# Progetto del corso di Sicurezza informatica e Internet A.A. 2017-2018

Andrea Graziani (0273395)[1], Alessandro Boccini (0277414)[1], and Ricardo Gamucci (0274716)[1]

[1]Università degli Studi di Roma Tor Vergata

27 marzo 2019

# Indice

# Capitolo 1

# Descrizione dell'attacco

Secondo il rapporto stilato dalla NCCIC[1], il malware denominato **FASTCash** è composto da una serie **12 file** i quali, attraverso tecniche di **code injection** tali da alterare il normale comportamento di uno o più processi legittimi, che hanno consentito l'ispezione e alterazione dei dati trasmessi durante transazioni basate su **protocollo ISO 8583**, hanno permesso agli attaccanti di eseguire operazioni di prelievo fraudolento di denaro dagli ATM. Tra questi file, di cui riportiamo una lista completa in 1.1, spiccano per importanza:

- Tre file progettati per essere eseguibili su sistemi operativi AIX, uno dei quali responsabile dell'esecuzione della code injection contro i processi operanti sul server bersaglio. Due di essi sono stati analizzati nelle sezioni 2.2.3 e 2.2.1
- Due versioni di un malware capace di modificare le impostazioni del firewall.
- Un **trojan** capace di consentire **accesso remoto completo** al sistema bersaglio.

[1] Cfr. `https://www.us-cert.gov/ncas/analysis-reports/AR18-275A`

Tabella 1.1: Lista dei file del malware FASTCash

| **Nome file** | **SHA256 digest** |
|---|---|
| Lost_File.so | 10ac312c8dd02e417dd24d53c99525c29d74dcbc84730351ad7a4e0a4b1a0eba |
| Unpacked_dump_4a740227eeb82c20... | 10ac312c8dd02e417dd24d53c99525c29d74dcbc84730351ad7a4e0a4b1a0eba |
| Lost_File1_so_file | 3a5ba44f140821849de2d82d5a137c3bb5a736130dddb86b296d94e6b421594c |
| 4f67f3e4a7509af1b2b1c6180a03b3... | 4a740227eeb82c20286d9c112ef95f0c1380d0e90ffb39fc75c8456db4f60756 |
| 5cfa1c2cb430bec721063e3e2d144f... | 820ca1903a30516263d630c7c08f2b95f7b65dffceb21129c51c9e21cf9551c6 |
| Unpacked_dump_820ca1903a305162... | 9ddacbcd0700dc4b9babcd09ac1cebe23a0035099cb612e6c85ff4dffd087a26 |
| 8efaabb7b1700686efedadb7949eba... | a9bc09a17d55fc790568ac864e3885434a43c33834551e027adb1896a463aafc |
| d0a8e0b685c2ea775a74389973fc92... | ab88f12f0a30b4601dc26dbae57646efb77d5c6382fb25522c529437e5428629 |
| 2.so | ca9ab48d293cc84092e8db8f0ca99cb155b30c61d32a1da7cd3687de454fe86c |
| Injection_API_executable_e | d465637518024262c063f4a82d799a4e40ff3381014972f24ea18bc23c3b27ee |
| Injection_API_log_generating_s | e03dc5f1447f243cf1f305c58d95000ef4e7dbcc5c4e91154daa5acd83fea9a8 |
| inject_api | f3e521996c85c0cdb2bfb3a0fd91eb03e25ba6feef2ba3a1da844f1b17278dd2 |

# Capitolo 2

# Analisi dell'attacco di Process Injection

## 2.1 Introduzione

In questo capitolo analizzeremo dettagliatamente la tecnica di attacco usata dai cyber-criminali per alterare a loro vantaggio il corretto funzionamento dei server bancari presso le quali erano in esecuzione le applicazioni di *payment switch*.

### 2.1.1 Definizione della forma di attacco

I rapporti pubblicati dalla NCCIC[1] [2] e dalla Symantec[3] indicano che la forma di attacco adottata per compromettere i server dell'istituto bancario fosse stata una **process injection**.

Con la locuzione *process injection* si intende una tecnica che rende possibile *l'esecuzione di codice arbitrario precedentemente introdotto all'interno dello spazio d'indirizzamento di un processo distinto in esecuzione.*[4]

L'esecuzione di codice maligno nel contesto di un processo legittimo, oltre a garantire ai cyber-criminali l'accesso a tutte le risorse assegnate al suddetto processo da parte del SO (memoria, risorse di rete, dati ecc.), non viene generalmente individuata dai prodotti commerciali per la sicurezza informatica, essendo l'esecuzione del malware *nascosta*.[5]

### 2.1.2 Descrizione della variante di attacco adottata

Esistono molte varianti di attacco di process injection che, sfruttando diverse tipologie di vulnerabilità esposte dal sistema operativo, sono in grado di introdurre con successo codice arbitrario all'interno di un processo; la variante adottata dai cyber-criminali nel malware FASTCash è conosciuta come **SIR**, acronimo di **Suspend-Inject-Resume**.

[1] `https://www.us-cert.gov/ncas/analysis-reports/AR18-275A`
[2] `https://www.us-cert.gov/ncas/alerts/TA18-275A`
[3] `https://www.symantec.com/blogs/threat-intelligence/fastcash-lazarus-atm-malware`
[4] `https://attack.mitre.org/techniques/T1055/`
[5] *Ibid.*

Come facilmente intuibile dal nome, tale tipologia di attacco prevede 3 fasi:[6]

1. La **sospensione del processo** bersaglio o, più specificatamente, di tutti i suoi thread.
2. **Alterazione dello stato del processo** attraverso la modifica del suo spazio di indirizzamento (*Address Space*) o dei valori contenuti nel suo PCB, come il valore che detiene l'indirizzo della successiva istruzione (*Program Counter/Instruction Pointer*) o i dati contenuti nei registri.
3. Il **riavvio del processo** attaccato in modo tale che esegua il codice maligno precedentemente introdotto.

### 2.1.3 Gli strumenti usati per l'analisi

Prima di procedere con la descrizione dettagliata dell'attacco perpetrato dal malware FASTCash, riportiamo di seguito i vari tool utilizzati durante le nostre analisi:

`strings` [7] Usato per l'estrazione di tutte le stringhe stampabili contenuti in un file.

`stat` [8] Utilizzato per ottenere alcune informazioni di base dei file tra cui nome, dimensione, data di ultima modifica, ecc.

`file` [9] Usato per determinare la tipologia di appartenenza di uno specifico file.

`onlinedisassembler` [10] Il de-assemblaggio dei file è stato eseguito utilizzando il servizio cloud **onlinedisassembler** che ci ha permesso di ricavare facilmente i listati di codice assembly dei file scritti per le architetture PowerPC™.

[6] `https://www.endgame.com/blog/technical-blog/ten-process-injection-techniques-technical-survey-common-and-tre`
[7] Cfr. `https://linux.die.net/man/1/strings`
[8] Cfr. `https://linux.die.net/man/1/stat`
[9] Cfr. `https://linux.die.net/man/1/file`
[10] Cfr. `https://onlinedisassembler.com/`

## 2.2 I file di FASTCash responsabili dell'attacco

In questa sezione analizzeremo i file del malware FASTCash responsabili dell'attacco di process injection contro l'istituto bancario cercando di comprenderne il funzionamento.

Nel corso di questo capitolo analizzeremo nel dettaglio i seguenti file, i cui sample sono stati ottenuti mediante download dal database di *Hybrid-Analysis*:[11]

`Injection_API_executable_e` Tale file contiene l'*injection tool*.

`Injection_API_log_generating_script` Un file di log.

`2.so` Questo file, insieme a quelli denominati dalla NCCIC come `Lost_File1_-so_file` e `Lost_File.so`, rappresenta una *shared library* contenente i metodi usati per manomettere le transazioni finanziarie ed invocati dal codice maligno introdotto durante la process injection. Non partecipa direttamente alla process injection, ma le funzioni esportate dalle suddette librerie vengono invocate direttamente dal codice introdotto durante l'attacco.

