Vulsploit: A Module for Semi-automatic Exploitation of Vulnerabilities

Arcangelo Castiglione, Francesco Palmieri, Mariangela Petraglia, Raffaele Pizzolante

Vulsploit: a module for semi-automatic exploitation of vulnerabilities

Arcangelo Castiglione[1][0000−0002−7991−2410], Francesco Palmieri[1][0000−0001−7100−9748], Mariangela Petraglia[2], and Raffaele Pizzolante[1][0000−0001−5722−1179]

[1] Dipartimento di Informatica, Università degli Studi di Salerno, Fisciano, Salerno, 84084, Italy
{arcastiglione,fpalmieri,rpizzolante}@unisa.it
[2] Università degli Studi di Salerno, Fisciano, Salerno, 84084, Italy
m.petraglia9@studenti.unisa.it

Abstract. Penetration testing (PT) is nowadays one of the most common and used activities to evaluate a given asset's security status. Penetration testing aims to secure networks and highlights the security issues of such networks. More precisely, PT, which is used for proactive defense and information systems protection, is a structured process, made up of various phases that typically needs to be carried out within a limited period.
In this work, we first define a modular semi-automatic approach, which allows us to collect and integrate data from various exploit repositories. These data are then used to provide the penetration tester (i.e., the *pentester*) with information on the best available tools (i.e., *exploits*) to conduct the exploitation phase effectively. Also, the proposed approach has been implemented through a proof of concept based on the *Nmap Scripting Engine (NSE)*, which integrates the features provided by the Nmap *Vulscan* vulnerability scanner, and allows, for each vulnerability detected, to find the most suitable exploits for this vulnerability.
We remark that the proposed approach is not focused on the vulnerability mapping phase, which is carried out through *Vulscan*. Instead, it is focused on the automatic finding of the exploits that can be used to take advantage of the results achieved by such a phase.

Keywords: Penetration testing · Automation of security testing · Security assessment · Security monitoring · Nmap · Nmap Scripting Engine (NSE).

1 Introduction

The increasing diffusion of hardware and software technologies have significatively extended the attack surface available to malicious users since such technologies are not perfect and can be affected by bugs, vulnerabilities, and other security issues. An attack could damage an organization, impacting the economic, social, and image aspects of such an organization. Attacks are typically

carried out by finding a weakness in the systems and then exploiting it. Thus, this is the reason why this weakness is called *vulnerability*.

Nowadays, information is more vulnerable than ever. Every technological advance raises new security threat that requires new security solutions. For example, *Internet of Things (IoT)* objects offer new services and pose new security threats. The IoT paradigm application to smart communities, smart cities, and smart homes, connecting objects to the Internet, brings to light new security challenges [11, 12, 20, 4]. These challenges make smart communities/cities/homes extremely vulnerable to several kinds of attacks [2]. Again, cloud and *mobile cloud computing (MCC)* enables mobile devices to exploit seamless cloud services via offloading, offering several advantages but increased security concerns [7, 8, 1]. More precisely, one of the most critical issues for the security assessment of cloud-based environments is the lack of control over the involved resources. Besides, this kind of assessment requires knowledge of the possible security tests to carry out and the hacking tools used for the analysis. To address this issue, in [8, 7] the authors propose a methodology that allows them to efficiently carry out a security assessment of cloud-based applications, by automating the set-up and execution of penetration tests. This methodology is based on the knowledge of the application architecture and security-related data collected from multiple sources. However, unlike the method proposed by us, the authors' methodology focuses more on coarse-grained architectural aspects, rather than on host-based scans. Besides, security threats are increasing in mobile payment environments, and in particular, in *mobile banking applications (MBAs)* since such applications store, transmit, and access sensitive and confidential information [6]. Moreover, many everyday life activities are based on interconnected electronic devices. Most of such devices are based on *Third-Party Intellectual Property (3PIP)* cores that could not be trustworthy. Therefore, if one of these 3PIP cores is vulnerable, the security of the device could be affected [14]. Finally, the rapid growth of *Artificial Intelligence (AI)* has made this even more challenging since machine learning algorithms are now used to attack such systems. On the other hand, the current defense mechanisms protect such systems by using traditional security facilities [10]. In detail, the primary use of AI in the penetration testing field concerns using reinforcement learning (RL) techniques [10, 16, 15, 25, 29] to perform regular and systematic security testing, saving human resources. Using RL techniques, penetration testing systems can learn and reproduce average and complex penetration testing activities. We point out that the system proposed by us does not use intelligent techniques, even if, as future developments, we aim to integrate these techniques in our proposal.

