EasyChair Preprint

№ 10077

NIF: Reactive Injection Attack via Nmap Piggybacking

Alessandro Bonfiglio, Gabriele Costa and Silvia De Francisci

May 12, 2023

NIF: Reactive Injection Attack via Nmap Piggybacking

Alessandro Bonfiglio[1], Gabriele Costa[2], and Silvia De Francisci[2]

[1] University of Florence, Florence 50121, Italy
alessandro.bonfiglio@stud.unifi.it
[2] IMT School for Advanced Studies, Lucca 55100, Italy
{gabriele.costa,silvia.defrancisci}@imtlucca.it

Abstract

Network scanning is a common task in cybersecurity. For instance, penetration testers often scan a target system during the initial stage of their vulnerability detection process, e.g., for profiling machines and services. On the other hand, attacker scan remote systems looking for exploitation opportunities. Network scans are generally considered harmless for the victim, as they only consist of a few requests that cause no service interruption or degradation. Nevertheless, as shown in [19], scanning is risky for its author.

In this paper, we present a general attack framework that takes advantage of network scans for injecting remote systems. In particular, our proposal leverages the widely adopted scanner Nmap [15] for transmitting attack payloads through the scan responses. If the output of Nmap is processed by an injectable application, e.g., a web browser of a SQL DBMS, our payloads are executed and the scanning system gets compromised.

1 Introduction

Roughly, network scanning amounts to testing whether a target system accepts connections to certain ports. For instance, a web server might accept requests on ports 80 (HTTP) and 443 (HTTPS), while a mail server could use ports 25 (SMTP) and 143 (IMAP). The list of available services is extremely important for security-related activities. As a matter of fact, services that listen to incoming connections are part of the perimeter that, e.g., an attacker might want to penetrate. Scanning the active ports of a remote system is thus fundamental for collecting crucial information such as the type and version of the running services.

The most famous and adopted scanning tool is Nmap [15]. Analysts/attackers can rely on various scanning strategies. A common one is based on establishing a TCP connection with the target service. Briefly, every TCP connection starts with the notorious 3-way handshaking. Upon completion of the handshake, most services send some banner message containing information about the service type and version. This message is thus processed by Nmap to identify the scanned service. For instance, connecting to `ftp.gnu.org` one gets the banner message `"220 GNU FTP server ready"`.

Clearly, when establishing a direct connection, the target server can read the client's IP address, which might be useful for attribution purposes. However, to avoid it, scan authors can resort to various techniques, depicted in Figure 1. The first scenario consists of a client performing a scan through a proxy server running Nmap. For instance, the client may resort to one among many network scanning web applications such as `https://nmap.online/`. Also, an attacker might have gained control over a remote server where she can install and run tools. In both cases, the target would attribute the scan operation to the proxy server, rather than the actual author. An alternative approach consists of scanning through the TOR network. Since the traffic goes through a number of relay nodes, the target can only observe the last, exit relay. Under these scenarios, attributing a Nmap scan to its author is, in general, undoable.