### 2.2.1 Il file `Injection_API_executable_e`

Cerchiamo ora di dimostrare come il file `Injection_API_executable_e` sia quello responsabile dell'attacco di process injection attraverso gli strumenti citati nell'introduzione.

L'output ottenuto dal tool `file` indica che il suddetto file, di cui abbiamo riportato alcuni dettagli nella tabella 2.1, è un **eseguibile** di tipo **eXtended COFF** (**XCOFF**), ovvero una versione migliorata ed estesa del formato **Common Object File Format** (**COFF**), il formato standard per la definizione dei file a livello strutturale nei sistemi operativi UNIX[12] fino al 1999[13], anno della definitiva adozione dello standard **Executable and Linkable Format** o **ELF**.

Il formato XCOFF è uno standard proprietario sviluppato da IBM[14] ed adottato nei sistemi operativi **Advanced Interactive eXecutive** o **AIX**, una famiglia di sistemi operativi proprietari basati su Unix sviluppati dalla stessa software house.[15]

Come confermato anche dalla stampa internazionale, da queste informazioni possiamo affermare senza ombra di dubbio che il sistema operativo in esecuzione sui server bancari sia stato proprio un AIX; ma cosa possiamo dire della versione?

#### 2.2.1.1 Analisi delle stringhe

Per rispondere alla precedente domanda dobbiamo analizzare il malware studiando alcune delle stringhe estratte ricorrendo al tool `strings`.

Osservando innanzitutto il formato della directory di installazione predefinita delle librerie del compilatore **GCC** nei sistemi operativi AIX, riportato per

[11] `https://www.hybrid-analysis.com/`
[12] Cfr. `https://it.wikipedia.org/wiki/COFF`
[13] Cfr. `https://en.wikipedia.org/wiki/Executable_and_Linkable_Format`
[14] Cfr. `https://www.ibm.com/support/knowledgecenter/ssw_aix_72/com.ibm.aix.files/XCOFF.htm`
[15] Cfr. `https://www.ibm.com/it-infrastructure/power/os/aix`

Tabella 2.1: Dettagli del file `Injection_API_executable_e`

| **Descrizione** | **Valore** |
|---|---|
| Nome | `Injection_API_executable_e` |
| Dimensione (*byte*) | `89088` |
| Data ultima modifca | `2018-11-09 11:08:40.000000000 +0100` |
| Tipo di file | `64-bit XCOFF executable or object module` |
| MD5 digest | `b3efec620885e6cf5b60f72e66d908a9` |
| SHA1 digest | `274b0bccb1bfc2731d86782de7babdeece379cf4` |
| SHA256 digest | `d465637518024262c063f4a82d799a4e40ff3381014972f24ea18bc23c3b27ee` |
| SHA512 digest | `a36dab1a1bc194b8acc220b23a6e36438d43fc7ac06840daa3d010fddcd9c316 8a6bf314ee13b58163967ab97a91224bfc6ba482466a9515de537d5d1fa6c5f9` |

comodità nel listato 2.1[16], possiamo facilmente conoscere dalle stringhe estratte mostrate nel listato 2.1 sia la versione di GCC che quella del sistema operativo AIX utilizzati per eseguire la *build* del malware, le quali risultano essere pari a 4.8.5[17] e 7.1[18] rispettivamente. Dallo stesso listato si può apprendere inoltre l'architettura del sistema: la PowerPC™.

Listing 2.1: Formato della directory di installazione predefinita delle librerie GCC nei sistemi operativi AIX

```
/opt/freeware/lib/gcc/<architecture_AIX_level>/<GCC_Level>
```

Listing 2.2: Stringhe estratte dal file `Injection_API_executable_e` (1)

```
348 /opt/freeware/lib/gcc/powerpc-ibm-aix7.1.0.0/4.8.5/ppc64:/
    opt/freeware/lib/gcc/powerpc-ibm-aix7.1.0.0/4.8.5:/opt/
    freeware/lib/gcc/powerpc-ibm-aix7.1.0.0/4.8.5/../../..:/
    usr/lib:/lib
```

Benché non sono state trovate in rete dati ufficiali a riguardo, dal momento che il malware stesso è stato compilato per la versione 7.1 di AIX, possiamo presupporre che la versione del sistema operativo attaccato fosse *almeno* pari a tale versione.

Sfortunatamente non è stato possibile risalire alla versione degli aggiornamenti, identificati dalla stessa IBM con il nome di *Technology Levels* (TLs) [19], installati sul sistema operativo bersaglio al momento dell'attacco, pertanto non possiamo escludere lo sfruttamento di una qualche vulnerabilità nota da parte degli attaccanti.

Ciò che è veramente importante ricordare è che il supporto ufficiale da parte di IBM nei confronti della versione 7.1 di AIX TL0, sostituita dalla ben più moderna versione 7.2 rilasciata nel dicembre 2015, è stato già terminato nel novembre 2013, benché la versione 7.1 TL5 riceverà ancora aggiornamenti fino ad aprile 2022.[20]

[16] Cfr: `http://www.perzl.org/aix/index.php%3Fn%3DMain.GCCBinariesVersionNeutral`
[17] Ulteriori dettagli su: `https://gcc.gnu.org/gcc-4.8/`
[18] Ulteriori dettagli su: `https://www-01.ibm.com/support/docview.wss?uid=isg3T1012517`
[19] Cfr: `http://ibmsystemsmag.com/aix/tipstechniques/migration/oslevel_versions/`
[20] Cfr: `https://www-01.ibm.com/support/docview.wss?uid=isg3T1012517`

Listing 2.3: Stringhe estratte dal file `Injection_API_executable_e` (2)

```
945 IBM XL C for AIX, Version 11.1.0.1
```

Il particolare mostrato nel listato 2.3 dimostra l'uso da parte degli attaccanti del software **XL C/C++ for AIX** versione 11.1.0.1, un compilatore C/C++ appositamente ottimizzato dalla IBM per i propri sistemi operativi[21], confermando, insieme ai numerosi riferimenti alle ben note librerie standard di C, come il C/C++ sia stato il linguaggio di programmazione scelto per implementare il malware.

#### 2.2.1.1.1 Le attività di logging di FASTCash

Osservando ora il frammento mostrato nel listato 2.4, è possibile notare un insieme di stringhe, aventi formato `[FUNCTION_NAME]` *`info`*, le quali, come avremo modo di notare anche durante l'analisi del codice assembly, fanno parte certamente di un **meccanismo di logging** sfruttato dagli attaccanti; tale aspetto è stato confermato dalla già citata analisi della NCCIC ed è noto anche il file di log stesso, di cui abbiamo riportato alcuni dettagli nella sezione 2.2.2.

Con ogni probabilità, le suddette stampe sono state realizzate per mezzo della funzione della libreria standard `snprintf`, come dimostrato dal codice assembly e dai numerosi riferimenti alla suddetta funzione presenti nel file. E' interessante notare come molte delle stampe coinvolgano numeri interi senza segno in forma esadecimale, come dimostrato dall'uso dei *conversion specifier* (le speciali sequenze di caratteri usati abitualmente nella definizione del formato di output nelle funzioni `printf`) nella forma `%llX`[22].

Queste stampe di log coinvolgono gran parte delle funzioni implementate nel file e presumibilmente sono state utilizzate dagli attaccanti per motivi di debug e racconta di informazioni arricchite anche da indicazioni temporali, come dimostrano l'uso delle funzioni `gettimeofday` e `localtime`.

Inoltre, la presenza di una procedura denominata `out_log`, analizzata in dettaglio in 2.2.3, dimostra come le suddette stampe vengano scritte in memoria di massa.

Infine le righe 333, 334 e 335 del listato 2.4 indicano che il malware sia stato implementato sotto forma di una **command-line utility interattiva**; tale supposizione è stata confermata anche dall'analisi NCCIC.