Penetration testing (PT) is an activity carried out to assess the security posture of an organization by safely exposing its vulnerabilities [9]. PT also helps evaluate the efficiency of the defense methodologies, policies, and tools used by an IT organization (or IT *asset*). Penetration testing is conducted regularly to identify risks and manage them to achieve higher security standards. Nowadays, penetration testing is one of the most common and used activities to assess the health of a given IT asset in terms of security. This activity deals with de-

tecting the vulnerabilities of a given asset in a completely legal and structured way, trying to exploit them by mimicking an attacker (i.e., a *black hat hacker*). The attacker could be driven by various malicious objectives, such as compromising or disabling the system, stealing data from it, etc [13]. The penetration tester (*pentester*) takes care of detecting potential vulnerabilities existing in the system, trying to fix them. The main aim of PT is to provide the most appropriate solutions to fix or mitigate the consequences deriving from the exploitation of vulnerabilities found, thus increasing the security posture of the organization. Furthermore, due to the recent regulations introduced by the *General Data Protection Regulation (GDPR),* [28], and the *Cybersecurity Act* [21], penetration testing activity finds an ever-increasing application in the context of the so-called cyber-physical systems and, in particular, in cyber-physical critical infrastructures [2].

We remark that PT is a structured process, made up of various phases, typically conducted over a limited time [24]. The success of this process depends mainly on two factors: the skills of the person who conducts it, typically called *penetration tester* or *pentester* for short, and the information available to him. On the other hand, one of the main problems with this process is that the data needed to conduct it often lies in distinct and heterogeneous sources. Therefore, these data must first be collected, then integrated, and finally used appropriately. This problem assumes great importance, especially in the *vulnerability mapping* and *exploitation* phases of the penetration testing process. Through these phases, the pentester, given a vulnerability, tries to find the most appropriate tools to exploit it [26], typically called *exploits*.

However, PT is a very time- and money-consuming activity that needs specialized security skills and tools. More precisely, penetration testing is a typically human-driven procedure that requires an in-depth knowledge of the possible attack methodologies and the hacking tools available to carry out the security assessment. Therefore, automated tools to assist the pentester in his activity are increasingly needed, and often become crucial to the success of the PT activity [3].

It is essential to point out that our approach is not focused on the vulnerability mapping phase, which is carried out through *Vulscan*. However, it is focused on the automatic finding of the exploits that can be used to take advantage of the results achieved by such a phase.

1.1 Contribution

This paper defines a modular semi-automatic approach, which allows us to collect and integrate data from various exploit repositories. These data will then be used to provide the pentester with indications on the best available tools (i.e., exploits) to conduct the exploitation phase effectively. We stress that since potentially a broad set of automatic tools can support this phase, it could be challenging to choose the most suitable one. Furthermore, the architecture of the proposed approach has been implemented and integrated into a module for the *Nmap* vulnerability scanner. In detail, this architecture has been realized

through an NSE (Nmap Script Engine) script [23], which integrates the features provided by the *Vulscan* vulnerability scanner. This script provides the pentester with the most suitable tools to exploit these vulnerabilities for each detected vulnerability. The experimental results obtained from the evaluation of our proposal have shown its effectiveness. As a future development, we intend to parallelize the proposed approach to improve performance in terms of detection times and choose the most appropriate exploits. Again, we intend to explore the potential of artificial intelligence to enhance penetration testing and vulnerability identification in complex network scenarios.

1.2 Organization

The paper is organized as follows. In Section 2, we present the proposed semi-automated approach and highlight its main characteristics and the motivations behind it. In Section 3, we introduce the Vulsploit module, which is a Nmap module implementing our semi-automated approach. In Section 4, we show and discuss the preliminary test results achieved by evaluating the Vulsploit module. Finally, in Section 5, we give some final considerations and draw future research directions.

2 The proposed semi-automated approach

In this section, we show the reasons that led us to the introduction of our proposal. Furthermore, we describe the architecture underlying the proposed approach.

