TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

TANELI UUSIKORPI

IMPROVING TEST COVERAGE OF STANDALONE VULNERABILITY SCANNER WHEN SCANNING HTTP API

Master of Science thesis

Examiners: Professor Hannu Koivisto and
University Teacher Mikko Salmenperä
Examiners and topic approved by the
Faculty Council of the Faculty of
Engineering Sciences
on 1st November 2017

ABSTRACT

TANELI UUSIKORPI: Improving Test Coverage of Standalone Vulnerability Scanner when Scanning HTTP API
Tampere University of Technology
Master of Science thesis, 85 pages
March 2018
Master's Degree Programme in Automation Engineering
Major: Automation and Information Networks
Examiners: Professor Hannu Koivisto and University Teacher Mikko Salmenperä
Keywords: Security Testing, Standalone Web Application Vulnerability Scanner, DAST, OWASP ZAP, REST

Information security and security testing is important for software companies because even if their own information has not been compromised one bad news about badly implemented security can cause irreparable affects to market value and future sales. Standalone vulnerability scanners can be used to implement security testing easily but real effectiveness and functionality how they work remain in the dark if the security tester does not know enough about penetrations testing, technologies used in the software and methods that is used to reveal vulnerabilities.

A Finnish software company M-Files has developed M-files Web Service that is REST like HTTP API for communicating between their document management system and client applications for web and mobile platforms. The goal of my master thesis was to study web application security testing and understand functionality, limitations and technical challenges of standalone vulnerability scanners for web applications. Also it was suggested and implemented solutions that improve the results when M-Files Web Service was scanned by using a standalone vulnerability scanner. The reconnaissance phase was improved by creating an application that retrieves information of every method of M-Files Web Service directly from the source code and sends valid requests to the scanner by using this information. The attack surface is always up-to date and any shortcomings caused by inadequate documentation or environment are defeated. This component improved testing coverage compared to the previous manual solution by 125 percent. Second component that was created was POC extension to OWASP ZAP that improves active scanning by ensuring that the environment is in the best state to reveal vulnerabilities by executing presteps before every attack request sent by OWASP ZAP. The solution required modification to core source codes of OWASP ZAP but it was proofed that suggestion could work and produce better result. Some issues caused by the created customization to OWASP ZAP left without solutions so this component is not yet ready for production.

TIIVISTELMÄ

TANELI UUSIKORPI: Testikattavuuden parantaminen testattaessa HTTP rajapintaa Web-sovelluksille tarkoitetulla automaattisella haavoittuvuusskannerilla
Tampereen teknillinen yliopisto
Diplomityö, 85 sivua
Maaliskuu 2018
Automaatiotekniikan DI-tutkintoohjelma
Pääaine: Automaatio- ja informaatioverkot
Tarkastajat: Professori Hannu Koivisto ja yliopisto-opettaja Mikko Salmenperä
Avainsanat: Tietoturvatestaus, tietoturvatyökalut, DAST, OWASP ZAP, REST

Tietoturva ja tietoturvatestaus on tärkeää ohjelmistoyrityksille, koska vaikka heidän oma datansa ei vaarantuisikaan, yksikin huono uutinen liittyen huonosti toteutettuun tietoturvaan voi aiheuttaa korjaamattomia vaikutuksia markkina-arvoon ja tuleviin kauppoihin. Automaattisia haavoittuvuusskannereita voidaan helposti ottaa käyttoon tietoturvatestauksessa, mutta niiden todellinen toimivuus ja toimintaperiaatteet jäävät pimentoon, jos tietoturvatestaajalla ei ole tietämystä penetraatiotestauksesta, ohjelmistossa käytetyistä teknologioista ja menetelmistä, joita käytetään haavoittuvuuksien paljastamiseen.

Suomalainen ohjelmistoyritys M-Files on kehittänyt julkisen REST arkkitehtuuria seuraavan HTTP ohjelmistorajapinnan M-Files Web Servicen kommunikoidakseen kehittämänsä dokumenttienhallintaohjelman ja web- sekä mobiilikäyttöliittymien välillä. Tämän työn tavoitteena on tutkia web-sovellusten tietoturvatestausta, ymmärtää web-sovelluksille tarkoitettujen haavoittuvuusskannereiden toimintaperiaatteet, rajoitukset sekä tekniset haasteet sekä ehdottaa ja toteuttaa ratkaisuja, jotka parantavat tuloksia, kun M-Files Web Servicen tietoturvaa testataan käyttämällä automaattisia haavoittuvuusskannereita. Tiedusteluvaihetta kehitettiin luomalla sovellus, joka hakee tiedon kaikista M-Files Web Servicen metodeista suoraan lähdekoodista ja lähettää sopivat viestit skannerille käyttämällä haettuja tietoja. Tällä tavalla hyökkäyspinta on aina ajan tasalla ja dokumentaation sekä ympäristön aiheuttamista puutteista päästään eroon. Sovellus paransi testikattavuutta edelliseen manuaaliseen menetelmään verrattuna 125 prosenttia. Toinen toteutettu komponentti oli demolaajennus OWASP ZAP:iin, joka paransi aktiivista skannausta varmistamalla että ympäristö on parhaassa tilassa paljastamaan haavoittuvuuksia lähettämällä viestejä, jotka toteuttivat esiehtoja, ennen jokaista OWASP ZAP:n lähettämää hyökkäysviestiä. Ratkaisu edellytti, että OWASP ZAP:n lähdekoodia muutettiin, joten ratkaisu ei ole yleisesti toteutettavissa. Saatiin kuitenkin todistettua, että ratkaisu voisi toimia ja parantaa testaustuloksia. Toteutukseen jäi vielä muutamia ongelmia, jotka liittyivät OWASP ZAP:iin tehtyyn muutokseen, joten komponentti ei ole vielä valmis tuotantokäyttöön.

PREFACE

Creating this master thesis has been long and educational journey. Firstly, I would like to thank my instructor in M-Files Minna Vallius for giving me this challenging subject and allowing me to concentrate in the security testing of web applications and test different solutions in my work.

Additionally I want to thank Professor Hannu Koivisto and University Teacher Mikko Salmenperä for their guidance and giving helpful comments how this thesis could be improved. Finally I want to give warm thanks to Anni who has supported me in this journey and never stopped believing me.

Tampere, 27.3.2018

Taneli Uusikorpi

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS AND SYMBOLS

API	Application Programming Interface
AST	Application Security Testing
ASCII	American Standard Code for Information Interchange
CIA	The Confidentiality, Integrity and Availability Triad for Security
CRUD	Create, Read, Update and Delete
CWE	Common Weakness Enumerator
DAST	Dynamic Application Security Testing
DoS	Denial of Service
ECM	Enterprise Content Management
EIM	Enterprise Information Management
HATEOAS	Hypermedia as the Engine of Application State
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IAST	Interactive Application Security Testing
JSON	JavaScript Object Notation
MFWS	M-Files Web Service
NDA	Non-Disclosure Agreement
OSSTMM	Open Source Security Testing Methodology Manual
OWASP	The Open Web Application Security Project
PoC	Proof of Concept
REST	Representational State Transfer
RPC	Remote Procedure Call
SAST	Static Application Security Testing
SDLC	Software Development Life Cycle
SSL	Secure Sockets Layer
TUT	Tampere University of Technology
TLS	Transport Layer Security
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
XML	Extensible Markup Language
XSS	Cross-Site Scripting
XEE	XML External Entities
ZAP	OWASP Zed Attack Proxy

1. INTRODUCTION

It is easy to make a vulnerable web application. Even an experienced software developer makes mistakes and creates unknowingly vulnerabilities to the software. Vulnerability issues are also very difficult to find because they do not have any remarkable effects to normal functionality and are difficult to find in ordinary testing. The application can work perfectly without any security details have been considered. For this reason, security testing is always necessary. Its purpose is to find vulnerabilities by ethically attacking the application or using vulnerability scanners that automatically identify known security vulnerabilities. Software companies can order security testing from third party companies, if the company feels that they do not have enough money, time or knowledge to test security properly itself. In any case it is important that software companies also execute their own security testing to ensure that security of the software is really at the appropriate level and not blindly trust that these third party penetration testers and external security audits can find every vulnerability of the software.

Standalone vulnerability scanners for web applications seems to be quite easy to use but actual principles how they work remain in the dark if the tester does not have needed knowledge about security testing, penetration testing or technologies used in the software. These tools are also designed to provide best results for generic web applications. If the structure or functionalities of a tested web application is not ordinary or it acts differently as a testing tool expects it leads worse testing accuracy and coverage. Problems are also difficult to notice only by looking the results of the testing tool because these does reveal any info how many vulnerability have not been found.

The goal of my master thesis was to study web application security testing and understand the basic functionalities of standalone vulnerability scanners for web applications. Collected information was then used to improve security testing of M-Files Web Service that is REST-like HTTP API used in M-Files Web and M-Files Mobile applications and is also weakest link before M-Files DMS (Document Management System) server that is used to store confidential information. The main goal was not to verify what is the best vulnerability scanner in the market but to clarify common problems and technical challenges related to all vulnerability scanners and made suggestions how these problems could be overcome by creating additional tools and extensions.

For proving the suggestions it was created two components in this master thesis. These both improve security testing coverage of M-Files Web Service when it was tested by using a standalone vulnerability scanner. The first created component is an application that automates previously manually executed reconnaissance phase. It retrieves details of all methods of M-Files Web Service by using source code where the web service is implemented. Parsed information was used to teach a standalone vulnerability scanner as well as possible by using always up to date information without human intervention. The methods were decided to retrieve directly from source code so that shortages in the documentation and previous manual teaching method can be bypass. The second component was an proof of concept extension to one vulnerability scanner OWASP ZAP (The Open Web Application Security Project - Zed Attack Proxy). It uses the information learned in the first component to maintain preconditions of the methods while active vulnerability scanner is on-going.

Second chapter of this master thesis defines generally what is meant by information security, different methods to model information security and basics about cyber attacks and threat modeling. Common technologies and vulnerabilities related to web applications are explained in the third chapter. The fourth chapter explains what is meant by security testing techniques vulnerability assessment and penetration testing and described the functionality and typical structure of vulnerability scanners for web applications. Strengths, limitations and technical challenges were also discussed in this chapter. In the end of this fourth chapter it is explained features of OWASP Zed Attack Proxy that is free open source intercepting proxy and vulnerability scanner. It is used to test created solutions. The fifth chapter clarifies the security testing of M-Files Web and problems, which are encountered when security testing is executed by using OWASP ZAP. The sixth chapter defines how the testing coverage was improved by explaining in detail how the created components were implemented. In last chapter there is conclusion.

2. INFORMATION SECURITY

The purpose of information security is to prevent access to information from those who should not have access but still provide a safe and reliable connection for those who have rights. There is a well-known quote that says: "The only truly secure system is one that is powered off, cast in a block of concrete and sealed in a lead-lined room with armed guards and even then I have my doubts"[25]. The system described in the quote would be indeed secure but how about productivity? The system wouldn't be usable anymore and productivity would be near to zero. When developing information security it is important to understand that when security level is increased it is often done by destroying productivity. The cost of security should not exceed the value of secured information. Security planning is finding balance between protection, usability and cost. [1]

2.1 Modeling Information Security

It is difficult to discuss security issues and information security without common terminology. For this reason, it is helpful to use predefined model as a baseline of discussion. Information security and security issues can be modeled in various ways but two generally familiar models are CIA (Confidentiality, Integrity, Availability) triad and the Parkerian Hexad. CIA triad categorize information security to three parts: confidentiality, integrity and availability. The Parkedian Hexad adds three categories more to CIA model, which are possession, authenticity and utility.

2.1.1 CIA triad

Information security is traditionally categorized to three parts: confidentiality, integrity and availability. This is also known as the confidentiality, integrity and availability (CIA) triad (Figure 2.1). These principles are explained in following paragraphs in more detail.

Confidentiality is sometimes wrongly perceived as the same thing as privacy but these don't entirely mean the same. Confidentiality is one primary component of privacy but it

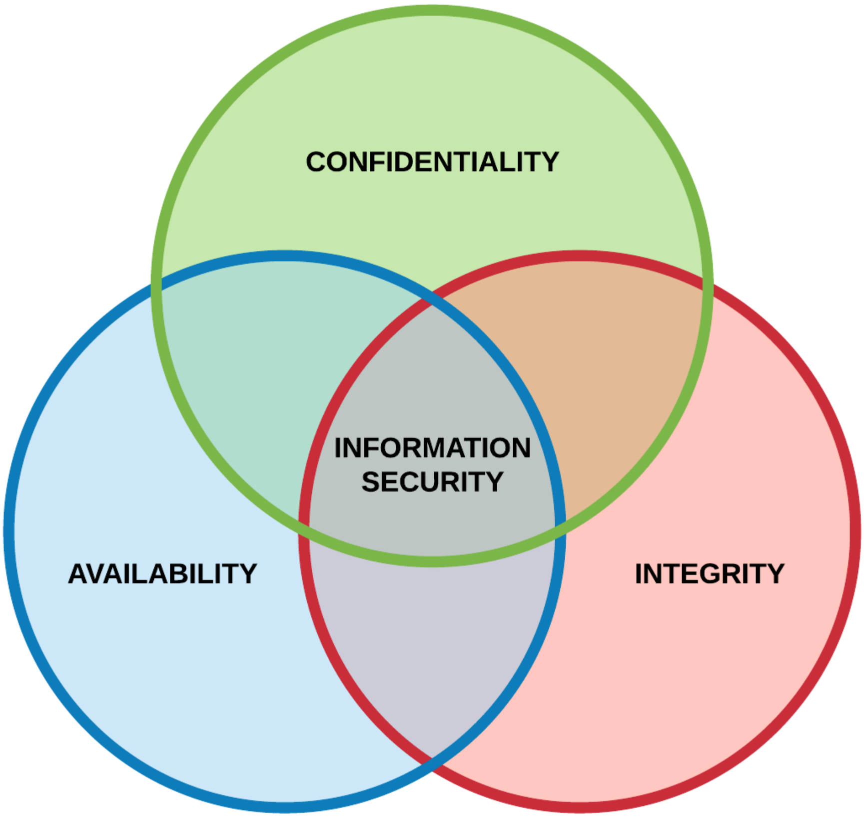

Figure 2.1 *Principles of CIA triad. Classical way how information security is described.*

also contains other elements that isn't always included in privacy. For example, confidentiality is lost also when you lose a laptop that contain confidential data, when a hacker has penetrated in your system or when you sent an e-mail to a wrong person. Not only when your confidential information is used without your permission like in privacy. The confidentiality contains all things, which privacy has, but includes also things when information has been reached by a crime or by an accident. [1]

Integrity of a system is lost when a data or portions of data has been changed or deleted in an unauthorized or undesirable way. It is also lost when there isn't any way to reverse authorized changes that need to be undone. To maintain integrity, it is needed to prevent all unauthorized changes to the data but also add ability to rollback authorized changes. Integrity is important especially for data that is used for decisions because otherwise attackers has the ability to change decisions according their preferences by changing information so that wrong decisions are made. [1]

Availability of information is lost when any authorized person does not have access to the data when needed. It is required that at least one part in the chain to the information has been broken. For example, these breaks can happen in power loss, operating system or application problems, network attacks, compromise of a system or other problem which prevent authorized users to view and modify a data they need. Attacks against availability principle are called DoS (Denial of Service) attacks. [1]

2.1.2 The Parkerian Hexad

The Parkerian hexad is more complex version of classic CIA. It has a total of six principles. First three principles are same as in CIA triad. Other three are possession or control, authenticity and utility. The reason for this model is to fill the gaps of CIA triad because CIA is too technology driven and does not consider the human element of information security. Difference of principles in the Parkedian hexad can be understood more easily if they are grouped in pairs. First group is confidentiality and possession, second is integrity and authenticity, and third availability and utility. These principles and pairs are shown in figure 2.2 where principles are drawn around security and pairs are drawn with the same color. These pairs and how principles added in the Parkerian hexad modify the classic CIA triad model is explained next.

Confidentiality and Possession

Possession or control principle differs from confidentiality principle so that even if information is stolen confidentiality is not always breached. For example, when an encrypted memory stick is stolen the confidentiality of data is not compromised because criminal cannot open the encryption and see what is written to the memory stick. The possession of data is however compromised because the owner of data does not have access to it and the control of data is in the hands of criminals. [1]

Integrity and Authenticity

In the Parkerian hexad integrity principle refers to integrity of data and the integrity of authentication information is divided to the separate authenticity principle. For example, if the user information of event log entry is altered to different user, as who has made the operation, the authenticity of event log is violated. Digital signatures is usually used to enforce authenticity of data. [1]

Figure 2.2 *Principles of the Parkerian hexad. Colors of principles describe principle pairs*

Availability and Utility

There are two principles in the Parkerian hexad which explains how the using of the data can be prevented. These are availability and utility. Availability refers to that the whole service is unavailable and it means the same thing as in CIA model while utility refers to the usability of the data. This refers to a situation where criminals have got an access to the confidential information but the information is not in the form that criminals can use. For example, if the information is encrypted the information is not very usable for criminals. Passwords are usually in encrypted form in the database because of that principle. Even if password information is compromised criminals can't use this information. [1]

2.2 Cyber Attacks and Threat Modelling

Cyber-attack is an attack, which is targeted to computer systems, technology-dependent enterprises or networks [28]. There are many different types of cyber-attacks. Before we can find out which particular cyber-attack can have an effect to our business it is important to understand what is meant by terms threat, vulnerability and risk.

A threat is an existing situation that can cause harm. For example, there can exist a virus which might pose a threat to a Windows operating system. Threats are often environment specific because it is usual that the same virus hasn't any affect to Linux operating systems. Vulnerabilities are holes in the system, which enable the attack specified in a threat. A vulnerability might exist in the operating system, in the application or in the building where the office is located. For example, a threat can be that criminals break into the warehouse and the vulnerability, which provide this attack, is that the door of the warehouse can be opened by a screwdriver. If both a threat and a vulnerability exist then we have a risk that something bad might happen. If either is missing, then the risk does not exist. For example, if there aren't any valuable articles in the warehouse then there isn't any threats that a burglary happens and if there aren't any vulnerabilities, which allow that criminals can break into the warehouse, but the threat still exist the risk disappear. So that the risk can be mitigated we should improve either physical, logical or administrative control. [1]

Different helpful questions about the system can help the administrator of the system to figure out the cyber-attacks, which the system might encounter. Questions can be such as:

How the system can be exploit?

Where the most confidential information is located?

If this is not enough for good results these questions can be formed more specifically by using some existing threat model, which categorizes these threats. With these models it is easier to find every place where these threats can locate. They also help one to think threats in right perspective. There are different methods to model threats. One way, which Microsoft introduced, is STRIDE model [14]. This model propose that threats can be grouped to six categories:

- **Spoofing Identity** refers to a threat where the authentication information of a user, such as a username and a password, is illegally accessed and then used in order to present as another user. [14]
- **Tampering with data** refers to a threat where a confidential information has been criminally modified via unauthorized access without that it has been noticed. This contains changes in the database where the information is held and the alteration of data when it has been sent between two computers over the public network. [14]
- **Repudiation** refers to a threat where a user denies performing an action and there isn't any way to prove otherwise. For example, if this user has executed illegal

action in the system and the system does not have capability to trace this illegal action. Nonrepudiation refers to the ability of a system to prevent repudiation threats. In this case the system would have perfect event log where this illegal action could have been traced and ability to point the action to the right person. [14]

- **Information Disclosure** refers to a threat where individuals who aren't supposed to have access to a resource have ability to view or read it. For example, if users can open a document, which they were not granted to open, or criminals can listen the transmission between two computers. [14]
- **Denial of service** refers to a threat where a system is temporary unavailable or unusable because of a natural disaster or a denial of service attack caused by criminals. [14]
- **Elevation of privileges** refers to a threat where an unprivileged user has a way to gain privileged access and can compromise and destroy the entire system. [14]

Questions which can be formulated by using STRIDE model can be for example:

S *How an attacker can access the authentication information?*

T *Can someone modify messages when these are transmitted over Internet?*

R *Is every important operation which user can made logged to the event log?*

I *Can someone listen messages when these are transmitted over Internet?*

D *How our system can manage a DoS attack?*

E *Are every administrator interfaces in the system secure?*

If some of these threats has been established always at least one principle in CIA or the Parkerian hexad is going to be violated. When spoofing identity threat has been established, it violates particularly authenticity principle but in the worst case also many other principles can be violated if this spoofed identity can be used to login in the system. Confidentiality principle is violated if documents can be opened and Integrity principle is violated if documents can be modified. If there is possibility to change the password of the user, the criminal can take the account under control and possession principle is violated.

When tampering with data threat exists it violates integrity principle because then there is possibility that data is modified. When repudiation threat exists, it violates authenticity principle because then the system lacks user information for example in event log. Any user can say that they didn't make some specific operation and there isn't any evidence prove that otherwise. When information disclosure threat has been occurred, it violates confidentiality principle because then confidential information is leaked. When denial of service threat exists, there is possibility that availability principle can be violated. When elevation of privileges has been established, it can cause big disaster. Attacker can take whole system under a control, destroy whole system, read every document, modify documents but stay still invisible, modify authenticity information in event log or make whole information unusable. So in the worst case every principle in information security can be violated.

Cyber attacks can be very dangerous also when these are done against a software that you have sold. Even if your own information is not compromised in the cyber attack the information of your customers have been compromised. News related to this causes irreparable affects to a market value and future sales of your company and in a worse case can even lead to bankruptcy. For this reason, it is important for software companies that they invest in the security of their software and ensures that their software complies with current security regulations. Security of the software can be improved by raising awareness about security among developers and invest in a good security testing team or purchase the security testing from a third part company that is focused and are experts in security testing.

3. VULNERABILITIES IN WEB APPLICATIONS

It is easy to make a vulnerability to a web application. The created web application can work perfectly without that any security details have been considered. Security issues in these applications can also remain unfixed if the developers or the software testers are not familiar how vulnerabilities are exploited and what technologies used in web applications enable these vulnerabilities. The security can be improved by increasing awareness about the technologies which are used in web application, learning methods that are used reveal vulnerabilities and understanding main enablers that make vulnerabilities possible.