Listing 2.4: Stringhe estratte dal file `Injection_API_executable_e` (3)

```
320 ...
321 [main] Inject Start
322 [main] SAVE REGISTRY
323 [main] proc_readmemory fail
324 [main] toc=%llX
325 [main] path::%s
326 [main] data(%p)::%s
327 [main] Exec func(%llX) OK
328 [main] Exec func(%llX) fail ret=%X
329 [main] Inject OK(%llX)
330 [main] Inject fail ret=%llX
331 [main] Eject OK
```

[21] Cfr. `https://www.ibm.com/it-it/marketplace/xl-cpp-aix-compiler-power`
[22] Cfr. `http://man7.org/linux/man-pages/man3/printf.3.html`

```
[main] Eject fail ret=%llX
Usage: injection pid dll_path mode [handle func toc]
       mode = 0 => Injection
       mode = 1 => Ejection
[main] handle=%llX, func=%llX, toc=%llX
[main] ERROR::g_pid(%X) <= 0
[main] ERROR::load_config fail
[main] ERROR::eject & argc != 7
[main] ERROR::g_dl_handle(%llX) <= 0
[main] WARNING::func_addr(%llX), toc_addr(%llX)
...
```

#### 2.2.1.1.2 La gestione dei processi in AIX

Prima di analizzare nel dettaglio l'attacco vero e proprio, è indispensabile dapprima comprendere come vengono rappresentati e gestiti i **processi** nei sistemi operativi AIX.

Sia dato un processo con identificatore pari a `pid`. Ogni particolare aspetto di tale processo, come, ad esempio, il suo stato, i sui livelli di privilegio o il proprio spazio di indirizzamento, viene descritto da un insieme di file raccolti all'interno della directory `/proc/pid`.

Di questi file, alcuni dei quali riportati nella tabella 2.2[23], ricordiamo in particolare i seguenti:

**/proc/pid/as** Contiene l'immagine dello spazio degli indirizzi del processo e può essere aperto sia per la lettura che per la scrittura e supporta la subroutine `lseek` per accedere all'indirizzo virtuale di interesse.[24]

**/proc/pid/ctl** Un file di sola scrittura attraverso cui è possibile modificare lo stato del processo e alterare dunque il suo comportamento. La scrittura avviene per mezzo di opportuni **messaggi** scritti direttamente sul file con effetti immediati.[25]

**/proc/pid/status** Contiene informazioni sullo stato del processo. [26]

Un aspetto molto importante da considerare è che, in virtù di tale sistema di gestione dei processi, è possibile:

1. Conoscere i `pid` di **tutti** i processi del sistema attraverso il listing nella directory `/proc`.
2. Accedere alle informazioni di un dato processo attraverso semplici operazioni di lettura e scrittura sui suddetti file, utilizzando ad esempio le *system call* standard come `open()`, `close()`, `read()` e `write()`. [27]

[23] La lista completa è disponibile in *ivi* pag. 246
[24] *Cfr. ivi* pag. 232
[25] *Cfr. ivi* pag. 232
[26] *Cfr. ivi* pag. 232
[27] Cfr. IBM - *AIX Version 7.1: Files References* - pag. 232-246

Tabella 2.2: Sottoinsieme dei file contenuti in `/proc/pid`

| File | Descrizione |
|---|---|
| `/proc/pid/status` | *Status of process* `pid` |
| `/proc/pid/ctl` | *Control file for process* `pid` |
| `/proc/pid/as` | *Address space of process* `pid` |
| `/proc/pid/cred` | *Credentials information for process* `pid` |
| `/proc/pid/sigact` | *Signal actions for process* `pid` |
| `/proc/pid/sysent` | *System call information for process* `pid` |

#### 2.2.1.1.3 L'accesso del malware ai file dei processi

Ora che abbiamo compreso alcune delle caratteristiche più elementari riguardo la gestione dei processi nei sistemi operativi AIX è molto più facile comprendere l'attacco compiuto da FASTCash.

Infatti, come mostrato nel listato 2.5, sono state individuate all'interno del malware tre stringhe che fanno riferimento ai sopracitati file descrittori di processo ed, in particolare, ai file `ctl`, `status` e `as`.

Come dimostrato dall'analisi della NCCIC e dalla nostra, la presenza del *conversion specifier* `%d` indica che il malware, dopo aver individuato l'identificatore del processo bersaglio, ricostruisca, per mezzo della funzione `sprintf`, i percorsi completi verso i suddetti file per poi ispezionare e manipolarne il contenuto.

Listing 2.5: Stringhe estratte dal file `Injection_API_executable_e` (4)

```
320 /proc/%d/ctl
321 /proc/%d/status
322 /proc/%d/as
```

A questo punto è legittimo chiedersi cosa può essere effettivamente scritto all'interno dei suddetti file.

La documentazione ufficiale rilasciata dalla IBM riporta l'esistenza di un insieme di **messaggi strutturati**[28], ognuno dei quali identificato da un codice operativo, rappresentato da un valore `int`, e da una serie di argomenti (se presenti)[29]. Un aspetto importante che bisogna tenere in mente è che questi messaggi possono essere scritti direttamente nel file `ctl` di un dato processo al fine di alterare lo stato del processo.

Osservando il listato 2.6, notiamo una stampa del logger all'interno è presente la stringa in cui compare la parola **PCWSTOP**; PCWSTOP è il nome di un messaggio definito nei sistemi operativi AIX che viene usato per sospendere l'esecuzione di un processo il cui `pid` viene passato come argomento.[30] I risultati della NCCIC e le nostre analisi sul codice assembly indicano che il malware usi questo ed altri messaggi per interrompere dapprima il processo bersaglio, accedere al suo spazio di indirizzamento, effettuare la code injection per poi riavviare il processo affinché esegua effettivamente il codice malevolo.

[28] La documentazione IBM usa in modo intercambiabile il termine *messaggio* e quello di *segnale*

[29] *Cfr. ivi* pag. 242

[30] Cfr. *Ibidem*

Listing 2.6: Stringhe estratte dal file `Injection_API_executable_e` (5)

```
...
[proc_wait] PCWSTOP pid=%d, ret=%d, err=%d(%s)
[proc_wait] tid=%d, why=%d, what=%d, flag=%d, sig=%d
...
```

Gli altri tipi di messaggi usati dagli attaccanti sono visibili nel listato 2.7 tra cui spiccano per importanza:

**PCSET** Serve per passare una serie di flag ad un processo (PR_ASYNC, PR_FORK, PR_KLC ecc.) per modificarne lo stato.[31]

**PCRUN** Riesegue un thread dopo essere stato arrestato.

**PCSENTRY** Il thread corrente viene interrotto nel momento in cui richiama una specifica system call.

**PCSFAULT** Definisce un insieme di *hardware faults* "tracciabili" nel processo. Il thread si interrompe quando si verifica una fault.[32]

Listing 2.7: Stringhe estratte dal file `Injection_API_executable_e` (6)

```
...
[proc_attach] PCSET pid=%d, ret=%d, err=%d(%s)
[proc_attach] PCSTOP pid=%d, ret=%d, err=%d(%s)
[proc_attach] PCSTRACE pid=%d, ret=%d, err=%d(%s)
[proc_attach] PCSFAULT pid=%d, ret=%d, err=%d(%s)
[proc_attach] PCSENTRY pid=%d, ret=%d, err=%d(%s)
[proc_detach] PCSTRACE pid=%d, ret=%d, err=%d(%s)
[proc_detach] PCSFAULT pid=%d, ret=%d, err=%d(%s)
[proc_detach] PCSENTRY pid=%d, ret=%d, err=%d(%s)
[proc_detach] PCRUN pid=%d, ret=%d, err=%d(%s)
...
```

Come dimostrano i log mostrati nei listati 2.8 e 2.9, il malware non si limita solo alla scrittura dei messaggi nei file di controllo dei processi ma raccoglie ed altera le informazioni presenti nei registri del processore, parte dei quali sono riportati nella tabella 2.3[33]

Sfortunatamente, non avendo la possibilità di eseguire il malware direttamente, ignoriamo quali dati siano stati prelevati e/o scritti attraverso i registri, benché è possibile fare una serie di supposizioni che riteniamo molto probabili.