2.1 Motivations

The Penetration Testing (PT) process is quite costly and takes a considerable amount of time, ranging from days to weeks. The main phases of a typical PT process are the following: *information gathering*, *network scanning*, *target enumeration*, *vulnerability assessment*, *target exploitation*, *target post exploitation*, and *final reporting*. Since each phase takes time and may require procedures usually carried out manually, our goal is to automate some of these phases. This way, the whole process turns out to be faster and less prone to human error. This paper aims to automate the phases ranging from the network scanning to the exploitation as much as possible. Notice that when the target asset's vulnerabilities are detected through the vulnerability assessment phase, the next step is to understand if and how they can be exploited. More precisely, to take advantage of a detected vulnerability, we need to look for an exploit that allows us to exploit such vulnerability, typically gaining access to the vulnerable asset. Nowadays, this process is mostly done manually. Besides, we remark that the process described above should be done for all the vulnerabilities detected during the vulnerability assessment phase. Therefore, when the vulnerabilities found are few, the time taken to find all the possible exploits for such vulnerabilities is not

relevant. However, when there are thousands of vulnerabilities, such a manual approach becomes impractical. In fact, by calculating with hypothetical data, we can obtain roughly the following time estimate:

- `1` minutes to search for the exploit of a given vulnerability;
- `1000` total vulnerabilities;
- `1 × 1000 = 1000 minutes ≈ 16.6 hours`.

The semi-automated approach proposed in this paper is intended to significantly reduce the search time effort required to find suitable exploits for the found vulnerabilities. In particular, the main objective of our approach is to merge and automate, in a single phase, the following phases of a typical PT process, so that they result in the eyes of the pentester as a single phase:

1. *Network Scanning*
2. *Enumeration*
3. *Vulnerability Assessment*
4. Mid-stage of *Exploitation*, because it searches for the exploit, but does not execute it.

The idea behind our proposal is shown in Figure 1(a) and Figure 1(b).

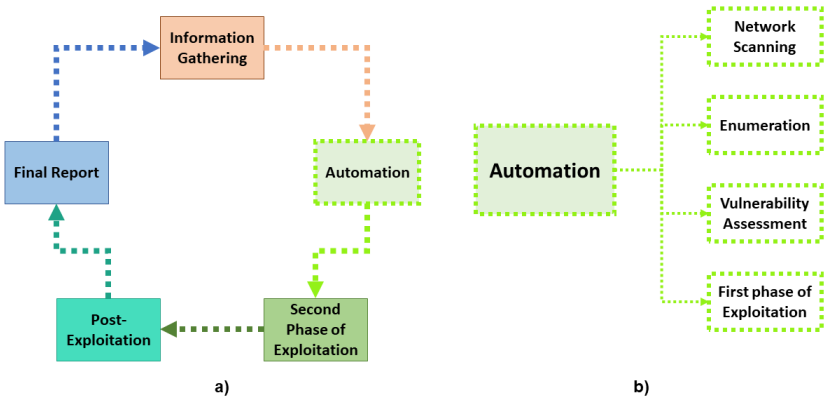

Fig. 1. (a) Phases of the PT process with the *automation*; (b) The subphases of the *automation*.

The benefits arising from the use of the proposed approach are twofold. First of all, it allows an experienced pentester to save resources in terms of time and costs, reducing the likelihood of making errors. On the other hand, our approach allows inexperienced pentesters to conduct the penetration testing process more efficiently and effectively since various stages of this process are automated [27].

2.2 Architecture

In Figure 2, we show abstraction and generalization of our proposal. We first provide an abstract architectural view of our proposal, which abstracts the approach we have introduced from the relative implementation details. In this way, the proposed approach can be adapted according to the evolution of technologies and methodologies and based on the specific operational requirements of the context in which this approach will operate.

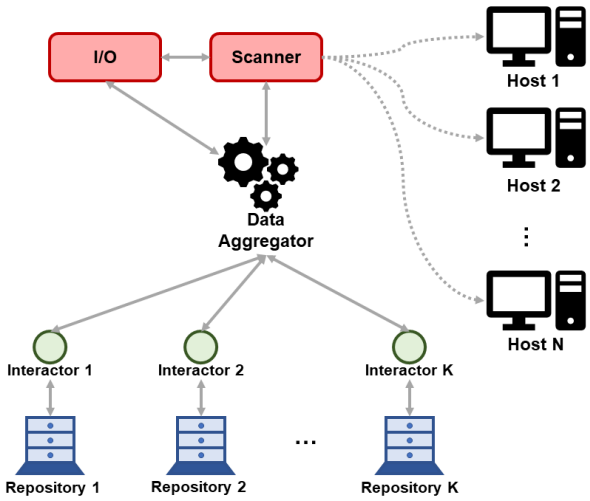

Fig. 2. The architecture of the proposed approach.