3.1 Web Applications

In the past, the World Wide Web contained only static web sites and other static documents linked to the sites. Communication was mostly one-way, from server to browser and authentication was rarely used because it was usually only wanted that every user saw the same content on a web site. Attackers also rarely gained access to sensitive information because everything was already public. Today the majority of web sites are web applications. They support multiple features like registration, login, financial transactions, search, dynamically generated content that is often user specific and they rely on two-way information exchange. In other words, web applications are client-server softwares in which the client is run in a web browser. Web applications have been created a lot nowadays for many reasons. Most important reason is that there is no need to install separate client software because every web user already have a browser installed on their every web device and they already know how to use it. Changes to user interface can be done on the server and these are immediately taken in use. HTTP (Hypertext transfer protocol) that is used as communication protocol in web application brings many useful features to a web application like connectionless communication, tunneling over other protocols and possibility for secure communication. Lots of open source code and other resources are also publicly available for helping to develop web applications more easily. Various HTTP APIs like RESTful web services are used for communication between a client and a server. JavaScript is used in the client side to create dynamically generated menus and lists and sending requests to other services. [27, p. 2-6]

3.1.1 HTTP

Hypertext transfer protocol (HTTP) is a message-based communication protocol used widely in the World Wide Web and in all web applications created these days. The protocol was originally used for transferring static text-based resources but nowadays it is also used in complex distributed applications because it is easy to utilize to different situations. In HTTP protocol, there is always a client and a server. The client sends a HTTP request message, which the server process and returns a HTTP response message back to the client. HTTP is a connectionless protocol but stateful TCP (Transmission Control Protocol) can be used as a transport mechanism for HTTP messages. It makes each request and response an autonomous transaction. [27, p. 39–40]

Both a HTTP request and a HTTP response message have a start-line, zero or more header field lines and an empty line to inform that the header of a message is ended. After the empty line there can be optional message body. HTTP message types have divergent start-lines. In a HTTP request it is called request-line and in a HTTP response it is called status-line. Format of start-line of HTTP request is:

```
[HTTP method] [URL] [HTTP version]
```

and for example it can look like this:

```
GET /index.html HTTP/1.1
```

HTTP method is verb which specify the operation that should be executed in the server for the resource specified in the URL. Most common methods are GET, POST, PUT and DELETE. Get method is used when a resource is intended to retrieve from the web server, POST method is used to execute actions PUT method is used to upload a resource and DELETE is used to delete a resource. Every method has their intended purpose but there aren't any restrictions to use these differently. [4, p. 21–33]

URL (Uniform Resource Locator) defines the resource where the operation defined in HTTP method should be targeted. URLs in web follows format:

```
protocol://hostname[:port]/[path/]resource[?param=value]
```

and for example can look like this

```
http://www.domain.com:8080/folder/main.html?search=cat
```

where **protocol** defines communication protocol and **hostname** defines domain or IP address of the web server. **Port** is optional parameter and is necessary to define only if non-default port is used. After that the **path** to the resource and **resource** name is specified and after the question mark it is possible to specify query parameters.

HTTP version specifies the protocol version, which the client uses and the web server should also be followed.

The status line that is the start line of HTTP response message has three element and follows a format:

```
[HTTP version] [Status-code] [Reason phrase]
```

and for example it can look like this:

```
HTTP/1.1 200 OK
```

It starts with HTTP version and follows with a 3-digit **status-code** indicating the result of the request and an **reason phrase** that describes the status code in human readable form. Status-codes have been divided to five groups according their first digit:

1xx — Informational

2xx — The request succeeded.

3xx — The request is redirected to a different resource.

4xx — The request fails because the request has error.

5xx — The request fails because server encountered an error.

For example status-code **201** means that the resource was created successfully, **404** means that the asked resource does not exist and **503** means the service is unavailable. [27, p. 48-49]

Cookies are data items in HTTP protocol which are stored in the client when HTTP response with **Set-Cookie** header is received from the server. The cookie resubmitted back to the server of every HTTP request, which are sent to this same server domain, in **Cookies** header. These are used specially to identify a client when a user returns to a same web application. [27, p. 47]

HTTP requests and HTTP responses can be transmitted through a certain server by configuring a browser to use HTTP Proxy server. It is a server that acts as an intermediary between the web browser and the web server. The proxy forwards all messages requested by the browser to the server and convey their responses back to the browser. The proxies are used for caching, authentication, access control or intercepting and modifying requests for security testing purposes. When an unencrypted HTTP request is sent via a proxy server, whole URL is readable and the proxy server can use this information to forward the request to the correct server. Same thing cannot be done with an encrypted

HTTPS (Hypertext Transfer Protocol Secure) request because the URL is encrypted. The SSL handshake between the browser and the proxy server cannot be performed either because this would break secure tunnel and make connection vulnerable to interception attacks. A pure TCP-level relay is used for this reason. This allows the browser to perform an SSL handshake normally with the web server as the proxy server passes all data in both directions. At first for starting this relay, the browser sends an HTTP CONNECT request to the proxy server where the destination hostname and a port number is specified as the URL. The proxy accepts this by returning an HTTP response with a 200 status and keeps that TCP connection open. After that this connection is used as a pure TCP-level relay to the destination. [27, p. 49–50]

3.1.2 Encoding Schemes

Encoding schemes are used in text based protocols and programming languages so that unusual characters, binary data or characters, which have special meaning, can be safely handled. Various technologies used in web applications like the HTTP protocol or the HTML language are historically text based so these have various encoding schemes to handle difficult situations. Without encoding the meaning of data can be changed and messages cannot be safely transported. [27, p. 66–67]

URLs can contain only the characters in the US-ASCII (American Standard Code for Information Interchange) character set and multiple characters in this set are also restricted because these has special meaning in URL schema or in HTTP protocol. The URL-encoding is created to encode the special characters and characters that does not belong to US-ASCII character set inside HTTP messages. The URL-encoded character starts with % prefix and is followed by the two-digit ASCII code of the character in hexadecimal. For example, the space is encoded with %20 and the equal sign (=) is encoded with %3d. [27, p. 67]

HTML encoding is used to escape characters that are used to define a document structure in HTML. Without HTML encoding some special characters couldn't be used inside the content of a HTML document but would instead be understood as part of the structure or the functionalities of the web page. For example these important characters for HTML can escaped in the following manner:

< $\rightarrow$ **<**
> $\rightarrow$ **>**
" $\rightarrow$ **"**
’ $\rightarrow$ **'**
& $\rightarrow$ **&**

Characters can be HTML encoded also by using form **<** → **<** where 60 presents ASCII code of the character as decimal or form **<** → **<** where 3c presents ASCII code of the character as hexadecimal. [27, p. 68–69]

3.1.3 RESTful Web Services

RESTful web services and APIs (Application Programming Interface) are services, which follow principles and constrains of Representational State Transfer (REST) architectural style. RESTful web services, which are implemented by using HTTP are often used by web applications to communicate between a client and a server. It is important to notice that REST is just an architectural style and not a technology, a communication protocol or a set of standards that have specific rules to implement. REST has set of constraints and principles, which have been found to be good guidelines when programming interfaces for distributed applications and hypermedia systems. It is designed to improve performance, scalability, simplicity, reconfigurability, portability and reliability of interfaces. Simply REST-compliant or RESTful web services follow client-server model where the connections are stateless. All in all, REST is a hybrid of multiple different architectural styles and can be defined by six constraints:

1. **Client-Server**
   By requiring separate client and server solutions, the user interfaces in the client solutions and data storage in the server solution will be isolated and these can be developed independently. Portability will be improved because client solutions for multiple platforms can be created without that these should worry how the data storage is managed. Scalability will be improved because now the server can be much simpler. [5, p. 76][8, p. 27]

2. **Stateless**
   Stateless communication means that each request sent to the server should contain all necessary information to be understood without any client context is stored on the server. Clients are responsible to store session state. These improve reliability of the service because without state in the server it is easier to roll back when failure happens. This makes possible to scale the server for bigger environments and make scaling simple because the server can release the resource as soon as the related request is handled and it does not have to wait following requests to understand the full purpose of the request. However, statelessness increase the repetitive data in a series of requests, which may decrease network performance, because any data about context is not stored in the server. Cache constrain, which is explained next, is added to REST to overcome this issue. [5, p. 78][8, p. 28]

3. **Cache**
   Cache constrain is added to REST for improving network efficiency by eliminating some client interactions to the server and reducing average latency in series of interactions. For fulfill this constrain there should be possible to implicitly or explicitly set data elements in the response to be cached or not. The disadvantage here is that this may decrease the reliability of the data.[5, p. 79]

4. **Uniform Interface**
   By requiring uniform interface, the system architecture is simplified and the visibility between interactions is improved. Also by defining uniform interface, REST adds most visible and recognizable guiding principles to the architectural style. These are identification of resources, resource manipulation through representations, self-descriptive messages and HATEOAS (Hypermedia as the Engine of Application State). These are explained better in following chapters. [5, p. 81-82]

5. **Layered System**
   In layered system, components and resources are divided to hierarchical layers and it is constrained that each component can see only the layer which they are interacting with. These layers prevent the system to become too complex by guiding to divide matching functionalities to own layers. Components can be simplified by moving rarely used functionalities to another layer or legacy services and new services can be divided to own layers so that new and legacy clients can be used simultaneously. This also allows that the API can be split to multiple software and hardware. Proxies, gateways and firewalls can be used at various points without that the interface of the API is necessary to change. [5, p. 82-83]

6. **Code-On-Demand**
   In REST, it is allowed but not forced that client functionality can be extended by executable code downloaded from the server. This optional code-on-demand constrain improves system extensibility and simplifies implementation of client solution because some features can be created and administered in the server side and these does not have to be implemented one by one to every client solution. [5, p. 82-84]

By implementing these six constraints, it is possible to achieve main design goals of REST architectural style which are independent deployment of the components, encapsulation of legacy systems, general interfaces, reduced latency, high security of service interactions, scalability and high performance. [5][8]

It is not restricted what communication protocol is used when RESTful API is implemented but because REST architectural style was created beside HTTP protocol and especially for network based applications it is very popular in the HTTP world. Better results are often achieved when REST is implemented by using HTTP protocol. REST relies in existing features in HTTP and for this reason the constraints and principles of REST are easier to fulfill by using HTTP. [5, p. 100][8]

REST is just architectural style and does not specify exactly how it should be utilized by using HTTP protocol. Lack of clarity in the right implementation has caused multiple different constantly evolving best practices, opinions and false conclusions how RESTful web services should be implemented by using HTTP. In the next, REST principles are explained in detail by following one recommended best practices documentation. [6]

In REST, functionalities and data are divided to nouns so-called resources and these resources are managed and found via resource identifier by using standard methods. Resource can be any information that can be named such as document, customer, order, image or service but also collection of things like documents, customers or orders and so on. Good resource identifiers should follow easily understandable hierarchical structure, which also define relationships between resources and should also be predictable in consistent way so that they can be predict in any time. Selected resource names should increase understandability of RESTful API by providing context for a request. When resource hierarchy and resource names are designed well understandability increases, the service is easy to use and is intuitive even without documentation. [5, p. 88][6, p. 6, 14]

URIs are used as resource identifiers of RESTful API when REST is utilized to create HTTP API. For example in a system that contains customers and orders created by the customer an appropriate URI to retrieve information of all customers in the system could be listed with request.

```
GET http://my.domain.com/restapi/customers
```

and informations of customer which id is 1234 can retrieved with request

```
GET http://my.domain.com/restapi/customers/1234.
```

It can be possible that all orders can be listed by using request

```
GET http://my.domain.com/restapi/orders
```

but this is not probably as intuitive to use as a request

```
GET http://my.domain.com/restapi/customers/1234/orders,
```

which also defines the relationship between customer and orders and will return only orders which are created by that specific customer. It is also possible that the system providers both resource hierarchies at the same time because in REST it is allowed that multiple URIs points to same resource. Continuing this, both requests

```
GET http://my.domain.com/restapi/customers/1234/orders/5678
GET http://my.domain.com/restapi/orders/5678
```

could point to the same order resource in the example system. [6, p. 11-15]

Multiple different HTTP features are used for utilizing the uniform interface constrain of REST. As explained above URIs are used to identify resources. In additional to this HTTP methods are utilized as standard method for handling CRUD (Create, Read, Update, and Delete) operations for a resource general way. Most important method for RESTful Web Services are GET, POST, PUT and DELETE and best practices suggests that these should be used for following actions:

GET method should be used when a specific representation of a resource is requested.

POST method should be used when a new resource is created to the collection.

PUT method should be used when a existing resource is updated or when a new resource is created with the ID which is selected by the client.

DELETE method should be used when a resource is deleted.

Both POST and PUT methods can be used to create new resources but they should be used differently. When POST is used, the creation request is targeted to a collection resource so that when a request like

```
POST http://my.domain.com/restapi/customers
```

is executed it creates a new customer to customers collection with a ID that is chosen by the server. When PUT is used for creation the client choose the ID for the resource that will be created if the id does not yet exist. For example a reguest

```
PUT http://my.domain.com/restapi/customers/1234
```

creates new customer with the ID defined in the URI if the id does not yet exist. If the ID already exist the details of this customer is updated with values given in the body. [6, p. 11–15]

Best practices and REST API implementations have biggest differences in how POST and PUT method should be used. Which one should be used for resource creation and which one should be used for updating an existing resource or should both creation and update be possible for each method. Before it is possible to understand actions which were suggested here for POST and PUT it is necessary to understand what is meant by idempotence.

So that an operation is idempotent it should not have difference if the client makes identical request once or multiple times. End results should be always same on the server side.

Methods GET, HEAD and OPTIONS are idempotent because these should not change anything on the server. It is also defined that PUT and DELETE methods should be idempotent and POST method should be used for non-idempotent requests. For this reason, POST method should be used when a new resource is created directly to the collection by using URI of collection resource. If the identical POST creation request is sent multiple times it causes multiple new resources to the server, which means that the operation is not idempotent. This does not happen with PUT method in the suggested creation usage because a new resource is created only at the first request and after that identical request only update values with same information as in the creation. This does not change anything on the server so the operation is idempotent. DELETE method is idempotent on server side but a client sees usually different response if the same resource is tried to delete second time. The resource is not findable anymore and the second deletion causes not found error.

All requests and responses in a RESTful API should be self-descriptive and have enough information so that these can be processed without additional information. For example, messages should specify Internet media type of content so that a correct parser can be selected to read body data or responses should explicitly indicate how these can be cached. The services can support multiple different media types and clients can choose which one it uses. Usually the services support at least JSON (JavaScript Object Notation) but also XML is quite typical. These both are text-based formats for serializing and transferring data objects.

In RESTful web services, hypermedia technologies are used as engine for delivering application state between clients and the web service. This so-called HATEOAS (Hypermedia as the Engine of Application State) means that hyperlinks and hypertext are used to deliver resource statuses and relationships between resources. Client uses body content, query-string parameters, request headers and URI to deliver client state and the service uses body content, status codes and response headers to deliver the service state to the client. URIs are also used in the response body as links to related resources and to resource itself. [6, p. 7]

In REST, resources are updated by sending full representation of a resource with updated information back to the server by using PUT method and URI for the resource. Client knows the representation of a resource because the similar presentation can be retrieved by using GET method for the same URI and has for this reason enough information to update or delete the resource.

HTTP status codes are used to indicate how the request succeeded. For REST there is commonly-accepted usage how status codes should be used that follows pretty much how

Table 3.1 Usage of HTTP methods, URI and status codes in REST. [6, p. 13-14]

HTTP Method	/customers	/customer/{Customer_ID}
GET	**200** (OK) Returns list of customers.	**200** (OK) Returns one customer. **404** (Not Found) Customer ID does not exist or it is invalid.
POST	**200** (OK) Location header contains link to the created resource.	**404** (NOT FOUND)
PUT	**404** (NOT FOUND) If it is not possible to update/replace whole collection. **201** (CREATED) If it is possible to replace whole collection.	**200** (OK) The resource updated successfully and the representation of the updated resource is returned in the response body. **204** (NO CONTENT) The resource is updated successfully but the representation of the updated resource is not returned in the response body. **404** (NOT FOUND) Customer ID does not exist or it is invalid.
DELETE	**404** (NOT FOUND) Usually it is not desirable that it is possible to delete whole collection.	**200** (OK) Resource deleted successfully. **404** (NOT FOUND) Customer ID does not exist or it is invalid.

status codes should be used in HTTP protocol. Usage of most important status codes are explained in table 3.1. Other commonly used status codes are **401** (UNAUTHORIZED) for indicating that a response has missing or invalid authentication token and **403** (FORBIDDEN) for indicating that user does not have rights to access the resource. Sometimes these two are replaced with **404** for security reasons so that it is impossible to resolve existing resources in the system. [6, p. 39]

3.2 Vulnerabilities and Security Risks in Web Applications

Web applications can contain multiple different types of vulnerabilities. The vulnerabilities not implemented purposely to the web application but are created by accident. Usually the easiest way to create a feature to a web application is not the most secure way. Creating a secure web application is not easy. It always requires paying attention to information security details and making additional implementations that fills security holes.

Existing and discovered vulnerability types are listed in many services. Often they also explain how the vulnerability works, what harms it can cause, how it can be exploited and how it can be prevented. OWASP Top 10 (The Open Web Application Security Project - Top 10) project keep up-to-date list of ten most critical security risks in web applications

[20]. In their newest list published in 2017, top 10 was:

1. Injection
2. Broken Authentication
3. Sensitive Data Exposure
4. XML External Entities (XXE)
5. Broken Access Control
6. Security Misconfiguration
7. Cross-Site Scripting (XSS)
8. Insecure Deserialization
9. Using Components with Known Vulnerabilities
10. Insufficient Logging and Monitoring

These security risks contains multiple different vulnerability types. OWASP uses CWE (Common Weakness Enumerator) ID to identify these different vulnerability and weaknesses types in web applications [15]. CWE upholds a community-developer list of software weakness types that are enumerated with CWE-ID. CWE contains own ids for weakness categories, weakness classes, weakness bases and weakness variants. For example, the CWE ID of Improper Authentication weakness class is CWE-287.

Many important languages used to develop web applications are interpreted languages like SQL, Perl and PHP. These languages are not compiled but these are executed in real time by using an interpreter and there is multiple use cases where the code is formed by using user inputs in a real time. **Injection** vulnerabilities exist because a web application does not sanitize user-supplied or hostile data and this input is directly used or concatenate to construct code for a interpreted language. [20]

For example following SQL call contains injection vulnerability.

```
var query = "SELECT * FROM students WHERE studentID = '"
    + request.getParameter("studentid") + "'";
```

The call is vulnerable because user input is not sanitized. The vulnerability can be exploit for example with input `studentid=' or '1'='1` that change the WHERE clause to form `WHERE studentID='' or '1'='1` that is always true. This causes that every record of students table is returned. In addition to viewing confidential data, SQL injection vulnerabilities can cause data loss, data corruption, loss of accountability, denial of access an in the worst case complete host takeover. [20]

The input value that is used to exploit a vulnerability is usually also called as crafted input, attack string or attack vector. The input parameter that is vulnerable is called as input vector. One vulnerability can have multiple different attack strings that are possible to use for exploiting the vulnerability. [20]

In top 10, there is two almost similar security risks **broken authentication** and **broken access control**. **Broken authentication** security risk refers to vulnerabilities, which cause weaknesses in authentication and password handling. For example, the handling is not secured in any way to use brute force or other automated attacks against the system to get valid username-password pairs. Also if default administrator passwords haven't been changed, passwords are saved as plain text or weak credential recovery process is used refers that the authentication of the system is broken. **Broken Access Control** security risk refers to vulnerabilities, which allows to bypass the access control checks or elevation of privilege. [20]

XSS vulnerabilities enable attacks against the users of a web application. There attackers can include own code in the web page that is executed in users' browser. These are caused by invalidated or unescaped used inputs that is used as part of HTML. If the input contains for example valid JavaScript code this is executed in browsers when the web page is shown. XSS vulnerabilities can be split to three forms: Reflected XSS, Stored XSS and DOM XSS. In reflected XSS vulnerability, the code is implemented immediately and only in a HTML of the next response after the crafted input is sent. A crafted input locates for example as a query parameter in a malicious link that the attacker has lured to click by an unaware user. That method can be used for example to hijack a session. In stored XSS vulnerability, the code is stored to a database and is the part of HTML also when other users or administrator view the web page. DOM XSS vulnerability is exploited similarly as reflected XSS by using malicious links. Difference is that the code is not put into HTML by the server but the client side scripting of the web application has access to the browser's document object model (DOM) and uses a parameter value in the URL without unescaping to construct the current page. [20][27, p. 431–442]

4. VULNERABILITY ASSESSMENT AND PENETRATION TESTING

Vulnerabilities and other security issues are difficult to find because these don't usually change normal functionalities of the system and for this reason weren't occurred in normal use or found in ordinary testing. For this reason, it is needed special testing for finding security issues. Its purpose is to find all security issues and vulnerabilities in the system by either scanning and seeking vulnerabilities or ethically attacking to the tested system. Security testing is important for software companies because that way the sold product has less security issues. This decreases the likelihood of malware attacks, service breaks, network intrusion and data theft. Such events have effect to customers loyalty and may cause loss in the company's market value or cause fines and lawsuits from poorly implemented security practices. [26]

4.1 Testing Techniques in Security testing

There is two main testing techniques for security testing: vulnerability assessment and penetration testing. The aim of vulnerability assessment is to review services, systems and used applications for potential security issues, whereas penetration testing goes further by simulating hackers' activity and actually performing attacks. Penetration testing is used because often the found vulnerability and its consequences are understood better when the real exploitation is demonstrated with PoC (Proof of Concept) attacks. Vulnerability assessment is carried out at beginning of penetration testing and this phase is explained as a phase in penetration testing in a following section. [3][7]

More precisely penetration testing is a legal and authorized attempt to locate vulnerabilities of computer systems and successfully demonstrate that vulnerabilities really exist. This is done by executing real attacks in a tested system or a test environment by using same tools and approach as hackers. When security issues is found and demonstrated this information is not used for crimes but these are reported to the client. For this reason, penetration testers are often called as ethical hackers or white hat hackers. The proper penetration testers also always report recommendations how and why these issues should be fixed. More precisely the proper report should include identification of vulnerabilities, risk rating of each vulnerability and quantity of vulnerabilities. Identification of vulnerability should contain information where and how the vulnerability is discovered and the vulnerability type so that the vulnerability is possible to fix. Risk rating of vulnerability can be for example Critical, High, Medium or Low which helps in fix prioritization. Without this comprehensive report both vulnerability assessment and penetration testing is quite useless because just stating that the system is vulnerable doesn't provide any helpful information. [3][7]

