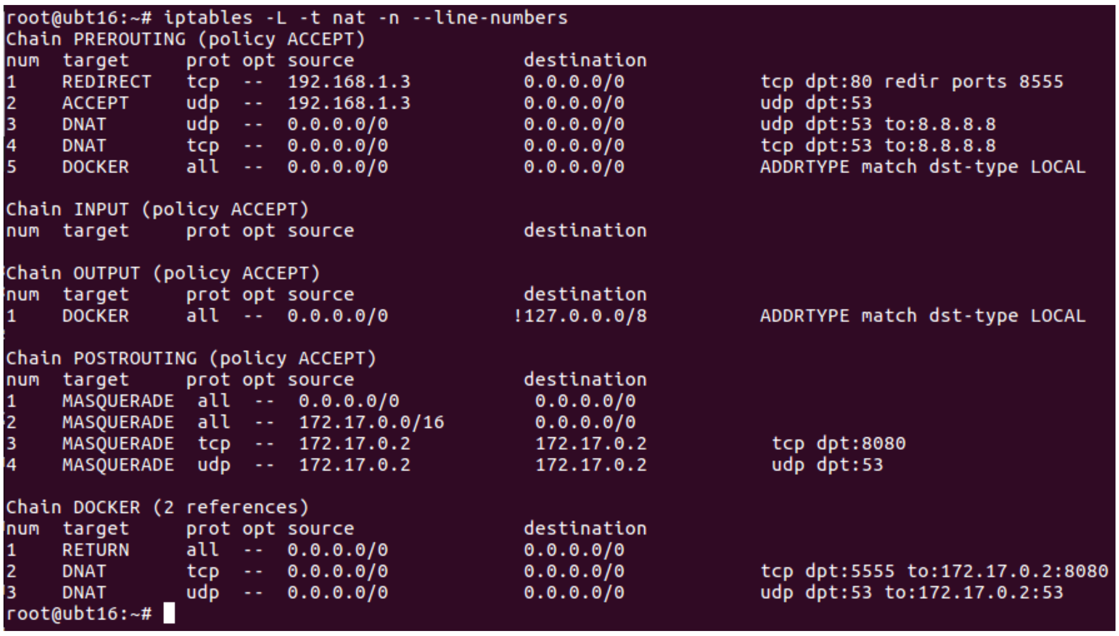
```
#listen on container interface
listen-address=0.0.0.0
interface=eth0
user=root

#only use these namesservers
no-resolv
server=8.8.4.4
server=8.8.8.8

#static entries
address=/phising-website.itsape/192.168.1.12
```

Figure 29: ***Test Stage 2. & 3.*** *The dnsmasq configuration responds to a name request for* `phising-website.itsape` *with the containers IP address.*



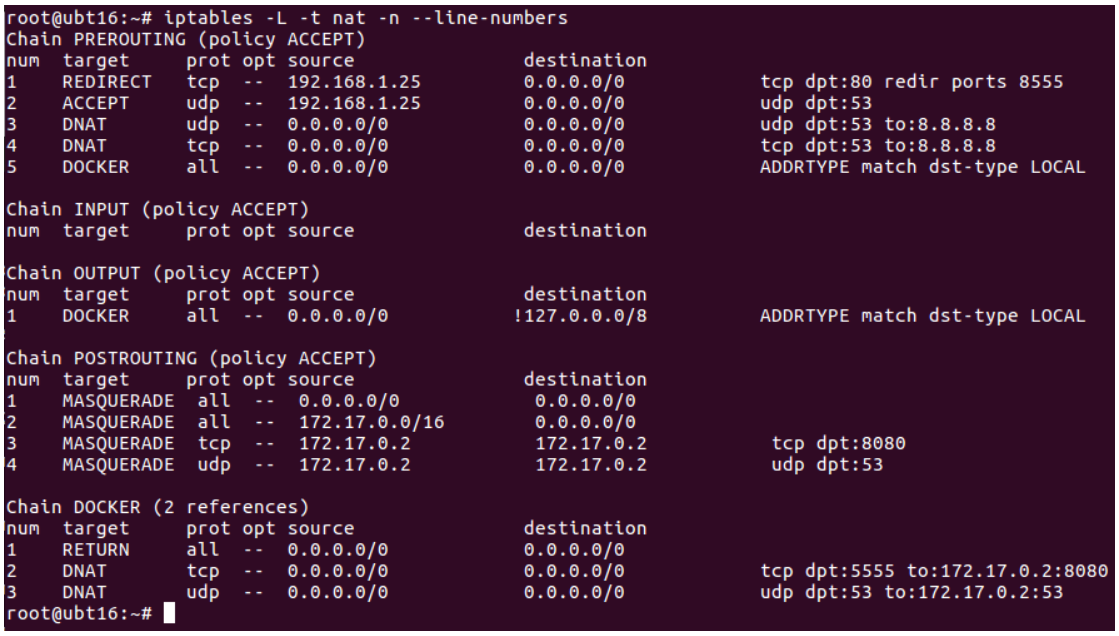
```
root@ubt16:~# iptables -L -t nat -n --line-numbers
Chain PREROUTING (policy ACCEPT)
num  target     prot opt source               destination
1    DNAT       udp  --  0.0.0.0/0            0.0.0.0/0            udp dpt:53 to:8.8.8.8
2    DNAT       tcp  --  0.0.0.0/0            0.0.0.0/0            tcp dpt:53 to:8.8.8.8
3    DOCKER     all  --  0.0.0.0/0            0.0.0.0/0            ADDRTYPE match dst-type LOCAL

Chain INPUT (policy ACCEPT)
num  target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
num  target     prot opt source               destination
1    DOCKER     all  --  0.0.0.0/0           !127.0.0.0/8          ADDRTYPE match dst-type LOCAL

Chain POSTROUTING (policy ACCEPT)
num  target     prot opt source               destination
1    MASQUERADE  all  --  0.0.0.0/0            0.0.0.0/0
2    MASQUERADE  all  --  172.17.0.0/16        0.0.0.0/0
3    MASQUERADE  tcp  --  172.17.0.2           172.17.0.2           tcp dpt:8080
4    MASQUERADE  udp  --  172.17.0.2           172.17.0.2           udp dpt:53

Chain DOCKER (2 references)
num  target     prot opt source               destination
1    RETURN     all  --  0.0.0.0/0            0.0.0.0/0
2    DNAT       tcp  --  0.0.0.0/0            0.0.0.0/0            tcp dpt:5555 to:172.17.0.2:8080
3    DNAT       udp  --  0.0.0.0/0            0.0.0.0/0            udp dpt:53 to:172.17.0.2:53
root@ubt16:~# 
```