As shown in Figure 2, the main functional units of the architecture of the proposed module are the following:

- **I/O**. The Input/Output unit is the component that allows us to send and receive data. More precisely, this unit is responsible for managing the *standard input (stdin)*, *standard output (stdout)* and *standard error (stderr)*.
- **Host**. Hosts are the target machines (i.e., the *asset*) scanned to find vulnerabilities.
- **Scanner**. The scanner is the unit that deals with finding and analyzing vulnerabilities on the asset.
- **Data Aggregator**. The Data Aggregator is the unit that takes care of collecting, aggregating, and transferring data to the I/O unit.
- **Interactor**. This unit takes care of interacting with the Repository.
- **Repository**. The Repository is the unit that contains a list of vulnerabilities, categorized by port, service, and service version.

The information flow between the functional units is shown in Figure 2 and can be characterized as follows:

1. The *Scanner* is started using the *I/O* unit.
2. Once started, the *Scanner* interacts with the *Hosts* and finds vulnerabilities for each of them.
3. These vulnerabilities are then sent to the *Data Aggregator*, which deals with:
   - Collecting the data achieved through the searches carried out to find vulnerabilities;
   - Aggregating them in a structured way, for example, through *JSON* structures;
   - Showing them to the user through the I/O unit.

More precisely:

- Each *Repository* interacts with a dedicated *Interactor*, the output of each *Interactor* is then passed to a *Data Aggregator* once the processing is finished.
- The *Data Aggregator* aggregates the outputs of all the *Interactors* and then returns the output to the user.
- *Interactors* can be distributed, and their execution can be parallelized.
- The primary role of the *Interactors* is to mitigate the performance issues deriving from the massive query execution for each detected vulnerability.

3 The Vulsploit Module

This section shows the design and implementation choices that led us to create the proposed Nmap module (script), called `Vulsploit`, implementing our semi-automated approach. Vulsploit has been realized through an NSE (Nmap Script Engine) script, called "vulsploit.nse", which integrates the features offered by the *Vulscan* vulnerability scanner. This script provides the most suitable tools for the exploitation of the detected vulnerabilities.

In Figure 3 we show an overview of the `Vulsploit` module.

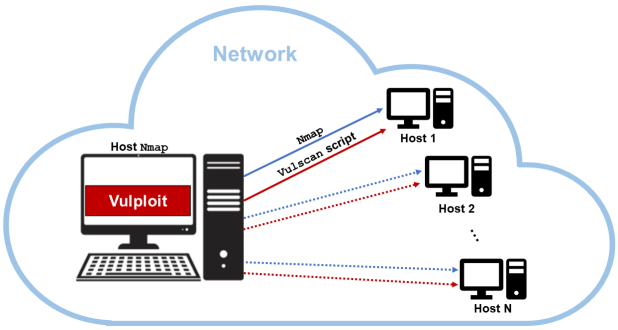

Fig. 3. An overview of the logical functioning of the proposed module.

Vulsploit automatically provides the exploits available for the vulnerabilities detected on a given asset. In detail, the general functioning of Vulsploit can be summarized as follows.

- We have a network scenario composed of $N + 1$ connected hosts.
- The component represented on the left side of Figure 3 is the host where the `Nmap` scanner is running. This scanner will start the execution of `Vulsploit`. More precisely:
  1. `Nmap` scans the asset (i.e., the hosts);
  2. The Nmap Scripting Engine (NSE) executes `Vulsploit`.
- On the right side of Figure 3 there is the asset (i.e., the N hosts) on which the target enumeration phase is carried out through `Nmap`. More precisely, Nmap
  1. using `Vulscan` searches for vulnerabilities of each port/service returned by Nmap;
  2. using `Vulsploit` searches for exploits related to the vulnerabilities found on each host.

3.1 Execution Flow

Vulsploit is composed of several logic components that interact with each other to carry out its duties. The actions performed by Vulsploit can be summarized as follows:

1. Execution of `Vulscan`, whose output is redirected to a file. Each line of this file roughly characterizes a vulnerability;
2. For each vulnerability present in this file, Vulsploit searches the relative exploits, making calls to local and/or remote repositories.

The following diagram outlines the detailed execution flow of `vulsploit.nse`.

As shown in Figure 4, the general execution flow of `Vulsploit` can be summarized by the following main actions:

1. `Vulsploit` runs `Vulscan` to find the vulnerabilities on a given host, and redirects the output to a file.
2. After running `Vulscan`, the output file is read, row by row, by `Vulsploit`.
3. For each row:
   - If the row does not characterize a vulnerability but just a port number, a service, and a service version, this row is printed without performing any other action.
   - If the row corresponds to a vulnerability database:
     - If the database is `Exploit-DB`, `Vulsploit` makes a local query and returns the exploits.
     - If the database is `OSVDB` or `MITRE CVE` or any other database, `Vulsploit` makes a remote query and returns the exploits.
   - The exploits found by each local or remote query (or call) are eventually manipulated, returned, and then printed.