Penetration testing is also divided to white box penetration testing and black box penetration testing. White box penetration testing is used for finding as many vulnerabilities as possible whereas black box testing is used for testing capability of intrusion detection systems. The goal of white box penetration testing is to test the system security entirely. This is done by giving as much information as possible to penetration testers and allowing them to attack to the target system in peace so that they have best possibilities to find multiple vulnerabilities. The testing isn't done stealthy and does not test intrusion detection systems, incident response or early-alert systems at all. Black box penetration testing simulates stealthy actions of real attacker. Penetration tester has just the information what a skilled attacker would have or can stealthy collect. The security of the system is not tested entirely and it isn't intended to find all vulnerabilities. After one working vulnerability is discovered it is prove that the system is not secure. Advantage of black box penetration testing is to test system's ability to detect intrusion attempts and how these intrusion attempts is responded. Basically, it is tested that is it possible to attack to the target system without that users or administration team notice the attack or attack attempt. [3]

Penetration testers do not always get all possible information because often the testing is purchased from an external company and the customer does not want to give confidential information to testers. It is also thought that the attackers do not have access to this information so it isn't important for penetration testers. However it is more reasonable explore the system better than any hacker could do by giving more information to penetration testers than hackers could ever have. [2]

The work of penetration testers differs from black hat hacking also in elapsed time. Time which is allocated to penetration testing is limited somewhere between one week to couple months. Hackers can use as much time as they want so they have more time to find one vulnerability than penetration testers have time to find all vulnerabilities. It would be good to allocate time for penetration testing and execute it periodically. Usually organization do that before product releases or major upgrades but it is recommended to conduct penetration testing and security testing more often. [26]

Penetration testing and vulnerability assessment can be executed in different environments. For example, you can test security of a building or a network. One of most common things to test is a software. This application Security Testing (AST) contains all security testing which is executed against an application or a software. Vulnerabilities can be sought by investigating the source code of the application or simulating attacks against an application and then analyzing responses for vulnerabilities. When vulnerabilities are seek directly from source code it is called SAST (Static Application Security Testing) and when these are sought by simulating attacks it is called DAST (Dynamic Application Security Testing). [29]

4.2 Phases of Penetration Testing

Recommended phases and tasks for penetration testing are described in penetration testing methodologies. The best-known methodology, which is used also as de facto standard, is OSSTMM (Open Source Security Testing Methodology Manual) but there are also other methodologies that can be followed while executing penetration testing like NIST or Backtrack. These define tasks of penetration testing slightly differently but the same main phases can be found in every methodology. These have common that there is a phase or phases which should be done before actual exploitation, a phase where the exploitation or attack is executed and a phase or phase which are executed after the exploitation. In this writing, penetration testing is divided to three phases which are called:

1. The pre-attack phase
2. The attack phase
3. The post-attack phase

These phases simulate the phases of a real attacker but there are some differences because motivation of attackers and penetration testers are different. [2][26]

The pre-attack phase that is also known in some methodologies as Reconnaissance and Scanning contains all operations which penetration testers or hackers do before any real attacks or other suspicious operations are made to target system. This phase consists planning and attempts to investigate, explore and scan the target system for discovering possible vulnerabilities. These reconnaissance operations can be categorized into two types: passive reconnaissance and active reconnaissance. Passive reconnaissance contains operations, which does not do any detectable events to the target system. The aim is to gather as much information as possible about the target system from secondary sources without any possibilities to getting caught. After all possible information is got via passive reconnaissance it is usually moved to active reconnaissance, which actually do events, which may be detectable in the target system. In this part it is gathered information for example a provisional map of the network infrastructure or site map of the web application. All information in the pre-attack phase is used when actual attack is planned and usually hackers spend more time on this phase than on the actual attack. [2]

The attack phase or Exploitation phase in some methodologies is the phase where the actual simulated attack is executed and where all logical and physical vulnerabilities found in the pre-attack phase are tested. This is also the phase, which differ most when it is executed by hackers than by penetration testing team. Hackers need only one vulnerability to compromise the whole system whereas penetration testers are interesting to found and test as many vulnerabilities as possible because it is not known which vulnerability an attacker is going to use at first. [2]

The last phase in penetration testing is the post attack phase, which is also known as Post Exploitation and Maintaining Access. It is the phase where all systems, which are altered in attacks, are restored to the original state. This phase is not necessary to do if the penetration testing is executed in the testing environment which is created only for this purpose and is going to be destroyed in the end. However penetration testing is often executed in the product environment so there the resetting to initial state is essential. [2]

4.3 Tools for Web Application Penetration Testing and Vulnerability Scanning

Manual penetration testing is very slow. This has been improved by developing multiple different tools for seek information, speed up penetration testing by automating time-consuming tasks and facilitate vulnerability assessment. These tools are made by an extensive team of professionals with diverse security testing skills and are very useful for all security testers regardless of their professional skills. Professional penetration testers can speed up their testing but also not so professional security testers can reveal severe security issues. Fully automated or in other words standalone security testing tools are popular in many organization which does have money to have professional security testing team or want to automate their security testing.

Tools for application security testing can split tools for dynamic AST (DAST) and tools for static AST (SAST). Tools for DAST contains all tools, which analyze the reactions of the target application while simulated attacks are executed against the application. These tools are also called vulnerability scanners, penetration testing tools or web hacking frameworks or testing suites. DAST tools for web applications analyze HTTP requests to find inputs locations, modifies requests by putting attack strings to input locations, send the modified request and analyses responses for vulnerability signatures. Tools for SAST are source code analysis tools. These tools seek vulnerabilities directly from the source code, bytecode or binary of the application. Additional of this there is nowadays also interactive AST (IAST) tools that are somewhere between SAST and DAST tools. These tools observe operations of an application and identify vulnerabilities within the application typically as agent installed in runtime environment. [29]

4.4 Web Application Vulnerability Scanners and Intercepting Proxies

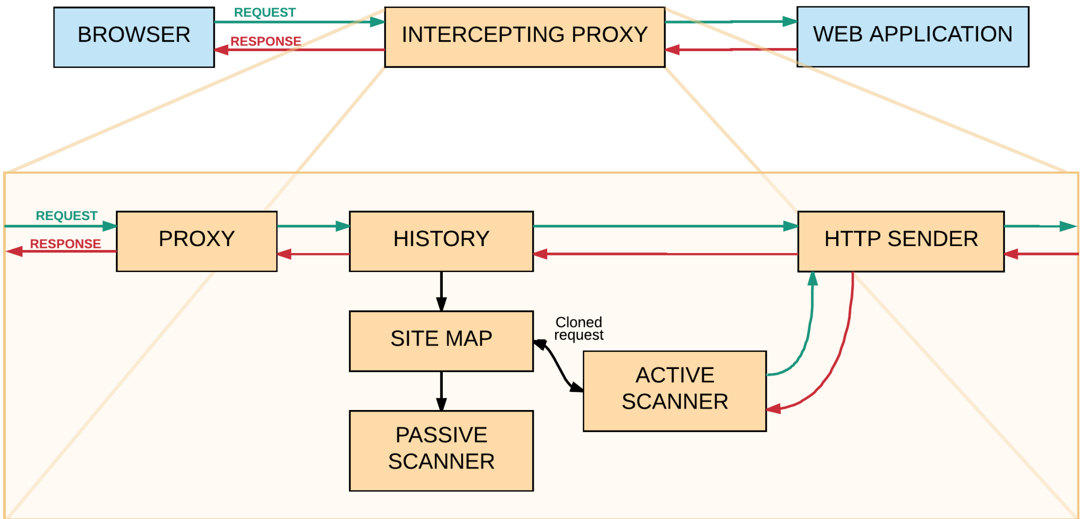

Figure 4.1 *Main mechanism and features of general intercepting proxy*

The most seminal tool for dynamic web application security testing is intercepting proxy. Main mechanism and features of this tool are explained in a figure 4.1. It is an application that is typically configured in a browser as a proxy server. When the target application is used in the normal way by using this configured browser the intercepting proxy monitors network traffic between the target web application and the browser. This allows penetration tester to edit the requests, like an attacker, before they are sent to the target application. Simultaneously intercepting proxies collects relevant information such as history about all requests and responses and a site map that is presentation of discovered content in table or tree form. Additional to this, intercepting proxies provides multiple useful functions for helping penetration testing tasks. For example web application spider, web application fuzzers, automatic vulnerability scanner, passive vulnerability scanner and possibility to manually send requests and analyze responses. [27, p. 751–752]

4.4.1 Features

The big task at the beginning of using intercepting proxy is to define attack surface by introducing all the web pages, directories and other requests that belongs to the target web application. This can be performed manually by using the web application normally but intercepting proxies have also automatic spidering tools for crawling the application without human intervention. These tools request already visited or given web pages, analyze HTML code of responses for additional links in the scope of the target application and then requests found URLs until it does not find any new URLs, which belong to the target web application. All found links are seen in the proxy history and added to the site map to increase attack surface. When all request types are listed in one place these can be reviewed and passed to other tools for further analysis. [2]

A traditional web spider looks only HTML code for new links and requests but this is rarely enough for web applications that uses forms, drop-down lists, buttons, text inputs, JavaScripts and so on for its functionalities. JavaScript is used in the web application for dynamically generated menus and lists or sending request to other services. Requests sent by these functionalities are not run into by using traditional spider because traditional spider looks only the HTML code of the responses so these needs additional approach for that whole attack surface can be retrieved. For this reason it is created spiders, which work better with web applications. These can for example use tools of UI test automation to fill every input and click every clickable element in the web page to execute all possible JavaScript code. These launch requests to other services, which are collected by intercepting proxy and added to the site map. Some web application spiders can analyze also JavaScript codes to find additional links. [27, p. 760]

In the best case, the whole web application can be mapped by using the spider only by giving one URL and unleash the tool. However, it is very difficult to ensure that whole application is really mapped. Especially when the web application is very big and it is impossible to manually ensure that every request type is really added to the site map. Also executing spiders can cause side effects if it finds a link which executes unwanted operation like a operation that deletes a user account or some other important information. For this reason, it is important that spiders are used very carefully in product environment. Also log out requests are annoying for spiders because if these requests are executed the spider deletes its own session and the crawling is not completed perfectly. The traditional spider can never be used silently because it makes hundreds or even thousands request. For this reason, there is also passive spiders that parse links from the responses and add these to the site map but do not requests these for additional site map entities. [27, p. 760]

Web application fuzzer is a tool for creating attack payloads to requests. It keeps inside multiple different built-in sets of attack payloads, where one or more multiple payload can be select and use in requests to reveal vulnerabilities or use custom or built-in functions to generate arbitrary payloads. These payloads can be based on malformed encoding, character substitution, brute force, or data retrieved in a previous attack. [27, p. 762-763]

Vulnerability scanning executed in intercepting proxies can split into two categories; passive and active scanning. Passive scanning does not send any additional requests and for this reason can be executed without trace. It monitors all requests and responses, which pass through the proxy and identify potential vulnerabilities such as cleartext password submission, cookie misconfiguration, and cross-domain referrer leakage. Active scanning executes actual penetration testing. It cannot be executed without trace because vulnerabilities are detected by sending multiple crafted request with different standard attack strings designed to trigger the signature of a vulnerability in the response. [27, p. 764]

Intercepting proxies in the market are made for different purposes and for different level of knowledge about web application security testing. Tools, which are created for advanced penetration testers are usually more user-driven and need more manual work and knowledge to get good results. These tools are called penetration testing tools, integrated web application security testing suites or web hacking frameworks. Testers which does not have so much knowledge cannot get good results with these testing suites because it is required to know special details how vulnerabilities are revealed. For example, what attack payloads should be used in requests and what results should be except in responses. However, with testing suites the application can be tested more precisely. It allows to choose which requests are scanned first and how the scanning is performed. It also gives real time feedback that makes possible to find more advanced vulnerabilities if the tester has enough experience to notice these vulnerabilities. Every attack request can be sent manually or use repeater tools which sends multiple requests with different attack payload and location combination. [27, p. 764]

Intercepting proxies usually have also features for automatic scanning. This includes passive scanner that work automatically in the background and active scanner that can be executed manually for selected requests. Passive scanner analyze every request and response, which go through the proxy while attack surface is introduced and report vulnerabilities immediately. Active scanner is needed to start manually. It can be executed for one requests or multiple request simultaneously. It seeks common vulnerabilities by parsing parameter locations from the request and then send the request again multiple times while blindly trying multiple different attack payloads one by one in every found parameter location. Vulnerabilities are detected by analyzing the response for vulnerability signatures that are something strange in the response that reveal that vulnerability exist or might exist. [27, p. 764]

If all or most phases in web application penetration testing is automated, these tools are called standalone web application scanners. These tools crawl the application automatically for example by using spider tools, execute passive and active scanners and then produces a report describing each vulnerability it has discovered. These tools are good for security testers, which does not have yet much experience. These test wide range of vulnerabilities and tester does not have to know special details how vulnerabilities occur and can be revealed. Using standalone vulnerability scanner seems to be quite easy because tester does not have to know anything about security testing but real functionality how they work remain in the dark. Vulnerability scanners can never be perfect so even if the report produced by standalone vulnerability scanner says that the application does not have any vulnerabilities it does not say anything that it couldn't have. All alarms reported by scanner should also be reviewed because there can be false alarms. Experienced penetration testers know the limitations of vulnerability scanners but they use these tools to get easily a general overview about the application. It might prompt interesting occasions which would need manual investigation. This is important information especially when the application is very wide and is difficult to test entirely. [27, p. 764, 783–784]

4.4.2 Strengths, Limitations and Technical Challenges

Like said before standalone or automatic vulnerability scanners can never find every vulnerability in the web application but if the vulnerability causes clear signature in the response it is possible to found. Vulnerability signature is programmatically identifiable evidence in the response which tells that a vulnerability is successfully triggered. Scanners are usually implemented to detect vulnerabilities at least from following categories:

- Reflected cross site scripting
- Some SQL injection vulnerabilities
- Some path traversal vulnerabilities
- Some command injection vulnerabilities
- Identify straightforward directory listing
- Cleartext password submission

The detection accuracy of the scanner depends how well attack strings and signature detection were implemented and especially how the tested web application fits to the implementation of the scanner. The vulnerability can be left undetected even if an attack string really triggers a vulnerability because the application responses different signature as the scanner assumes. Also the attack string may be sanitized and for this reason do not trigger a vulnerability even if a skilled attacker could discover a way to bypass the input validation. [27, p. 773–776]

Vulnerability scanners cannot detect several important categories of vulnerabilities because these do not have detectable signature or cannot be exploit by using standard set of attack strings. Some of these would be easily detected by any attacker with modest skills. These are for example:

- Broken access control
- Vulnerabilities, which require that the meaning of a parameter in the applications is understood.
- Logic flaws
- Weak password quality rules
- Session hijacking attacks
- Leakages of sensitive information

Some vulnerability scanners claim that they can detect also some of these vulnerabilities but real detection accuracy depend a lot on the application what is tested. If the application does not response or function like the scanner assumes these causes huge number false positives and false negatives alarms. [27, p. 773–776]

Vulnerability scanners are designed and implemented by the experts of security testing, but even if they would have skills to find manually every vulnerability in every web application, it is impossible to create an automatic scanner, which would do that perfectly. Existing scanners are created to found vulnerabilities as many web applications as possible but every web application is different and so that the scanner should find every vulnerability these should understand the logic of the web application. This would be possible only if the scanner is full-blown artificial intelligent or by adding human intervention. For this reason, existing scanners work best against web sites and web applications which works expected ways and find only vulnerabilities, which does not require to understand logic of the application. [27, p. 776–777]

Because the scanners do not understand the real meaning of parameters and the logic of the web application it have to blindly test every parameter in every HTTP requests in the site map by using all known attack strings. The scanner does not improvise if it notices something strange because it only does what it is implemented. This causes that many application specific vulnerabilities will be left undetected. For example, scanners are not able to understand that being able to modify price of the product before form is sent is a vulnerability whereas modifying some other parameter is not. Using vulnerability scanners against product environment can be very dangerous also for this reason because the scanner cannot understand what it is doing. It can reset user password, delete accounts or corrupt all data by accident.[27, p. 776–779]

Mapping the attack surface of a modern web application can be challenging. Web applications have usually content that are using the same core set of functionalities. These are mapped unnecessarily multiple times to attack surface because if a vulnerability is found in one content element it will be found also in other elements, which uses the same functionality. The web application spider can also create new content while spidering the web application. This causes more and more new elements to map that are usually also duplicates. This can cause also that spidering newer ends. If the spidering is needed to manually stop, because it is in the loop, it causes that some content in the web application is not mapped. Web application spider that actually executes client-side codes, fill forms and press buttons would require lot of intelligent and understanding about logic of the application so that they would work perfectly. This is usually impossible. They should for example understand when invalid input is given so that it is possible to proceed forward and execute important functionalities beyond that. As explained automatic spidering tools for web sites or web applications are not perfect and they often left some parts of application unmapped. Especially, when the web application is big and complicated or use lot of client-side functionalities. It is also very difficult to notice when some part is missing from the site map. Every content of the site map should be verified manually and it would be as laborious as manually crawl the application. [27, p. 778–780]

Active scanner tests requests from the site map individually one by one. It selects one request, use a crafted input in one parameter in the request and send the crafted request for that potential vulnerability is revealed. This is continued until every test are executed in every parameter in the request before the scanner switch to a next request. This means that vulnerabilities in multi-step functionalities are not tested. Some vulnerabilities require that a crafted input is submitted in one or more step and then it is observed the rest of process for vulnerability signature. Then there are also vulnerabilities, which occur only when sequence of steps is changed or when multiple crafted input parameter values is used at the same time in one request. [27, p. 776–777]

Request isolation cause problems also in web applications, which hold data in both client side and server side, and update its state to the server via asynchronous requests. These requests are added to site map when the application is crawled and tested one by one by active scanner. However, the application use these usually in a certain order or before some preconditions are achieved. The scanner should understand multistage request processes to reveal all possible vulnerabilities. In other words, the scanner should be able to change the state of the application so that it is in the desired state to handle a particular attack request. This problem can be understood by thinking different code paths that are executed, when the application is in different states. Vulnerabilities exist in some code path and the vulnerability is never revealed if the execution cannot reach that specific path for example because some pre-condition has been verified. This logic is explained more

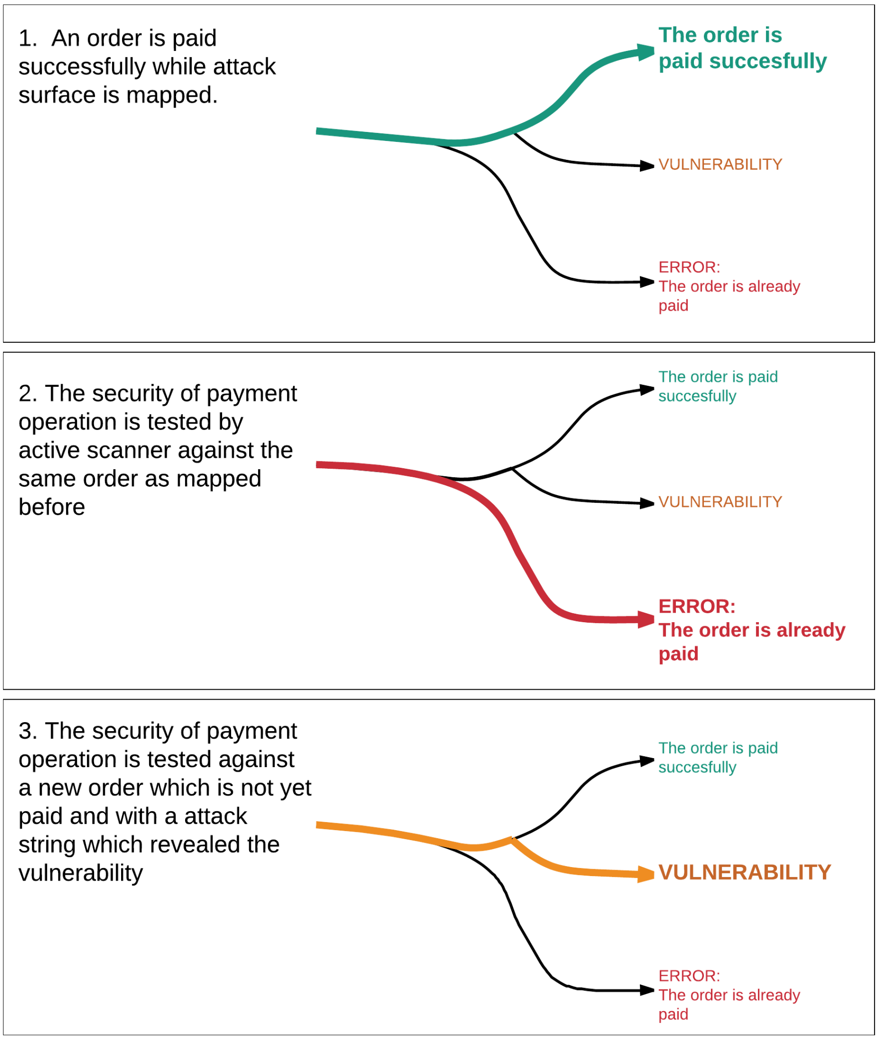

***Figure 4.2** Code paths of payment operation*

specifically in the figure 4.2 that shows the possible code paths of order payment operation. In the first picture, the order is paid successfully while the application is crawled manually. In the second picture, the same request is tried to execute again by the active scanner but it does not succeed because that request still points to the same order as crawled in the first picture. The execution won't never go to the code path, where the vulnerability locates, even if the correct attack string is used in the correct parameter because it is noticed in the code that the order is already paid and the execution goes to the other code path. In the third picture it is explained the situation where that vulnerability can be revealed. There a new order, which is not yet paid, is used in the request and a right attack string is used to reveal the vulnerability. So that these situations can be tested it always require human involvement that understand the requirements of the application, knows how to configure the scanner appropriately and monitor how well the scanning is performed. Explained situation can be common because usually in an application implementation the pre-conditions have been verified at first and only, when these are fulfilled, it is continued to the actual implementation that do modification operations. These operations are more important for safety because these more often touch a database. This problem exists in requests, which does not return same results when these are executed multiple times. [27, p. 780-781]

Any automated vulnerability scanner never provides absolute assurance that the application does not contain anymore any vulnerabilities that scanner claims to be able to detect. The effectiveness of using vulnerability scanner depends largely on the application you are testing but generally it can be said that the scanners are capable to discover approximately half of these, where a standard signature exists. They cannot find the more subtle and unusual cases with standard signature. Usually these deficiencies are related to uncommon structure of web application or lack of understanding the functionality of the web application. However, could it be possible to configure or extend vulnerability scanner to work better with a specific web application if it is well known how the application works. For example if the vulnerability scanner can itself establish different preconditions in the system would it be possible to get better results and reach a level of reliability where the results of the scanner can be really trusted. Also, could it be possible to detect more advanced application specific vulnerabilities automatically if it is possible to create tailored scans that understand the logic of the application. These requires at least a good security testing team with wide scale excellencies about security testing and programming. [27, p. 783–784]

4.5 Fully Automated Penetration Testing

Automatic testing tools are widely used for helping penetration testing because otherwise testing is not effective [26]. These automate the worst time and money consuming operations but do not fully automate every task so that whole security testing round can be executed without human intervention. The modern trend is that all testing should be automated but removing all human intervention may cause more disadvantages than advantages if the automation is not made well enough. For example, if automatic testing causes too much false positive alarms, correct alarms remain in the shadow of false alarms. This causes also that no one wants or have time to check and analyze the results to find the correct issues. If this continues for a long time, step by step the results of automatic testing system are not trusted anymore and it becomes useless. Before the security testing can fully automated it should be sure that the quality of the security testing is good enough. Selected system or tool can really find vulnerabilities automatically and there isn't false alarms or the amount these is very low.