Figure 30: ***Test Stage 1. & 4.*** *All DNS requests are routed to the networks standard DNS server* `8.8.8.8` *via PREROUTING rule 1, 2 and POSTROUTING rule 1.*



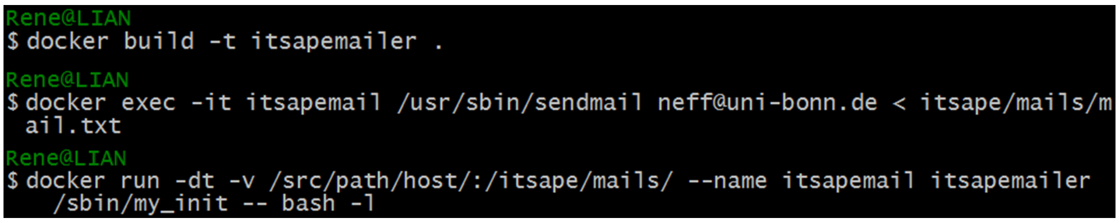
```
root@ubt16:~# iptables -L -t nat -n --line-numbers
Chain PREROUTING (policy ACCEPT)
num  target     prot opt source               destination
1    REDIRECT   tcp  --  192.168.1.3          0.0.0.0/0            tcp dpt:80 redir ports 8555
2    ACCEPT     udp  --  192.168.1.3          0.0.0.0/0            udp dpt:53
3    DNAT       udp  --  0.0.0.0/0            0.0.0.0/0            udp dpt:53 to:8.8.8.8
4    DNAT       tcp  --  0.0.0.0/0            0.0.0.0/0            tcp dpt:53 to:8.8.8.8
5    DOCKER     all  --  0.0.0.0/0            0.0.0.0/0            ADDRTYPE match dst-type LOCAL

Chain INPUT (policy ACCEPT)
num  target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
num  target     prot opt source               destination
1    DOCKER     all  --  0.0.0.0/0           !127.0.0.0/8          ADDRTYPE match dst-type LOCAL

Chain POSTROUTING (policy ACCEPT)
num  target     prot opt source               destination
1    MASQUERADE  all  --  0.0.0.0/0            0.0.0.0/0
2    MASQUERADE  all  --  172.17.0.0/16        0.0.0.0/0
3    MASQUERADE  tcp  --  172.17.0.2           172.17.0.2           tcp dpt:8080
4    MASQUERADE  udp  --  172.17.0.2           172.17.0.2           udp dpt:53

Chain DOCKER (2 references)
num  target     prot opt source               destination
1    RETURN     all  --  0.0.0.0/0            0.0.0.0/0
2    DNAT       tcp  --  0.0.0.0/0            0.0.0.0/0            tcp dpt:5555 to:172.17.0.2:8080
3    DNAT       udp  --  0.0.0.0/0            0.0.0.0/0            udp dpt:53 to:172.17.0.2:53
root@ubt16:~# 
```

Figure 31: ***Test Stage 2.*** *All DNS requests are routed to the networks standard DNS server 8.8.8.8. DNS requests from Alice (192.168.1.3) are forwarded to the ITS.APE DNS server. HTTP requests (Port 80) from her machine are forwarded to the ITS.APE webserver container on port 8555.*