We remark that in general, for each database that requires making a remote query via API, the type of query to perform could change. In the following, we focus on the main components of the Vulsploit module.

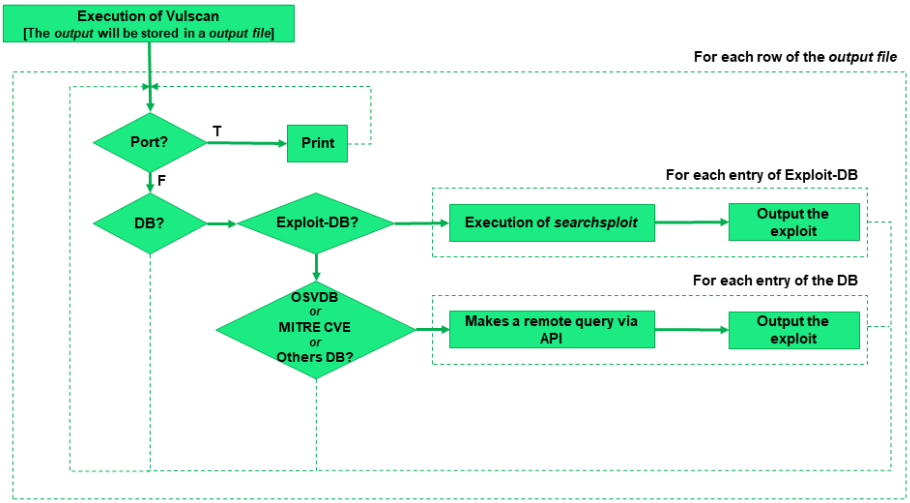

Fig. 4. `vulsploit.nse` execution flow.

3.2 Main Components

Scan and search for vulnerabilities In detail, the first step performed by Vulsploit during its execution is searching for vulnerabilities. This search, whose output is redirected to a file, is performed through a script, called *Vulscan*, that provides the vulnerabilities found for a given host (or asset). An example of this step is shown in Figure 5.

```
Starting Nmap 7.80 ( https://nmap.org ) at 2020-03-06 13:11 CET
Nmap scan report for 10.0.2.6
Host is up (0.00022s latency).
Not shown: 977 closed ports
PORT     STATE SERVICE     VERSION
21/tcp   open  ftp         vsftpd 2.3.4
| vulscan: VulDB - https://vuldb.com:
| [43110] vsftpd up to 2.0.4 Memory Leak denial of service
|
| MITRE CVE - https://cve.mitre.org:
| [CVE-2011-0762] The vsf_filename_passes_filter function in ls.c in vsftpd before 2.3.3
allows remote authenticated users to cause a denial of service (CPU consumption and process
slot exhaustion) via crafted glob expressions in STAT commands in multiple FTP sessions, a
different vulnerability than CVE-2010-2632.
|
```

Fig. 5. Initial output of the `Vulscan` script.

Data parsing As mentioned above, for each database that requires making a remote query via API, the type of query to perform could change. Therefore, before searching for the exploits through the appropriate query, depending on

how this query needs to be structured, Vulsploit performs a different type of parsing of the data generated by Vulscan:

– *Vulnerability ID*: get the number in the square brackets, such as "*17491*", as shown in Figure 6.

```
| Exploit-DB - https://www.exploit-db.com:
| [17491] VSFTPD 2.3.4 - Backdoor Command Execution
| [16270] vsftpd 2.3.2 - Denial of Service Vulnerability
|
```

Fig. 6. Vulnerabilities characterized by an ID.

– *Phrase identifying the vulnerability* (Figure 7): eliminates all the special characters, such as parentheses and quotes, since they could cause problems in requests, and removes all words whose length is less than two characters since they may be irrelevant. In detail, the processing of the strings shown

```
[13563] linux/x86 overwrite MBR on /dev/sda with `LOL!' 43 bytes
[13553] Linux - linux/x86 execve() - 51bytes
```

Fig. 7. Vulnerabilities characterized by a phrase.

in Figure 7 produces the following strings:
  - ‘‘linux/x86 overwrite MBR/dev/sda with LOL43 bytes’’
  - ‘‘Linux linux/x86 execve 51bytes’’

– *CVE*, if present: The processing of the string shown in Figure 8 produces

```
| MITRE CVE - https://cve.mitre.org:
| [CVE-2011-0762] The vsf_filename_passes_filter function in ls.c in vsftpd before 2.3.3
allows remote authenticated users to cause a denial of service (CPU consumption and process
slot exhaustion) via crafted glob expressions in STAT commands in multiple FTP sessions, a
different vulnerability than CVE-2010-2632.
|
```

Fig. 8. Vulnerabilities characterized by CVE.

the following string: `CVE-2011-0762`.

