Hochschule für Technik
und Wirtschaft Berlin

University of Applied Sciences

Developing a Software for Automated Module-based Configuration of Virtual Machines for Penetration Testing

Abschlussarbeit

zur Erlangung des akademischen Grades:

Bachelor of Science (B.Sc.)

an der

Hochschule für Technik und Wirtschaft (HTW) Berlin
Fachbereich 4: Informatik, Kommunikation und Wirtschaft
Studiengang *Angewandte Informatik*

1. Gutachter: Prof. Dr.-Ing. Piotr Wojciech Dabrowski
2. Gutachter: Dr. rer. nat. Tom Ritter

Eingereicht von Nader Alhalabi [561121]

Datum

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel verfasst habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Berlin, den

Nader Alhalabi

Danksagung

Ich bedanke mich bei allen Personen die mich während meines Studiums und besonders bei der Erstellung dieser Arbeit unterstützt haben. Ich bedanke mich bei meinen Betreuern Herrn Prof. Dr. Piotr Wojciech Dabrowski und Herrn Dr. Tom Ritter für die Unterstützung während der Arbeit und die Bereitstellung von technischen Ressourcen.

Zusammenfassung

Die Beherrschung von Penetrationstests kann eine schwierige Aufgabe sein, da so viele Informationen aufgenommen werden müssen und so viele Schwachstellen zu lernen und zu analysieren sind. Penetrationstester müssen dieses Wissen auch in der Praxis anwenden, und virtuelle Maschinen können eine der besten Umgebungen sein, um die Fähigkeiten und das Verständnis eines Penetrationstesters anzuwenden, aber das Einrichten einer VM, um viele Schwachstellen zu üben, kann eine schwierige und zeitraubende Aufgabe sein.
Diese Arbeit zielt darauf ab, die Hindernisse bei der Einrichtung einer VM zu beseitigen, indem eine Software implementiert wird, die diese Aufgabe automatisiert. Diese Software sollte in der Lage sein, eine Liste von benutzerdefinierten Modulen zu installieren, die frei konfigurierbar sind, um eine bestimmte Schwachstelle auf der VM einzurichten, wodurch diese VM zu einer einsatzbereiten Sandbox für Penetrationstests und Hacking wird.

Abstract

Penetration Testing can be a difficult skill to master, with so much information to absorb and so many vulnerabilities to learn and analyze. Penetration testers also need to practice this knowledge, and virtual machines can be one of the best environments to apply the skills and understanding of a penetration tester, but setting up a VM to exercise many vulnerabilities can be a tough and time-consuming job.
This work aims to remove the obstacles of setting up a VM, by implementing a software that automates this task, this software should be able to install a list of user-defined modules, which are freely configurable for the purpose of setting up a specific vulnerability on the VM, making this VM a ready-to-go sandbox for penetration testing and hacking.

Contents

1 Introduction **1**
1.1 Motivation 1
1.2 Objective 1
1.3 Approach and Structure 1

2 Fundamentals **3**
2.1 Virtual Machines 3
2.2 Penetration Testing 4
2.3 Python 4
2.4 YAML 5

3 Conception and Design **6**
3.1 Modules 6
3.2 Process 7
3.3 Validation 7
3.4 Template Engine 8

Bibliography **9**

Chapter 1

Introduction

This introductory chapter provides a summary of the motivation, the desired aim, and the structure of this work.

1.1 Motivation

Getting into the world of penetrating testing is a big challenge, especially when it comes to applying what was learned theoretically to an actual machine.
New learners should not apply their knowledge to real targets, and setting up a testing environment can be a daunting and time-consuming task, but a program that automates this process can lift this obstacle, and with the power of Python and Bash scripts, configuring a virtual machine for pen testing can be turned into a straightforward and effortless process.

1.2 Objective

This work aims to create a simpler way to set up a virtual machine for penetration testing, in addition, it is intended to enable the user to pass a particular set of configurations through metadata.
This could be achieved by implementing a python script that installs a user-defined list of modules to a specific virtual machine.

1.3 Approach and Structure

This thesis can be divided into five main chapters. In the beginning, the challenges that led and inspired this work are introduced and illustrated. Chapter 2 gives an overview of the basics to understand the methods and techniques of the work, Then, in Chapter 3, the conception and design of the intended software are established.

Afterward, a detailed explanation of the implementation and the structure of the designed program is provided in Chapter 4. Lastly, chapter 5 gives a brief rundown on the tests and the evaluation of the development process, this gets concluded with a summary and potential future development.