Infatti riteniamo plausibile che sia stato alterato il registro `IAR` affinché, dopo il riavvio del processo bersaglio, venga eseguito il codice malevolo introdotto durante l'attacco.

Presumibilmente una serie di indirizzi relativi allo spazio d'indirizzamento del processo bersaglio sono stati ricavati dai registri al verificarsi di certe condizioni, come la chiamata ad una specifica syscall; ciò spiegherebbe l'uso, pare per ben due volte, del messaggio PCSENTRY e PCSFAULT.

[31] *Cfr. ivi* pag. 234

[32] *Cfr. ibidem*

[33] Cfr. *AIX Version 7.1: Assembler Language Reference* per una lista completa oppure visita `https://www.ibm.com/support/knowledgecenter/en/ssw_aix_71/com.ibm.aix.alangref/idalangref_arch_overview.htm` o `https://www.ibm.com/support/knowledgecenter/en/ssw_aix_71/com.ibm.aix.kdb/kdb_registers.htm`

Listing 2.8: Stringhe estratte dal file `Injection_API_executable_e` (7)

```
299 ...
300 [proc_getregs] GETREG pid=%d, ret=%d, err=%d(%s)
301 [proc_getregs] GETSTATUS pr_syscall=%d, pr_why=%d, pr_what=%
        d, pr_flags=%d, pr_cursig=%d
302 [proc_setregs] SETREG pid=%d, ret=%d, err=%d(%s)
303 ...
```

Listing 2.9: Stringhe estratte dal file `Injection_API_executable_e` (8)

```
320 [out_regs] IAR=%llX
321 [out_regs] MSR=%llX
322 [out_regs] CR=%llX
323 [out_regs] LR=%llX
324 [out_regs] CTR=%llX
325 [out_regs] GPR%d=%llX
```

Tabella 2.3: Breve descrizione dei registri ispezionati dal malware

| Registro | Nome esteso | Descrizione |
|---|---|---|
| `LR` | Link Register | E' usato per ospitare l'indirizzo dell'istruzione successiva ad una operazione di salto. E' usata principalmente per ospitare l'indirizzo di ritorno al termine di una funzione. |
| `CR` | Condition Register | Un registro da 32 bit usato per specificare varie classi di operazioni. |
| `CTR` | Control Register | Un registro da 32 bit usato per specificare varie classi di operazioni. |
| `IAR` | Instruction Address Register | Usato per contenere l'indirizzo dell'istruzione successiva. |
| `MSR` | Machine State Register | Registro da 32 bit usato per specificare varie classi di operazioni. |
| `r0-r31` | General Purpose Registers (GPRs) from 0 through 31 | Registri per usi generici. |

Mostriamo infine nel listato 2.10 un log che dimostra come il malware dapprima accede ispezionando l'area di memoria riservata di un processo per poi alterarla eseguendo un'operazione di scrittura, completando in tal modo l'attacco di code injection che si conclude definitamente con il riavvio del processo attaccato.

Listing 2.10: Stringhe estratte dal file `Injection_API_executable_e` (9)

```
308 ...
309 [proc_readmemory] ret=%d, err=%d(%s), addr=%p, len=%d, data
        =%p
310 [proc_readmemory] (%X~%X) %02X %02X %02X %02X %02X %02X %02X
         %02X %02X %02X %02X %02X %02X %02X %02X %02X %02X
311 [proc_writememory] ret=%d, err=%d(%s), addr=%p, len=%d, data
        =%p
312 ...
```

#### 2.2.1.2 Analisi del codice assembly

Segue ora una breve analisi di alcune linee codice assembler estratte dal malware allo scopo di confermare alcuni aspetti finora.

Tabella 2.4: Alcune istruzioni assembly disponibili nell'architettura PowerPC™

| Istruzione | Nome | Argomenti | Descrizione |
|---|---|---|---|
| `bl` | *Branch Link* | *`target_address`* | *Branches to a specified target address.* |
| `mfcr` | *Move From Condition Register* | `RT` | *Copies the contents of the Condition Register into a general-purpose register.* |
| `std` | *STore Doubleword* | `RS,`*`Offset`*`,RSML` | *Store a doubleword of data from a general purpose register into a specified memory location.* |
| `stw` | *STore Word* | `RS,`*`Offset`*`,RSML` | *Stores a word of data from a general-purpose register into a specified location in memory.* |
| `li` | *Load Immediate* | `RT,`*`Value`* | *Copies specified value into a general-purpose register.* |
| `ld` | *Load Doubleword* | `RT,`*`Offset`*`,RS` | *Load a doubleword of data into the specified general purpose register.* |
| `mr` | *Move Register* | `RT,RS` | *Copies the contents of one register into another register.* |
| `addi` | *ADD Immediate* | `RT,RS,`*`Value`* | *Place the sum of the contents of RA and the 16-bit two's complement integer value, sign-extended to 32 bits, into the target RT.* |
| `mtrl` | *Move To Link Register* | `RS` | *Copies the contents of RS register into Link Register.* |
| `extsw` | *Extend Sign Word* | `RT,RS` | *Copy the low-order 32 bits of a general purpose register into another general purpose register, and signextend the fullword to a doubleword in size (64 bits).* |

##### 2.2.1.2.1 La procedura `main`

Essendo `Injection_API_executable_e` un file eseguibile, è naturale cominciare dall'analisi della procedura `main`.

Nelle primissime linee di codice della procedura sono state trovate varie istruzioni di salto incondizionato verso la funzione `atoi` (riga 7015 e 7041), probabilmente utilizzata, come confermato anche dalla NCCIC, per gestire gli input ricevuti da terminale, dimostrando così la natura interattiva del file malevolo.

Successivamente il malware intraprende varie azioni, di cui ignoriamo le finalità, per la manipolazione di stringhe, come dimostrano la serie di istruzioni di salto condizionato verso le funzioni `strlen` (riga 7028), `strncpy` (riga 7035) e `strtoull` (riga 6973, 6984 e 6995).

Infine, dopo una serie di istruzioni di salto verso procedure che riteniamo avere scopo di inizializzazione (`load_config`, `get_func_addr`), viene raggiunta

la porzione di codice mostrata nel listato **??** in cui viene eseguita una istruzione di salto verso una procedura denominata `inject` che, come vedremo, contiene il codice operativo per l'esecuzione della process injection.

#### 2.2.1.2.2 La procedura `inject`

Tra tutte le procedure presenti all'interno del malware, quella contenente il codice per l'esecuzione dell'attacco di process injection è stata denominata `inject`.

Osservando il frammento di codice mostrato nel listato 2.11, possiamo distinguere le seguenti azioni preliminari compiute dal malware:

1. Copia dell'indirizzo di ritorno della procedura chiamante, presente all'interno dal registro `r0`, all'interno del link register attraverso l'uso dell'istruzione `mflr`.

2. Inizializzazione di una serie di registri e aree di memoria ricorrendo alle istruzioni `std` e `mr` contenenti con ogni probabilità indirizzi di memoria da passare come argomento alla procedura `memset` chiamata successivamente.

3. Alterazione dello spazio d'indirizzamento del processo bersaglio attraverso numerose chiamate alla procedura `memset`; le funzioni `memset` qui invocate, come avremmo modo di intuire successivamente, hanno il solo compito di inizializzare aree di memoria a preparazione della successiva copia del codice malevolo.