Also, regarding the parsing of the other rows of the file which are not vulnerabilities:

– if it is a row of the type shown in Figure 9 this line is printed;
– if instead it is a row of the type shown in Figure 10 this line is ignored.

```
21/tcp   open  ftp         vsftpd 2.3.4
```

Fig. 9. Row of type:`PORT STATE SERVICE VERSION`

```
Starting Nmap 7.80 ( https://nmap.org ) at 2020-03-06 13:11 CET
Nmap scan report for 10.0.2.6
Host is up (0.00022s latency).
Not shown: 977 closed ports
```

Fig. 10. Initial script output.

Vulnerability analysis and exploit research After the parsing of each row of the file, local and/or remote queries are performed by Vulsploit to search for exploits related to detected vulnerabilities.

Data presentation We use the `JSON` format for the presentation of the data. More precisely, the result of both local and remote calls is returned in JSON format.

In detail, for remote calls, since no other manipulation of the result of such calls is performed, this result is printed. On the other hand, for local calls, the result is manipulated as follows:

1. Check if there are any working exploits, or if the response is empty or contains only the payloads of the found exploits.
2. Isolate the result if there are multiple exploits in the response. Notice that for each request, there may be multiple exploits that contain the ID between the fields, for example, in the date, in the title, etc.
3. Retrieve the exploit which matches the vulnerability.

```
[17491] VSFTPD 2.3.4 - Backdoor Command Execution

    *** Exploits ***

       {"Title":"vsftpd 2.3.4 - Backdoor Command Execution
       (Metasploit)""EDB-ID":"17491""Date":"2011-07-05""Author":"Metasploit""Type":"remote""
       Platform":"unix"
       "Path":"/usr/share/exploitdb/exploits/unix/remote/17491.rb"}
```

Fig. 11. Vulnerability and related exploit (*partial output*).

3.3 Vulsploit usage

The `Vulsploit` script can be executed with the following command:
`nmap --script vulsploit/vulsploit.nse --script-args ‘‘IP = 10.0.2.6’’ 10.0.2.6`
where:

- `--script vulsploit/vulsploit.nse` allows to execute the `vulsploit.nse` script;
- `--script-args ‘‘IP = 10.0.2.6’’` pass arguments to the script, `‘‘IP = 10.0.2.6’’` is the IP address of the scanned hots;
- `10.0.2.6` is the IP address passed to `Nmap` for network scanning.
  - We remark that the IP passed to the script and the one passed to `Nmap` must be the same, otherwise the `Vulsploit` script analyzes one machine and `Nmap` another.

4 Preliminary test results

In this section, we show and discuss the preliminary experimental results obtained from the evaluation of Vulsploit. In particular, the purpose of the testing phase is twofold. First, we evaluate the effectiveness of Vulsploit in finding the appropriate exploits for the detected vulnerabilities. Second, we assess the performance of Vulsploit in terms of time when it deals with a large number of vulnerabilities to analyze. Before showing and discussing the experimental results obtained from the evaluation of our proposal, we first define the hardware and software environment used for the testing phase.

4.1 Hardware and software environment

To carry out the testing phase, we used `Kali Linux`, a distribution explicitly designed for security analysis. Besides, we used `Metasploitable 2`, a vulnerable by-design server distribution, which is based on `Ubuntu (32- bit)`. This distribution was created by the *Rapid7 Metasploit* team. Again, `Kali Linux` is virtualized using `VirtualBox`, on the `Microsoft Windows 10` operating system. Vulsploit is executed by Kali Linux on the `Metasploitable 2` distribution.

In detail, we assigned the following resources to the virtual machine:

- 4 GB of RAM DDR3;
- 2 virtual cores on Intel® Core™ i5 4200M;
- 20 GB of memory, out of 237 GB of SSD available.

4.2 Execution Times and Discussions

The execution of the `Vulscan` script on `Metasploitable2` found 23 open ports and roughly 220000 vulnerabilities.

Moreover, using the testing environment mentioned above, Vulsploit was able to process about half (i.e., 113489 vulnerabilities) of the file produced by Vulscan, containing the found vulnerabilities. The processing of such part of the file took

considerable time, i.e., about 51 hours, that is, 2 days and 3 hours. Although this is a large amount of time, it is nevertheless a tolerable amount. Notice that the PT activity is generally not conducted in real-time, but periodically, at dates typically not very close to each other.

