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 2.2.2 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:[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.

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

`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.

Ricordiamo che i sample sono stati ottenuti mediante download dal database di *Hybrid-Analysis*.[11]

2.2.1 Il file `Injection_API_executable_e`

L'output ottenuto dal tool `file` indica che il file `Injection_API_executable_e`, 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 IBM.[15]

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`

[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`

2.2.1.1 Analisi delle stringhe

Cominciamo l'analisi studiando alcune delle stringhe più importanti che è possibile estrarre ricorrendo al tool `strings`.

Osservando innanzitutto il formato della directory di installazione predefinita delle librerie del compilatore **GCC** nei sistemi operativi AIX, riportato per comodità nel listato 2.1[16], possiamo facilmente conoscere dal frammento mostrato 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
```

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; in ogni caso, dal momento che il malware è stato compilato per la versione 7.1 di AIX, possiamo presupporre che la versione del sistema operativo attaccato fosse almeno pari alla 7.1. E' importante ricordare 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 stata già terminata nel novembre 2013, benché la versione 7.1 TL5 riceverà ancora aggiornamenti da parte di IBM fino ad aprile 2022.[20]

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.

Osservando il frammento mostrato nel listato 2.4, è possibile notare un insieme di stringhe, aventi formato (`[FUNCTION_NAME]` *info*), le quali, come

[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
[21] Cfr. https://www.ibm.com/it-it/marketplace/xl-cpp-aix-compiler-power

avremo modo di notare 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

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 procedura denominata `out_log`, analizzata in dettaglio in 2.2.2, dimostra che le suddette stampe siano state scritte in memoria di massa.

Infine le righe 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
332 [main] Eject fail ret=%llX
333 Usage: injection pid dll_path mode [handle func toc]
334        mode = 0 => Injection
335        mode = 1 => Ejection
336 [main] handle=%llX, func=%llX, toc=%llX
337 [main] ERROR::g_pid(%X) <= 0
338 [main] ERROR::load_config fail
339 [main] ERROR::eject & argc != 7
340 [main] ERROR::g_dl_handle(%llX) <= 0
341 [main] WARNING::func_addr(%llX), toc_addr(%llX)
342 ...
```

Prima di analizzare nel dettaglio l'attacco di code injection vero e proprio, è indispensabile dapprima comprendere come vengono rappresentati e gestiti i **processi** nei sistemi operativi AIX. Ogni particolare aspetto di un processo, come, ad esempio, il suo stato, i sui livelli di privilegio o il proprio spazio di indirizzamento, è descritto da un insieme di file. Quest'ultimi, dato un processo

[22] Cfr. `http://man7.org/linux/man-pages/man3/printf.3.html`

il cui identificatore sia pari a `pid`, sono tutti raccolti nella directory `/proc/pid`. Tale sistema di gestione dei processi adottato da AIX permette di:

- Conoscere i `pid` di **tutti** i processi del sistema attraverso il listing nella directory `/proc`.
- 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()`. [23]

Di questi file, alcuni dei quali sono riportati a titolo di esempio nella tabella 2.2[24], ricordiamo in particolare:

/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.[25]

/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.[26]

/proc/pid/status Contiene informazioni sullo stato del processo. [27]

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`

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

Come dimostrato dall'analisi della NCCIC, dalla nostra analisi del codice assembler e anche dalla presenza del *conversion specifier* `%d`, non c'è dubbio 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
```

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

Analizziamo ora nel dettaglio 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]. Come già detto, questi messaggi possono essere scritti direttamente nel file `ctl` di un dato processo, alterandone lo stato.

Osservando il listato 2.6, notiamo una stampa del logger all'interno è presente la stringa **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.

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

```
319 ...
320 [proc_wait] PCWSTOP pid=%d, ret=%d, err=%d(%s)
321 [proc_wait] tid=%d, why=%d, what=%d, flag=%d, sig=%d
322 ...
```

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)

```
299 ...
300 [proc_attach] PCSET pid=%d, ret=%d, err=%d(%s)
301 [proc_attach] PCSTOP pid=%d, ret=%d, err=%d(%s)
302 [proc_attach] PCSTRACE pid=%d, ret=%d, err=%d(%s)
303 [proc_attach] PCSFAULT pid=%d, ret=%d, err=%d(%s)
304 [proc_attach] PCSENTRY pid=%d, ret=%d, err=%d(%s)
305 [proc_detach] PCSTRACE pid=%d, ret=%d, err=%d(%s)
306 [proc_detach] PCSFAULT pid=%d, ret=%d, err=%d(%s)
307 [proc_detach] PCSENTRY pid=%d, ret=%d, err=%d(%s)
```

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

[29] *Cfr. ivi* pag. 242

[30] Cfr. *Ibidem*

[31] *Cfr. ivi* pag. 234

[32] *Cfr. ibidem*