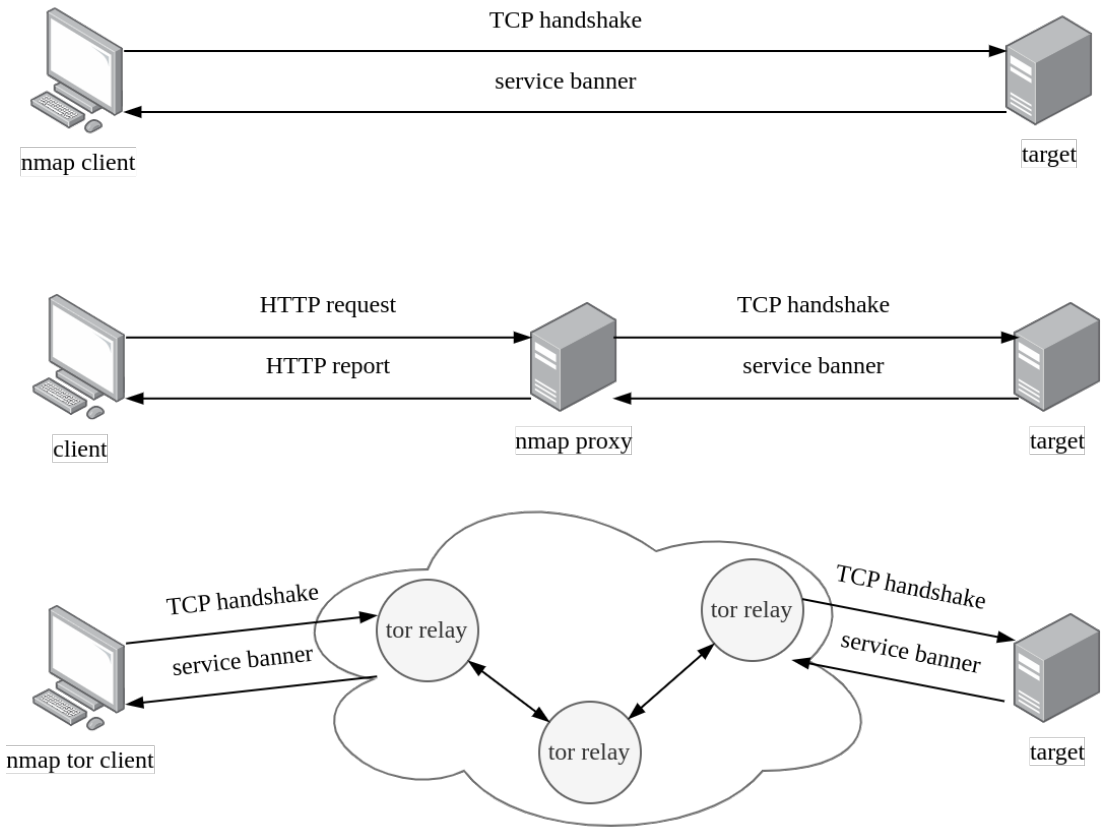

Figure 1: Nmap TCP scanning scenarios.

For the reasons stated above and since they generate a negligible amount of traffic, network scans are usually tolerated. Nevertheless, banner grabbing is part of the attackers' kill chain and it should be considered a risky operation. In [19], the authors showed that this is not only true for scan victims, but also for their authors. Under their attacker model, they demonstrated that HTTP service banners may contain cross-site scripting payloads that the scan target uses to inject the authors' browser. The reason is that the service banner is processed by Nmap to find critical information, e.g., service name and version. In many cases, Nmap is used as a component in some exploitation frameworks. Thus, returned details may be integrated into a HTML report without a proper sanitization.

In this paper, we present an extension of the attack scenario presented in [19]. Starting from the same attacker model, we extend it by generalizing (i) the set of injectable protocols and (ii) the family of code injection payloads. The first generalization is based on a reverse engineering of the Nmap service reconnaissance method. In particular, we develop a payload generation technique that returns injected banners following the exact syntax expected by Nmap. This ensures that Nmap propagates our payloads to the upper level, i.e., the attack or penetration testing framework, if it exists. On the other hand, our attack strategy introduces a generic and configurable set of attack payloads. As a matter of fact, analysts can provide their attack payloads via configuration files. For instance, a set of payloads may consist of SQL injections,