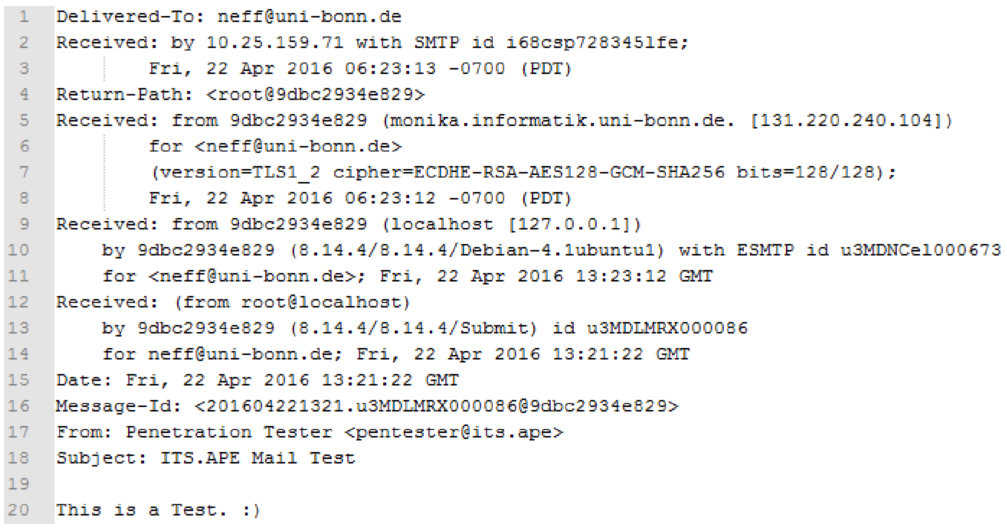
```
root@ubt16:~# iptables -L -t nat -n --line-numbers
Chain PREROUTING (policy ACCEPT)
num  target     prot opt source               destination
1    REDIRECT   tcp  --  192.168.1.25         0.0.0.0/0            tcp dpt:80 redir ports 8555
2    ACCEPT     udp  --  192.168.1.25         0.0.0.0/0            udp dpt:53
3    DNAT       udp  --  0.0.0.0/0            0.0.0.0/0            udp dpt:53 to:8.8.8.8
4    DNAT       tcp  --  0.0.0.0/0            0.0.0.0/0            tcp dpt:53 to:8.8.8.8
5    DOCKER     all  --  0.0.0.0/0            0.0.0.0/0            ADDRTYPE match dst-type LOCAL

Chain INPUT (policy ACCEPT)
num  target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
num  target     prot opt source               destination
1    DOCKER     all  --  0.0.0.0/0           !127.0.0.0/8          ADDRTYPE match dst-type LOCAL

Chain POSTROUTING (policy ACCEPT)
num  target     prot opt source               destination
1    MASQUERADE  all  --  0.0.0.0/0            0.0.0.0/0
2    MASQUERADE  all  --  172.17.0.0/16        0.0.0.0/0
3    MASQUERADE  tcp  --  172.17.0.2           172.17.0.2           tcp dpt:8080
4    MASQUERADE  udp  --  172.17.0.2           172.17.0.2           udp dpt:53

Chain DOCKER (2 references)
num  target     prot opt source               destination
1    RETURN     all  --  0.0.0.0/0            0.0.0.0/0
2    DNAT       tcp  --  0.0.0.0/0            0.0.0.0/0            tcp dpt:5555 to:172.17.0.2:8080
3    DNAT       udp  --  0.0.0.0/0            0.0.0.0/0            udp dpt:53 to:172.17.0.2:53
root@ubt16:~# 
```

Figure 32: ***Test Stage 3.*** *All DNS requests are routed to the networks standard DNS server 8.8.8.8. DNS requests from Bob (192.168.1.25) are forwarded to the ITS.APE DNS server. HTTP requests (Port 80) from his machine are forwarded to the ITS.APE webserver container on port 8555.*



```
Rene@LIAN
$ docker build -t itsapemailer .

Rene@LIAN
$ docker exec -it itsapemail /usr/sbin/sendmail neff@uni-bonn.de < itsape/mails/m
  ail.txt
Rene@LIAN
$ docker run -dt -v /src/path/host/:/itsape/mails/ --name itsapemail itsapemailer
    /sbin/my_init -- bash -l
```

Figure 33: *Commands to send an e-mail with the provided mail sender container, which is first build and started.*



```
Delivered-To: neff@uni-bonn.de
Received: by 10.25.159.71 with SMTP id i68csp728345lfe;
        Fri, 22 Apr 2016 06:23:13 -0700 (PDT)
Return-Path: <root@9dbc2934e829>
Received: from 9dbc2934e829 (monika.informatik.uni-bonn.de. [131.220.240.104])
        for <neff@uni-bonn.de>
        (version=TLS1_2 cipher=ECDHE-RSA-AES128-GCM-SHA256 bits=128/128);
        Fri, 22 Apr 2016 06:23:12 -0700 (PDT)
Received: from 9dbc2934e829 (localhost [127.0.0.1])
    by 9dbc2934e829 (8.14.4/8.14.4/Debian-4.1ubuntu1) with ESMTP id u3MDNCe1000673
    for <neff@uni-bonn.de>; Fri, 22 Apr 2016 13:23:12 GMT
Received: (from root@localhost)
    by 9dbc2934e829 (8.14.4/8.14.4/Submit) id u3MDLMRX000086
    for neff@uni-bonn.de; Fri, 22 Apr 2016 13:21:22 GMT
Date: Fri, 22 Apr 2016 13:21:22 GMT
Message-Id: <201604221321.u3MDLMRX000086@9dbc2934e829>
From: Penetration Tester <pentester@its.ape>
Subject: ITS.APE Mail Test

This is a Test. :)
```

Figure 34: *RFC822 Message body of received test e-mail.*

5.8.2 Evaluation of Website Service

The provided webserver image is created in the same way as the mail server infrastructure. An instance of this container is started exposed to port 8555, a folder is linked into the container. Said folder contains a phishing website used as an artifact presented in [53].

From a different computer within the same network a *curl* process is invoked. It requests a website from the containers machine, said website is expected to be the provided phishing website. A HTTP response status code 200 is expected.

A browser is used to request the website; it is displayed as expected and shown in Figure 35. The received website's hash sum is compared with the known phishing website's hash sum. Both hash sums are identical and therefore website artifact delivery with the provided website infrastructure is proven to be functional. The *curl* command and hash sum comparison are shown in Figure 36. The container's Apache access log file is checked to reflect the recent request. Figure 37 shows that the expected request is recorded.