However, we remark that manually searching for exploits would take a significantly longer amount of time. The following rough calculation can show the amount of time spent by the pentester using a manual approach. In detail, assuming that an exhaustive manual exploits search for each vulnerability takes about 1 minutes, that search would take 113489 minutes $\approx$ 1891.48 hours $\approx$ 78.81 days.

5 Conclusions and future work

The process of finding an exploit is a very time-consuming activity, even more so when the vulnerabilities detected are many, and the pentester is not very experienced or prepared. One of the most challenging tasks of this activity is searching for exploits to take advantage of the detected vulnerabilities. This activity becomes even more challenging when, for each detected vulnerability, there could be many exploits.

First of all, this paper aimed to define a general approach that allows the collection and integration of data from various exploit repositories. Any exploits collected are then used to provide the pentester with indications on which exploit to choose to conduct the *exploitation* phase of the Penetration Testing process effectively.

We highlight that our approach is not focused on the vulnerability mapping phase, which is carried out through *Vulscan*. On the other hand, our approach is focused on the automatic finding of the exploits that can be used to take advantage of the results achieved by such a phase.

A proof of concept of the proposed approach has been implemented and integrated through a `Nmap` script. More precisely, our proposal has been implemented through a `NSE` script. This script, called `Vulsploit`, is based on technologies such as `Nmap` [19], the `Lua` language [17, 18], `searchsploit`, `cURL`, and the `Shodan` API [5]. The information provided by the `Shodan` API is important and will be increasingly important, given the ever-increasing diffusion and heterogeneity of connected devices. In detail, `Vulsploit` makes use of another `NSE` script, which is a *vulnerability scanner* provided by Nmap called `Vulscan`, to find the exploits available for the vulnerabilities detected by this scan.

The results obtained were quite satisfying since we could obtain the exploit information on thousands of vulnerabilities in a tolerable amount of time, obtaining a substantial saving compared to the manual approach. This time improvement is a considerable advantage, especially if we consider that penetration testing is a routine activity that is typically not conducted in real-time.

Although the results obtained are satisfactory, we could do better in future work, considering more efficient approaches. A first improvement that could be applied is to parallelize the execution of Vulsploit, in order to make calls to the

various exploit repositories in parallel. Furthermore, after having parallelized the script, it could be distributed across multiple hosts, using a distributed or fully distributed paradigm. Besides, we want to improve the performance of Vulsploit through caching mechanisms. The three improvements mentioned above could increase the efficiency of our proposal, reducing execution times. Besides, some other data sources could be integrated to provide a broader set of available exploits. Moreover, current PT practice is becoming repetitive, complex, and resource-consuming despite automated tools. Therefore, since there is an ongoing interest in exploring the potential of *artificial intelligence (AI)* to enhance penetration testing and vulnerability identification of systems [22], we intend to investigate AI techniques to enhance the automation of the PT process further. In particular, we intend to investigate intelligent PT approaches using *reinforcement learning (RL)* that will allow regular and systematic testing, saving human resources [16, 15]. Using trained machine learning agents to automate this process is an important research area that still needs to be explored. Moreover, to decrease the time required by remote fetches, we want to use pre-fetching techniques to get the full remote repositories and keeping them updated.

To conclude, we remark that despite all the extensions and improvements described above to integrate and enhance our proposal, Vulsploit may still be very useful at present to those who carry out PT activities.

References

1. Al-Ahmad, A.S., Kahtan, H., Hujainah, F., Jalab, H.A.: Systematic literature review on penetration testing for mobile cloud computing applications. IEEE Access **7**, 173524–173540 (2019)
2. Ali, B., Awad, A.: Cyber and physical security vulnerability assessment for iot-based smart homes. Sensors **18**(3), 817 (Mar 2018)
3. Almubairik, N.A., Wills, G.: Automated penetration testing based on a threat model. In: 2016 11th International Conference for Internet Technology and Secured Transactions (ICITST). pp. 413–414 (2016)
4. Ankele, R., Marksteiner, S., Nahrgang, K., Vallant, H.: Requirements and recommendations for iot/iiot models to automate security assurance through threat modelling, security analysis and penetration testing. In: Proceedings of the 14th International Conference on Availability, Reliability and Security. ARES '19, Association for Computing Machinery (2019)
5. Bodenheim, R., Butts, J., Dunlap, S., Mullins, B.: Evaluation of the ability of the shodan search engine to identify internet-facing industrial control devices. International Journal of Critical Infrastructure Protection **7**(2), 114–123 (2014)
6. Bojjagani, S., Sastry, V.N.: Vaptai: A threat model for vulnerability assessment and penetration testing of android and ios mobile banking apps. In: 2017 IEEE 3rd International Conference on Collaboration and Internet Computing (CIC). pp. 77–86 (2017)
7. Casola, V., De Benedictis, A., Rak, M., Villano, U.: Towards automated penetration testing for cloud applications. In: 2018 IEEE 27th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE). pp. 24–29 (2018)