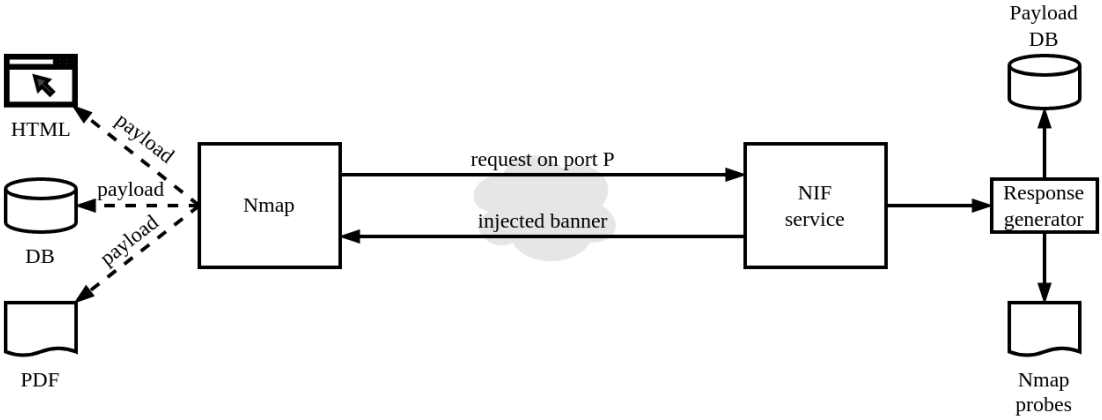

Figure 2: Abstract workflow of NIF.

while another may be for cross-site scripting. Our methodology is then demonstrated through a prototype implementation. The prototype is released as an open source project for the security community.

The rest of this paper is structured as follows. Section 2 illustrates our framework. In Section 3 we discuss a usage scenario of NIF. Finally, we survey the related literature in Section 4 and conclude the paper in Section 5.

2 Framework

In this section, we describe our attack framework and its implementation details.

2.1 Overview

The abstract workflow of our framework is depicted in Figure 2. The scanning platform appears on the left. Briefly, it consists of Nmap and some reporting systems. The reporting system can belong to various categories. For instance, it can be an HTML report or a PDF document. Possibly, the scan result may even be stored in a database.

On the other hand, the NIF platform exposes a frontend service. When a Nmap scan occurs on port P, the NIF service triggers a response generation utility that uses the Nmap service probes (see Section 2.2) and a collection of payloads, e.g., stored in a database. The generated response resembles a legal service banner, but it carries the injection payload. When Nmap processes the injected banner message, the payload is eventually returned to the reporting system.

2.2 Reversing Nmap Probes

Nmap service recognition relies on a list of service probes. Each probe amounts to a rule for parsing a service banner. In this way, even if a service is not running on its usual port (e.g., SSH on port 22) Nmap can correctly identify it. For instance, consider the following service probe rule.

```
match ssh m|^SSH-([\d.]+)-OpenSSH[_-]([\w.]+)\r?\n|
    p/OpenSSH/ v/$2/ i/protocol $1/ cpe:/a:openbsd:openssh:$2/
```

The rule states that the received banner message matches an SSH service if it follows the syntax between `m|...|`. The syntax is given through a Perl-compatible regular expression (PCRE). For instance, we can observe that the string `SSH-1.2-OpenSSH_v5` matches the PCRE. The second part of the rule tells Nmap how to interpret a successful match. Every segment refers to specific information that Nmap infers from the banner message. In this case, for instance, `p/OpenSSH/` is for the product name. More interestingly, `v/$2/` tells Nmap that the service version is taken directly from the banner content. In particular, `$2` identifies the second *capture group* parsed by the regular expression, i.e., the part of the string that matches the expression inside the second pair of rounded parentheses. In the previous example, this value is `v5`. Finally, segments `i/.../` and `cpe:/.../` are for info and platform (e.g., OS and hardware) identification, respectively.

Since Nmap extracts the information matching capture groups and includes it in its output, we can expect these details to be propagated to the reporting system. Hence, they represent the ideal target for placing a malicious payload. For instance, consider the banner `SSH-1.2-OpenSSH_PAYLOAD`. Nmap would extract from it the product version `PAYLOAD`.

Clearly, not every capture group is a suitable candidate for placing a payload. As a matter of fact, the payload must match the regular expression between parentheses. As an example, consider the first capture group in the previous rule, i.e., `([\d.]+)`. Since only digits and the `'.'` symbol can appear there, we cannot use it for any meaningful injection. Nevertheless, since all the matching rules and capture groups are listed in the source code of Nmap, we can easily filter those that admit payload injection.