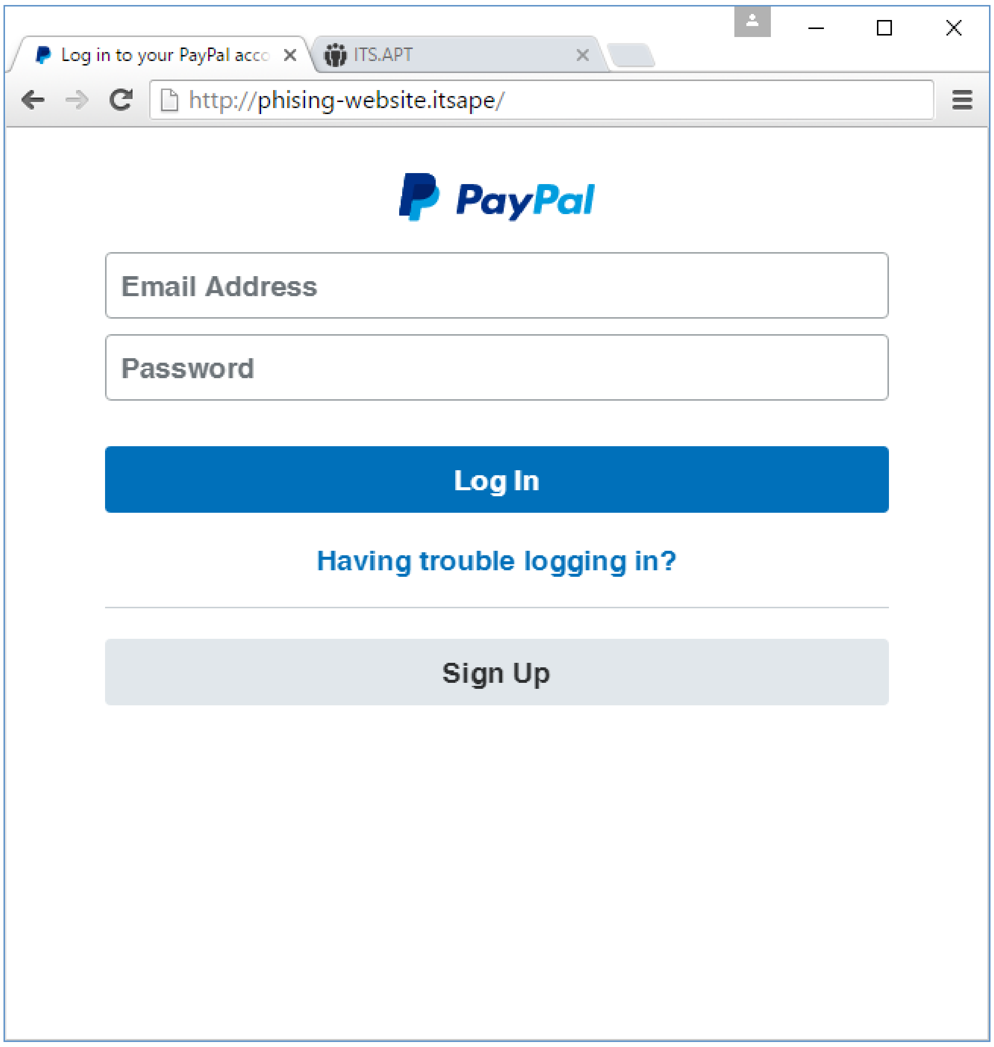
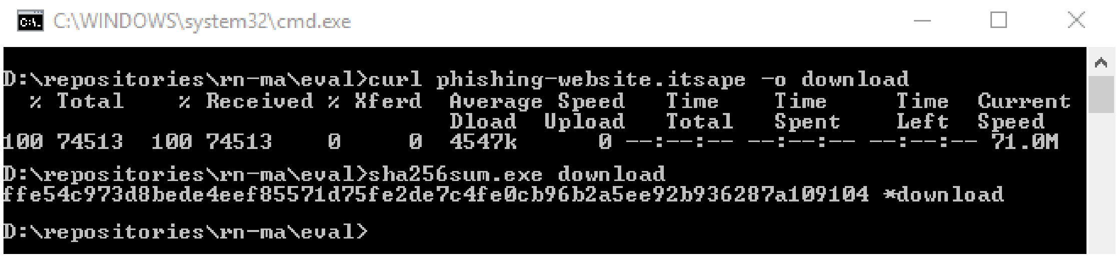
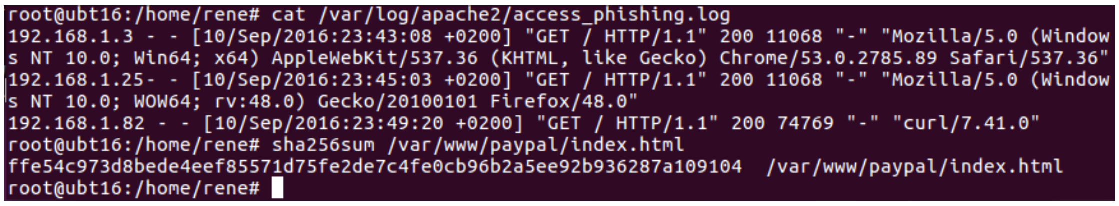

Log in to your PayPal acco
ITS.APT
http://phising-website.itsape/
PayPal
Email Address
Password
Log In
Having trouble logging in?
Sign Up