Listing 2.11: Codice assembly estratto dal file `Injection_API_executable_e` (1)

```
308 mflr    r0
309 std     r0,16(r1)
310 std     r31,-8(r1)
311 stdu    r1,-1520(r1)
312 mr      r31,r1
313 std     r3,1568(r31)
314 mr      r9,r4
315 std     r5,1584(r31)
316 stw     r9,1576(r31)
317 li      r9,0
318 stw     r9,120(r31)
319 li      r9,0
320 std     r9,144(r31)
321 addi    r10,r31,152
322 li      r9,384
323 mr      r3,r10
324 li      r4,0
325 mr      r5,r9
326 bl      0x10002f00 <.memset>
327 nop
328 addi    r10,r31,536
329 li      r9,384
330 mr      r3,r10
331 li      r4,0
```

```
332 mr      r5,r9
333 bl      0x10002f00 <.memset>
334 nop
335 addi    r10,r31,920
336 li      r9,256
337 mr      r3,r10
338 li      r4,0
339 mr      r5,r9
340 bl      0x10002f00 <.memset>
```

Successivamente, osservando il frammento riportato in 2.12, possiamo innanzitutto osservare l'intensa attività di log eseguita dal malware durante la sua attività; vengono infatti eseguite molte istruzioni `bl` per invocare due procedure denominate `out_log` e `out_regs`, invocate dopo ogni passo dell'attacco. Osservando il codice assembler, è facile convincersi come la procedura `out_log` venga invocata per effettuare la scrittura di una serie informazioni di interesse su un file esterno mentre `out_regs` per scrivere nel log il contenuto dei registri.

Successivamente viene intrapreso l'attacco vero e proprio. In accordo a quanto detto nell'introduzione a proposito della forma di attacco di process injection di tipo SIR, il processo bersaglio viene sospeso ed, a tale scopo, viene invocato la procedura `proc_wait`; dopo l'arresto avviene la raccolta dei valori dei registri attraverso l'invocazione delle procedure `proc_getregs` ed `out_regs`.

Listing 2.12: Codice assembly estratto dal file `Injection_API_executable_e` (2)

```
312 li      r3,0
313 li      r4,0
314 bl      0x10001b44 <.proc_wait>
315 ld      r3,728(r2)
316 bl      0x10000674 <.out_log>
317 addi    r9,r31,152
318 mr      r3,r9
319 bl      0x10001ee4 <.proc_getregs>
320 addi    r9,r31,152
321 mr      r3,r9
322 bl      0x10000c80 <.out_regs>
```

Quindi, dopo aver raccolto altre informazioni attraverso la procedura `proc_readmemory`, come mostrato nel listato 2.13, viene introdotto il codice malevolo nello spazio di indirizzamento del processo attraverso l'invocazione della procedura `proc_writememory`.

Successivamente vengono eseguite azioni per alterare il contenuto dei registri, probabilmente per fare in modo che il processo bersaglio esegua il codice malevolo appena introdotto attraverso l'esecuzione di un procedura denominata `proc_setregs`, concludendo in tal modo l'attacco.

Listing 2.13: Codice assembly estratto dal file `Injection_API_executable_e` (3)

```
308 bl      0x10002460 <.proc_writememory>
309 addi    r9,r31,536
310 mr      r3,r9
311 bl      0x10000c80 <.out_regs>
312 addi    r9,r31,536
```

```
313 mr      r3,r9
314 bl      0x10002068 <.proc_setregs>
315 li      r3,3
316 bl      0x10001a28 <.proc_continue>
317 li      r3,6
318 li      r4,11
319 bl      0x10001b44 <.proc_wait>
320 addi    r9,r31,536
321 mr      r3,r9
322 bl      0x10001ee4 <.proc_getregs>
323 addi    r9,r31,536
324 mr      r3,r9
325 bl      0x10000c80 <.out_regs>
```

### 2.2.2 Il file `Injection_API_log_generating_script`

Il file denominato `Injection_API_log_generating_script`, secondo i rapporti dalla NCCIC[34], rappresenta il file di log generato automaticamente durante l'esecuzione dell'injection tool `Inject_API_executable_e` analizzato nella precedente sezione.

Il contenuto del file, di cui omettiamo un'analisi specifica in quanto di scarso interesse ai fini della nostra trattazione, è ovviamente pari a quanto descritto durante l'analisi dell'injection tool.

Tabella 2.5: Dettagli del file `Injection_API_log_generating_script`

| **Descrizione** | **Valore** |
|---|---|
| Nome | `Injection_API_log_generating_script` |
| Dimensione (*byte*) | 2337 |
| Tipo di file | `ASCII text` |
| MD5 digest | `844eec0ff86c10f5f9b41648b066563b` |
| SHA1 digest | `5d0fd2c5f58dcbc51e210894e8698bc14ccd30e2` |
| SHA256 digest | `e03dc5f1447f243cf1f305c58d95000ef4e7dbcc5c4e91154daa5acd83fea9a8` |
| SHA512 digest | `199dee05b602039e480f62963cb0ec3b96393e37bb78ff1475e6dfc5857e4849` `24a476dbe73f02de96670ff488eb26f53ca9c600dd44390cf767a4aa510869a4` |

[34] https://www.us-cert.gov/ncas/analysis-reports/AR18-275A

### 2.2.3 Il file `2.so`

Come suggerisce la sua l'estensione, il file `2.so` rappresenta una **shared library** usata per esportare funzioni in grado di interagire con i messaggi basati su protocollo **ISO 8583**, con lo scopo di alterare le transazioni finanziare a proprio favore.

Tabella 2.6: Dettagli del file `2.s0`

| **Descrizione** | **Valore** |
|---|---|
| Nome | `2.so` |
| Dimensione (*byte*) | `110592` |
| Data ultima modifca | `2018-11-09 11:08:40.000000000 +0100` |
| Tipo di file | `64-bit XCOFF executable or object module` |
| MD5 digest | `b66be2f7c046205b01453951c161e6cc` |
| SHA1 digest | `ec5784548ffb33055d224c184ab2393f47566c7a` |
| SHA256 digest | `ca9ab48d293cc84092e8db8f0ca99cb155b30c61d32a1da7cd3687de454fe86c` |
| SHA512 digest | `6890dcce36a87b4bb2d71e177f10ba27f517d1a53ab02500296f9b3aac021810 7ced483d70d757a54a5f7489106efa1c1830ef12c93a7f6f240f112c3e90efb5` |

#### 2.2.3.1 Analisi delle stringhe

Listing 2.14: Stringe estratte dal file `2.so` (1)

```
...
DL_ISO8583_MSG_Init
DL_ISO8583_MSG_Free
DL_ISO8583_MSG_SetField_Str
DL_ISO8583_MSG_SetField_Bin
DL_ISO8583_MSG_RemoveField
DL_ISO8583_MSG_HaveField
DL_ISO8583_MSG_GetField_Str
DL_ISO8583_MSG_GetField_Bin
DL_ISO8583_MSG_Pack
DL_ISO8583_MSG_Unpack
DL_ISO8583_MSG_Dump
DL_ISO8583_MSG_AllocField
DL_ISO8583_COMMON_SetHandler
DL_ISO8583_DEFS_1987_GetHandler
DL_ISO8583_DEFS_1993_GetHandler
DL_ISO8583_FIELD_Pack
DL_ISO8583_FIELD_Unpack
...
GenerateRandAmount
GenerateResponseTransaction1
GenerateResponseTransaction2
GenerateResponseInquiry1
Crypt
...
```

Osservando l'output ottenuto usando il tool `strings` di cui ne riportiamo un piccolo frammento in 2.14, è facile convincersi che la maggior parte delle funzioni esportate, di cui è abbastanza facile intuirne lo scopo, riguardano il protocollo ISO 8583.

I metodi esportati non si limitano ovviamente alla sola gestione dei messaggi ISO 8583 sebbene siano la maggioranza; esistono metodi per la gestione di tabelle hash, di generazione di numeri casuali, metodi per gestire le risposte ricevute dai sistemi bancari durante le transazioni e molto altro ancora.