2.3 Payload Management

As stated above, the first step is finding capture groups that we can instantiate with payloads. For instance, we might look for `(.*)`, i.e., the capture group matching any finite sequence of characters. Performing this search on the Nmap service probe file returns 89 entries such as the following one.

```
match ftp m|^220 vsFTPd (.*) ready\.\.\.\r\n| p/vsftpd/ v/$1/
```

Nevertheless, other regular expressions can be used as well. For instance, we might search for `(.*` and `.*)`, i.e., capture group admitting a preamble or a trail (respectively) made of any sequence of chars. This results in rules such as this one.

```
match ssh m|^SSH-([\d.]+)-OpenSSH_([\w._-]+)[ -]{1,2}Debian[ -_](.*ubuntu.*)\r\n|
```

In this case, the capture group `(.*ubuntu.*)` can be used for our purposes.

With this approach, we can already identify 153 injectable probes. Nevertheless, more refined searches may be implemented. The basic idea is that one might use different charsets for each payload category. For instance, the character `'/'` may be necessary for XSS payloads, but irrelevant for SQLi ones. Thus, we may test each capture group against the charset of a specific injection attack for checking whether it can be injected.

Independently from the used method, NIF is eventually provided with a list of injectable probes. For each probe, the NIF service exposes a service on the corresponding port. When a scan occurs on port P, the payload generation proceeds in the following way. First, a probe PCRE is selected among those that (*i*) corresponds to a service running on port P (e.g., FTP if P = 21), and (*ii*) contains at least an injectable capture group. Briefly, filter (*i*) is implemented by using the predefined service mapping included in the source code of Nmap. In particular, the configuration file `nmap-services` consists of a finite list following the syntax below.

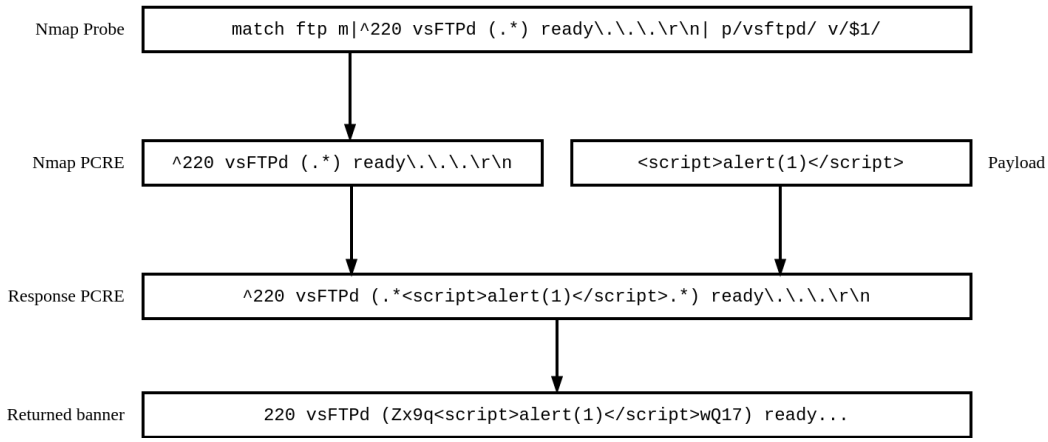

Figure 3: NIF response generation flow example.

```
service port/protocol frequency # comment
```

There, `service` identifies the service, e.g., `ftp`, `port/protocol` is for the service port and protocol type, e.g., `21/tcp`, `frequency` denotes the service likelihood, i.e., how often Nmap expects to find the service running on the given port, and `comment` is free text. Instead, filter (*ii*) is implemented by considering the specific type of injection attack (given as an input). To check whether a capture group is injectable with a payload for a certain attack, e.g., XSS, we compute the intersection between PCRE of the capture group and a signature PCRE for the attack. Briefly, the signature PCRE is `.*C.*` where `C` is the payload charset described in Section 2.2. If the intersection between the two PCRE is not empty, we consider the probe injectable. As an example, consider again the capture group `(.*ubuntu.*)` discussed above and imagine that `C` = `<>()alert/""123`, i.e., a simplified charset for XSS. When computing `.*ubuntu.*` ∩ `.*<>()alert/""123.*` we find that, for instance, it includes the string `<>()alert/""123ubuntu`.