Chapter 2

Fundamentals

In this chapter, the technical basics of this thesis are presented, initially, an introduction to virtual machines and penetration testing is given, followed by some explanation on python and YAML files.

2.1 Virtual Machines

A virtual machine (VM) is a virtual environment that works like a computer system with its own resources, like CPU, memory, and storage, created on an actual physical hardware system. With the help of a software called "hypervisor", the machine's resources get separated from the hardware so they can be provided in the right manner to be used by the VM.
The physical machines, ones equipped with a hypervisor, are called host machines (host), while the many VMs that utilize its resources are guest machines (guest). The hypervisor treats the host's resources as a pool of resources that can be simply distributed and relocated between existing guests as well as new virtual machines. VMs are also isolated from the rest of the system, and multiple of them can co-exist on a single physical piece of hardware. They can be dynamically relocated between host servers depending on demand.
One of the advantages of virtual machines is allowing numerous operating systems to run on a single computer at the same time, and each operating system runs as if it's running on the host hardware, thus the user experience within the VM is almost identical to that of a real-time operating system experience running on a physical machine[5].

This allows penetration testers to apply their knowledge on disposable sandboxes that are as real as host systems, with no worry of potentially damaging hardware or harming people/organizations.

2.2 Penetration Testing

A penetration test, or a pen test, is a simulated cyberattack against a computer system for the purpose of checking for security vulnerabilities. Pen testing can expose various types of security weaknesses in an application system (e.g. APIs and servers), it can also identify unsanitized inputs that are vulnerable to code injection attacks.
The pen testing process can be broken down into five stages:

1. **Planning and reconnaissance:** Defining the goals and scope of a test, including the testing methods to be used and the systems to be addressed.
2. **Scanning:** Understanding the target application and how will it respond to several intrusion attempts. This can be done by inspecting an application's code to estimate the way it behaves while running.
3. **Gaining Access:** This stage consists of exploitation techniques that expose web application vulnerability, such as cross-site scripting and SQL injection. Attackers then use these techniques to escalate privileges and steal data to understand the scope of damage they can create.
4. **Maintaining access:** This aims to see if an exploit can be used to gain a persistent presence in an exploited system, long enough for a bad actor to gain in-depth access.
5. **Analysis:** Results of the pen test are then compiled into a report containing the vulnerabilities that were exploited, sensitive data that was accessed, and the amount of time the pen tester was able to remain in the system undetected[6].

2.3 Python

Python is a high-level programming language that has a variety of object-oriented features. Its flexible and high-level structure makes it very attractive for developing rapid application development. Its simple and easy-to-learn syntax helps minimize program maintenance.
The rapid edit-test-debug cycle of Python makes it very easy to debug programs. When an error occurs, the interpreter prints a stacktrace, which tells the program which of the available exceptions has been encountered.
The source level debugger simplifies the debugging process by allowing the program to inspect and evaluate the code at a time[7].
Python was the language of choice for this software, for the fact that it has most of the packages that are necessary for this work, in addition to its automation capabilities and ease of implementation.
Python 3.8 is the language and version of choice for the implementation of this work.

2.4 YAML

YAML is a data serialization language that is often used to create configuration files, It stands for yet another markup language and evolved into ain't markup language, which highlights that YAML is for data and not for documents. It is also easy to understand and is human-readable.
YAML is a superset of JSON, so JSON files are valid in YAML, but it uses Python-style indentation to indicate nesting, as there are no usual format symbols, such as braces, square brackets, YAML files use a .yml or .yaml extension.
The structure of a YAML file is a map or a list. Mappings allow you to group key-value pairs into distinct values. Order is not relevant, and each key must be unique. A map needs to be resolved before it can be closed. A new map can then be created by either creating an adjacent map or increasing the indentation level.
A list sequence is a type of object that contains values in an order. It can contain multiple items, and starts with a dash (-) and a space, while indentation separates it from the parent. Naturally, YAML also contains scalars that can be used as values such as strings, integers, or booleans[8].

Example of YAML syntax:

```
---
name: max
enrolled: True
languages:
  - english
  - german
marks:
  - programming: failed
  - math: 1.0
```

Code snippet 2.1: YAML example

YAML is the dominant file type for writing configuration files and metadata, it has the benefit of easier human readability, which helps module creators to step into writing metadata rapidly and comfortably.