Listing 2.15: Stringhe estratte dal file `2.so` (2)

```
545 Blocked Message(msg=%04x, term=%02x, pcode=%06x, pan=%s)
546 Passed Message(msg=%04x, term=%02x, pcode=%06x, pan=%s)
547 [recv] ret=%d
548 send ret = %d, err = %d
549 /tmp/.ICE-unix/context.dat
550 /tmp/.ICE-unix/tmp%d_%d.log
551 [%04d-%02d-%02d %02d:%02d:%02d][PID:%4u][TID:%4u] %s
552 /tmp/.ICE-unix/config_%d
553 /tmp/.ICE-unix/tmprd%d_%d.log
554 /tmp/.ICE-unix/tmpwt%d_%d.log
555 [DetourInitFunc] dlopen error(%s)
556 [DetourInitFunc] org_func(%p) %02X %02X %02X %02X %02X %02X
        %02X %02X %02X %02X %02X %02X
557 [DetourInitFunc] new_func(%p) %02X %02X %02X %02X %02X %02X
        %02X %02X %02X %02X %02X %02X
558 [DetourInitFunc] dlsym error(%s)
559 Success
560 Failed
561 DetourInitFunc(%s, %s) %s
562 [DetourInitFunc] org_func=%p new_func=%p
563 [DetourAttach] hook_func_addr=%p, new_func_addr=%p
564 [DetourAttach] after mmap=%p
565 [DetourAttach] copy_func(%x) %02X %02X %02X %02X %02X %02X
        %02X %02X %02X %02X %02X %02X
566 [DetourAttach] hook_func_addr(%x) %02X %02X %02X %02X %02X
        %02X %02X %02X %02X %02X %02X %02X
567 [DetourDetach] hook_func_addr(%x) %02X %02X %02X %02X %02X
        %02X %02X %02X %02X %02X %02X %02X
```

Osserviamo ora l'insieme di stringhe riportate in 2.15

Analogamente al file `Injection_API_executable_e`, la libreria esegue una notevole attività di log delle proprie attività; come avremmo modo di notare successivamente nella sezione 2.2.3.2.2, i dati di log vengono scritti direttamente in memoria di massa attraverso la procedura `out_dump_log`.

Dal listato possiamo osservare un'interessante riferimento al percorso `/tmp/.ICE-unix/`: in accordo alla documentazione relativa alla versione R6.8.2 di X11[35] (la famosa implementazione del X Window System[36]) il suddetto percorso viene usato per ospitare una serie di `socket` sfruttate dal protocollo **Inter-Client Exchange** (**ICE**), utilizzato per la risoluzione di varie problematiche come quelle legate

[35] Cfr. `https://www.x.org/releases/X11R6.8.2/doc/RELNOTES5.html`
[36] Cfr. `https://www.x.org/wiki/`

all'autenticazione o al *byte order negotiation*[37]. Pertanto, sebbene ne ignoriamo le motivazioni, abbiamo motivo di ritenere che gli attaccanti abbiano interagito con la GUI session manager di X11 attraverso il protocollo ICE per svolgere le loro attività.

#### 2.2.3.2 Analisi del codice assembly

Benché naturalmente sprovvista di una procedura `main` trattandosi di una libreria, il file `2.so` assume un ruolo centrale per il corretto svolgimento dell'attacco in quanto, come già detto, esporta tutte le procedure necessarie per manipolare le transazioni elettroniche dei sistemi finanziari.

Come già detto i metodi esportati dalla libreria sono molto numerosi e vari, comprendendo, ad esempio, procedure di ispezione dei campi dei messaggi (`DL_ISO8583_MSG_GetField_Bin`,`DL_ISO8583_MSG_GetField_Str`) o manipolazione (`DL_ISO8583_MSG_SetField_Bin`, `DL_ISO8583_MSG_RemoveField`).

##### 2.2.3.2.1 `DL_ISO8583_MSG_SetField_Bin`

La procedura denominata `DL_ISO8583_MSG_SetField_Bin` sia stata implementata dagli attaccanti poter alterare il contenuto di un particolare campo dei messaggi ISO 8583 scambiati fra i sistemi bancari.

Come mostrato nel frammento riportato nel listato 2.16, la manipolazione del messaggio incomincia con una fase di ispezione finalizzata alla ricerca dell'indirizzo di memoria corrispondente al campo del messaggio che s'intende modificare.

Come si può notare, l'ispezione del messaggio avviene per mezzo di un ciclo come dimostrano la presenza di diverse istruzioni di salto, come `beq` (*Branch On Equal*) e `ble` (*branch if Less or Equal*), e di comparazione, come `cmpdi` (*Compare Doubleword Immediate*) o `cmplwi` (*Compare Logical Word Immediate*).

Una volta trovato l'indirizzo, l'operazione di scrittura vera e propria viene eseguita invocando dapprima la procedura `DL_ISO8583_MSG_AllocField`, attraverso la quale viene allocata sufficiente memoria per contenere il nuovo campo, e utilizzando successivamente la procedura `memmove`, usata per copiare i dati nel messaggio.

Listing 2.16: Codice assembly estratto dal file `2.so` (1)

```
347 cmplwi  cr7,r0,128
348 ble     cr7,0x10001b84
349 li      r0,1
350 std     r0,128(r31)
351 b       0x10001c08
352 lwz     r0,208(r31)
353 clrldi  r9,r0,32
354 lwz     r0,224(r31)
355 clrldi  r0,r0,32
356 addi    r11,r31,120
357 mr      r3,r9
358 mr      r4,r0
359 ld      r5,232(r31)
360 mr      r6,r11
```

[37] Cfr. `https://www.x.org/releases/X11R7.7/doc/libICE/ICElib.html#Overview_of_ICE`

```
361 bl      0x100026e0 <._DL_ISO8583_MSG_AllocField>
362 nop
363 mr      r0,r3
364 std     r0,112(r31)
365 ld      r0,112(r31)
366 cmpdi   cr7,r0,0
367 bne     cr7,0x10001c00
368 ld      r9,120(r31)
369 lwz     r0,224(r31)
370 clrldi  r0,r0,32
371 mr      r3,r9
372 ld      r4,216(r31)
373 mr      r5,r0
374 bl      0x1000034c <.memmove>
```

#### 2.2.3.2.2 out_dump_log

La procedura denominata `out_dump_log` è stata usata, come inequivocabilmente suggerisce il nome, per finalità legate alle attività di log del malware.

L'importanza della procedura risiede nella quantità delle invocazioni eseguite: ben 32 volte. Curiosamente l'invocazione di `out_dump_log` è quasi sempre preceduta da un'altra verso una funzione denominata `ReadRecv`; dal codice assembler è facile intuire che gli attaccanti ispezionavano il contenuto dei messaggi ricevuti per poi salvarne il contenuto nel file di log.

Il frammento di codice mostrato in 2.17 suggerisce che il file di log, creato ed aperto invocando la funzione `fopen`, veniva arricchito anche da informazioni temporali, come dimostrano le istruzioni di salto verso le procedure `gettimeofday` e `localtime`.

Listing 2.17: Codice assembly estratto dal file `2.so`

```
350 mr      r3,r0
351 ld      r4,856(r2)
352 bl      0x10000770 <.fopen>
353 ld      r2,40(r1)
354 mr      r0,r3
355 std     r0,152(r31)
356 ld      r0,152(r31)
357 cmpdi   cr7,r0,0
358 beq     cr7,0x1000a0ec
359 addi    r0,r31,4264
360 mr      r3,r0
361 li      r4,0
362 bl      0x10000798 <.gettimeofday>
363 ld      r2,40(r1)
364 addi    r0,r31,4264
365 mr      r3,r0
366 bl      0x100007c0 <.localtime>
```

# Capitolo 3

# Analisi degli impatti subiti

L'attacco perpetrato dal gruppo Lazarus ha incontestabilmente causato una serie di danni diretti ed indiretti nei confronti dell'istituto bancario, comportando impatti considerevoli a livello economico-finanziario, politico-sociale e di reputazione, aggravati non appena la notizia dell'avvenuto attacco è divenuta di dominio pubblico.