If some existing penetration testing tool is wanted to use as base for fully automated security testing system the tool should have features to automate every three phases in the penetration testing:

1. There should be way to automatically retrieve and map the attack surface and teach it to the selected penetration testing tool. In some cases, it is enough that the content of previous testing round can be maintain and reuse for next testing round. This can be used if the application interface does not change much and the changes is easy to add but usually security testing is more important when new items is added to the interface. It would be good if the attack surface would stay up-to-date automatically.
2. There should be way to automatically start the scanners after the first phase is done. Scanner should work as perfect as possible so that it would report only actual issues.
3. There should be way to programmatically go through all find issues to make a report and analyze these so that potential false alarms can be discarded. The best case would be that there won't be any false alarms and all false alarms would be fixed by improving scans.

So that the these can be achieved, the team who maintains the system and analyze results is needed to be excellent in both security testing and software engineering so that they can find the reasons for false alarms and fix these to the automatic system for that the system can provide better results next time. Systems cannot also automatically find new vulnerability types without that an existing test is reviewed and added to the system or a new test is implemented to the system.

Automated security testing would be important nowadays in continues integration where SDLC (Software Development Life Cycle) is very short and a new version is published every month or even every week. By using manual penetration testing it is not possible to test the security of every published version well enough at least when the application is really large.

4.6 OWASP Zed Attack Proxy Project

OWASP (The Open Web Application Security Project) is a not-for-profit charitable organization, which goal is to add visibility about web application security and improve the safety and security of the softwares in the world. OWASP has multiple projects which develop security tools, produce documents about security and provide place for forums that all are free and open to everyone interested in improving application security. Core values of OWASP are to be honest, truthful, transparent and vendor neutral, encourage to innovate solutions to software security challenges and encourage anyone around the world participate in the OWASP community. [18][23]

Multiple project under OWASP brand are related to developing security software [19]. One of developed software is OWASP ZAP (Zed Attack Proxy), which is a DAST tool working as intercepting HTTP proxy. ZAP describes itself as integrated penetration testing tool, which is easy to use when finding vulnerabilities in web applications. Like all projects under OWASP brand ZAP is open source project and it is completely free without that user needs paid for a better pro version. Development of OWASP is started based on source code of Paros Proxy that had been popular intercepting proxy in early 2000 but its development was ended around 2006 [24]. OWASP ZAP has become popular since its first version was released in 2010. It has very active community and forums for users and developers and it is developed actively. [21][22]

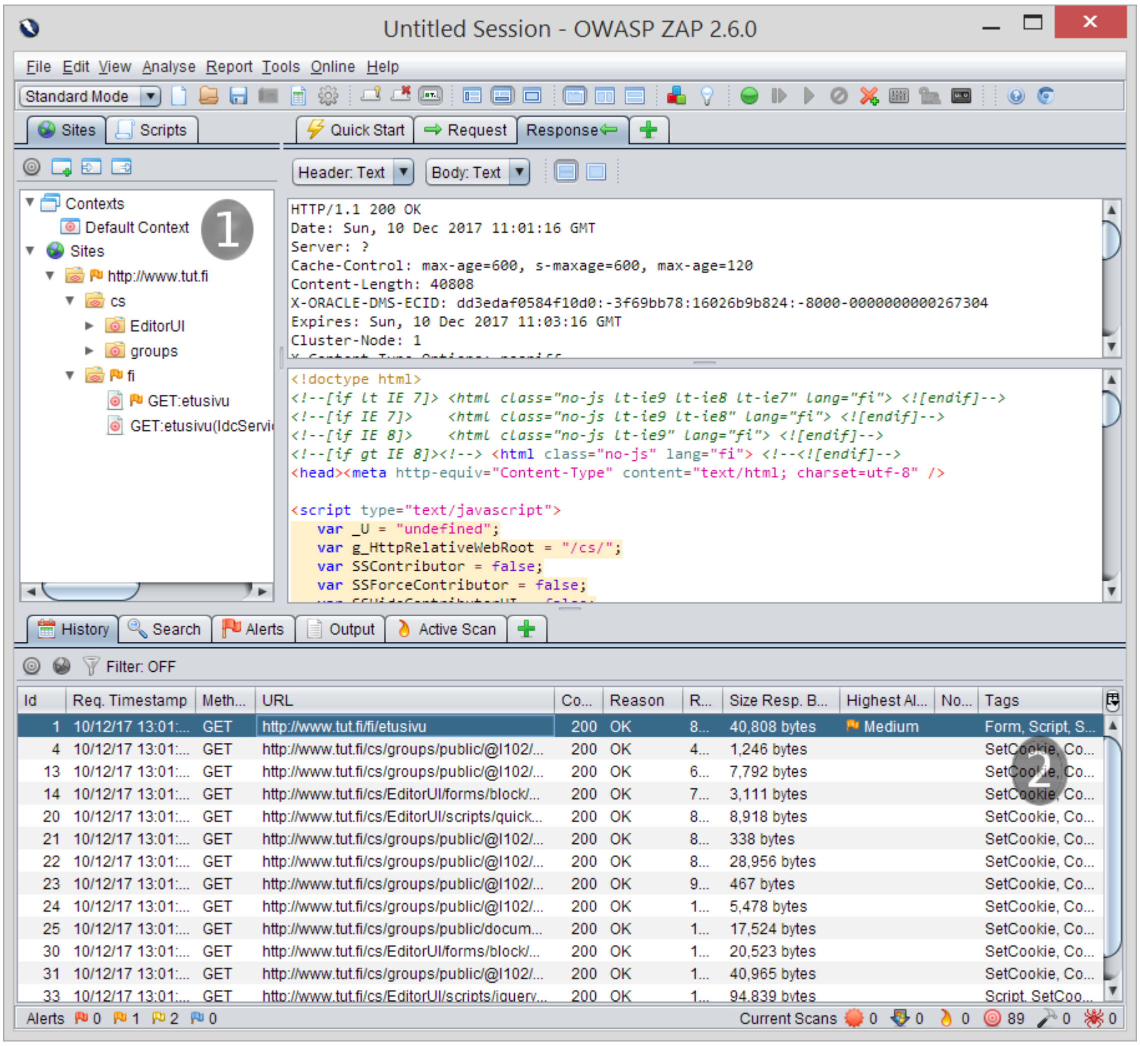

Figure 4.3 *UI of OWASP ZAP*

OWASP ZAP is helpful tool for all penetration testers regardless of their experience.

It can be used for automated vulnerability scanning without actual understanding about vulnerabilities but it has also multiple helpful tools for manual security testing. All in all, OWASP ZAP has multiple features that are common for all intercepting proxies but it has also other features. Some of these features are created as add-ons but in OWASP ZAP it means the same as a built-in feature. Next it is listed and explained some of most important features of OWASP ZAP:

- **Intercepting HTTP proxy** that is started simultaneously with OWASP ZAP to the default port that is 8080 or freely selected custom port. It allows user to see all requests and responses, which have gone through the proxy and modify these on the fly in set break points. [21]
- **History** pane where is seen all requests and their responses that have gone through the proxy. Pane 2 in figure 4.3. [21]
- **Site Map** that describe the structure of a web application in tree view mode. Pane 1 in figure 4.3. [21]
- **Spider** that work like a traditional web site spider. It finds additional URLs from links in HTML code. Also some known pages and files, which are added to web application for revealing the structure of the site, were read. For example Robots.txt. [21]
- **AJAX Spider** that is a web application spider. It uses web UI test automation tool Selenium to open inside the browser so that all client-side scripts are executed and then click and fill all UI elements on the page so that client sided functionalities of the web application are executed. [21]
- Possibility to manually **resend** a request with any change wanted to test. [21]
- **Fuzzer** is a tool for submitting lots of requests with different attack payloads to a target. Payloads can be based built-in sets, custom scripts or payload generators. The functionality of this tools is based from the OWASP JBroFuzz project and includes files from the fuzzdb project. [21]
- **Passive Scanner** executes passive scans for all requests and responses have gone through the proxy or only messages which are in the selected scope. It is performed in a background for that it does not slow down the exploration of an application. [21]
- **Active Scanner** is needed to start manually. It can be performed one by one for every message or recursively for all message under specific node selected in the site map. [21]

Functionality of OWASP ZAP can be expand by installing free extensions from Marketplace. Some extensions are already installed in default installation and some previously mentioned features was implemented as extension for example AJAX spider. All scans for Passive Scanner and Active Scanner are also implemented as Extensions. By default, release scan sets for Passive and Active Scanner are installed. These contains scanner rules, which are verified to be high quality. Additional of this, there is possible to install beta or alpha quality scanner rules. Beta quality indicates that these scanner rules has reasonable quality but are not yet ready or need further testing. Alpha quality indicates that these scanner rules are in an early stage of development. Beta and alpha quality scanner rules can also find new vulnerabilities but there are no absolute certainty because these are still in progress. [21]

OWASP ZAP can be used also as standalone or fully automated vulnerability scanner. It has very comprehensive REST API and the same information is used also to create APIs for Java, Python, PHP, NodeJS and .NET languages. The API seems to have methods for all operations that are possible manually so all operations, which are necessary to fully automate the standalone vulnerability scanner, are possible. [21]

4.6.1 Adding Additional Functionalities to OWASP ZAP

OWASP ZAP has very good extensibility support that is implemented by creating add-on support. These add-ons have full access to all ZAP internal functionalities and are for this reason powerful tool for creating new features and modifying existing functionalities. All new features of OWASP ZAP like AJAX Spider are already created as extension that should explains how comprehensive the extension support really is. Add-ons are created by Java language. Existing add-ons can be upload from Add-on Marketplace that is built-in feature inside OWASP ZAP and many basic extensions are installed by default if the ZAP is installed by using the default installation package. [21]

One installed extension package can have multiple extension items that are loaded when the extension is installed or when ZAP is restarted. Extensions items are listed in **zapaddon.xml** file that should be locate in the root folder of the extension package. There are three different type for extension items:

- Extension
- Active Scanner Rule
- Passive Scanner Rule

Extension items are executed when application is started. These items are for new feature implementations, API additions and UI modifications. For example, for new buttons, menu items and dialogs that are used to operate the new feature. It is also possible to register listeners for modifying existing functionality to different situation. For example, it is possible to create listeners to check and modify every HTML-request which is received before it is sent to ZAP and again just before ZAP send it forward to a web application. All locations, where a request or a response can be check, modified or own implementation can be added, are seen in figure 4.4. [21]

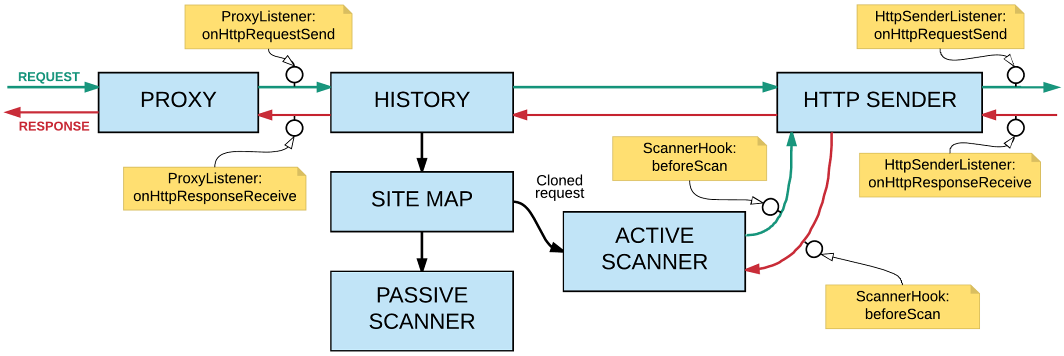

Figure 4.4 *Locations of event handlers which can be used to check and modify requests and responses or add additional features when the execution is reached to this situation.*

Event handlers are registered by inheriting a class from a base class (i.e. HttpSenderListener), implementing its abstract methods and then register the new object from that class by using the correct registration methods. Methods *onHttpRequestSend* and *onHttpResponseReceive* in *ProxyListener* base class can be used to prevent a request or a response to be sent forward, clean requests from parameters, which are not wanted seen in the site map or add a note that is built-in feature of ZAP for sending additional information with a request to other components in ZAP like between these listeners. Similar methods in *HttpSenderListener* are called just before a request is sent to a web application and just after a response is received from the web application This is the last gate before the web application and is good place to make last modifications to requests and responses. Methods *beforeScan* and *afterScan* in *ScannerHook* can be used to implement similar functionalities as with methods in HttpSenderListener but these are executed only while the active scanner is executed. These messages are also gone through HttpSender so also all modifications implemented HttpSenderListnerers are executed. The developer of the extension needs to be careful to that similar operations are not executed twice by accident. [21]

Active Scanner Rule and **Passive Scanner Rule** extension items are used to define new scans to the scanners. It is possible to create as good scan as exiting scans because all existing scanner rules are also created as extension and use same tools. Custom scanner rule are created by implementing a class and inheriting it from a correct base class. A passive scanner rule is created by inheriting the class from *PluginPassiveScanner* and implementing methods *scanHttpRequestSend(HttpMessage msg, int id)* that is called for every request and *scanHttpResponseReceive(HttpMessage msg, int id, Source source)* that is called for every response. Developer can create two different types of active scanner rule.

A rule, which is executed for all requests, is created by inheriting a class from *AbstractAppPlugin* and implementing *scan()* method and a rule, which is executed for all parameters in the requests, is created by inheriting a class from *AbstractAppParamPlugin* and implementing *scan(HttpMessage msg, String param, String value)* method. Parameters, which are tested in this rule and in all existing rules, are selected from UI before active scanner is started. [16] [17]

Active scanner rules are intended to make requests to the tested web application. A copy of the current HTTP message is retriever by using *getNewMsg()* method. This returns a copy of the original request from the site map that is currently tested. This request can be modified in all possible ways before these are sent to the tested web application by using command *sendAndReceive(...)*. After the response is received it can be analyzed and possible findings can be inform by using *bingo(...)* method. The copied HTTP request cannot be reused so if the scan requires to make multiple requests *getNewMsg()* is needed to call for each request. [17]

The easiest way to make small modification to OWASP ZAP is use scripting possibilities added in Script Console extension. This adds scripting support with JSR 223 scripting languages. For example JavaScript. Then there is also two other extension, which add scripting support also with Python and Ruby. [21]

This scripting console contains 8 different script types. These scripts allows developers to make additions and modifications to existing functionalities like with extensions but in limited areas. [21]

Active Rules With active rules script a new active scans by using scripting languages can be created similarly as with the extensions. [21]

Authentication If active scanner or spider has sent logout request, relogin can be automatized with authentication script. For that this is done in correct time, it is needed to define a string or regular expression (Regex) to the setting which tells that the user is not logged in. This authentication script is important because testing fails if request do not have same possibility to provide similar results as when these are mapped. Attack strings used in active scan usually cause different a response message but if the user has been logged out and some vulnerabilities can be left unrevealed. [21]

HTTP Sender HTTP Sender script allows to do similar functionality changes as with HttpSenderListener implemented in a ZAP extension. [21]

Passive Rules Passive rules allows to create custom passive scan by using scripting languages similarly as with an extension. [21]

Proxy Proxy script allows allows to create similar functionality changes as with ProxyListener implemented in a ZAP extension. [21]

Script Input Vector Script Input Vectors are used to define the places in a HTTP request what ZAP should attack. [21]

Stand Alone Stand Alone scripts are scripts that can be executed only manually. [21]

Targeted Targeted scripts are similar as Stand Alone scripts but these are targeted to the specific URLs. [21]

It is important to notice also that additional of these extension possibilities the core of OWASP ZAP can be also modified because it is open source application. Of course, this should never be the first act but if the needed changes can be made easily and change cannot be done by using extension possibilities it is worth of considering. After the changes are made and these are useful for others

5. SECURITY TESTING OF M-FILES WEB

Security testing is important part in software development. Even if security testing can be bought from third party companies, which are professionals in their field, it is necessary that every company that develops software performs also own security testing. That way companies ensures that third party security testing do their work properly. Internal security testing team can execute their testing more freely because more confidential information can be given for them because information leaks does not have to be considered. This additional information can improve testing results and testing coverage. At the same time this will tests also the effectiveness of the purchased service.

5.1 M-Files Product

M-Files is a product especially for Microsoft Windows developed by Finnish software company M-Files. M-Files itself calls their product as EIM (Enterprise Information Management) system because M-Files wants to stand out from old ECM (Enterprise Content Management) system and tell that all information is equally important, needs control and a system to manage it. Not just documents. In other words, M-Files develops similarly named product for corporations to handle all their information. This information includes in addition to documents all meta data related to the document but also all information related to i.e. customers, products, projects, employees and other information that usually do not have any specific document. In M-Files, document is just one object type among the other like customers or projects. [13]

The place where the information is in M-Files is called a vault that all Microsoft Windows users are seen as M-Files (M:) drive on their local computer. Physically the information is located on M-Files Server application that is running on external server and client applications are connected to this server over the local or external Internet. One M-Files server can manage multiple vaults simultaneously.

Even if M-Files drive is seen in the same way as other drives in Windows its differ a lot from folder-based systems like File Explore in Windows. To M-Files, it is more important what the document is instead of where it is stored so M-Files has rejected hierarchical folder based system, where it was necessary to know specific path to find or store a document. In M-Files, the specific document can be categorize based on its meta data. For example, a specification document, which is related to a specific customer and project, can be found by searching all specification documents which are related to this specific project and customer. In folder-based system there can be many good places to save or seek this document. For example, good places are

`D:/Customers/CustomerABC/SpecificationDocuments`

or

`D:/SpecificationDocuments/ProjectXYZ`

and if the system has both folder structures it is difficult to know, which structure should be used in certain situation. Choosing incorrectly can lead to situations where other employees cannot be sure where the correct document can be found or save their own version from the document in the other path and then no one can be sure, where the newest document locates. Main problem there is that this is not automatic and it needs always common agreements that each must respect. In M-Files, correct document version can be found if it has enough meta data. M-Files has also a feature which emulates folder structure called views so the documents can be found also without using search. At underneath, these views are like saved meta data based searches where can be defined different meta data based grouping rules that emulates sub folders. Outward a view looks like a folder structure and it is possible to create views that looks like the folder structures presented in examples above. Because views are based on meta data searches the same document can be found in two places in same time and if the modification is made in one place same modifications are visible immediately in the other place. This difference is explained in figures 5.1 and 5.2. The first figure shows how difficult it can be to find the correct document from folder structure and the second figure explains how in M-Files this is impossible because the same object is found from various paths. [11][13]

M-Files has also simple built in version control that has only four commands: check out, check in, undo check out and roll back. Version control works so that before document or some other file in another object type is wanted to modify user needs to check out for modification. After that moment other users cannot modify the same document or see modifications which the user, who checked out the document, has made. Document is opened always inside M-Files so users never have local copies that could confuse. After the user is ready he checksthe document in and other users can see the modifications and made own modifications. If it is wanted to return to back to the version where modification was started user can undo check out the document and if it is wanted to return back to even older version user can use roll back option. These features force to make modifications so that it is always known, which is the newest version of the information. [11][13]

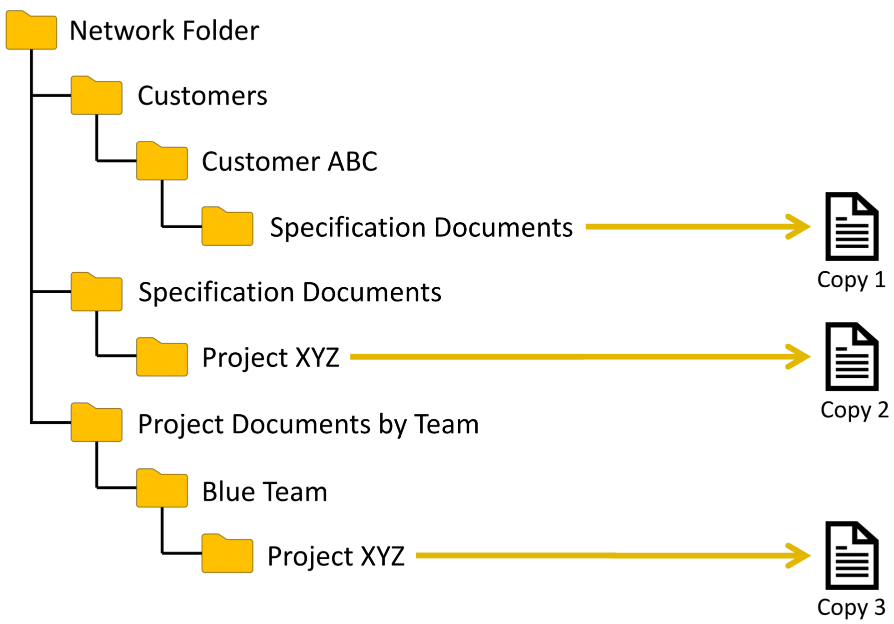

Figure 5.1 *The document can have multiple potential locations in folder based document management system*

Every documents and information object saved in M-Files have individual permissions that restrict how users can see and use the information. Permissions is split to four easily understandable access right that are *read*, *edit*, *delete* and *change permissions* that every can be explicitly allow or deny. There is many ways how to configure automatic adaptive permissions and if the vault is configured well user itself does not have to care about permissions. For example, permissions can be meta data driven so that if a document relates to a project it gets automatically permissions from metadata of the project. It gives all access rights except change permission access right to team member, which working on the project, project leader gets all permissions including change permission access right and other employees can only see the document. [11]

M-Files is not focused to any specific business area. Out of the box M-files have only basic features presented above and only couple object types. However, M-Files has comprehensive modification possibilities. The aim of M-Files is to adapt to workflows and practices of a corporation and does not require that the corporation should adapt practices that M-Files propose. For this reason, the large part of the deployment of M-Files is set up and customizing the vault like defining used meta data structure elements, workflows and tailor-made vault extensions. Meta data structure of the vault can be modified by adding and defining following meta data elements:

Object Type Object type is main element to separate differently typed objects. By default, in a new vault there is only two object types: Docu-

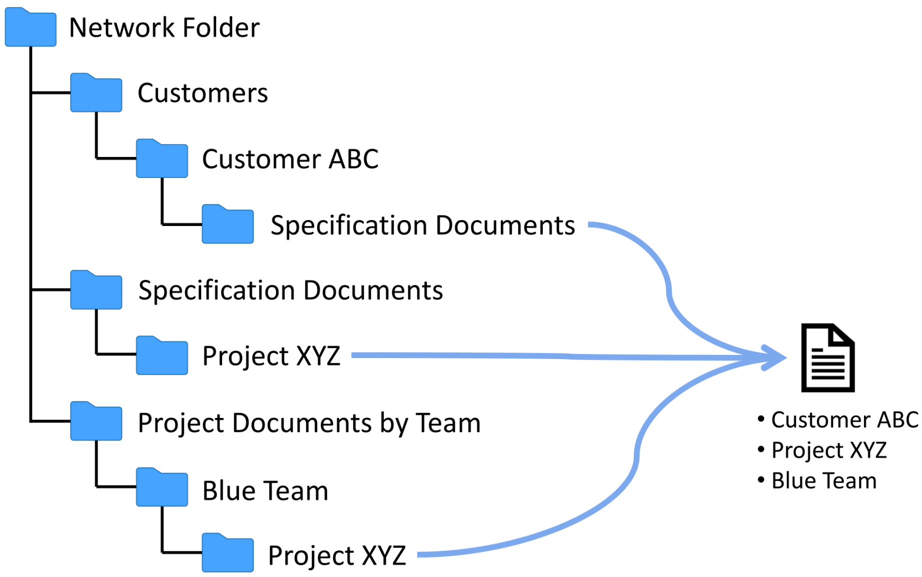

Figure 5.2 *M-Files can have multiple views that emulates different folder structures and the same document is accessible via multiple paths.*

ment and Assignment. Additional object types can be created freely. Good example for object types are Project, Customer, Product or Employee.