```
[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]

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

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

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

```
[out_regs] IAR=%llX
[out_regs] MSR=%llX
[out_regs] CR=%llX
[out_regs] LR=%llX
[out_regs] CTR=%llX
[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.

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

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

2.2.1.2 Analisi del codice assembly

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

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).*

2.2.1.3 La procedura `main`

La parte iniziale della procedura `main` è caratterizzata da una serie di operazioni che coinvolgono 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) che probabilmente sono state utilizzate per raccogliere informazioni. Sono presenti due istruzioni di salto vero le funzioni `atoi` (riga 7015 e 7041) usate per convertire i parametri passati dagli attaccanti attraverso la linea di comando il che dimostra la natura interattiva del malware.

Dopo una serie di istruzioni di salto verso procedure varie procedure di inizilizzazione, tra cui spiccano `load_config` e `get_func_addr`, viene raggiunta la porzione di codice mostrata nel listato 2.11 dove, dopo aver copiato i dati necessari in alcuni registri attraverso le apposite istruzioni mr, vengono eseguite a cascata due istruzioni di salto verso una procedura chiamata inject, che contiene il codice operativo per l'esecuzione della code injection.

Listing 2.11: Codice assembly estratto dal file `Injection_API_executable_e`

```
308 mr      r3,r10
309 mr      r4,r9
310 addi    r5,r2,-728
311 bl      0x1000256c <.inject>
312 li      r3,120
313 bl      0x10003468 <.sleep>
314 ld      r2,40(r1)
315 addi    r10,r31,120
316 lwz     r9,116(r31)
317 extsw   r9,r9
318 mr      r3,r10
319 mr      r4,r9
320 li      r5,0
321 bl      0x1000256c <.inject>
322 bl      0x10001154 <.CloseHandle>
```

2.2.1.4 La procedura `inject`

In questo paragrafo descriveremo le operazioni eseguite dalla procedura chiamata `inject` a cui gli autori del malware hanno affidato il compito di eseguire l'attacco di code injection a danno del processo bersaglio. Nel listato 2.12 viene mostrato il frammento corrispondente alla prima parte della suddetta funzione, all'interno del quale possiamo distinguere le seguenti operazioni:

- Copia nel *link register* dell'indirizzo di ritorno dal registro `r0` attraverso l'istruzione `mflr`.
- Inizializzazione di vari registri necessari attraverso varie istruzioni `std` e `mr` che coinvolgono i registri `r4`, `r5`,`r9` e `r31`, quest'ultimo probabilmente adibito ad ospitare l'indirizzo di memoria di base da cui viene computato l'indirizzo da cui prelevare i dati dalla memoria. Si presupponi che i registri precedentemente menzionati ospiteranno gli indirizzi alle celle di memoria contenenti il codice malevolo che verrà poi scritto all'interno dello spazio di indirizzamento del processo bersaglio.
- Esecuzione della code injection vera e propria attraverso una serie di istruzioni di salto (`bl`) verso l'indirizzo `0x10002f00` corrispondente alla prima istruzione della funzione `memset` preceduta dalle necessarie inizializzazioni dei registri di input attraverso varie istruzioni `mr`.

Listing 2.12: Codice assembly estratto dal file `Injection_API_executable_e`

```
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>
```

Dopo una serie di istruzioni di salto verso la funzione `memset`, ed aver dunque conclusa le operazioni di modifica della memoria del processo attaccato, possiamo osservare le successive operazioni eseguite dal listato 2.13 in cui apprendiamo che:

- Vengono eseguite ben tre istruzioni `bl` per permettere l'esecuzione della procedura `out_log` per effettuare la scrittura delle informazioni di interesse su un file esterno.
- Vengono diverse istruzioni di salto per eseguire varie procedure tra cui quella denominata `proc_attach`, usata probabilmente per modificare alcune informazioni di stato del processo, la `proc_wait`, usata probabilmente per arrestare l'esecuzione del processo bersaglio, `proc_getregs` ed `out_regs` usate rispettivamente per leggere i valori contenuti nei registri e successivamente scriverli in un file di log.

Listing 2.13: Codice assembly estratto dal file `Injection_API_executable_e`