Figure 35: *The received phishing website usable as artifact in a HPT.*



```
D:\repositories\rn-ma\eval>curl phishing-website.itsape -o download
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 74513  100 74513    0     0  4547k      0 --:--:-- --:--:-- --:--:-- 71.0M

D:\repositories\rn-ma\eval>sha256sum.exe download
ffe54c973d8bede4eef85571d75fe2de7c4fe0cb96b2a5ee92b936287a109104 *download

D:\repositories\rn-ma\eval>
```

Figure 36: *Commands used to request and compare the provided website.*



```
root@ubt16:/home/rene# cat /var/log/apache2/access_phishing.log
192.168.1.3 - - [10/Sep/2016:23:43:08 +0200] "GET / HTTP/1.1" 200 11068 "-" "Mozilla/5.0 (Window
s NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/53.0.2785.89 Safari/537.36"
192.168.1.25- - [10/Sep/2016:23:45:03 +0200] "GET / HTTP/1.1" 200 11068 "-" "Mozilla/5.0 (Window
s NT 10.0; WOW64; rv:48.0) Gecko/20100101 Firefox/48.0"
192.168.1.82 - - [10/Sep/2016:23:49:20 +0200] "GET / HTTP/1.1" 200 74769 "-" "curl/7.41.0"
root@ubt16:/home/rene# sha256sum /var/www/paypal/index.html
ffe54c973d8bede4eef85571d75fe2de7c4fe0cb96b2a5ee92b936287a109104  /var/www/paypal/index.html
root@ubt16:/home/rene#
```

Figure 37: *Apache access log of the webserver container.*

6 Summary & Future Work

This master thesis presented an extension for infrastructure provisioning for the ITS.APE framework. The provided extension simplifies the work of the penetration tester to perform a *Human Penetration Test (HPT)*. Required Infrastructure can be defined via text based configuration and generated containers are consolidated if possible. Compared to the legacy script-based infrastructure generation process, a resource-efficient management is provided. Furthermore, the extension consists of a routing system which supports automated traffic redirection for test element presentation.

To provide necessary background, this thesis presented the idea of *Human Penetration Testing (HPT)* and set out goals of the ITS.APT project. The *Social-Engineer Toolkit (SET)* was identified to provide only rudimentary tools, useable to perform a HPT. IT services present in an office environment are identified and their network protocols are presented. A comparison between different virtualization techniques provides feasible background for later design decisions. Detailed descriptions of ITS.APE framework components are given and allow to identify relevant models and modules used to provide this thesis' extension.

A requirement analysis is performed and resulted in ten requirements to meet for the desired extension.

An extension design based on software design patterns and conform to the identified requirements was developed and presented. It includes the design of a redirect system, used to ensure test conform exposure of subjects to test elements. Furthermore, designs of three concrete services, used for HPTs, were provided and explained.

During implementation of the designed class-based system, the idea for a more generic system came up. The former system was revised and a type-based infrastructure scheme was designed. The type-based implementation was finalized and supports the definition of infrastructure via a configuration file. Additionally, it allows for a resource saving consolidation of similar *generic* infrastructure demands by a single container.

The implemented extension is capable of managing HPT infrastructure with a full support of legacy script-based infrastructure management. Configuration can be handled via black- and whitelisting of managed infrastructure types. A simple configuration, which requires only four parameter stated in a text based configuration file can be used to define new managed infrastructure. Managed infrastructure is based on Docker containers, implemented through a replaceable container class. If compatible commissioned infrastructure demands are consolidated with already running containers. Containers support adding and removing of test elements of a specific HPT, even for consolidated containers. The implemented redirect system uses IP and DNS based routing, it is capable of routing specific subjects to designated infrastructure based on a provided schedule.

Two manageable infrastructure compositions were provided and tested to show their utilization in a HPT test scenario. The implemented routing system was evaluated to provide precise traffic manipulation of subject's traffic. It was further tested to form no obstacle to other user's Internet use. Evaluation of the implemented extension confirmed compliance to the formulated requirements. This includes RSpec unit tests of implemented and extended legacy models and modules, a fully described test environment and test results of performed test cases. The following outlook on future work concludes this master thesis.

Future Work

A broad spectrum for future development in context of HPT and the ITS.APE framework exists, starting from new kinds of artifacts beside the presented e-mail and website type. These new artifacts might require additional infrastructure types. Especially, provisioning of executable or other file types as offered by the designed and described *File Sharing* infrastructure container allows for new kind of artifacts. Other infrastructure could be used to target subjects not only via their computer but for example by sending SMS which include a phishing link. For the provided webserver container a modified version to support secured request provided by TLS could be created. Automated generation of certificates and keys could be made part of the container script.

Future development of the framework and IME could improve port and IP provisioning. Therefore, available ports have to be provided to define a predefined container within the configuration of IME. This hard requirement could be softened to support automatic selection of an internal port by the framework. A check whether this port is freely available and a conform handling of subsequent redirecting operations are necessary. If a HPT is performed in a large company network separation might be desired. The IME uses the first interface's IP address for its communication, a future version could use a configured IP interface and IP address instead. This IP address could be stored in the test's *Test Season* and thereby a specific IP address could be used for each HPT instance.

The current implementation uses always the first container from the list of suited containers, but additional parameters could be used to make a more sophisticated selection. An improvement to the consolidation process could involve re-consolidation. The implemented process only checks for possible consolidations for a newly commissioned infrastructure onto running containers. A complex type of infrastructure is very unlikely to find a suitable match, therefore a new container is generated and used. In addition to the existing check, a check for consolidation of running containers into other running containers could be added. This check must ensure a seamlessly artifact delivery throughout the relocation of served *Infrastructure Elements*.