Once the injectable probe is selected, an actual response message that both (*i*) follows its syntax and (*ii*) contains an attack payload must be generated. This is achieved by means of a PCRE generator that, given a regular expression, returns a random string belonging to its language. In particular, the PCRE used is that of the selected probe, where the injectable capture groups are modified as follows. Imagine the capture group PCRE is `E` and the selected payload is `P`, we replace `E` with the intersection `E` ∩ `.*P.*`. Again, if we consider the example above, if `E` = `.*ubuntu.*` and `P` = `<script>alert(1)</script>`, we obtain `[[.*ubuntu.*]&&[.*<script>alert(1)</script>.*]]`[1]. The overall response generation flow described above is exemplified in Figure 3.

3 NIF Demonstration

In this section, we present our implementation, called NIF. Furthermore, we demonstrate NIF through an attack scenario where the adversary establishes remote control over the victim's

[1] Different implementations of PCRE may use different operators for intersection. Here we refer to the Java implementation using `[[...]&&[...]]`.

```
┌──(kali㉿kali)-[~/Desktop/NIF/TextFiles/injectableProbes]
└─$ head injectable-service-probes.txt
match ftp m|^220-.*\r\n220-\r\n220 using FileZilla FileZilla Server version ([^\r\n]+)\r\n|s
 p/FileZilla ftpd/ v/$1/ o/Windows/ cpe:/a:filezilla-project:filezilla_server:$1/ cpe:/o:mic
rosoft:windows/a
match ftp m|^220 AXIS ([\d\w]+)V(\d\S+) (.*?) ready\.\n| p/AXIS $1 Webcam ftpd/ v/$2/ i/$3/
d/webcam/ cpe:/h:axis:$1/a
match ftp m|^220 AXIS (.+) FTP Network Print Server V([-\w_.]+) | p/AXIS $1 print server ftp
d/ v/$2/ d/print server/ cpe:/h:axis:$1/a
match ftp m|^220- (.*) WAR-FTPD ([-\w.]+) Ready\r\n220 Please enter your user name\.\r\n| p/
WAR-FTPD/ v/$2/ i/Name $1/ o/Windows/ cpe:/o:microsoft:windows/a
match pop3 m|^\+OK ([-\w_.]+) Cyrus POP3 v(\S+?)[-_]?Debian\S+ server ready| p/Cyrus pop3d/
v/$2/ i/Debian/ o/Linux/ h/$1/ cpe:/a:cmu:cyrus_imap_server:$2/ cpe:/o:debian:debian_linux/
```

Figure 4: Output generated by the filter utility.

browser. All the material presented below is available at https://github.com/iAleKira/Nmap-Injection-Framework.

3.1 Implementation

NIF is available as an open-source, Java-implemented project consisting of a few components. We briefly describe them below.

Payloads. This package contains a collection of injection payloads. Payloads are stored in each row of text files. Text files are organized according to their type. For instance, `xss_payloads.txt` contains a collection of XSS payloads. New injection attacks can be implemented by adding payloads' files to this package. NIF retrieves these payloads when generating malicious responses to Nmap scans.

Filter. The filter utility is a stand-alone Java executable used to extract reversible Nmap probes, as explained in Section 2.2. Filtered probes are stored in a text file that NIF Server can access. Figure 4 shows the output generated by the following command.

```
java -jar filter.jar "<script>alert(1)</script>"
```

In Figure 4, we report four rules for FTP and one for POP3 (out of 37 total rules).

Server. This component is in charge for delivering the attack payloads once receiving a Nmap scan. Briefly, it implements the NIF service and the response generator described in Section 2. The NIF Server instantiates Java sockets listening on the ports associated with the injectable probes, e.g., port 25 for the FTP service.