Class Every object in the vault belongs to some class. Class is object type specific and contains information about which properties need to be defined to object meta data when object is created.Property can be mandatory or non-mandatory. For example, project object type can have classes Internal Project and Customer Project. Customer project have for example properties Name, Customer, Responsible Person, Starting Date and End Date.

Value List Value list is a list of string value items. Value list items is like objects but these does not have additional meta data. Just the name. For example, countries or languages are good examples for value list.

Property Definition Property definition defines a property which is used in a class or added additionally to a metadata of an object. Property definition has a data type which can be text, multi-line text, Integer, a double-precision floating point, date, time, boolean, lookup or multi-line lookup. Lookup properties are properties which add relation to a value list item or to an object. For example, responsible person property can be lookup property which contain values of Employee objects.

Workflow Every object in M-Files can have to a workflow, which is sequence of states, where the document belongs during its lifetime. Various workflows can be freely configured and set as a default or mandatory workflow for objects which belong to a specific class. Various operations can be made to the object while its state is changed. For example, it can be converted to PDF or change permission.

User group Users created in M-Files or another user groups can be grouped to bigger user groups which then can be used in access control lists or in metadata of object.

Operations in M-Files can be divided in to two groups. These are client operations and administration operations. Client operations are operations that a user can do without any administrator privileges such as viewing and modifying object meta data and possible delete or delete objects. Client operations for an object or metadata structure elements can be restricted defining permissions for this specific object. Permissions of metadata structure element can be changed so that some users or user groups cannot see that metadata element or cannot modify for example a property. Administrator operations are operations which always need some specific privileges before a user is able to execute these. For example, undeleting deleted items, modifying metadata structure are administrator operations. [11]

5.1.1 Connecting to M-Files Vault

Before any client or server operations can be executed in the vault the user needs a login account. The user can have M-Files specific login account but also windows account can be mapped to M-Files.

Users in M-Files can login to the vault by using different platforms. At the moment M-Files has client applications for Microsoft Windows, web browser, Android and IOS that are only for client operations. Additional of this there is M-Files Admin for Windows that is used for admin operations. All these application and how these communicate with M-Files Server is seen in the figure 5.3. Desktop client for Windows, M-Files Admin and M-Files API uses RPC (Remote Procedure Call) over TCP protocol. M-Files API is COM interface that contains methods for all client and admin operations that are possible also manually. It has been used for creating M-Files Web Service (MFWS), which is REST like HTTP API and is the main target in this master thesis, which security testing

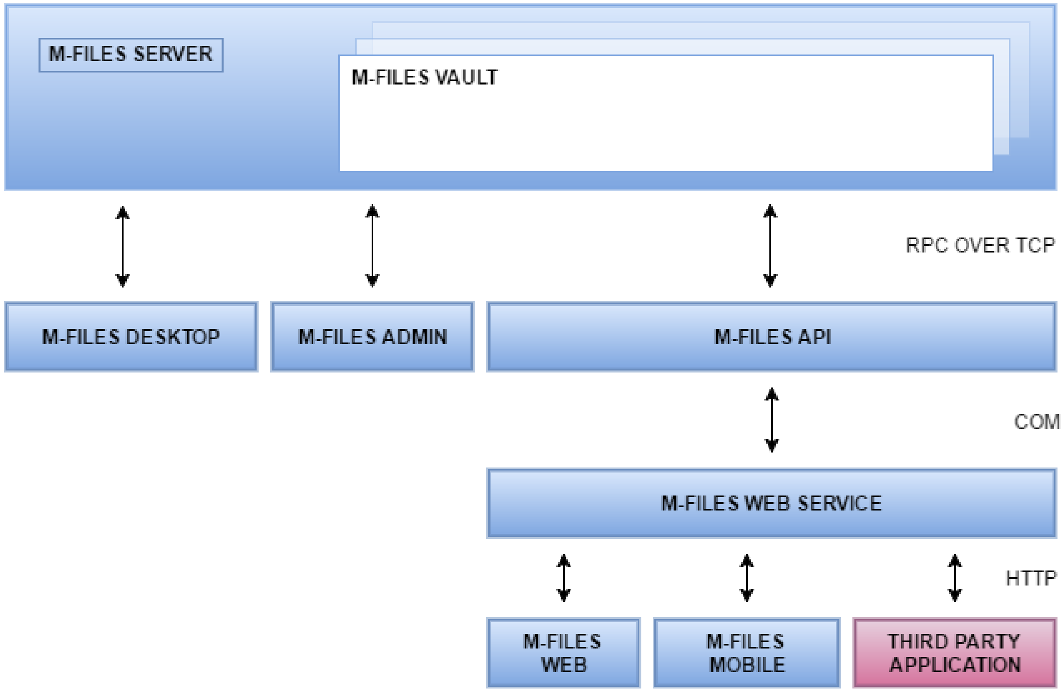

Figure 5.3 *Different M-Files clients uses different technologies and protocols to access the vault in M-Files server. M-Files Desktop, M-Files Admin and M-Files API uses RPC over TCP. Mobile application and M-Files Web uses M-Files Web Service which is implemented by using M-Files COM API.*

is improved. MFWS is used for creating M-Files Web client for all popular web browsers and mobile applications for Android and IOS. M-Files Web Service like also M-Files API are public interfaces and has public documentations so these are used also for creating third party applications. MFWS is not enabled by default but is configured and enabled simultaneously when M-Files Web is configured for the M-Files server. [9] [11]

M-Files Web Service (MFWS)

M-Files Web Service is a HTTP API for executing client operations to the vault. It describe itself as "REST like" interface and it resembles a lot REST interface but does not follow every detail of REST architecture style. For example, MFWS uses URIs to identify resources and HTTP methods almost correctly to utilize uniform interface but HATEOAS is not used to deliver relationships between resources and communication is not always stateless because some requests expects that the vault is in the specific state before they succeed. For example, some requests returns error if the object in M-Files is not checked out before the request is executed.[12]

Because M-Files Web and M-Files Mobile applications use this interface it is not possible to use M-Files Web or M-Files mobile applications without that this interface is enabled and available for everyone. Of course M-Files credentials are needed before it is possible to access to information in the vault but the interface is still visible and anyone can try access and diagnose its security.[12]

MFWS has public documentation that means it cannot be sure how many third-party applications use it. This makes harder to modify the service because it has not been wanted to break third-party applications. For this reason MFWS contains methods, which are deprecated and are not used anymore either in M-Files Web or M-Files mobile applications. The documentation is not also updated for a while so it has deficiencies. MFWS has multiple methods, which are used in both M-Files Web and M-Files Mobile, but it has also methods that are used only in one of the application because M-Files Web and M-Files Mobile applications support differing features. [11][12]

Like in REST API M-Files Web Service consist of resources and these resources can be accessed via resource hierarchy. There are five big hierarchies where most of resources belongs. These are objects hierarchy, views hierarchy, vault structure hierarchy, server hierarchy and session hierarchy. Resources in body are always presented as JSON and there isn't support for XML. [12]

For example a new file can be added to a M-Files object, which is checked out, by using command

```
POST /objects/<object type id>/<object id>/<version>/files,
```

properties of an object can retrieved with command

```
GET /objects/<object type id>/<object id>/<version>/properties
```

and an object can be checked out with command

```
GET /objects/<object type id>/<object id>/checkout.
```

The last command does not follow REST architecture quite well because this change the state of the object by using GET method that should be idempotent and should not change anything on the server. Additional of this MFWS contains multiple method, which are allowed only for objects, which are checked out. Like the first command.

Requests to MFWS can be authenticated by using three different methods. Credentials can be passed in HTTP header of each request, use cookie-based session or use authentication tokens. Passing the username and the password in the HTTP header is the easiest way but it is secure only when using secure HTTPS connection because credentials are passed as plain text in each request. Cookie-based authentication is most useful in browser environments where the browser can set and store cookies automatically. Authentication tokens are best solution when browser is not used because this encrypt the credentials by using asymmetric encryption and does not have same insecurity problems as when credentials are passed in HTTP header as plain text. [12]

5.1.2 M-Files Extensibility

As told previously M-Files has very comprehensive extensibility support. Most of customizations are possible to implement by using built-in features but more advanced customizations is needed to implement by using vault applications or scripting possibilities in various situations. The UI of Windows and web clients can be extended by using UI Extensibility framework that can be used to create new commands, buttons, dashboards and new functionalities to UI. All in all features of M-Files can split to three groups:

1. Features, which exist in every implemented system i.e. version control, file operations, metadata, permissions and authentication.
2. Features, which providers changing and reformation possibilities by changing metadata structure of the vault. For example, by creating and configuring new Object Types, Classes, Properties and Workflows. All reformation possibilities of these features can be somehow known beforehand even if these allows multiple different combination.
3. Features, which allows to modify and extend basic functionalities arbitrary ways. For example, by implementing vault applications, using scripting support or using UI Extensibility framework. These modifications cannot be predict so these can cause strange incidents like additional issues and vulnerabilities that are difficult to prevent.

A vault application is a dll (Dynamic-link library) created by using C#. Methods in this dll are called by using VBScript in event handlers that are triggered in various situations. There are over 70 different event handlers in M-Files that can be implement. For example, it is possible to make own modifications to the vault after a new object is created or after an existing object is modified or check object properties before the changes are saved. Additional of these it is possible to implement Vault Extension Methods. These are manually triggered methods that are implemented by using VBScript but the execution can be delegated to a vault application similarly as with event handlers. These can be called by using M-Files API or MFWS so vault execution methods can be used to implement new methods to M-Files API and MFWS. It is easy way to execute complex operations via API, which are not related to any specific event in the vault. Vault Extension Method has a name and can have one string input parameter and can produce string output. [9][10][11][12]

5.2 Security Risks in M-Files Products

M-Files offer secure place for most important and in the same time also the most confidential information of a corporation. It is very important that the security of M-Files product is in good shape. If M-Files product causes information leak this causes big problems to a corporation, which confidential information is leaked, but also this causes as big problems to M-Files because customers lose their faith for M-Files and switch to another product. For this reason, even one negative case can cause permanent effect to deals.

M-Files is affected by many security risks. Some of these are very general but some are related to the specific client platform. M-Files has for example following security risks:

- Someone has an opportunity to get access to information in M-Files vault without any credentials.
- Someone who has credentials to M-Files vault has opportunity to access information, which should be permitted for his/her.
- It is possible to break the system with or without credentials.
- It is possible to take the system under control.
- Privacy of users can be compromised.
- Vault application or UI Extension application has made a hole, which endangers the entire system.
- M-Files Web client, M-Files Web Service or some other client software of M-Files has a vulnerability, which endangers the user, M-Files Server or confidential information in the vault.

M-Files Web and Web service are most vulnerable parts in M-Files because it is easy and familiar for attackers to start seeking vulnerabilities. Browser and HTTP communication specific vulnerabilities are investigated a lot and these have a lot known vulnerabilities. There is also a lot of existing user-friendly and free tools to intercept and investigate HTTP request and responses. Communication between Windows desktop client and M-Files Server has been implemented by using self-made RPC protocol. This traffic is more difficult to investigate because there are not any existing tools and there are not existing vulnerabilities. For these reason it can be considered more secure even if it can contain vulnerabilities.

Vault applications and UI Ext applications can create additional vulnerabilities and security risks that are difficult to prevent by M-Files itself. This is due to chosen approach that does not prevent any extensibility functionalities from vault applications. It is not wanted to exclude any operations in vault applications for that it may pose a risk because this reduce extensibility possibilities. For this reason, vault applications have privileges that is not always necessary. Vault applications and UI Ext applications can be made so that these does create additional vulnerabilities but this is not forced any way. It is chosen that it is responsibility of the developer of vault application that it does not contain any security holes.

Invalidly configured permissions cause also security risks because permissions of M-Files object can be based different configurations and it is not easy task to configure them correctly. It can be for example based on related objects metadata, object class or manually selected users and user groups. When these are configured invalidly some users can accidentally see information, which should be available only for small group only because object permission has an invalid rule that allows users selected in related object's metadata. M-Files is also very difficult to prevent these and it is responsibility of the vault administrator to configure permissions correctly. Responsibility of M-Files is to ensure that the permissions works as has been said.

5.3 Security Testing in Software Company Compared to Purchased Service

When security testing is purchased as service from third party company that is specialized to penetration testing both parties sign NDA (Non-Disclosure Agreement) and so called "Permission to hack"agreement. It is also defined following details:

- What part of the organization or what software should be tested.
- When the penetration testing is started and when it should be ended.
- Methodology that should be used when a penetration testing is executed.
- What is goals of a penetration testing.
- The allowed and disallowed penetration testing techniques. For example, does DoS testing allowed.
- Liabilities and responsibilities if something breaks because penetration testing activities.

Definitions and restrictions are necessary because usually the organization who has purchased the service has made the test environment and locates the internal network of the organization or the penetration testing is conducted against a product environment. [26]

These agreements are unnecessary if the security testing is conducted by the software company's own security testing team. They have already sign NDA when they have began in the company and they have also more knowledge about the software going to test so they usually know how to build the testing environment or at least are allowed to manage the environment itself because the whole testing is performed inside the company's internal network. The testing is seldom carried out against any production environment.

That way testers are more freely to test different situation and do have to worry about if the environment is not usable after penetration tests.

The security testing differs also how much information is given for the testing team. If security testing is bought from a third-party company, all possible information is not always wanted to give. Some information is considered too confidential to be given to third party even if this information could be important revealing vulnerabilities and NDA agreements are signed. There is two different school of thoughts. Some wants that penetration tester and ethical hackers have just that information what a malicious hacker would have and then there is a group who things that all possible information should be given to penetration testers so that they would have the best chances to find all vulnerabilities. First way of thinking tests how likely a hacker can break in the system and how ell the detection systems detect cyber-attacks. Second way of thinking give a change to find as much vulnerabilities as possible. The security testing performed inside software company resembles always more the second method. [2]

The information, which is not usually given to third party penetration testers, are non-public documentations, source codes and technical details about the software. However, these are information, which internal penetration testers can see. Technical details how the applications is implemented helps to create more suitable attacks to find technology specific vulnerabilities. Source code is important because vulnerabilities can be found directly from the code but it can be used also to analyze how the application really works. Non-public documentation can contain details and features, which are hidden from public documentation. There are many reasons why some features are hidden from publicity. The feature can be deprecated but it is still used by some important customers and for this reason this feature cannot be totally removed. Of course, if the feature is not publicly documented it is not very likely that attacker can found this information but if undocumented, deprecated or otherwise hidden feature has vulnerabilities it is still security issue. Information about deprecated features can be seek for example by examining old public documentations.

5.4 Problems with vulnerability scanners while testing M-Files Web

As explained previously, the performance of a vulnerability scanner is highly related to how well the scanner is implemented tp understand the structure and functionality of the tested web application. Most scanners try to work best with general web applications. If structure or functionality are far from what the tool except the test results can be very bad. This is more probable if the tested application is very complex or it is implemented by using advanced technologies. Compatibility of M-Files Web with vulnerability scanners was investigated by using above-mentioned OWASP ZAP. It wasn't focused to investigate how many different vulnerabilities OWASP ZAP can find from M-Files Web Service but identifying the problems and technical challenges, which are common for all vulnerability scanners, when the scanner is used for scanning M-Files Web Service. These causes deficiencies to overall quality of security testing and if these are not known these causes either illusion that the application is safe or a feeling that the scanner only produce false alarms. The problems that were common for vulnerability scanners and were encountered when OWASP ZAP was used to test the security M-Files Web Service are explained in next chapters..

Problems when Crawling MFWS

Different built in crawling methods of ZAP were tested and it was discovered that best solutions at this moment was still obtained to crawl M-Files Web Service manually through M-Files Web like before. It was only way to introduce every request type used in M-Files Web. Traditional spider that try to find additional links from source code didn't work at all because the main page of M-Files Web does not contain any links to another pages. Different pages are opened by uploading different content to the same page by using methods in MFWS that were executed by using JavaScript. The most important would be introduce every request type in MFWS but automatic spider does have enough information to crawl these by using responses of MFWS because the resources of MFWS does not contain information about sub resources like in the REST specification told they should have.

Ajax spider of OWASP ZAP that really push every button and link in the web page and executes JavaScript has theoretical possibility to crawl every corner of M-Files Web but the procedure how the spider clicks different buttons and links is so random and because clicking possibilities in M-Files Web is so huge that time which this is going to take is too long. It never ends and even if the execution was ended by force after couple hours the spider was crawled only basic functionalities and these were crawled multiple times because same operations can be made for all objects in M-Files. The spider also creates new content to M-Files that makes even more difficult that spider might find mode complex features in M-Files Web. Some features need exact series of operations and it is reasonable to wait that AJAX spider finds them by accident.

Either using manual crawling through M-Files Web is not the best possible solution because it never introduces every possible methods of MFWS. It contains every time human error and some parts can be left taught by accident. Using the same session from a previous testing round where requests are already introduced as well as possible is not good solution either because this does not take in to account new or changed requests and most often new vulnerabilities are found in these requests. Even if the manual teaching would be perfect every time for example by using some automatic UI testing tool, which can define to programmatically click wanted buttons in M-Files Web in specific order and emulating to usage which touch every feature, only request used in M-Files Web can be introduced. This approach misses all requests which is used only in M-Files mobile clients and requests which is not used in any application created by M-Files. Some not used methods have been marked also as deprecated. The best way to minimize vulnerabilities from these methods would be remove these from the interface but this cannot be done because is in not known if these are used in some third-party applications. These missing requests are still important because the interface is public and available for everyone after M-Files Web is configured.

Manual crawling cause also that requests, which executes the same method in MFWS, are added to site map unnecessary multiple times. This happen because some methods of MFWS are very commonly used in M-Files Web. These requests manage different object so they have different URL but OWASP ZAP does not understand that these are used as parameters in the same method and add these to site map. This does not cause any harm to testing coverage but extends testing time noticeably because these are unnecessary tested separately.

Scanning XSS-Vulnerabilities Cause Lot of False Alarms

The scan in OWASP ZAP that checks XSS-vulnerabilities, cause lot of false alarms and never finds real XSS-vulnerabilities from M-Files Web. OWASP ZAP tests requests individually and assume that every response would be a web page that is shown and executed in a browser. If OWASP ZAP notices that the response has javascript, which is not escaped, it alarms about XSS vulnerability.

MFWS sends responses without escaping and leaves it to the developer's responsibility that uses the service to escape the responses before these are shown in the browser. In M-Files, Web escaping is implemented this way just before showing. XSS-vulnerabilities are not revealed because it is not tested how MFWS responses are shown in the website. This problem is explained in figure 5.4. XSS vulnerabilities are quite impossible to test by using OWASP ZAP so this vulnerability should be covered by using some other tool.