The IME prepares and runs all infrastructure on the ITS.APE host system. Within a future version designated container hosts might exist. An adapted *ITSAPE Container* module can interact with remote Docker containers, which would allow for load balancing this system and could also support external containers. These external containers are placed in the penetration tester's controlled colocation center. Consequently, only a reduced version of the ITS.APE framework would be required at the tested company's location.

The designed routing system could be extended to support deep packet inspection. Thereby a single satellite installed in this enhanced routing system could track all subject reactions. As all company's DNS traffic is expected to be handled via the routing system this might be a violation of privacy rights.

The presented IME is easily integrated into the existing framework, yet source code manipulation is required. A generic extension system, easily configured via a configuration file, could pave the way for new extensions. Extensions based on the principle of exchanging and extending legacy framework modules could be provided with just one configuration line.

Bibliography

[1] *A minimal Ubuntu base image modified for Docker-friendliness: phusion/baseimage*. https://phusion.github.io/baseimage-docker/; last accessed 05. September 2016. Sept. 2016.

[2] *Akamai - IPv6 Adoption visualization - Data last updated: 24.04.2016*. https://www.akamai.com/de/de/our-thinking/state-of-the-internet-report/state-of-the-internet-ipv6-adoption-visualization.jsp; last accessed 05. June 2016. June 2016.

[3] Bagnulo et al. *Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers*. RFC 6146. IETF, Apr. 2011. URL: https://tools.ietf.org/html/rfc6146.

[4] Sheffer et al. *Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS)*. RFC 7457. IETF, Feb. 2015. URL: https://tools.ietf.org/html/rfc7457.

[5] *Amit Singh - kernelthread.com - "An Introduction to Virtualization"*. http://www.kernelthread.com/publications/virtualization/; last accessed 14. June 2016. June 2016.

[6] P. Borgnat et al. “Seven Years and One Day: Sketching the Evolution of Internet Traffic”. In: *INFOCOM 2009, IEEE*. Apr. 2009, pp. 711–719. DOI: 10.1109/INFCOM.2009.5061979.

[7] Robert Braden. *Requirements for Internet Hosts – Communication Layers*. RFC 1122. RFC Editor, Oct. 1989. URL: http://www.rfc-editor.org/rfc/rfc1122.txt.

[8] Peter F. Brown and Rebekah Metz Booz Allen Hamilton. *Official OASIS Standard: Reference Model for Service Oriented Architecture 1.0*. 2006.

[9] *Bundesamt für Sicherheit in der Informationstechnik - Allianz für Cyber-Sicherheit - "Cyber-Sicherheits-Umfrage 2015"*. https://www.allianz-fuer-cybersicherheit.de/ACS/DE/_/downloads/cybersicherheitslage/umfrage2015_ergebnisse.pdf?__blob=publicationFile&v=4; last accessed 16. June 2016. June 2016.

[10] *Can I use... Support tables for HTML5, CSS3, etc Website - "HTTP/2 protocol"*. http://caniuse.com/#search=http2; last accessed 13. June 2016. June 2016.

[11] *Christopher Hertel - Implementing CIFS: The Common Internet File System, Published Aug 11, 2003 by Prentice Hall*. http://www.ubiqx.org/cifs/; last accessed 08. June 2016. June 2016.

[12] *darkhttpd Project Website*. https://unix4lyfe.org/darkhttpd/; last accessed 02. June 2016. June 2016.

[13] Network Working Group Deering & Hinden. *Internet Protocol, Version 6 (IPv6) Specification*. RFC 2460. IETF, Dec. 1998. URL: https://tools.ietf.org/html/rfc2460.

[14] *Docker Inc. - "Docker Engine"*. https://www.docker.com/products/docker-engine; last accessed 14. June 2016. June 2016.

[15] *Docker Inc. - "Docker Hub"*. https://hub.docker.com/explore/; last accessed 14. June 2016. June 2016.

[16] *Docker Inc. - "Docker Remote API"*. https://docs.docker.com/engine/reference/api/docker_remote_api/; last accessed 14. June 2016. June 2016.

[17] Network Working Group Durand et al. *IPv6 Tunnel Broker*. RFC 3053. IETF, Jan. 2001. URL: https://tools.ietf.org/html/rfc6146.

[18] *FastMail Pty Ltd. "SSL vs TLS vs STARTTLS"*. https://www.fastmail.com/help/technical/ssltlsstarttls.html; last accessed 08. June 2016. June 2016.

[19] Alessandro Finamore and Konstantina Papagiannaki. "Is the Web HTTP/2 Yet?" In: *Passive and Active Measurement: 17th International Conference, PAM 2016, Heraklion, Greece, March 31-April 1, 2016. Proceedings*. Vol. 9631. Springer. 2016, p. 218.

[20] Erich Gamma et al. *Design Patterns: Elements of Reusable Object-oriented Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995. ISBN: 0201633612.

[21] *GitHub - Social-Engineer Toolkit*. https://github.com/trustedsec/social-engineer-toolkit; last accessed 19. June 2016. June 2016.

[22] *Github swipely/docker-api - "A lightweight Ruby client for the Docker Remote API"*. https://github.com/swipely/docker-api; last accessed 14. June 2016. June 2016.

[23] *Google IPv6 - Statistics*. https://www.google.com/intl/en/ipv6/statistics.html; last accessed 05. June 2016. June 2016.

[24] Charles David Graziano. "A performance analysis of Xen and KVM hypervisors for hosting the Xen Worlds Project". In: (2011).

[25] Crispin - Network Working Group. *INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1*. RFC 3501. IETF, Mar. 2003. URL: https://tools.ietf.org/html/rfc3501.