Limitations. The general approach discussed in Section 2 and the current implementation of NIF mainly differ for a single aspect, i.e., the payload injection strategy. As discussed in Section 2.3, the general strategy is based on computing the intersection between two regular languages, i.e., one for the service probe and one for the injection payload. However, for the time being, NIF only generates payloads belonging to the regular language of the probe. For instance, consider the case where the probe's regular expression is `.*ubuntu.*` and the payload is `<script>alert(1)</script>`. The approach described in Section 2.3 would allow generating, e.g., the response `<script>alert(1)</script>ubuntu`. However, the current implementation of NIF cannot find this match. Although this reduces the number of potential injection vectors, NIF can already identify several injectable probes for most payloads. We plan to extend NIF with this functionality as part of our future work.

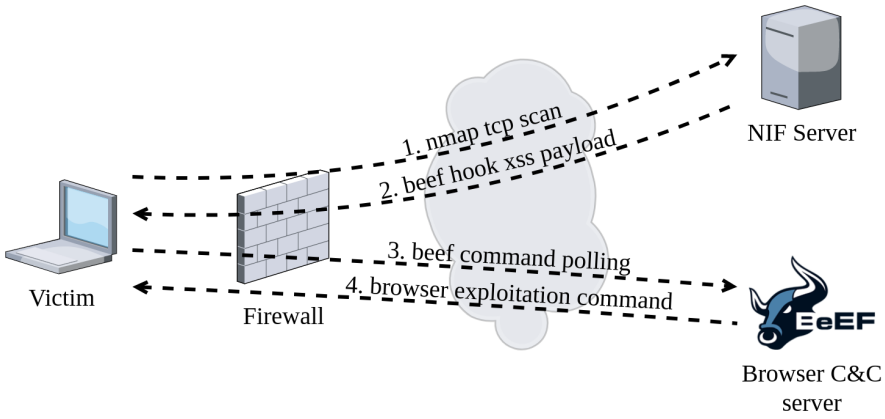

Figure 5: Remote browser takeover scenario with NIF and BeEF.

```
└─$ ./Injector_server.jar
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Server listening on ports: [20, 22, 25, 80]
Client kali.homenet.telecomitalia.it opened connection on local port:80.
Payload to deliver: <script src="http://192.168.1.17:3000/hook.js"></script>
Response delivered: HTTP/1.1 547 }çLdºwÖÜ¾`¿~
Connection: Keep-Alive
Server: Siemens Gigaset <script src="http://192.168.1.17:3000/hook.js"></script>

Client kali.homenet.telecomitalia.it opened connection on local port:22.
Payload to deliver: <script src="http://192.168.1.17:3000/hook.js"></script>
Response delivered: SSH-21-OpenSSH_zn0O- DebianX<script src="http://192.168.1.17:300

Client kali.homenet.telecomitalia.it opened connection on local port:22.
Payload to deliver: <script src="http://192.168.1.17:3000/hook.js"></script>
Response delivered: SSH-86299967.9-OpenSSH_PC- DebianE<script src="http://192.168.1.
cript>
```

Figure 6: NIF startup and delivering BeEF's XSS hook.

3.2 Demonstration: Injecting BeEF's XSS Hook

In this demonstration, we show how NIF can be used to inject the victim with a XSS payload to hijack the target browser. The scenario is depicted in Figure 5. The attacker controls both the NIF server and a BeEF Command and Control (C&C) server. Briefly, BeEF [3] is a browser exploitation framework that relies on XSS injection to install a control module on a target browser. The control module is configured to periodically connect to the C&C server and retrieve commands to be executed. Since the victim's browser does active polling, BeEF can effectively bypass firewall controls in most cases.

The attacker configures NIF to inject the payload

```
<script src="HTTP://[IP]:3000/hook.js"></script>
```

where `[IP]` is the IP address of the BeEF C&C server (exposed on port 3000). NIF identifies four injectable service probes and configures a listening port for each of them. When the victim's machine scans one of these ports, it receives the payload. If the Nmap report is then flushed into an HTML report, the victim's browser connects to the C&C server, as discussed above. Figure 6 shows the output of NIF when injecting Nmap under our scenario.