## 3.1 Impatti socio-economici

Essendo stati compromessi i processi governanti funzionalità critiche del sistema, l'attacco ha determinato innanzitutto un'**interruzione dei servizi** legittimi offerti dall'istituto bancario, come quelli aventi funzione di credito sulle quali si basano le transazioni finanziare, comportando conseguenze economico-sociali molto gravi tra cui:

- Danni economici diretti a danno dell'istituto per mancati introiti.
- Danni economici indiretti, difficilmente quantificabili, a danno del tessuto economico sociale ed, in particolare, alle varie attività economiche che usufruiscono quotidianamente dei servizi offerti dall'istituto; si pensi, ad esempio, alle transizioni finanziare indispensabili alle aziende per eseguire attività basilari come il pagamento delle forniture, degli stipendi dei dipendenti, le richieste di credito ecc.

L'introduzione di codice maligno all'interno dei sistema ha avuto molteplici conseguenze di notevole impatto economico quali:

- Furto di denaro a danno dei clienti dell'istituto verso i quali quest'ultima ha dovuto rispondere con operazioni di risarcimento.
- Danni economico-finanziari dovuti alle operazioni di ripristino del sistema, aggiornamento dei software e di tutti i meccanismi di sicurezza.

## 3.2 Impatti sulla reputazione

La diffusione della notizia riguardante l'avvenuto attacco attraverso vari canali di informazione ha indubbiamente causato un danno alla reputazione dell'istituzione bancaria per via degli scarsi sforzi rivolti alla sicurezza informatica,

esponendo a gravi rischi i propri clienti sia dal punto di vista economico che di privacy, benché, dalle analisi, non risulta che il malware FASTCash sia stato concepito come spyware.

I danni all'immagine dell'istituto avranno inevitabilmente effetti di lungo termine a causa dalla perdita degli attuali e dei futuri clienti.

## 3.3 Impatti sul sistema

A causa delle modalità di attacco intraprese dal malware, è facile rendersi conto come l'impatto subito dal sistema sia stato molto grave:

- I processi critici sono stati compromessi in modo irreversibile.
- La compromissione dei processi ha determinato un mutamento del comportamento del sistema non conforme alle specifiche.
- Avendo avuto accesso privilegiato al sistema, come dichiarato anche dalla NCCIC, è possibile, benché non sia stato confermato ufficialmente, l'accesso e la manipolazione di dati sensibili sia dei clienti che dell'istituto bancario.
- La riservatezza di tutte le transazioni, e quindi di tutti dati ad essi associati, è stata compromessa poiché gli attaccanti sono riusciti ad accedere sia in lettura che in scrittura ai messaggi ISO 8583 scambiati tra i sistemi.

# Capitolo 4

# Contromisure

In questo capitolo forniremo una analisi dettaglia delle possibili contromisure capaci di contrastare le attività del malware FASTCash sia modo pro-attivo che reattivo.

## 4.1 Aggiornamenti software

Come suggerito dalla totalità delle aziende di sicurezza informatica, al fine di contrastare in generale gli attacchi informatici, è indispensabile una **regolare attività di aggiornamento** di tutto il parco software.

I ricercatori della Symantec hanno stabilito[1] che il mancato aggiornamento del sistema operativo AIX utilizzato dai payment switch server abbia compromesso la sicurezza del sistema poiché privata del supporto IBM relativamente alle patch di sicurezza, le quali avrebbero potuto contrastare o, nel migliore delle ipotesi, impedire l'attacco informatico.

### 4.1.1 Analisi dell'efficacia degli aggiornamenti software

Sebbene una regolare attività di aggiornamento rappresenti un requisito imprescindibile per garantire standard di sicurezza elevati, riteniamo tale attività, in virtù delle caratteristiche tecniche del malware e della forma di attacco adottata, poco efficace contro FASTCash.

Riteniamo infatti che l'attacco effettuato da FASTCash sia molto difficile da rilevare e contrastare poiché basa il proprio funzionamento sull'(ab)uso dei servizi essenziali offerti dal kernel del sistema operativo, in particolare la gestione dei processi/thread e meccanismi di lettura e scrittura.

Non avendo sfruttato una vera e propria vulnerabilità del sistema operativo, come stabilito dai report della Symantec e della NCCIC, è giustificabile ritenere che l'aggiornamento dei software, intesa come unica forma di sicurezza, non avrebbe contrastato efficacemente il malware FASTCash.

Benché si potrebbe pensare di rilasciare aggiornamenti di sicurezza che impongano restrizioni sull'uso delle *syscall* per l'accesso ai servizi del SO, riteniamo che FASTCash continuerebbe ad agire incontrastato in quanto "nascosto" all'interno di un processo ritenuto legittimo; oltre ad essere una misura in generale

[1] `https://www.symantec.com/blogs/threat-intelligence/fastcash-lazarus-atm-malware`

poco efficace, politiche basate su restrizioni riguardanti l'uso delle *syscall* potrebbero comportare il verificarsi di effetti collaterali legati al tentativo da parte di processi legittimi di accedere ai servizi del SO, comportando costi aggiuntivi per lo sviluppo di un software compatibile con le nuove impostazioni.

## 4.2 Principio del privilegio minimo

Per contrastare FASTCash, riteniamo molto efficace concentrare gli sforzi nell'impedire l'avvio dell'attacco ai fin dal principio.

Le contromisure più efficaci devono innanzitutto basarsi sul cosiddetto **principle of least privilege** (**PoLP**), in italiano **principio del privilegio minimo**, in cui si stabilisce che *un opportuno sistema di sicurezza deve fornire un meccanismo che assicuri che ogni processo in esecuzione sul sistema sia in grado di accedere solo ed esclusivamente alle informazioni di cui necessità per garantire il suo corretto e legittimo funzionamento.*

### 4.2.1 Le liste di controllo degli accessi

Una possibile applicazione del principio del privilegio minimo si basa sull'uso delle **access control list** (**ACL**), in italiano **lista di controllo degli accessi**, ovvero opportune strutture dati, generalmente tabelle, contenenti informazioni che specifichino quali utenti o gruppi hanno l'autorizzazione ad accedere alle risorse del sistema come file o risorse di rete.

Poiché il file `Injection_API_executable_e`, contenente l'*injection tool* di FASTCash, richiede, per poter funzionare, l'accesso in lettura/scrittura al pseudo-file system `/proc`, l'introduzione di restrizioni sull'accesso a tale directory avrebbe potuto contrastare efficacemente FASTCash.

Teoricamente, configurando opportunamente il sistema rendendo non eseguibili i file binari presenti nelle partizioni più vulnerabili, utilizzando i flag `noexec` o `nosetuid` all'interno delle stringhe per i mount delle partizioni presenti nel file `/etc/fstab`, sarebbe stato possibile impedire a priori l'attacco rendendo impossibile l'avvio del processo malware.[2]

Ovviamente tale forma di sicurezza è priva di utilità qualora gli attaccanti riescano ad ottenere privilegi amministrativi attraverso tecniche di *privilege escalation* che è necessario contrastare attraverso regolari attività di aggiornamento e forme di autenticazione resistenti.

### 4.2.2 Meccanismi di Whitelisting

Oltre a regolare le autorizzazioni di accesso alle risorse, una contromisura ancora più efficace consiste nell'adottare meccanismi di **Whitelisting** che consentano l'accesso alle risorse del sistema *solo ed esclusivamente* a processi attendibili.