```
308 bl      0x10000674 <.out_log>
309 bl      0x10001220 <.proc_attach>
310 li      r3,0
311 bl      0x10001a28 <.proc_continue>
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>
```

Dopo una serie di istruzioni di salto verso altre funzioni, tra cui figura una denominata `proc_readmemory`, avviene l'ultima fase della code injection durante la quale, come dimostrato dal listato 2.14, viene alterata la memoria del processo bersaglio attraverso istruzioni di salto verso le procedure `proc_writememory`, usata probabilmente per indurre il processo bersaglio a eseguire il codice malevolo copiato in precedenza, e la `proc_setregs` usata per alterare il contenuto dei registri e dunque modificare il futuro comportamento del processo. La procedura si conclude con il riavvio del processo e una lunga fase di log attraverso una grande quantità di istruzioni di salto verso la procedura `out_log`.

Listing 2.14: Codice assembly estratto dal file `Injection_API_executable_e`

```
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>
```

2.2.1.5 La procedura `proc_attach`

Analizziamo nel dettaglio l'attacco al processo la quale si compone in varie fare. Nel listato possiamo osservare come vengono dapprima eseguite delle operazioni di store ward con diversi offset con un registro comune come indirizzo sorgente;

Successivamente gli attaccanti utilizzano quello che probabilmente si tratti dell'indirizzo dell'area di memoria del processo bersaglio e con ripetute operazioni si store word muove il puntatore a quell'area di memoria con step da 4 byte. Alla fine, raggiunta la posizione desiderata, sposta il risultato in vari registri e esegue un'operazione di salto (bl) che punta all'indirizzo per la funzione memset.

```
1 [
2 frame=lines,
3 caption={Codice assembly estratto dal file \texttt{Injection
    \_API\_executable\_e}},
4 label={code:AssemblyFunction-inject-04},
```

```
firstnumber=308]
<.proc_attach>:
mflr    r0
std     r0,16(r1)
std     r29,-24(r1)
std     r30,-16(r1)
std     r31,-8(r1)
stdu    r1,-352(r1)
mr      r31,r1
li      r9,0
stw     r9,128(r31)
li      r9,0
stw     r9,132(r31)
li      r9,0
std     r9,136(r31)
li      r9,0
std     r9,144(r31)
li      r9,0
std     r9,152(r31)
li      r9,0
std     r9,160(r31)
li      r9,0
std     r9,168(r31)
li      r9,0
stw     r9,176(r31)
addi    r10,r31,180
li      r9,140
mr      r3,r10
li      r4,0
mr      r5,r9
bl      0x10002f00 <.memset>
```

Dopo aver richiamato la funzione `memset`, certamente utilizzata dagli attaccanti per eseguire la *code injection* alterando il contenuto dello spazio di indirizzamento del processo bersaglio, la funzione `proc_attach` incomincia una fase di logging durante la quale, attraverso ripetuti salti condizionati agli indirizzi `0x100031ec`, `0x1000319c` e `0x10000674`, corrispondenti agli indirizzi delle funzioni `write`, `sterror` (utilizzata certamente dagli attaccanti per verificare l'output della funzione `write`), `log_out`, vengono archiviati in un file esterno il contenuto dei registri di interesse che paiono essere i registri r31, r30, r29 e r9 che vengono copiati con ripetute istruzioni `mr` in registri ausiliari (r4, r5, r6 e r7 rispettivamente) prima di essere inviati come input alla funzione `log_out`.

```
[
frame=lines,
caption={Codice assembly estratto dal file \texttt{Injection
    \_API\_executable\_e}},
label={code:AssemblyFunction-inject-05},
firstnumber=308]
li      r9,14
stw     r9,136(r31)
li      r9,4
stw     r9,140(r31)
addi    r9,r2,-764
```

```
lwz     r9,0(r9)
extsw   r10,r9
addi    r9,r31,136
mr      r3,r10
mr      r4,r9
li      r5,8
bl      0x100031ec <.write>
ld      r2,40(r1)
mr      r9,r3
stw     r9,128(r31)
addi    r9,r2,-768
lwz     r9,0(r9)
extsw   r29,r9
ld      r9,128(r2)
lwz     r9,0(r9)
extsw   r30,r9
ld      r9,128(r2)
lwz     r9,0(r9)
extsw   r9,r9
mr      r3,r9
bl      0x1000319c <.strerror>
ld      r2,40(r1)
mr      r9,r3
lwz     r10,128(r31)
extsw   r10,r10
ld      r3,536(r2)
mr      r4,r29
mr      r5,r10
mr      r6,r30
mr      r7,r9
bl      0x10000674 <.out_log>
```

La fase di code injection si conclude con il caricamento nel registro `r0` dell'indirizzo della funzione chiamante copiato successivamente nel link register attraverso l'istruzione `mtrl`; vengono in seguito eseguite una serie di istruzioni ld per popolare i registri r29, r30 e r31 che conterranno probabilmente i valori di ritorno della funzione per poi eseguire una istruzione `blr` (*Branch Link Register*).