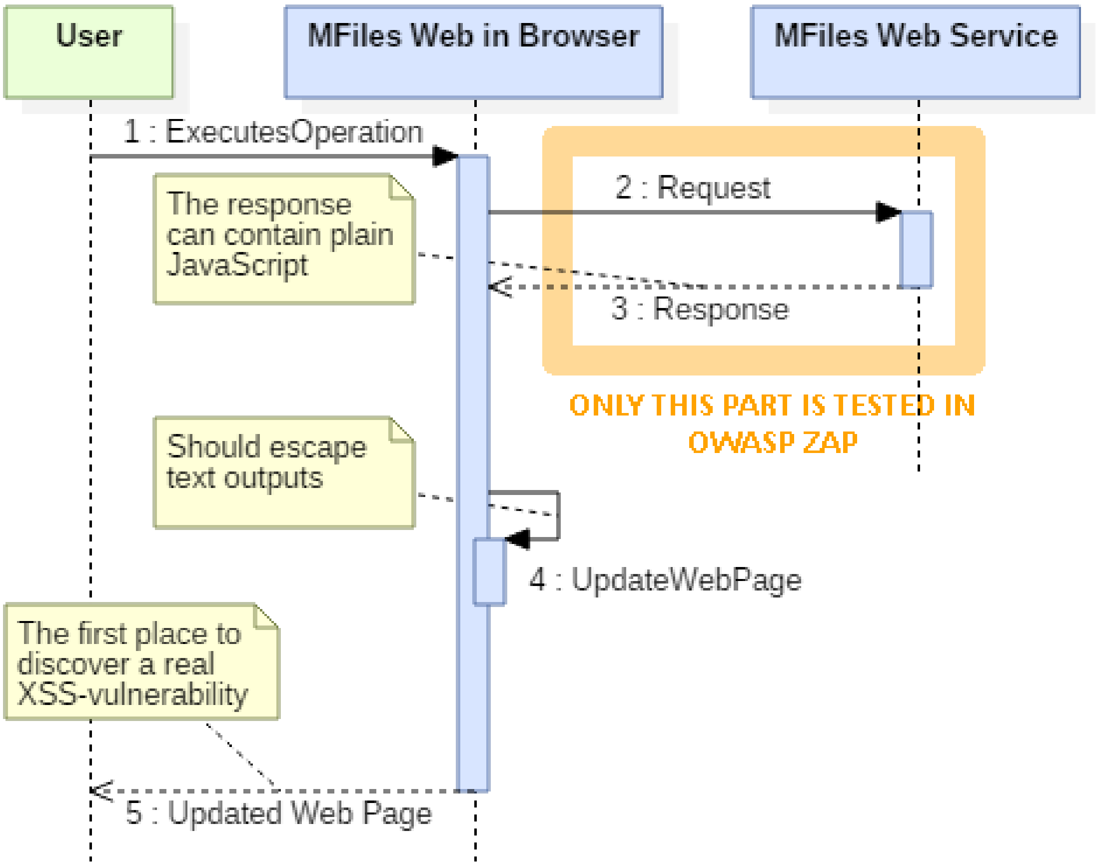

Figure 5.4 Real XSS vulnerabilities are not revealed because OWASP ZAP or other vulnerability scanners does not test how the responses are escaped in the website.

Plenty Unnecessary Attack Strings Was Tested

OWASP ZAP creates lot of unnecessary test executions because with default settings it assumes that web site can use all possible technologies and try to attack by technology specific attack vectors. For example, why it is necessary to test MySQL specific SQL injection attacks if it is known that MySQL cannot be used with M-Files Product. This is not a obligatory problem to fix because this cause only slowness in the test execution but if it is necessary to speed up the executions this is good and easy way to start.

The State of the Vault Does Not Stay Stable when Active Scanner Is Executed

As explained before, the vulnerability scanners have problems to understand multistage processes of today's web applications. These problems were encountered also, when M-Files Web was tested and it causes that many requests were not tested as carefully as other. When the scanner scans vulnerabilities by setting attack strings one by one in every parameter value in the request, it send hundreds almost similar requests. If the request modifies an object in the vault so that the same request responses differently next time it is not known anymore that every possible vulnerability is revealed. For example, if the request, which deletes an object, is tested the object is already deleted after the request is introduced to OWASP ZAP. The same request works differently after that because the object does not exist anymore. This causes doubt about testing coverage because it is possible that the method in M-Files Web Service is implemented so that at first it tests if the object exist and only after that tries to delete it. If the object does not exist ever it will be tested only the part that verifies that the object exist. If the vulnerability exists after that it will be left uncovered. Also situations where the request succeed but it does not do anything because requested operation is already done cause doubts because also then it cannot be sure that the execution goes as deep in the code path as when something is really done. These kind of problems are detected in M-Files Web at least in following situations:

- A document, which is transformed from single-file document to multi-file document, causes an error when the request tries to transform a document which already is a multi-file document. Similar problem exists when a document is tried to transform to single-file document.
- Object undeletion can be done successfully only for document which is deleted. Otherwise it causes error.
- An object can be marked as a favorite only once. The request succeeds but causes doubts.
- An object can be removed only once from the favorites. The request succeeds also but causes doubts.
- Multiple requests require either that the object is checked out or checked in before the request can be execute successfully. Biggest problems are caused by requests which check in or check out the object because these cause failures if the object is already in the requested state.
- Object cannot be moved to a next state in a workflow if it is already on that state.

A large part of methods in M-Files Web Service can be tested without problems. Only methods, which modify the content of the vault cause problems. Even if this problem is found only in fraction of tested methods these contain operations, which are most dangerous if these have vulnerabilities. So that this can be fixed there should be some procedure to return the object to the old state so that the next request has same change to succeed as previous one or make a copy of a suitable object and prepare it for testing and change the request so that it points to the new object.

6. IMPROVING COVERAGE OF M-FILES WEB SECURITY TESTING

Two components were implemented in this master thesis. They both improve testing coverage and tries to patch two found problems when security of M-Files Web Service is tested by using a DAST tool. The first it was created a tool for bypass the crawling problems. The tool learns everything about usage of M-Files Web Service from source code and use this information to teach that to a DAST tool. The second component is a PoC (Proof of Context) extension to OWASP ZAP that uses the same information to maintain the state of the vault and improve testing coverage further while scanning is on-going.

6.1 Background to the Chosen Solution

When teaching phase of DAST tool was managed manually it caused several problems that was wanted to overtake. It caused that all methods in MFWS had not been introduced to DAST tool and caused holes in testing coverage. These holes were due to human error, problems in environment, where teaching was executed, and lack of knowledge. It was wanted to find a solution, which would fill these holes permanently and ensures that all future additions and improvements of MFWS would be included automatically or these would be at least informed so that these could be added with minimum amount of work. It was chosen to create an external application that simulates manual crawling. It automatically introduces every request type for methods in MFWS to a selected vulnerability scanner by sending these with suitable values to the testing vault via chosen intercepting proxy. External tool was chosen to implement because this does not force to use any specific vulnerability scanner in future.

First problem that was encountered was to find the place, where the information about every method in MFWS, was retrieved. The web service has public documentation but it has shortages because the documentation was not updated regularly and even in the best case new requests will be added to documentation until the new M-Files version will be published. The security of new requests is wanted to test before the new version so it was never a good idea to find the information from the public documentation. The second option would have internal documentation or list that would contain same information but this documentation is not available at the moment and even if it would be available it should be updated manually. It has always possibility to human error and in the end some request will be forgotten. For these reasons it was decided that the whole MFWS interface was going to be parsed directly from the source code in order to ensure that every request type was certainly introduced.

Parsing is intended to execute always when the created teaching application is executed and it is going always to be used the source code of the M-Files version, which is going to be tested. These gives following benefits:

- It is not needed to trust development teams that they would inform a testing team when a new method is found from the interface.
- New methods are tested instantly after these are implemented to the interface and possible security issues can be fixed immediately.
- Solution retrieves always all possible methods in the interface even if they are deprecated or never used in M-Files Web including methods which are used only in M-Files Mobile.
- If older version was tested solution retrieves always only these methods, which are usable in this specific version.

The purpose was to implement this application very generally and self-learning way with minimum amount hard-coded parameter values. This way we minimum the amount of modification to the application, when new methods will be implemented to MFWS, and the application is not needed to rebuild after every addition to MFWS. Also, when the application is even a little self-learning it is not needed to implement solution for every method one by one but the application understand itself when the introducing succeed or how the introduction should be changed so that it succeed next time.

The application is focused to introduce methods in MFWS than web pages in M-Files Web. This was intended decision because security issues are more severe if these are found from MFWS. Issues in MFWS endangers M-Files Server directly because communication to the server are executed via MFWS. If issues are found only in M-Files Web it does not endanger M-Files Server directly if MFWS is working correctly.

Introducing was started by introducing the positive path. In other words, it is wanted that every introduced request succeed (Status code 200). Negative paths could also have situations, which could reveal additional vulnerabilities but this was left for future development. These are for examples situations where the request does not succeed because it requires specific privileges or all preconditions are not fulfilled.

The application provides that all possible methods in MFWS are introduced to a vulnerability scanner but does not fix every problem in test execution that reduces testing coverage. The second component that was created for this master thesis was a POC extension to OWASP ZAP that tries to solve problem that exist because the scanner assumes that the state of testing environment remains unchanged. The extension operates while scan is on-going. It creates test object before every test execution, updates a tested request to point to the created test object and ensures that all preconditions are fulfilled before ever test is executed. If this concept can be proofed the same idea can be used to implement similar extension also to other vulnerability scanner.

OWASP ZAP was selected for testing these solutions because it hasn't been entirely sure that the ideas that were tried to solve problems would work as expected or would be even possible. It would not be a good idea to choose a expensive program to realize that the ideas were not possible to implement. OWASP ZAP is open source, free to use and it is made so that it can be easily extend without restrictions. It was good and cheap tool to try new ideas. In the future, when the all ideas will be proven to be working and all necessary features will be known, it will be easier to find a new vulnerability scanner tool. OWASP ZAP has also big community so it is easy to ask and find help for problematic situations. It is also investigated that some free or cheap vulnerability scanners works as well as very expensive scanners [27, p. 782]. The open source codes helps also when the functionality of scans created by different developers in open source project are wanted to verify but also gives possibility to modify the core of the application to work better with special situations.

6.2 Areas which are Improved

Mainly it was improved the security testing of M-Files Web Service but simultaneously it was improved the security testing of M-Files Web and M-Files mobile application because this web service was used to implement these applications to discuss with M-Files Server. The purpose in this thesis was not to execute security testing itself and find new vulnerabilities but increase testing coverage so that in the future it will be possible to find new vulnerabilities.

Improved test coverage achieved by better introduction of test area is going to be tested by comparing the amount of introduced MFWS methods when introducing is made manually as good as possible and the amount of parsed MFWS methods by using the created external tool for parsing to source code. Improved test coverage achieved by second component that handle pre-conditions while active scanning is on-going is going to be tested by finding out how many requests can be handled better with the second component and were not tested comprehensive without the component.

6.3 Testing Environment

The environment where the created application and the custom OWASP ZAP extension were tested is shown in figure 6.1. It contains following applications and components:

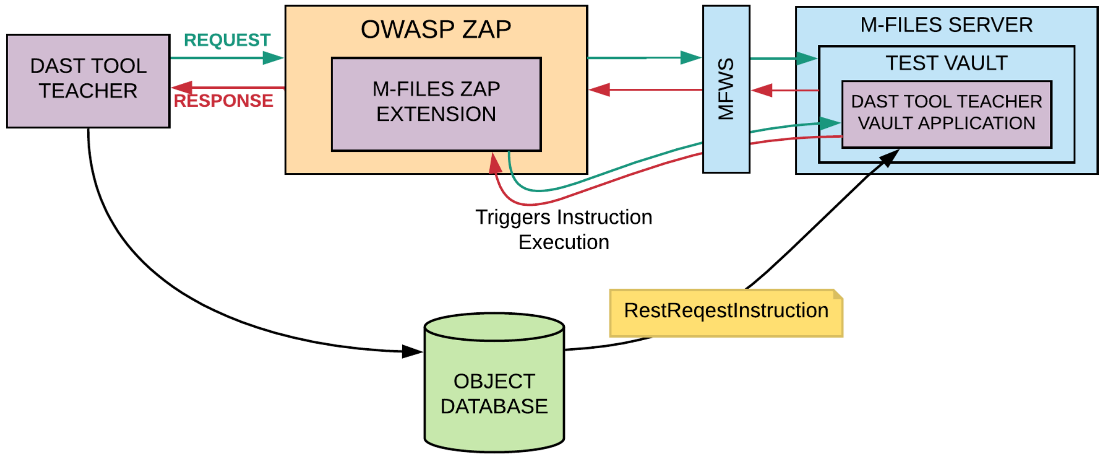

Figure 6.1 *Components and their information flows in testing environment.*

DAST-Tool Teacher is the created application that was used to learn everything necessary about MFWS and execute automatic session creation to DAST-tool so that testing can be started faster and it can be ensured that every time taught data is equally comprehensive.

OWASP ZAP is the DAST tool that was used in this thesis to test the created automatic DAST-tool Teacher and as a system for proofing how the security testing phase can be improved by implementing an extension that prepare vault objects every time before an attack is executed in the active scan.

M-Files ZAP Extension is the extension to ZAP that improves active scan of ZAP by sending requests by following instructions created in DAST-Tool Teacher.

M-Files Server keeps inside the vault which is built for test DAST-Tool Teacher. The vault is designed so that it is as simple as possible so that DAST-Tool Teacher is easy to configure to work with it. M-Files Web is configured in the server so the MFWS is also enabled and testing of MFWS is possible.

DAST Tool Teacher Vault Application is a vault application installed to the test vault. There is implemented various vault extension methods for helping execution of complex vault operations via MFWS.

Object Database is the database in Microsoft SQL Server that is used to convey information between DAST-Tool Teacher and the created extension in OWASP ZAP. This component is necessary so that same information, which is learned in DAST-Tool Teacher, can be immediately used in the extension.

All components in the testing environment are installed in the one virtual machine but in reality there isn't any barriers for that. All components can be separated to own virtual machines or host computers. Component locations are defined in the environment configuration file that is read by DAST-Tool Teacher.

6.4 DAST-Tool Teacher

DAST-Tool Teacher is the application that was created to handle and automate whole reconnaissance phase of DAST tool that was previously made manually. It retrieves and learns whole testing area, creates requests needed to introduce and sends these requests to a web application so that the selected DAST Tool is used as proxy Execution of DAST Tool Teacher can split to three main phases:

1. In the first phase, all methods from MFWS are retrieved from source code and request templates is created by using this information. This phase is explained with more detail in section **6.4.1 Retrieving Testing Interface**.
2. In the second phase, requests are built from templates by using manually defined or template specific values and sending instructions is collected by sending these requests to a test vault with different preconditions or using manually defined instructions. This phase is explained with more detail in section **6.4.2 Learning Phase**.
3. In the third phase, all methods in MFWS are introduced to DAST tool by sending built request following collected instructions or alone with instruction information in the header in that case where the DAST tool is capable to handle preconditions itself. The latter option requires always a custom extension in the DAST-Tool. This phase is explained with more detail in section **6.4.3 Teaching Phase**.

Class diagram of DAST-Tool Teacher is shown in figure 6.2. It contains only most important classes in the application and only methods and fields which are important for understanding the functions of the class.

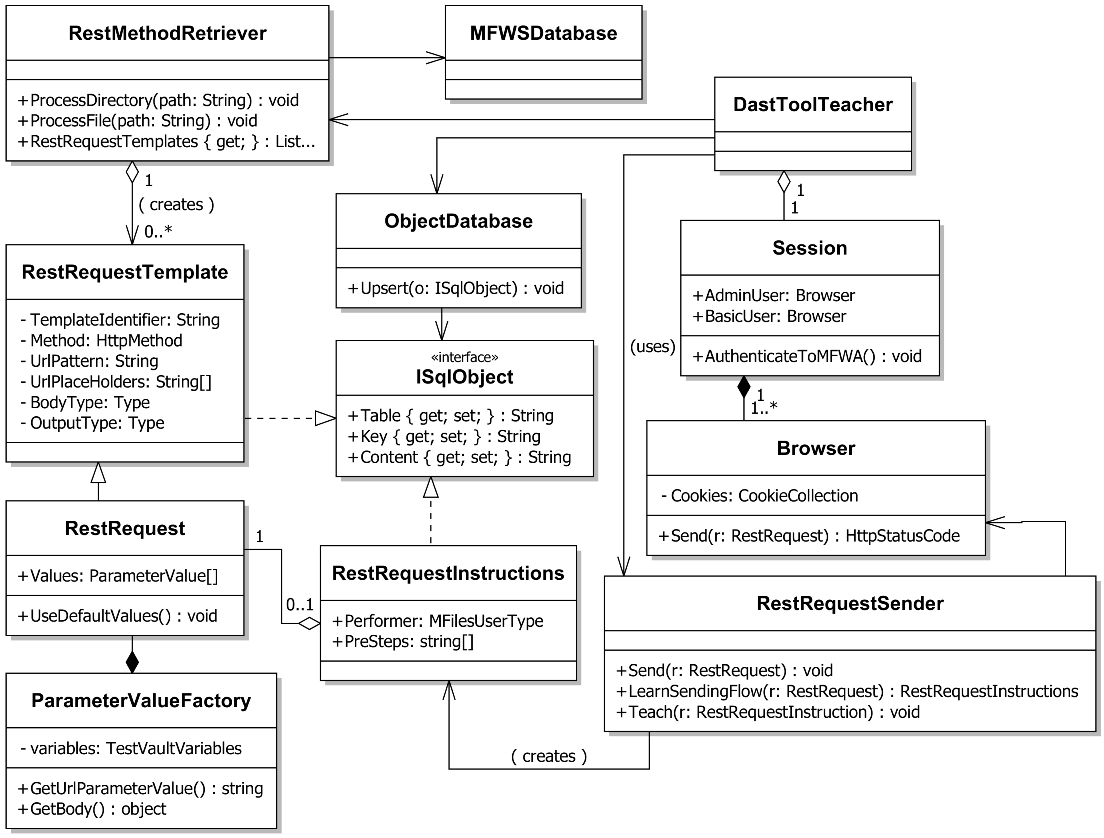

***Figure 6.2** Class diagram of DAST-Tool Teachers*

DastToolTeacher is the class where the execution is started and it orchestrates different phases of DAST-Tool Teacher.

RestMethodRetriever reads a given source file or files in a given folder and find and parse REST-method definitions and creates RestRequestTemplate objects for every method it finds.

MFWSDatabase contains information which is needed in order to resolve parameters and elements of MFWS from the source code. All information of this class is read from configuration file so that more information can be add without that the application is needed to change or a new application version is needed to build.

RestRequestTemplate contains definition of one REST-like method in MFWS without any specific parameter values.

RestRequest extends RestRequestTemplate by adding possibility to add values for parameters. If configured parameter values are wanted to use, values are asked from ParameterValueFactory instance that defines values which are configured to test vault configuration file. After parameter values are set, the object contains all necessary information that is needed to create a corresponding HTTP request.

ParameterValueFactory gives configured parameter values and object for HTTP body of one method when RestRequest is created from RestRequestTemplate.

Browser emulates functions of a web browser and stores cookies and tokens of login session to MFWS. It builds proper HTTP-request by using information in RestRequest object and sends these to M-Files vault via possible HTTP-proxy.

Session contains various Browser instances that are logged to M-Files by using different credentials.

RestRequestSended orchestrate learning and teaching phases of DAST-Tool Teacher.

RestRequestInstructions contains sending instructions of one request so that it can be sent to the vault so that the end condition specified in the instructions are reached.

ObjectDatabase is used to update RestRequestTemplates and RestRequestInstructions objects to database. These classes is needed to implement ISqlObject interface.

ISqlObject is the interface for classes, which is going to be stored in database by using ObjectDatabase.

Functionality of DAST-Tool teacher is configured and modified by using configuration files in JSON format. This approach was selected because that way the application is more suitable for different environments and situations. At total DAST-Tool Teacher follows three different configuration files:

Environment Configuration contains connection, credential and location information for every application and component in a testing environment. This includes connection information of the used intercepting proxy, M-Files Web, M-Files Server and SQL Server. These enables possibility to have every component in a separate machine. The environment configuration file contains also parameters that are related to these components. Like the path to the backup of the test vault, which is restored to M-Files Server and credentials of several users that are used to connecting to the vault after it is restored.

Parameter Value Configuration contains all manually configured default, template and group specific parameter values for building RestRequest instances. The configuration files is a test vault specific and is named in form <The name of test vault>.conf. A template of this configuration file is automatically created if a new test vault, which does not yet have a parameter value configuration file, is used. The structure of this file is seen in figure 6.3.

MFWS Parameters Configuration contains a search index that is used to find correct parameter placeholders to RestRequestTemplates by using parameter names that are used in the source code.

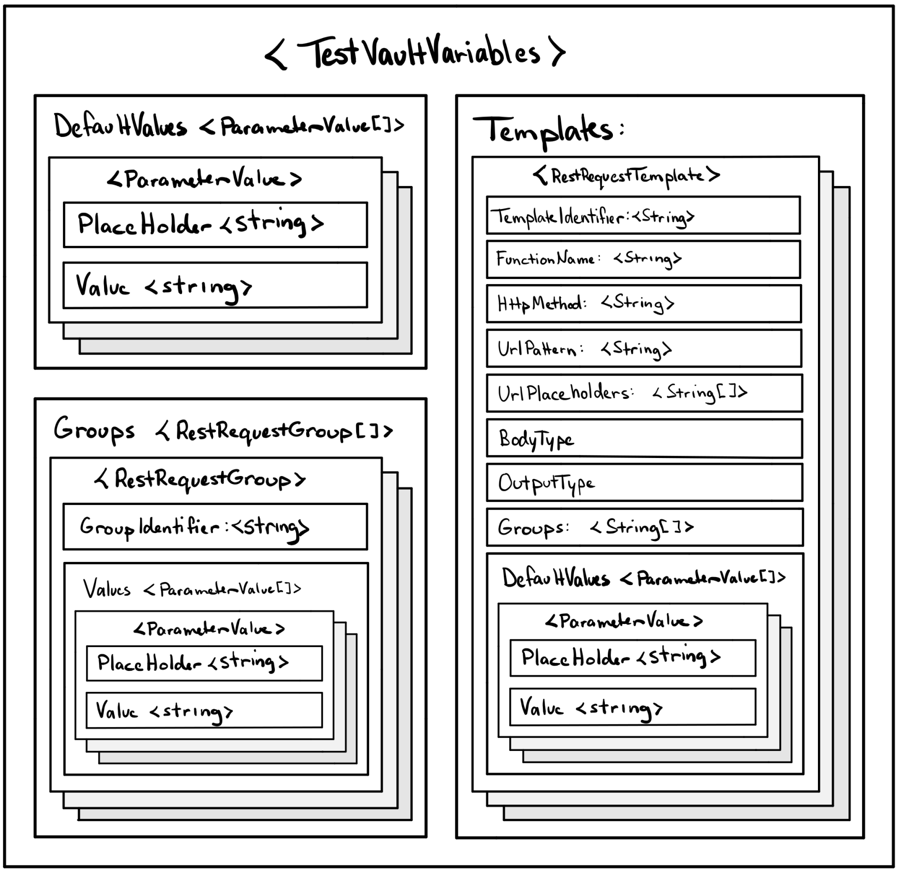

Figure 6.3 *Structure of test vault variables JSON.*

RestRequestTemplates are identified by using a string that is combination of a file name, where the REST-method has been found, a function name that is called, when REST method is sent and a growing number because sometimes multiple REST method calls same or similarly named function.

Datas of RestMethodTemplates are kept also in parameter value configuration so that identical templates from next parsing round can be easier to be identified and templates can have small changes without that all configured parameter values are overwritten. At the moment RestRequestTemplate identifier is the main identifier that means that procedure does not work very well if the function name called by REST-method is changed. This is considered as a different REST-method.