I software di whitelisting basano il proprio funzionamento sulla creazione preliminare di una lista, denominata *whitelist*, contenente gli identificati, solitamente stringhe hash, di tutti i file eseguibili autorizzati ad accedere a determinate risorse del sistema. Utilizzando un approccio denominato *default deny*, che si contrappone all'approccio *default allow* adottato dalla maggioranza degli

[2] `https://debian-administration.org/article/57/Making_/tmp_non-executable`

antivirus, questi software impediscono l'esecuzione di ogni file eseguibile sconosciuto, ovvero non presente nella whitelist, a prescindere dal livello di privilegio dell'utente.

Nei sistemi basati su Unix, di cui fa parte anche il sistema operativo AIX, esistono moltissimi tool di whitelisting come *AppAmour*, integrato nella maggior parte delle distribuzioni Linux; altri esempi noti sono *SELinux* e *grsecurity*.

Riteniamo che l'uso di un qualsiasi strumento di whitelisting, opportunamente configurato e testato, avrebbe potuto con buone probabilità contrastare le attività del malware FASTCash impedendo al processo maligno responsabile della code injection di avviarsi o di accedere alle risorse del sistema.

Tuttavia tale contromisura è lungi dall'essere una panacea; gli attaccanti avrebbero potuto, per mezzo di tecniche di *priviledge escalation*, riuscire a ottenere le autorizzazioni necessarie per alterare il contenuto della whitelist stessa allo scopo di consentire la successiva esecuzione del malware. Per tale motivo è indispensabile, allo scopo di non compromettere l'efficacia di questi strumenti, provvedere ad una opportuna protezione degli account responsabili della gestione della whitelist mediante forme di autenticazione più sofisticate e sicure.[3]

## 4.3 Riduzione della superficie di attacco

Dalle nostre analisi e da quelle pubblicate dalla NCCIC, risulta che il malware FASTCash sfrutti, per ragioni che purtroppo ignoriamo, la GUI session manager di X11 accedendo alla directory `/tmp/.ICE-unix/` all'interno della quale sono contenute tutti i dati riguardante la sessione corrente del gestore grafico.

Per quanto si possa controllare l'accesso non autorizzato alle risorse di X11 al fine di contrastare le attività di FASTCash, riteniamo che in generale l'utilizzo di un gestore grafico all'interno di sistemi critici, come i payment switch server dell'istituto bancario, rappresenti un grave rischio per la sicurezza.

Infatti l'uso del gestore grafico aumenta il numero di vulnerabilità sfruttabili dai malware poiché i bug di sicurezza del gestore si aggiungono a quelle già presenti nel sistema stesso, aumentando la superficie di attacco del sistema.

E' indubbio che la rimozione del gestore grafico avrebbe impedito al malware di funzionare a dovere anche qualora avesse compiuto con successo la process injection.

In base a quanto detto, riteniamo che all'interno di sistemi critici, al fine di aumentare la sicurezza del sistema, sia buona regola **installare ed eseguire solo ed esclusivamente le applicazioni indispensabili** per eseguire un certo servizio, disabilitando o rimuovendo ogni componente ridondate offerto dal sistema operativo. *Così facendo si riducono il numero di vulnerabilità del sistema e si facilità le operazioni di monitoring del sistema essendo più piccola la quantità di processi in esecuzione nel sistema.*

### 4.3.1 Gli Unikernel

Fortunatamente esiste uno strumento molto potente per ridurre al massimo la superficie di attacco minimizzando il numero di vulnerabilità del nostro sistema: gli **unikernel**.

[3] *Cfr.* `https://www.sans.org/reading-room/whitepapers/application/application-whitelisting-panacea-propaganda-33`

Gli unikernel sono sistemi operativi specializzati con unico spazio d'indirizzamento la cui principale caratteristica risiede nel possedere un set minimale di librerie e servizi, ossia quelli indispensabili per l'esecuzione delle applicazioni richieste.

Una descrizione dettagliata degli unikernel non ricade negli scopi della presente relazione, tuttavia basti ricordare che gli unikernel hanno una dimensione, in termini di linee di codice, pari al 4% di un sistema operativo tradizionale. Ciò comporta notevolissimi vantaggi in termini di sicurezza poiché, oltre alla riduzione delle vulnerabilità esposte, data la minore quantità di codice è possibile scoprire e risolvere le vulnerabilità con maggior velocità prima ancora di essere sfruttate da cyber-criminali.[4]

### 4.3.2 Ridondanza e virtualizzazione

Una contromisura che riteniamo efficace per aumentare la sicurezza del sistema e ridurre la probabilità di successo degli attaccanti è rappresentato dall'adozione di un certo grado di **ridondanza dei sistemi critici**.

Riteniamo che, subordinando l'approvazione di una transazione finanziaria all'approvazione di più sistemi di payment switch **indipendenti**, opportunamente coordinati attraverso algoritmi di consenso (Raft, Paxos ecc.), sarebbe stato possibile contrastare in modo molto efficace l'attività dei cyber-criminali, costretti ad attaccare un numero maggiore di sistemi.

La ridondanza dei sistemi può essere raggiunta anche attraverso tecniche di virtualizzazione, con vantaggi ben noti dal punto di vista della sicurezza grazie all'isolamento fra sistemi.

## 4.4 Monitoring

L'attività che più di tutte avrebbe contribuito a contrastare le azioni intraprese dal malware FASTCash è rappresentato dal *monitoring*, ovvero la possibilità, offerta da una variegata suite di applicazioni, di poter eseguire il *log di tutti gli eventi di interesse in un sistema* come, nel nostro caso specifico, l'esecuzione di transazioni finanziarie, con lo scopo di riuscire a rilevare le attività del malware e poter quindi reagire tempestivamente per limitare i danni.

Questi sofisticati software non si limitano semplicemente alla raccolta di dati ma sono in grado, previa un'opportuna configurazione, di poter reagire qualora rilevassero attività sospette attraverso l'esecuzione regolare di attività di audit sui log raccolti.

Qualora rilevassero attività insolite, questi software reagiscono emettendo i cosiddetti *alert*, avvisi finalizzati ad informare il personale incaricato della sicurezza del sistema della presenza di attività insolite nel sistema, la quale potrà infine prende provvedimenti per contrastate le attività del malware.

[4] `https://en.wikipedia.org/wiki/Unikernel`

## 4.5 Autenticazione

Sfortunatamente, le contromisure che abbiamo elencato finora sono vulnerabili a tecniche di *privilege escalation*, attraverso le quali gli attaccanti possono aggirare i nostri sistemi di protezione, firewall e antivirus inclusi.

La riduzione della superficie di attacco e una regolare attività di aggiornamento costituiscono il presupposto per un sistema resistente a queste forme di attacco; tuttavia è indispensabile adottare opportune forme di autenticazione, capaci di proteggere adeguatamente gli account con privilegi elevati.

Come ricordato anche nei rapporti NCCIC, sarebbe opportuno adottare *almeno* una **forma di autenticazione a due fattori** secondo la quale si devono combinare almeno due forme di autenticazione appartenenti a distinte classi, riportate di seguito:[5]

***Something an user knows*** Questa classe comprende tutte le forme di autenticazione che richiedono un "*qualcosa*" che l'utente deve conoscere per eseguire il login in un sistema, come, ad esempio, **usernames**, **ID**, **password** e **personal identification numbers** (**PIN**).

***Something an user has*** Fanno parte di questa categoria i cosiddetti **one-time password tokens** (**OTP tokens**), tessere magnetiche, carte SIM

***Something an user is*** In questa categoria fanno parte tutte le forme di autenticazione biometriche (scansione impronta digitale, retina ecc.)

***Something where user is*** Quest'ultima classe comprende forme di autenticazione legate alla posizione dell'utente, come dati GPS, indirizzi IP ecc.

Sebbene l'adozione di forme di autenticazione complesse possa essere controproducente sia dal punto di vista della sicurezza che della produttività, forme di autenticazione che comprendono 3 o più fattori sono molto efficaci nel proteggere gli account.

[5] `https://searchsecurity.techtarget.com/definition/four-factor-authentication-4FA`

# Elenco delle tabelle

# Listings