4 Related Work

To the best of our knowledge, RevOk [19] was the first and only proposal exploiting scan operations to perform injection attacks. In detail, RevOk uses a malicious web application in order to attack whoever scans it, injecting malicious payloads into the browser of the scan author. Injection payloads are embedded in HTTP response headers that Nmap parses to fingerprint the web server. Our work extends RevOk by using a general strategy that applies to every injectable protocol, rather than HTTP only. In the following three sections, we briefly revise the state-of-the-art of active defense techniques, security scanner weaknesses, and injection attacks on web applications. Although these works are not alternatives to our proposal, they are relevant w.r.t. the technologies involved in our reference attacker model.

4.1 Active Defense

Active defense refers to a set of cybersecurity measures that involve actively identifying, preventing, and responding to cyber threats. For a detailed review of the different types of active defense and related possible actions, we refer the interested reader to Glosson [8].

De Gaspari et al. [6] present AHEAD, an active defense approach based on deception. AHEAD allows redirecting back traffic received on a port, thanks to Rubberglue [1], generating attractive files that, once opened, permit the Web Bug Server [2] to reveal the attacker's identity and obtain the IP address of her machine. Unlike ours, this approach relies on analyzing the attacker's strategy and counterstriking. Similarly, Rana et al. [16] exploit rely on a honeypot to obtain attacker information. In particular, they consider the possibility of the attacker using a VPN and propose a method that can bypass it and get accurate information.

Another method is to directly identify and exploit vulnerabilities in the attacker's tool suite. For instance, Dereszowski [7] discovered a vulnerability in the Poison Ivy RAT. He built a buffer designed to overwrite the return address of the function and then drive the function back to a RET instruction. Thus, when the attacker uses this tool to spy on the victim, the manipulated buffer causes a buffer overflow and the execution of arbitrary shellcode.

4.2 Security Scanners

Reports about scanner vulnerabilities are not frequent in the literature. Yet, some authors have considered the security weaknesses of scanners. These works focus on issues such as incomplete scanning, which can leave some vulnerabilities undetected; false positives, which are reported vulnerabilities that do not exist; or performance issues.

For instance, Holm et al. [11] present a quantitative analysis of seven of the most used vulnerability scanners. Vieria et al. [20] perform an experimental evaluation of security vulnerabilities in 300 web services, demonstrating the advantages and limitations of these scanners. Along the same line, Bau et al. [5] investigate the effectiveness as well as the relevance of the discovered vulnerabilities. Idrissi et al. [12] compare the performance and efficiency of several commercial and free scanners. Nevertheless, none of the previous works consider whether a security scanner can convey injection attacks toward the scan author.

4.3 Injection Attacks on Web Applications

Web applications may be vulnerable to several types of injection attacks. One is the command injection attack that involves injecting malicious command-line code. The first formal definition of command injection attacks in the context of web applications is provided by Su and

Wassermann [18]. The XML injection attack involves injecting malicious XML code, which can be used to manipulate or access data stored in an XML file, as discussed in [9]. The LDAP injection attack is analyzed in depth by Alonso et al. [4]. This technique allows obtaining direct access to the hierarchical database underlying an LDAP tree. Another attack is SQL injection, among the most used techniques to violate web applications. Several authors survey these attacks, e.g., see [17, 14]. In particular, Halfond et al. [10] not only classify SQL injection attacks but compare techniques for detecting and preventing them. Similarly, Kumar and Pateriya [13] survey attacks and defense techniques for injection scenarios. All the injection attacks discussed above are compatible with our approach. As a matter of fact, our methodology treats injection payloads agnostically and an attacker can pick the specific payloads targeting the victim's infrastructure.

5 Conclusion

In this paper, we presented a general methodology for embedding injection attacks in Nmap scan responses. Our attack targets systems, e.g., penetration testing frameworks and reporting tools, that import the output of Nmap without proper sanitization. Although a systematic assessment of the impact of our attack strategy is yet to be carried out, we believe that many developers might excessively trust the output of Nmap when it is locally executed. In future work, we plan to experimentally validate this hypothesis by exploiting the tool presented in this paper for implementing a vulnerability detection campaign. Furthermore, such a campaign will be used to measure the impact of our methodology.