6.4.1 Retrieving Testing Interface

Whole interface of MFWS was retrieved by reading through every source file, which contains implementations of methods of MFWS. Parsing REST-methods from the source code weren't so complicated because all functions used as REST-method in the interface was marked with RestMethod-attribute which were easy to find from source files. Example about source code is seen in figure 6.4. All needed information of one method was got from this attribute line and from following line, where was function definition containing output type, function name and parameters.

```
[RestMethod( "/objects/{type}/{objectid}/{version}/properties", Method = "PUT" )]
public ExtendedObjectVersion SetPropertiesAllowingNameChange(
        string typeID,
        string objectID,
        string version,
        List<PropertyValue> properties )
```

```
{
    "TemplateIdentifier": "ObjectHandler:SetPropertiesAllowingNameChange1",
    "FunctionName": "SetPropertiesAllowingNameChange",
    "MethodText": "PUT",
    "UrlPattern": "/objects/%OBJTYPE_ID%/%OBJECT_ID%/%OBJECT_VERSION%/properties",
    "UrlPlaceHolders": [
        "%OBJTYPE_ID%",
        "%OBJECT_ID%",
        "%OBJECT_VERSION%"
    ],
    "BodyType": "System.Collections.Generic.List`1[[MFiles.Web.Service.Structs.PropertyValue, MFWS]], mscorlib",
    "BodyName": "properties",
    "BodyParameters": [
        "%PROPERTYDEF_ID%"
        "%PROPERTYVALUE_VALUE%",
    ],
    "OutputType": "MFiles.Web.Service.Structs.ExtendedObjectVersion, MFWS",
    "OutputName": "ExtendedObjectVersion",
    "DefaultValues": [],
    "Groups": null,
    "DefaultInstruction": null
}
```

Figure 6.4 *RestRequestTemplates were parsed from RestMethod-attribute and a following function definition.*

From parameters in RestMethod-attribute and in following function definition it was retrieved following information that were then used to build RestRequestTemplate-objects:

HTTP Method HTTP method of the request was specified in the attribute in method parameter if it was some else than GET.

URL Pattern The template for URL was also specified in the attribute. The template contains parameters that were wrapped to brackets. These parameters were replaced with placeholders that define the type parameter and make easier to replace parameters in URL pattern with a parameter value. For example, the placeholder of object type id was %OBJTYPE_ID%.

URL Parameters URL parameters lists placeholders which were used in the URL pattern. URL parameters were defined in two places in the source code. In addition to URL pattern the same parameters were also defined in the same order as function parameters. For resolving the right placeholder, it was created a simple search index where suitable placeholder was found when a parameter name in URL pattern or in a function parameter was given. In most cases these were enough but in some cases the parameters were not named systematically either places so it was not straight forwarding to define the right placeholder to URL pattern. For example in many cases parameters was named only as "id" and every element in M-Files has id so that parameter can be any element in M-Files. It was needed to add more information to search word from previous path elements in the URL pattern. In this case search word was a combination of the original search word and previous path element or elements. These additional searches were done only when previous search was inconclusive.

Request Body Type Every method, for example GET methods, didn't have request body. But if the request has a body the source code the type request body was specified as a last parameter in a function definition after parameters in URL. So if there was less parameters defined in URL template than there was in the function definition the specified method has a body. In MFWS bodies were objects which were written to requests in JSON format by using struct classes. For this reason, full class type used in that last function parameter was saved to identify the body. When request is built same struct class from source code of MFWS can be used for creating correct JSON for request body.

Body Parameters Parameters needed for a request body were retrieved by analyzing, which parameters were used when the request body was built in the implemented ParameterValueFactory.

Response Body Type The output type of the function definition is used to identify the body type in the response. This information was necessary if the response of previous requests was needed use in following requests.

Some functions in the source code have multiple RestMethod-attribute lines. These REST-methods use the same function and will be executed similarly. These should contains same vulnerabilities but it was still created separate RestRequestTemplate objects because interface was wanted to be perfect.

The solution was intended to do very generally so any special solution for one method was not created. All special situations were solved by doing general solution that should work also in similar special solutions in the future. This adds possibility that new methods can be parsed without additions to parsing code. New parameter types cannot be predicted so these needs manual work but these can be define by adding new placeholder in the search index.

By implementing these rules in DAST-Tool Teacher it was obtained templates for all possible requests. Parsed interface contains methods which contains also requests which are deprecated but still usable, requests which are used only in M-Files Mobile and requests which are not used yet anywhere. Totally 393 different REST-method has been found with this tool.

6.4.2 Learning Phase

The RestMethodTemplates that were parsed and created from source code did not have any parameter value information. If the templates had been sent without values the requests would never been succeeded. Random parameter values did not solve situation either because the requests would succeed only by accident. It was wanted that requests go deep enough in the code. Random values cause Not Found, Invalid parameter or other parameter value failures that were mostly checked at first and the execution wouldn't go deep enough. So in the next phase all parameter values are defined for every template so that request can be built. At the same time sending instructions are learned for every request for allowing that wanted end results were reached every time.

For parameter value definition it was created Test Vault Variables configuration file that contains all parameter value definitions for every RestRequestTemplate instance. This was followed when HTTP requests were built. Every parameter type had at least default parameter value but additional of this it was possible to add request, template, or group specific parameter values. Request specific parameter values were used only in internal functionalities and cannot be defined in the configuration file. Groups were parameter value collections where multiple parameter values were defined at once and the template can be belong to multiple groups.

Priority of parameter values is explained in figure 6.5. Parameter values were selected one by one for every parameter type in the RestRequestTemplate. The application can have specified request specific parameter values, which were used even if the template has configured parameter values. After that it was checked template specific parameter values. If the template didn't have parameter value for that parameter type it was next checked did the template belong to any group. If a group or groups were found it was checked these groups in order to find a correct parameter value. If all previous checks were useless default value was used instead.

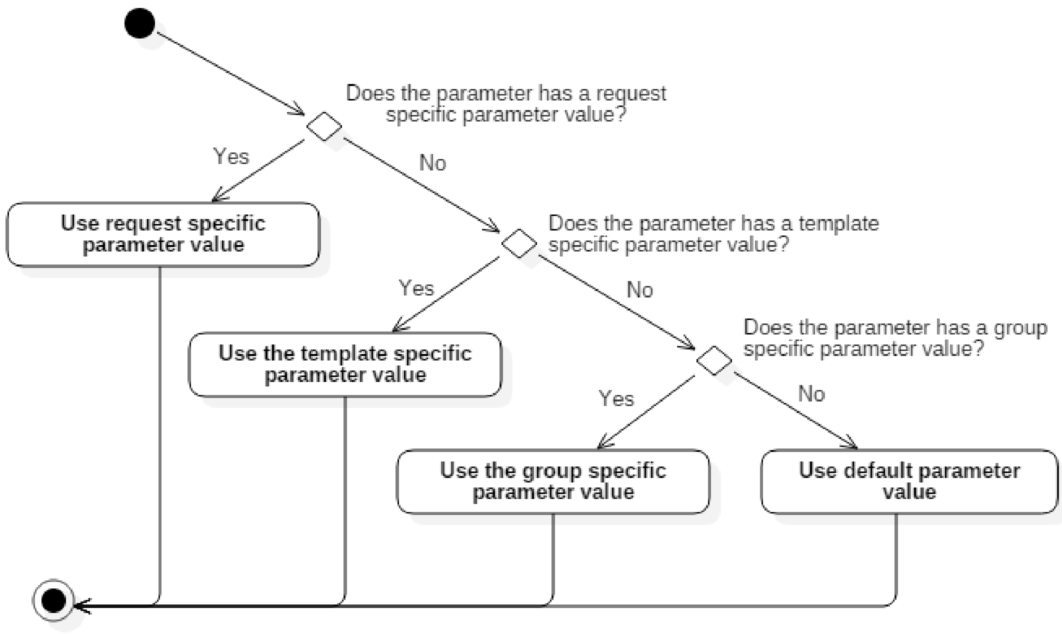

Figure 6.5 *Parameter value selection*

It would also had been possible to define own value for every templates but when default value definitions were used as much as possible it reduced workload notably. Parameter values for about half of templates were defined by using only default values and big part of rest templates were built by using parameter value groups.

RestRequests object were built from RestRequestTemplate objects by defining suitable parameter values to the template. These contains all needed information that is needed to build HTTP request for MFWS like a template for URL with necessary parameter types and a class that is used to define JSON for a HTTP body. In DAST Tool Teacher, HTTP requests were sent by using built-in HTTP libraries of .NET. Some part of MFWS implementation was also used as help for building HTTP requests. Especially was used struct classes for converting objects to JSON that were in request and response bodies of MFWS. The class name that should be used in the body is saved in the RestRequestTempate. This was the most important detail that influenced choice of programming language used in DAST Tool Teacher. Otherwise all class for JSON creation should be recreated from scratch.

OWASP ZAP had also a method in its API that could be used to send messages through OWASP but messages should be given in text format, which was not easiest way to construct messages generally in application code. Also, this would had able to be used only with OWASP ZAP. The current solution can be used with all vulnerability scanners, which works as intercepting proxy. Also with current solution it was easy to send requests also without proxy configured. Especially the learning phase was more convenient to execute without using proxy.

Simultaneously, when parameter values were configured for RestRequestTemplates, a test vault was built. This vault is restored to M-Files Server always before the learning phase is started so it is always in the same state at the beginning and all its values are predictable. It contains all M-Files objects and other settings that were used in the requests. The vault has minimal amount of objects and multiple request operated the same object because that way same object specific parameter values could be used in multiple requests. At first it was tried a procedure were the request can change these common objects. However, this was quickly abandoned because it requires that requests should be sent in a specific order. This was changed to a procedure were parameter values only points to an object template that were copied before the request is sent. Only copy of the common object was modified so the common object remained unchanged. Before the request was sent, the object specific parameter values were modified to point to this copied object. That way the requests could be sent in an arbitrary order.

There was lot of work in parameter value definition. It was started by creating one object to the test vault and its definitions were used as default values for every request. Then it was checked how many requests succeed. After that new parameter values were defined to default values. When it was encountered a situation, which couldn't been solved by defining default value or modifying the test vault it was started to create parameter value groups and templates were added to these groups. Only values, which were necessary were added to these groups. Otherwise default values were used. If parameter value was needed only in one template also template specific default values were defined.

Multiple methods in MFWS required specific preconditions before these could be sent successfully. For fulfilling these requirements, it was created RestRequestInstructions in this phase. These contains all information which was needed to execute requests again with the same steps and so that the same end condition was reached. One instruction contains following information:

- Relationship to a main RestRequest instance, which sending instructions is defined

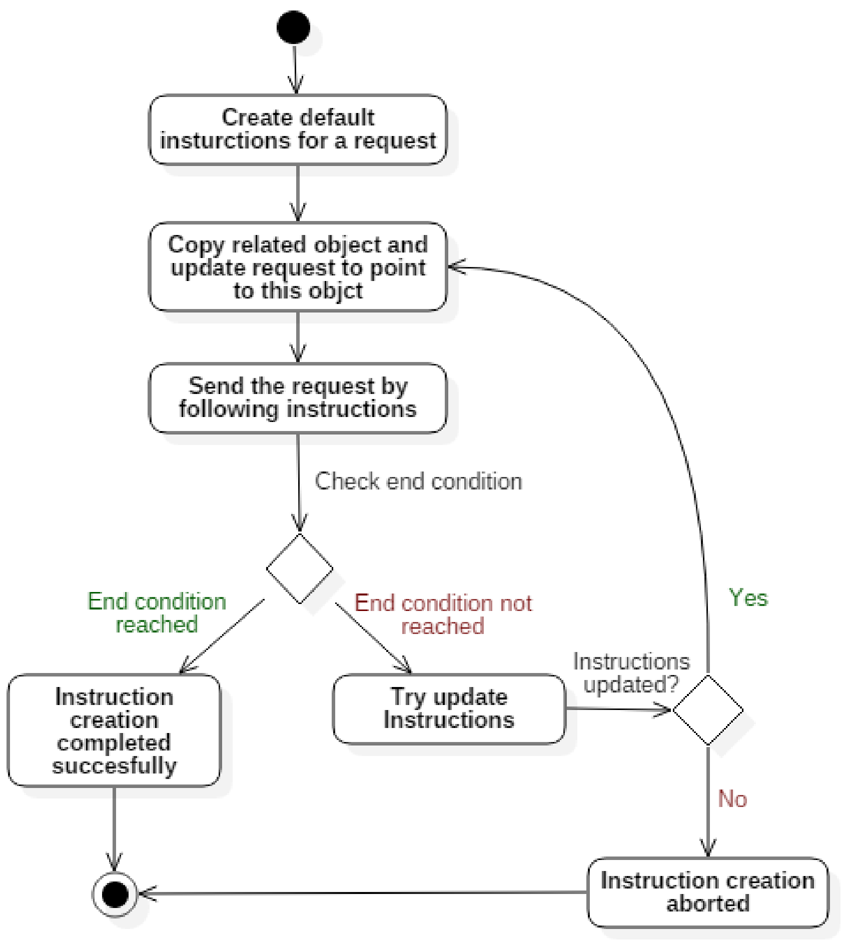

Figure 6.6 *The procedure that was used to construct RestRequestInstructions.*

in this RestRequestInstruction.

- The type of a user that should be used when executing the request. This is necessary because some methods for MFWS were intended for administrator operators. Possible user types at the moment are basic user and vault admin user.

- A list of presteps in an execution order. Presteps are defined in the list as identifier of the RestRequestTemplate that should be executed as a prestep. These templates are filled with parameter values related to the main request. That way the presteps are pointed to the same M-Files object as the main request.

- A list of poststeps in an execution order. The poststeps are defined the same way as presteps. These steps are sent after the actual request in the instruction is sent. Poststeps were not needed so much but when log out methods was tested it was needed to log back in after log out succeed.

Learning was performed by following method shown in figure 6.6. At first the request was sent with default instructions. without presteps or poststeps and it was sent by using a basic user without any specific privileges. Then a test object was created by finding the template object by using object related parameter values and copying it. New test object specific parameter values were updated to a test request. In the end the request was sent and response is analyzed. If the end condition was reached (Status code 200) instructions were already good enough and it is possible to continue to learn the instructions for a next request. Otherwise some details in the instructions were automatically identified from the error codes in responses. For example, if it is informed in the error that the object was needed to be checked out that was added as pre step in the instructions and whole instruction was tried again. This was continued until a end condition is reached or all ideas are used. For requests, which pre steps and post steps cannot be identified automatically it was added possibility to define these manually in the test vault configuration.

After the learning phase was executed and all property values and pre-steps were defined to the every RestRequestTemplate it was achieved instructions for executing every method in MFWS successfully. These RestRequestInstructions could be then used to introduce every method to a vulnerability as well as possible.

6.4.3 Teaching Phase

In teaching phase, all necessary information is introduced to the vulnerability scanner so that the attack phase can be started. This is done by following RestRequestInstructions learned previously. These are used to create suitable requests that generates desired end results. This also simulates better manual introduction and ensures that all complicated features are introduced as desired.

DAST-Tool Teacher contains two different options to execute the teaching phase that are selected by modifying environment configuration file. The first and easiest way is let DAST-Tool Teacher handle it (Figure 6.7). Messages are sent the same way as in the learning phase. DAST-Tools copies the template object pointed in the request, updated the object related parameters, sends pre-steps one by one by using a user configured in the instructions and after that sends the actual request that is going to be introduced. In the end, requests from post-steps are sent. This is suitable teaching method for every vulnerability scanners which work as intercepting proxy because it does not require any additional customizations to the scanner.

The requests are authenticated similarly as M-Files Web by using cookie based authentication. At the moment this is the only authentication method, which was implemented to DAST-Tool Teacher. In the future other authentication methods can be easily implement if these are seen as necessary. Cookie-based authentication was selected because it

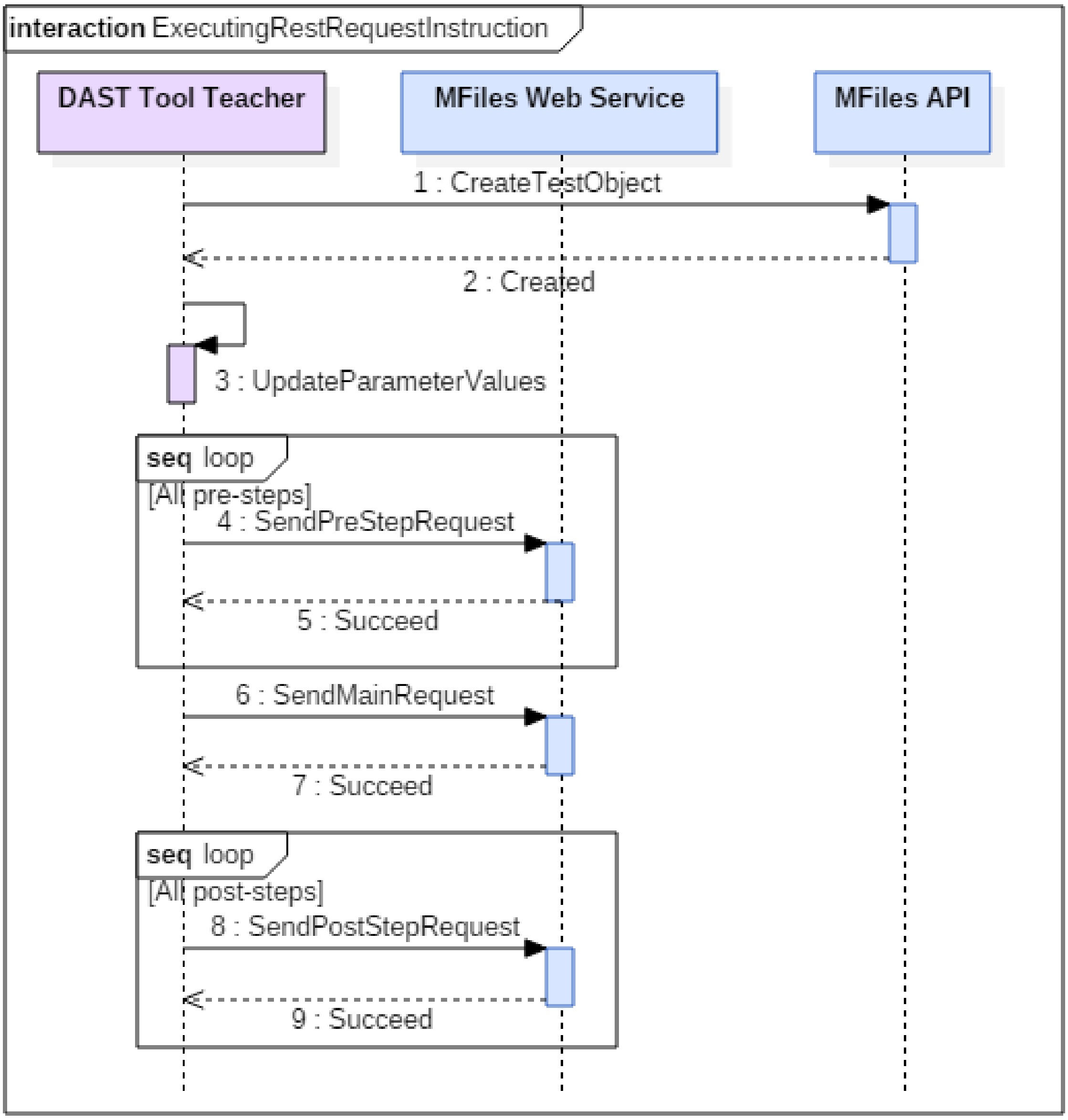

Figure 6.7 *The procedure that was used to execute teaching inside DAST Tool Teacher.*

is wanted to proof that is is possible to simulate similar communication as used in M-Files Web.

The second way to execute the teaching phase is let vulnerability scanner to follow RestRequestInstructions. Before teaching is started, all RestRequestInstructions are uploaded to the database that is visible for the vulnerability scanner. DAST-Tool Teacher does not follow instructions but sends only the main requests of the instructions with the identifier of the RestRequestInstruction in the request header. This same identifier was used as key in the database so the scanner can retrieve related instructions by using this identifier. This is more advanced option and it always requires that the scanner has custom extension that operates teaching phase by using the given instructions. Also, when the instructions are given to the scanner same information can be used also in another phases of scanning. For example, when active scanning is executed. In this master thesis this second teaching option is tested by creating the custom extension to OWASP ZAP.

DAST Tool Teacher has also other helping tasks for OWASP ZAP. Before the teaching phase is started a new session is created to OWASP ZAP by using .NET API of OWASP ZAP. That way all history and site map generated in learning phase are cleaned and the teaching phase can be executed without that garbage generated in learning phase cause disadvantages in the following attack phase. It is wanted that after teaching phase the site map contains all wanted items only once.

6.5 Custom OWASP ZAP Extension

The custom extension was created to OWASP ZAP so that it was able to test how preconditions of requests can be handled inside a vulnerability scanner. The extension is bound with the second teaching option of DAST Tool Teacher where the identifiers of RestRequestInstructions are given with request in HTTP request header. By using these identifiers, the extension is able to retrieve information how requests should be handled and operate so that desires end results are reached. In this extension this information was successfully used handling preconditions while introducing requests and it was proofed that preconditions handling can be possible also while active scan is executed. Support for postconditions was left for future development.

Reason for this extension was to improve the testing results and testing coverage when using vulnerability scanner. These are improved because if the scanner itself can handle preconditions in the attack phase the tested system is always in a better state to receive attack strings and trigger vulnerabilities that were otherwise possible trigger only with a manual penetration testing.

A big part of execution of the created ZAP extension is delegated to a vault application installed to the test vault because that way existing codes of DAST-Tool Teacher can be reused. For this reason, common parts of DAST-Tool Teacher was divided to own library and that library was used in both DAST-Tool Teacher and DAST-Tool Teacher Vault Application. This library contains all necessary codes, which are needed, to send requests by following RestRequestInstructions or almost everything except source code parsing.

The vault application was necessary to simplify the extension and reduce duplicate implementations that can cause unnecessary issues and workload. The extension part in OWASP ZAP should be written by using Java so M-Files API cannot be used. However significant part of teaching phase implemented in DAST-Tool Teacher uses M-Files API. Only interface, which could have been used in the extension to communicate with M-Files would have been MFWS that is not easiest solution for complex operations. For this reason, it was more reasonable to create a vault application where is implemented a necessary vault extension method that executes teaching procedures by using given instruction and uses same implementation as DAST Tool Teacher. This vault extension method are called by ZAP extension by using MFWS but now it is enough to implement only one method call to the extension.