8. Casola, V., De Benedictis, A., Rak, M., Villano, U.: A methodology for automated penetration testing of cloud applications. int. J. Grid Util. Comput. **11**(2), 267–277 (2020)
9. Ceccato, M., Scandariato, R.: Static analysis and penetration testing from the perspective of maintenance teams. In: Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. ESEM '16, Association for Computing Machinery (2016)
10. Chaudhary, S., O'Brien, A., Xu, S.: Automated post-breach penetration testing through reinforcement learning. In: 2020 IEEE Conference on Communications and Network Security (CNS). pp. 1–2 (2020)
11. Chen, C., Zhang, Z., Lee, S., Shieh, S.: Penetration testing in the iot age. Computer **51**(4), 82–85 (2018)
12. Chu, G., Lisitsa, A.: Penetration testing for internet of things and its automation. In: 2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS). pp. 1479–1484 (2018)
13. Denis, M., Zena, C., Hayajneh, T.: Penetration testing: Concepts, attack methods, and defense strategies. In: 2016 IEEE Long Island Systems, Applications and Technology Conference (LISAT). pp. 1–6 (2016)
14. Fischer, M., Langer, F., Mono, J., Nasenberg, C., Albartus, N.: Hardware penetration testing knocks your socs off. IEEE Design Test pp. 1–1 (2020)
15. Ghanem, M.C., Chen, T.M.: Reinforcement learning for intelligent penetration testing. In: 2018 Second World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4). pp. 185–192 (2018)
16. Ghanem, M.C., Chen, T.M.: Reinforcement learning for efficient network penetration testing. Information **11**(1), 6 (Dec 2019)
17. Ierusalimschy, R., De Figueiredo, L.H., Filho, W.C.: Lua - an extensible extension language. Software: Practice and Experience **26**(6), 635–652 (1996)
18. Ierusalimschy, R., de Figueiredo, L.H., Celes, W.: The evolution of lua. In: Proceedings of the third ACM SIGPLAN conference on History of programming languages. pp. 2–1 (2007)
19. Lyon, G.F.: Nmap network scanning: The official Nmap project guide to network discovery and security scanning. Insecure (2009)
20. Mahmoodi, Y., Reiter, S., Viehl, A., Bringmann, O., Rosenstiel, W.: Attack surface modeling and assessment for penetration testing of iot system designs. In: 2018 21st Euromicro Conference on Digital System Design (DSD). pp. 177–181 (2018)
21. Markopoulou, D., Papakonstantinou, V., de Hert, P.: The new eu cybersecurity framework: The nis directive, enisa's role and the general data protection regulation. Computer Law & Security Review **35**(6), 105336 (2019)
22. McKinnel, D.R., Dargahi, T., Dehghantanha, A., Choo, K.K.R.: A systematic literature review and meta-analysis on artificial intelligence in penetration testing and vulnerability assessment. Computers & Electrical Engineering **75**, 175 – 188 (2019)
23. Pale, P.C.: Mastering the Nmap Scripting Engine. Packt Publishing Ltd (2015)
24. Rahman, A., Williams, L.: A bird's eye view of knowledge needs related to penetration testing. In: Proceedings of the 6th Annual Symposium on Hot Topics in the Science of Security. HotSoS '19, Association for Computing Machinery (2019)
25. Schwartz, J., Kurniawati, H.: Autonomous penetration testing using reinforcement learning. CoRR **abs/1905.05965** (2019)

26. Shebli, H.M.Z.A., Beheshti, B.D.: A study on penetration testing process and tools. In: 2018 IEEE Long Island Systems, Applications and Technology Conference (LISAT). pp. 1–7 (2018)
27. Stefinko, Y., Piskozub, A., Banakh, R.: 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). pp. 488–491 (2016)
28. Voigt, P., Von dem Bussche, A.: The eu general data protection regulation (gdpr). A Practical Guide, 1st Ed., Cham: Springer International Publishing (2017)
29. Zennaro, F.M., Erdodi, L.: Modeling penetration testing with reinforcement learning using capture-the-flag challenges and tabular q-learning. arXiv preprint arXiv:2005.12632 (2020)