References

[1] Rubberglue. https://github.com/adhdproject/rubberglue. Accessed January 5, 2023.

[2] Web bug server. https://bitbucket.org/ethanr/webbugserver. Accessed January 5, 2023.

[3] Wade Alcorn. Beef: The browser exploitation framework. https://beefproject.com/. Accessed January 11, 2023.

[4] Jose Maria Alonso, Rodolfo Bordon, Marta Beltran, and Antonio Guzmán. Ldap injection techniques. In *2008 11th IEEE Singapore International Conference on Communication Systems*, pages 980–986. IEEE, 2008.

[5] Jason Bau, Elie Bursztein, Divij Gupta, and John Mitchell. State of the art: Automated black-box web application vulnerability testing. In *2010 IEEE symposium on security and privacy*, pages 332–345. IEEE, 2010.

[6] Fabio De Gaspari, Sushil Jajodia, Luigi V Mancini, and Agostino Panico. Ahead: A new architecture for active defense. In *Proceedings of the 2016 ACM workshop on automated decision making for active cyber defense*, pages 11–16, 2016.

[7] Andrzej Dereszowski. Targeted attacks: From being a victim to counter attacking. *Black Hat Europe*, 2010.

[8] Anthony Glosson. Active defense: An overview of the debate and a way forward. 2015.

[9] Abhinav Nath Gupta and P Santhi Thilagam. Attacks on web services need to secure xml on web. *Computer Science & Engineering*, 3(5):1, 2013.

[10] William G Halfond, Jeremy Viegas, Alessandro Orso, et al. A classification of sql-injection attacks and countermeasures. In *Proceedings of the IEEE international symposium on secure software engineering*, volume 1, pages 13–15. IEEE, 2006.

[11] Hannes Holm, Teodor Sommestad, Jonas Almroth, and Mats Persson. A quantitative evaluation of vulnerability scanning. *Information Management & Computer Security*, 2011.

[12] SE Idrissi, N Berbiche, F Guerouate, and M Shibi. Performance evaluation of web application security scanners for prevention and protection against vulnerabilities. *International Journal of Applied Engineering Research*, 12(21):11068–11076, 2017.

[13] Puspendra Kumar and RK Pateriya. A survey on sql injection attacks, detection and prevention techniques. In *2012 Third International Conference on Computing, Communication and Networking Technologies (ICCCNT'12)*, pages 1–5. IEEE, 2012.

[14] MA Lawal, Abu Bakar Md Sultan, and Ayanloye O Shakiru. Systematic literature review on sql injection attack. *International Journal of Soft Computing*, 11(1):26–35, 2016.

[15] Gordon Fyodor Lyon. *Nmap network scanning: The official Nmap project guide to network discovery and security scanning*. Insecure. Com LLC (US), 2008.

[16] Muhammad Usman Rana, Osama Ellahi, Masoom Alam, Julian L Webber, Abolfazl Mehbodniya, and Shawal Khan. Offensive security: Cyber threat intelligence enrichment with counterintelligence and counterattack. *IEEE Access*, 10:108760–108774, 2022.

[17] Amirmohammad Sadeghian, Mazdak Zamani, and Shahidan M Abdullah. A taxonomy of sql injection attacks. In *2013 International Conference on Informatics and Creative Multimedia*, pages 269–273. IEEE, 2013.

[18] Zhendong Su and Gary Wassermann. The essence of command injection attacks in web applications. *Acm Sigplan Notices*, 41(1):372–382, 2006.

[19] Andrea Valenza, Gabriele Costa, and Alessandro Armando. Never trust your victim: Weaponizing vulnerabilities in security scanners. In *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*, pages 17–29, San Sebastian, October 2020. USENIX Association.

[20] Marco Vieira, Nuno Antunes, and Henrique Madeira. Using web security scanners to detect vulnerabilities in web services. In *2009 IEEE/IFIP International Conference on Dependable Systems & Networks*, pages 566–571. IEEE, 2009.