[26] Dierks & Allen - Network Working Group. *The TLS Protocol - Version 1.0*. RFC 2246. IETF, Jan. 1999. URL: https://tools.ietf.org/html/rfc2246.

[27] Galbraith & Saarenmaa - Secure Shell Working Group. *SSH File Transfer Protocol - draft-ietf-secsh-filexfer-13*. RFC. IETF, July 2006. URL: https://tools.ietf.org/id/draft-ietf-secsh-filexfer-13.txt.

[28] Hoffman - Network Working Group. *SMTP Service Extension for Secure SMTP over Transport Layer Security*. RFC 3207. IETF, Feb. 2002. URL: https://www.ietf.org/rfc/rfc3207.txt.

[29] Klensin - Network Working Group. *Simple Mail Transfer Protocol*. RFC 5321. IETF, Oct. 2008. URL: https://tools.ietf.org/html/rfc5321.

[30] Myers & Rose - Network Working Group. *Post Office Protocol - Version 3*. RFC 1939. IETF, May 1996. URL: https://tools.ietf.org/html/rfc1939.

[31] Newman - Network Working Group. *Using TLS with IMAP, POP3 and ACAP*. RFC 2595. IETF, June 1999. URL: https://tools.ietf.org/html/rfc2595.

[32] Postel - Network Working Group. *Simple Mail Transfer Protocol*. RFC 821. IETF, Aug. 1982. URL: https://tools.ietf.org/html/rfc821.

[33] Rescorla - Network Working Group. *The Transport Layer Security (TLS) Protocol Version 1.3 draft-ietf-tls-tls13-13)*. RFC. IETF, May 2016. URL: https://tools.ietf.org/html/draft-ietf-tls-tls13-13.

[34] Resnick - Network Working Group. *Internet Message Format*. RFC 5322. IETF, Oct. 2008. URL: https://tools.ietf.org/html/rfc5322.

[35] SNIA CIFS Technical Work Group. *Common Internet File System (CIFS) Technical Reference Revision: 1.0 - SNIA Technical Proposal*. SNIA-DRAFT. SNIA, Feb. 2002. URL: https://www.thursby.com/sites/default/files/files/CIFS-TR-1p00_FINAL.pdf.

[36] *Heise Security - Locky*. http://www.heise.de/security/meldung/Erpressungs-Trojaner-Locky-schlaegt-offenbar-koordiniert-zu-3104069.html; last accessed 12. July 2016. July 2016.

[37] *ICANN, nanog mailing list, Leo Vegoda. "Five /8s allocated to RIRs – no unallocated IPv4 unicast /8s remain"*. https://mailman.nanog.org/pipermail/nanog/2011-February/032105.html; last accessed 08. June 2016. June 2016.

[38] University of Southern California Information Sciences Institute. *INTERNET PROTOCOL - DARPA INTERNET PROGRAM - PROTOCOL SPECIFICATION*. RFC 791. IETF, Sept. 1981. URL: https://tools.ietf.org/html/rfc791.

[39] *IPv6.com Inc, Kaushik Das. "IPv6 - The History and Timeline"*. http://www.ipv6.com/articles/general/timeline-of-ipv6.htm; last accessed 08. June 2016. June 2016.

[40] Mockapetris - ISI. *DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION*. RFC 1035. IETF, Nov. 1987. URL: https://tools.ietf.org/html/rfc1035.

[41] *ITS.APT: IT-Security Awareness Penetration Testing D1.1 - "State of the Art in IT Security Awareness Testing"*. https://tools.net.cs.uni-bonn.de/git/its-wg/itsapt/tree/master/AP1%20-%20Menschliche%20Faktoren; last accessed 01. June 2016. May 2016.

[42] *ITS.APT: IT-Security Awareness Penetration Testing - D1.2: ITS-APE-FRAMEWORK-KONZEPT*. https://tools.net.cs.uni-bonn.de/git/its-wg/its.apt/tree/master/framework/itsapt; last accessed 01. June 2016. May 2016.

[43] *ITS.APT: IT-Security Awareness Penetration Testing - Software Framework Repository.* https://tools.net.cs.uni-bonn.de/git/its-wg/its.apt/tree/master/framework/itsapt; last accessed 01. June 2016. June 2016.

[44] *ITS.APT: IT-Security Awareness Penetration Testing - Teilvorhabenbeschreibung und weitere interne Dokumentation.* https://tools.net.cs.uni-bonn.de/git/its-wg/itsapt/tree/master; last accessed 01. June 2016. May 2016.

[45] *ITS.APT: IT-Security Awareness Penetration Testing - UKSH Report on Services.* https://tools.net.cs.uni-bonn.de/git/its-wg/itsapt/tree/master; last accessed 01. June 2016. May 2016.

[46] *Kostya Kortchinsky - Immunity, Inc. - "CLOUDBURST - A VMware Guest to Host Escape Story".* https://www.blackhat.com/presentations/bh-usa-09/KORTCHINSKY/BHUSA09-Kortchinsky-Cloudburst-SLIDES.pdf; last accessed 14. June 2016. June 2016.

[47] Mills - COMSAT Laboratories. *Internet Name Domains.* RFC 799. IETF, Sept. 1981. URL: https://tools.ietf.org/html/rfc791.

[48] Naik - Network Working Group Leach. *A Common Internet File System (CIFS/1.0) Protocol - Preliminary Draft.* RFC-DRAFT. IETF, Dec. 1997. URL: https://tools.ietf.org/html/draft-leach-cifs-v1-spec-01.