Chapter 3

Conception and Design

In this chapter, the main parts of this program will be discussed, and the design will be explained.
The main idea of this program is to take a specific list of modules, and after some operations on them to ensure their validity, it should start to install the modules on VirtualBox[4]. Installing the modules and transferring files is done through SSH, which means realistically, that SSH module must always be present or rather be installed first for other modules to function.

3.1 Modules

A module is a user-defined package, that does one or more sets of tasks and it has 3 main components:

1. **Metadata**
2. **Main script**
3. **Resources**

The Metadata file is the core of the module, it consists of a YAML file that defines all of the features of the corresponding module, and especially the provided and/or the needed dependencies. The contents and rules of a metadata file will be covered later in the implementation chapter.

The main script is a Bash script, which acts as a start point to the module, it usually has the initial SSH connecting and resources copying commands.

Resources are everything else that the module needs to perform its intended task, for example, it can be website static files, SQL dumps, or other helper scripts.

3.2 Process

There are two considered ways to achieve the process to automate the install of modules: First by implementing a bash script that serves as an entry point to all modules, and then sending commands with a python script to it.
The second way is by controlling the modules from the main python script itself, this means that this python script is responsible for orchestrating the execution of bash scripts inside the modules.
Before the main python script executes the bash scripts in the modules, there need to be some necessary operations done.
First of all, metadata must be validated to ensure proper parsing of the YAML file, and afterward, the configurations available in the metadata must be parsed and then replaced with placeholders in the module's scripts.
The intended designed process will then look like the following figure:
TODO images

3.3 Validation

To ensure the success of the installation process, the metadata of the modules needs to be validated, a set of rules were declared to help creators of modules write a valid metadata file, this set of rules are written in a form of schema, and with the help of python package "cerberus"[3], this schema is validated against the metadata YAML file. In case of unsuccessful validation, the installing process will not begin.
These designed rules are explained in the following pseudo-code:

```
name:
provides:
    tech:  # list
      - entry:  # map
          name:  # string
          version:  # string
          config:  # list
    tech-config:
      - entry:
          name:
          version:
          config:
needs:
    tech:
      - entry:
          name:
          version:
          config:
```

Code snippet 3.1: YAML Schema

Each module has a “name” entity and provides one or more sets of dependencies and configurations, under “tech” are the dependencies that this module provides, and under “tech-config” are the configurations this module provides without providing the corresponding dependency for it.
Every “tech” entry has the name of dependency, the version of it, and all configurations it supplies. Same structure for “tech-config”.
For every configuration, the “name”, “file”, and “value” are listed.

1. **“name”** is the name of the placeholder that the parser will be replacing.
2. **“file”** is the name of the file, in which the placeholder should be replaced
3. **“value”** is a list of values that will be replacing the placeholder

```
config:  # list
    - name:  # string
      file:  # string
      value:  # list
```

Code snippet 3.2: YAML Schema

It could be noticed, that the “needs” entity does not have a “tech-config” entity of its own, and that’s for the reason that a module can not need a certain configuration without needing its dependency as well, thus deprecating the use of “tech-config” for it.

3.4 Template Engine

The metadata defines what dependencies the module has, and what configuration it provides and/or needs, and parsing the metadata file must be done in an organized way.

When a module has a configuration, it consists of a placeholder in the main bash script or any other scripts in the resources folder, these placeholders are the locations where the values from the metadata should be replaced. This is where a template engine is used.

“A template engine enables you to use static template files in your application. At runtime, the template engine replaces variables in a template file with actual values, and transforms the template into an HTML file sent to the client.”[2]

This can be done by using an internal package of python called “string”, this package has a helpful function, namely “Validation”[1].

Bibliography

[1] *String - common string operations¶*. URL: https://docs.python.org/3/library/string.html#template-strings.

[2] *Using template engines with Express*. URL: https://expressjs.com/en/guide/using-template-engines.html.

[3] *Welcome to cerberus¶*. URL: https://docs.python-cerberus.org/.

[4] *Welcome to VirtualBox.org!* URL: https://www.virtualbox.org/.

[5] *What is a virtual machine (VM)?* URL: https://www.redhat.com/en/topics/virtualization/what-is-a-virtual-machine.

[6] *What is Penetration Testing: Step-by-step process and Methods: Imperva*. Dec. 2019. URL: https://www.imperva.com/learn/application-security/penetration-testing.

[7] *What is python? Executive summary*. URL: https://www.python.org/doc/essays/blurb/.

[8] *What is YAML?* URL: https://www.redhat.com/en/topics/automation/what-is-yaml.