Currently the vault application contains only one vault extension method *ExecuteInstructions()* but in the future is can contain also other vault extension methods. That implemented method takes a rest request instruction identifier and session tokens as input parameters and returns an url path and body content if these are needed to update to a request. The rest request instruction identifier is used to retrieve the related RestRequestInstruction object from the database. This object contains all necessary information to execute pre-steps similarly as in the teaching phase executed by DAST-Tool teacher. The vault extension method requires session tokens for cookie authentication so that same session can be used when pre-steps are executed. This is necessary especially for checkout because this is needed to do with the same user and use the same authentication session. Otherwise the object is not accessible.

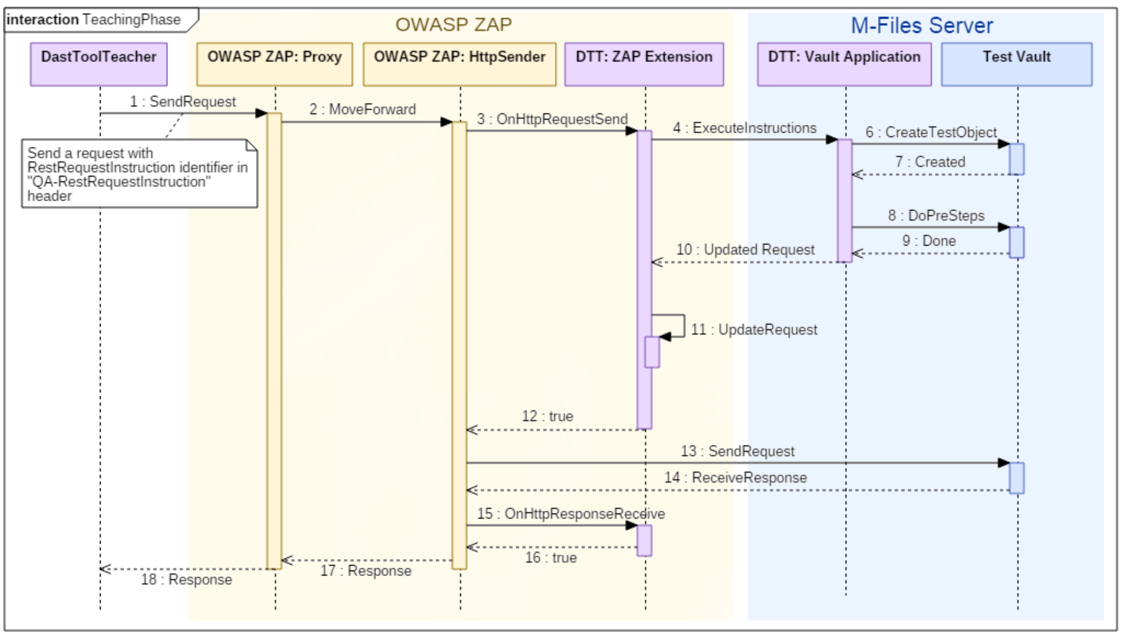

Figure 6.8 *In teaching phase the requests is sent with the instructions which OWASP ZAP then follows.*

Detailed sequence diagram how the teaching phase is executed inside OWASP ZAP is seen in the figure 6.8. When OWASP ZAP receives a request from DAST-Tool Teacher it goes through different components. The teaching phase operations was implemented as HttpSenderListener to event *OnHttpRequestSend*. This receives every request that was sent forward by OWASP ZAP. If the request contains *QA-RestRequestInstruction* header it is known that this request is from DAST-Tool Teacher and the teaching phase is not yet executed. The value of this header is received and it is sent to the vault application by sending an additional request that calls the vault extension method. It executes the teaching procedure by copying the object for testing from template object that is found from the instructions and executing all pre-steps if the instructions guide so. When the response from the vault extension method is arrived, it is known that all steps for correct preconditions are executed. If the request from DAST-Tool Teacher is needed to modify, the response from the vault extension method contains values for the URL path and the body. Otherwise these values in the response are empty. These modifications are needed for example if the request should point to a new object or a version of the object is increased because the object was check out. When preconditions are fine and the request from DAST-Tool teacher is updated the request can be sent to the test vault. This should now produce the wanted end results and wanted request and response to the site map that still contains information about the instructions in the same header field for repeating this sequence again. Current solution in the extension does not implement post conditions but if this is wanted to be supported in the future this can be implement to *OnHttpResponseReceive* event in *HttpSenderListener*.

With this extension, the preconditions of the request in the teaching phase were successfully handled inside OWASP ZAP. However this didn't gave much advantage comparing when preconditions were handled inside DAST-Tool Teacher because final conditions were almost identical. This solution cause little less unnecessary items to the site map because it does not contain request and responses, which are sent as pre-steps. This speed up a little the scanning because every request type found only once in the site map. Even if advantages were minimal this was a necessary step so that OWASP ZAP could handle pre-steps itself. Next it is explained how the same handling is implemented in active scan.

Teaching phase was implement in HTTPSenderListener but this was not suitable place when the same instruction procedure was wanted to use in the active scan. HttpSender-Listeners are executed after the active scan sends request so if the procedure had been executed in HttpSenderListener an attack that the active scan has made to the request would have been overwritten because the procedure overwrites almost every time the URL and the body of the request. For this reason, it was needed to find way to execute the procedure before the active scan has made its own changes. ScannerHook that is another place for custom implementation has event listeners that are closer to active scan but these are also triggered after the active scan has made its changes. Right place where the procedure was wanted to execute would be after the active scan copies a request from the site map but before the active scan has made an attack to the request. OWASP ZAP extensibility do not have yet possibility to add custom modifications here.

Because the wanted customization place wasn't available it was started to investigate the source code of OWASP ZAP and also tested how easy it would be to build OWASP ZAP from the source code with wanted modifications. There were good instructions how to build OWASP ZAP and it worked out easily. Modifications were wanted to implement as event handled that is possible to register in an extension so that modification does no change basic functionalities of OWASP ZAP only improves extensibility support.

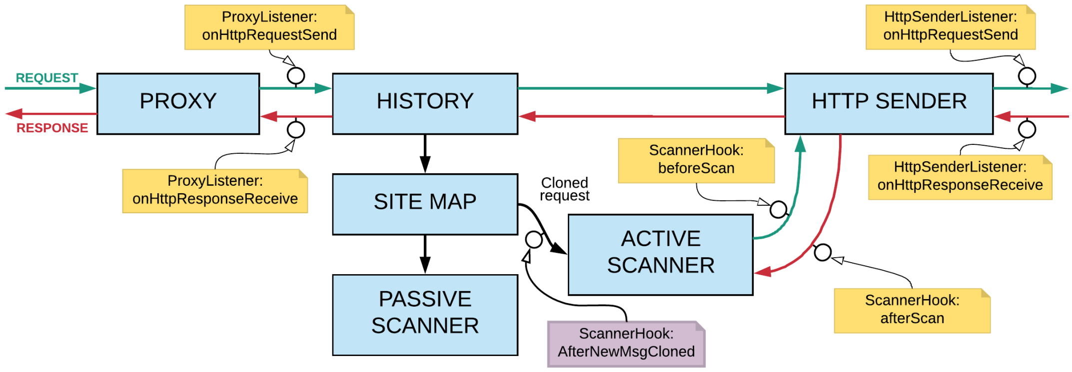

Figure 6.9 *Implemented AfterNewMsgCloned event listener to ScannerHook. It is executed before cloned request is given to Active Scanner.*

The custom event handler, which should add wanted extensibility support, was implemented to *ScannerHook* that already contains extensibility possibilities for active scan and was implemented to the same classes where the custom event handler had to be added. The name of the new event handled is *AfterNewMsgCloned()*. It is triggered while *newMsg()*-method is called by a scan just before the cloned request is given forward. The place of the method is seen also in figure 6.9. That way the preconditions can be executed and the request can be modified before it is given to the scan and any attacks created by the scan is newer overwritten. That new change brought a lot of unanswered questions because it wasn't clear how OWASP ZAP and all scans behaves if the request is changed in the middle of everything.

Parameters of the *AfterNewMsgCloned()* event handed method are similar as the parameters of other methods that were already exist in ScannerHook base class. Event handled allows to change the HTTP request that was going to be given forward and it has HTTP-Sender instance that can be used to for additional requests.

This new event handled was utilized in the custom OWASP ZAP extension where the teaching phase was already implemented. The implementation of *AfterNewMsgCloned()*method was quite simple and it resemble the similar implementation in HttpSenderListener. When this implementation was tested the pre-steps were successfully sent while active scan was executed but modifications contains also side effects. First it was notices that OWASP ZAP tests also sub paths of existing request in the site map. For example if the URL of a request is

`http://www.domain.com/REST/objects/0/123/3/properties`

it is also tested sub URLs

`http://www.domain.com/REST/objects/0/123/3`

`http://www.domain.com/REST/objects/0/123/`

`http://www.domain.com/REST/objects/0/`

`http://www.domain.com/REST/objects/`

and so on. Headers for these sub requests were copied from the original request so also the header *QA-RestRequestInstruction* was copied to all these sub requests. The custom extension blindly trust this header and thinks that all these request are the same request. It execute the same pre-steps and overwrite the URLs if necessary. This causes two problems. Sub requests is not tested well because often the URL is overwritten with URL that does not resemble the same request type. Also some request types are tested unnecessary multiple times.

This problem was fixed by creating a new API method to the REST API of OWASP ZAP. It reads through the whole site map and removes *QA-RestRequestInstruction* header from all requests, which were not directly sent by DAST Tool Teacher. This method can be called manually via OWASP ZAP web UI for REST API or any implemented API. For example, .NET API that is used in DAST Tool Teacher. The intention is that this would be called in DAST Tool Teacher after all requests is sent in the teaching phase.

Other problem that was encountered, when the solution was tested, was that when multiple requests were scanned simultaneously quite often the scan never completed. The reason was that the active scanner began to loop the same request types over and over again. The certain reason for never ending loop is not yet known but it is assumed that the reason is a new request in the site map that was added while active scanning is on-going because it has triggered a new vulnerability. This hasn't causes problems when the URL of the request hasn't been changed but when the custom extension changes the URL the request appears after the current request in the site map. The active scanner is encountered this issue again later and find the same vulnerability again. The triggering request is again added to the site map after the current request and so on. Because of this reliability problem the solution could not be tested very comprehensive. With single requests and small sets, the solutions seems to work as expected but when all request were tested at once it always ended up never ending loop.

6.6 Results

Retrieving test content from source code worked out perfectly. Every method of MFWS is now known and are in a form that can be used to build HTTP requests that calls these methods. Totally it was retrieved and introduced information of 393 methods. This amount was significantly bigger than amount achieved when the pre-attack phase was executed manually through M-Files Web. Amount of different request types were then somewhere between 175-200. When the improvement is calculated in percentage

```
393/175=2.246... → +125%

393/200=1.965 → +97%
```

it is got that totally testing coverage has been improved even 125 percent. Amounts in manual way differ because manual introduction can be executed in many ways and every execution is slightly different. Sometimes all features are remembered to crawl but sometimes the interface was introduced only partially because everything is not remembered. Also, multiple request types were multiple times in the site node that increased total testing time. By using the new solution, the site map is always complete and contains immediately also all new additions to MFWS even if there were not used anywhere.

By copying always the object that a method operates from template isolates the scanning of the method to one M-Files object that other methods can modify only by accident. This increase effectiveness of the scan because this reduces unexpected modifications to M-Files objects that may prevent that vulnerabilities are revealed. This method improves the scan effectiveness of 163 methods that operates a M-Files object directly.

The results of second component, the custom extension in OWASP ZAP, is a more difficult to diagnose because it isn't yet perfect. But in the future, it should improve the scanning of requests, which requires pre-steps. Totally MFWS has 39 methods, which requires pre-steps. After the custom extension is fixed it will improve the scanning of the these methods and the scanning will be improved by

```
39/393=0.099
```

10 percent.

6.7 Development ideas

In this master thesis, it was not focused to verify how well OWASP ZAP finds real vulnerabilities from web applications but it was ensured that there isn't any problems, which could prevent finding vulnerabilities, in a way how the attack surface was introduces or active scan was executed. DAST-Tool Teacher automates the reconnaissance phase of MFWS penetration testing but didn't automate rest phases so that MFWS can be scanned without any human intervention. It has been wanted that some day the penetration testing would be automated but there is still very long way to a situation where the whole penetration testing of MFWS is fully automated.

At first, before any automation could be though, it should be ensured that the scanner really finds vulnerabilities and does not give too much false positive alarms. This requires multiple steps. All problems exist in the custom extension while active scan is ongoing should be fixed. The existing scans in OWASP should be verify for that they really find vulnerabilities or own scans should be created that fits better to the structure or MFWS and tests application specific vulnerabilities. Testing time could be shortened by disabling unnecessary scans. For example, if it is known that only MSSQL and Firebird can be used as database for M-Files vault is not necessary to test attack vectors, focus for attacking MySQL server. It is also pointless to test XSS vulnerabilities by using OWASP ZAP or using some other DAST tool that scans requests one by one because the responses of MFWS always are sent without escaping because actual escaping is left for a web page, which shows the content. These vulnerabilities should be covered by using some other testing tool. It would be also important to consider other vulnerability scanners and review could one of these to function better in our case and plan. OWASP ZAP was selected only because it was good tool for testing different concepts. All requirements, which this new tool should have, are now better known so the selection will be easier than before this thesis. In any case any DAST tool cannot find every vulnerability from the application so manual penetration testing is always needed.

The penetration testing of MFWS can be automated by using OWASP ZAP but also other tools should be considered. However, OWASP ZAP has all required commands in its REST API. The commands can be called by using DAST-Tool Teacher or it can be created another tool that orchestrates every phase. The reconnaissance phase was already implemented in DAST-Tool Teacher so this phase is already done. The phases, which still require automation, is the attack phase and the post-attack phase. In the attack phase, the active scanner should be programmatically started and the execution should be followed so that its progress can be forwarded and it is known when it is completed. In the post attack phase all found issues should be gone through and all clearly false positive alarms should be discarded before results is shown. Additional of these, procedure that builds testing environment should be created. This install newest M-Files version, DAST-Tool Teacher and other necessary applications to a virtual machine, downloads related source codes of MFWS to a place where DAST-Tool Teacher can retrieve these and starts the testing procedure.

7. CONCLUSION

The goal of this master thesis was to study basic functionalities of standalone vulnerability scanners for web applications and identify problems and technical challenges that occurs when M-Files Web Service were tested by using these scanners. Problems were found from the way how the attack surface was currently introduced to the scanner and from common way how the active scan is executed in a vulnerability scanner by testing requests one by one without considering preconditions.

Vulnerability scanners are used as help in manual penetration testing but additional of this many vulnerability scanners can be used also as standalone mode that automatically test several vulnerabilities without that testers needed to now anything about security testing. It was learned that standalone vulnerability scanners are easy to use but have limitations that should be keep in mind when the scan is performed. Standalone vulnerability scanners can find only vulnerabilities, which has detectable signature and cannot found vulnerabilities related to logic flaws and vulnerabilities, which require that the functionality of web application is understood. Request are tested one by one that means that it is not revealed vulnerabilities, which requires specific steps and preconditions. If the security testers are not aware about limitations of standalone vulnerability scanner it causes false conclusions about the security level of the system because the system can still have vulnerabilities that aren't found via vulnerability scanner. Experienced manual penetration tester can usually find these vulnerabilities but if the system is big it is not reasonable or even possible to manually penetration test the system totally. For these situations vulnerability scanners are essential.

When current testing habits were analyzed, it was learned that current way to introduce the public M-Files Web Service HTTP API by manually crawling through M-Files Web didn't introduce whole M-Files Web Service. Main reason was that M-Files Web Service was used also in M-Files mobile applications and in third-party applications and for these reason it contains methods that were not used in M-Files Web. These holes in security testing was wanted to fill and to create a final solution that can be used also when the penetration testing will be automated. It was decided to create external application, that parses needed information directly from the source code where MFWS was implemented and send all needed requests through the vulnerability scanner with valid values and necessary pre-steps. That way introduced the attack surface will be always up-to-date and every new method to M-Files Web Service will be tested immediately.

Another problems that was identified was due to the common way how vulnerability scanners execute the active scan. When a scanner scans requests by sending each request multiple times it does take care that the tested system is every time in the best state to reveal vulnerabilities. When requests does not succeed, vulnerabilities were sought only from the part that validates parameters and object state and only small part of source code was covered. When penetration testing is executed manually penetration testers try different scenarios and sequences to reveal vulnerabilities. This same procedure was wanted to implement when penetration testing was executed automatically by using vulnerability scanner. This meant that the same pre-steps that were identified in the external application was needed to executed inside the vulnerability scanner when active scanner was executed. This method was tested by creating POC extension to OWASP ZAP. Even if OWASP ZAP has wide extensibility possibilities it was realized that it didn't has the needed event listener but because OWASP ZAP was open source application this custom event listener was able to implement and solution was able to test. It was proofed that it is possible to executed pre-steps before every attack executed by active scanner but custom extension causes side effect, which were not yet solved. The solution works with small sets but when wider attack surface was tested the scan ends to loop that never ends. This problem is needed to solve before the solution can be used in the production.

Information security and security testing is important for software companies like M-Files even if their own confidential information is not in danger because bad reputation in security reduces market value and future sales or causes fines and lawsuits from poorly implemented security practices. For M-Files the security is especially important because they create a software that is intended for storing all existing information of a corporation especially most important and confidential data. Their customers rely on that the security of M-Files is in good shape, but even one bad news causes that current and potential customers lose their confidence in M-Files. Security of the software can be improved by raising awareness about security among developers and invest a good security testing team or purchase the security testing from a third party company that is focused and are experts in security testing. This master thesis has improved security testing but there is still lot of work. In any case security testing cannot be forgotten. It should be always as important as ordinary testing.

BIBLIOGRAPHY

[1] J. Andress, *The Basics of Information Security: Understanding the Fundamentals of InfoSec in Theory and Practice, Second Edition*. Syngress Publishing, 2014.

[2] S. Bavisi, *Managing Information Security (Second Edition)*. Elsevier Inc, 2004, ch. Penetration Testing.

[3] P. Engebretson, *The Basics of Hacking and Penetration Testing: Ethical Hacking and Penetration Testing Made Easy, Second Edition*. Syngress Publishing, 2013.

[4] R. Fielding and J. Reschke. (2014) Hypertext transfer protocol (http/1.1): Semantics and content. [Online]. Available: https://tools.ietf.org/html/rfc7231

[5] R. T. Fielding. (2000) Architectural styles and the design of network-based software architectures. [Online]. Available: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[6] T. Fredrich, *RESTful Service Best Practices - Recommendations for Creating Web Services*, Pearson eCollege, 2013. [Online]. Available: https://github.com/tfredrich/RestApiTutorial.com/raw/master/media/RESTful%20Best%20Practices-v1_2.pdf

[7] A. Kakareka, *Managing Information Security (Second Edition)*. Elsevier Inc, 2004, ch. What is Vulnerability Assessment?

[8] J. Kanjilal, *ASP.NET Web API : Build RESTful web applications and services on the .NET framework*. Packt Publishing, 2013.

[9] M-Files Corporation. M-Files API: Overview. Accessed: 22.9.2017. [Online]. Available: https://www.m-files.com/api/documentation/latest/index.html

[10] M-Files Corporation. M-Files UI Extensibility Framework. Accessed: 22.9.2017. [Online]. Available: https://www.m-files.com/UI_Extensibility_Framework/

[11] M-Files Corporation. M-Files User Guide. Accessed: 30.9.2017. [Online]. Available: https://www.m-files.com/user-guide/latest/eng/

[12] M-Files Corporation. M-Files Web Service: Documentation. Accessed: 14.9.2017. [Online]. Available: https://www.m-files.com/mfws/

[13] M-Files Corporation. M-Files website. Accessed: 14.2.2017. [Online]. Available: https://www.m-files.com/en

[14] Microsoft. (2005) The stride threat model. [Online]. Available: https://msdn.microsoft.com/en-us/library/ee823878(v=cs.20).aspx

[15] MITRE. CWE - Common Weakness Enumeration. Accessed: 1.2.2018. [Online]. Available: https://cwe.mitre.org/

[16] OWASP. (2014) ZAP Blog: Hacking ZAP #3 - Passive scan rules. [Online]. Available: https://zaproxy.blogspot.fi/2014/04/hacking-zap-3-passive-scan-rules.html

[17] OWASP. (2014) ZAP Blog: Hacking ZAP #4 - Active scan rules. [Online]. Available: https://zaproxy.blogspot.fi/2014/04/hacking-zap-4-active-scan-rules.html

[18] OWASP. (2017) About the open web application security project. [Online]. Available: https://www.owasp.org/index.php/About_The_Open_Web_Application_Security_Project

[19] OWASP. (2017) Category:OWASP Project. [Online]. Available: https://www.owasp.org/index.php/Category:OWASP_Project

[20] OWASP, "OWASP Top 10 - 2017, The Ten Most Critical Web Application Security Risks," OWASP, Tech. Rep., 2017. [Online]. Available: https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf

[21] OWASP. (2017) OWASP ZAP User Guide. [Online]. Available: https://github.com/zaproxy/zap-core-help/wiki

[22] OWASP. (2017) OWASP Zed Attack Proxy Project. [Online]. Available: https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project

[23] OWASP. (2017) The Open Web Application Security Project (OWASP) Web Page. [Online]. Available: https://www.owasp.org

[24] Paros. (2017) Github page of Paros Proxy. [Online]. Available: https://sourceforge.net/projects/paros/files/Paros/

[25] E. H. Spafford. (2009) Quotable spaf, Gene Spafford's personal pages. Accessed: 22.2.2017. [Online]. Available: http://spaf.cerias.purdue.edu/quotes.html

[26] Y. Stefinko, A. Piskozub, and R. Banakh, "Manual and automated penetration testing. benefits and drawbacks. modern tendency," in *2016 13th International Conference on Modern Problems of Radio Engineering, Telecommunications and Computer Science (TCSET)*, Feb 2016.

[27] D. Stuttard and M. Pinto, *The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws, Second Edition*. John Wiley Sons, Inc, 2011.

[28] Techopedia. What is cyberattack? Accessed: 31.10.2017. [Online]. Available: https://www.techopedia.com/definition/24748/cyberattack

[29] D. Zumerle and A. Tirosh, "Magic quadrant for application security testing," Gartner, Report G00290926, 2017.