[49] *Lighttpd Project Website.* https://www.lighttpd.net/; last accessed 02. June 2016. June 2016.

[50] *Linux Containers Project - "Linux Containers - LXC - Introduction".* https://linuxcontainers.org/lxc/introduction/; last accessed 14. June 2016. June 2016.

[51] *Linux VServer Project - "Linux-VServer.org Homepage".* http://linux-vserver.org/Welcome_to_Linux-VServer.org; last accessed 14. June 2016. June 2016.

[52] *M. R. Endsley: Toward a Theory of Situation Awareness in Dynamic Systems. In: Human Factors: The Journal of the Human Factors and Ergonomics Society 37 (1995).* 1995.

[53] Thomas Maqua. “TITLE”. not yet published. Master’s Thesis. Germany: University of Bonn, 2016.

[54] *Metasploit - Penetration Testing Software.* https://metasploit.com/; last accessed 19. June 2016. June 2016.

[55] *Microsoft.com, MSDN - Microsoft SMB Protocol and CIFS Protocol Overview.* https://msdn.microsoft.com/en-us/library/windows/desktop/aa365233(v=vs.85).aspx; last accessed 08. June 2016. June 2016.

[56] *Microsoft.com, MSDN - [MS-CIFS]: Common Internet File System (CIFS) Protocol.* https://msdn.microsoft.com/en-us/library/ee442092.aspx; last accessed 08. June 2016. June 2016.

[57] *Microsoft.com, MSDN - [MS-SMB2]: Server Message Block (SMB) Protocol Versions 2 and 3.* https://msdn.microsoft.com/en-us/library/cc246482.aspx; last accessed 08. June 2016. June 2016.

[58] Bill Miller and Dale Rowe. “A Survey SCADA of and Critical Infrastructure Incidents”. In: *Proceedings of the 1st Annual Conference on Research in Information Technology*. RIIT ’12. Calgary, Alberta, Canada: ACM, 2012, pp. 51–56. ISBN: 9781450316439. DOI: 10.1145/2380790.2380805. URL: http://doi.acm.org/10.1145/2380790.2380805.

[59] *Net Applications: NetMarketShare.com - "Desktop Operating System Market Share"*. https://www.netmarketshare.com/; last accessed 14. June 2016. June 2016.

[60] *Netcraft Ltd.: May 2016 Web Server Survey*. http://news.netcraft.com/archives/2016/05/26/may-2016-web-server-survey.html; last accessed 02. June 2016. June 2016.

[61] Belshe et al. - Network Working Group. *Hypertext Transfer Protocol Version 2 (HTTP/2)*. RFC 7540. IETF, May 2015. URL: https://tools.ietf.org/html/rfc7540.

[62] Blake-Wilson et. al - Network Working Group. *Transport Layer Security (TLS) Extensions*. RFC 3546. IETF, June 2003. URL: https://tools.ietf.org/html/rfc3546.

[63] Fielding et al. - Network Working Group. *Hypertext Transfer Protocol – HTTP/1.1*. RFC 2616. IETF, June 1999. URL: https://tools.ietf.org/html/rfc2616.

[64] Thomson et al. - Network Working Group. *DNS Extensions to Support IP Version 6*. RFC 3596. IETF, Oct. 2003. URL: https://tools.ietf.org/html/rfc3596.

[65] *OpenVZ Project - "OpenVZ Virtuozzo Containers Wiki"*. https://openvz.org/Main_Page; last accessed 14. June 2016. June 2016.

[66] *Oracle Inc. - "Oracle Solaris Zones"*. https://docs.oracle.com/cd/E18440_01/doc.111/e18415/chapter_zones.htm#OPCUG426; last accessed 14. June 2016. June 2016.

[67] Sage Premier. *Human Factors: The Journal of the Human Factors and Ergonomics Society*. Sage, 1999.

[68] *Rescorla - Network Working Group - "Draft - The Transport Layer Security (TLS) Protocol Version 1.3" - June 08, 2016*. https://tlswg.github.io/tls13-spec/#rfc.section.1.2; last accessed 13. June 2016. June 2016.

[69] Dierks & Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.2*. RFC 5246. IETF, Aug. 2008. URL: https://tools.ietf.org/html/rfc5246.

[70] *rubyonrails.org: Guides - Active Record Basics*. http://guides.rubyonrails.org/active_record_basics.html; last accessed 01. June 2016. June 2016.

[71] Andrew S. Tanenbaum and David J. Wetherall. *Computer Networks*. 5th. Upper Saddle River, NJ, USA: Prentice Hall Press, 2010. ISBN: 9780132126953.

[72] *TrustedSec - Social-Engineer Toolkit*. https://www.trustedsec.com/social-engineer-toolkit/; last accessed 19. June 2016. June 2016.

[73] *TrustedSec - Social-Engineer Toolkit - David Kennedy - SET User Manual Made for SET 6.0*. https://github.com/trustedsec/social-engineer-toolkit/blob/master/readme/User_Manual.pdf; last accessed 19. June 2016. June 2016.

[74] Bellis - Nominet UK. *DNS Transport over TCP - Implementation Requirements*. RFC 5966. IETF, Aug. 2010. URL: https://tools.ietf.org/html/rfc5966.

[75] *Universitätsklinikum Schleswig-Holstein (UKSH) Website*. http://www.uksh.de/Das_UKSH.html; last accessed 02. June 2016. June 2016.

[76] ITU T X.200. *Information Technology Open System Interconnection Basic Reference Model: The Base Model*. July 1994.