```
ld      r0,16(r1)
mtlr    r0
ld      r29,-24(r1)
ld      r30,-16(r1)
ld      r31,-8(r1)
blr
```

```
bl      0x10000674 <.out_log>
bl      0x10001220 <.proc_attach>
li      r3,0
bl      0x10001a28 <.proc_continue>
li      r3,0
li      r4,0
bl      0x10001b44 <.proc_wait>
ld      r3,728(r2)
bl      0x10000674 <.out_log>
```

```
addi    r9,r31,152
mr      r3,r9
bl      0x10001ee4 <.proc_getregs>
addi    r9,r31,152
mr      r3,r9
bl      0x10000c80 <.out_regs>
addi    r8,r31,536
addi    r10,r31,152
li      r9,384
mr      r3,r8
mr      r4,r10
mr      r5,r9
bl      0x1000324c <.memmove>
nop
ld      r9,536(r31)
addi    r9,r9,-16
mr      r3,r9
li      r4,16384
bl      0x10000b48 <.file_dump>
```

2.2.2 Il file `2.so`

`2.so` è un file di tipo **eXtended COFF** che, come dimostreremo all'interno di questa sezione, è stato progettato per l'ispezione e la manipolazione dei dati contenuti nei messaggi basati sul protocollo **ISO8583** scambiati tra i sistemi informatici degli istituti finanziari. Come dimostrato anche dalla già citata analisi AR18-275A della NCCIC, il file, come suggerisce anche l'estensione `.so`, rappresenta una **shared library** che, esportando una grande quantità di metodi in grado di interagire con i messaggi basati sul suddetto protocollo, permette agli attaccanti di alterare le transazioni finanziare a proprio favore.

Tabella 2.5: 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`

2.2.2.1 Analisi delle stringhe

Listing 2.15: Stringe estratte dal file `2.so` (1)

```
465 ...
466 /opt/freeware/lib/gcc/powerpc-ibm-aix6.1.0.0/4.2.0/ppc64:/
    opt/freeware/lib/gcc/powerpc-ibm-aix6.1.0.0/4.2.0:/opt/
    freeware/lib/gcc/powerpc-ibm-aix6.1.0.0/4.2.0/../../..:/
    usr/lib:/lib
467 ...
```

Seguendo lo stesso ragionamento descritto in 2.2.1.1, comprendiamo dal listato 2.15 che gli attaccanti si siano serviti della versione 4.2.0 di GCC[34] compatibile con l'architettura PowerPC™ con sistema operativo AIX 6.1 (di quest'ultimo il supporto è terminato ufficialmente il 30 Aprile del 2017.[35]).

La libreria esporta una grandissima quantità di funzioni molte delle quali riguardanti la manipolazione dei messaggi basati su protocollo ISO8583, alcune delle quali riportate nel listato 2.16.

Listing 2.16: Stringe estratte dal file `2.so` (2)

```
545 ...
546 DL_ISO8583_MSG_Init
547 DL_ISO8583_MSG_Free
```

[34] Cfr. `http://www.gnu.org/software/gcc/gcc-4.2/`
[35] Cfr. `https://www-01.ibm.com/support/docview.wss?uid=swg21634678#AIX`

```
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
...
```

Listing 2.17: Stringhe estratte dal file `2.so` (3)

```
Blocked Message(msg=%04x, term=%02x, pcode=%06x, pan=%s)
Passed Message(msg=%04x, term=%02x, pcode=%06x, pan=%s)
[recv] ret=%d
send ret = %d, err = %d
/tmp/.ICE-unix/context.dat
/tmp/.ICE-unix/tmp%d_%d.log
[%04d-%02d-%02d %02d:%02d:%02d][PID:%4u][TID:%4u] %s
/tmp/.ICE-unix/config_%d
/tmp/.ICE-unix/tmprd%d_%d.log
/tmp/.ICE-unix/tmpwt%d_%d.log
[DetourInitFunc] dlopen error(%s)
[DetourInitFunc] org_func(%p) %02X %02X %02X %02X %02X %02X
    %02X %02X %02X %02X %02X %02X
[DetourInitFunc] new_func(%p) %02X %02X %02X %02X %02X %02X
    %02X %02X %02X %02X %02X %02X
[DetourInitFunc] dlsym error(%s)
Success
Failed
DetourInitFunc(%s, %s) %s
[DetourInitFunc] org_func=%p new_func=%p
[DetourAttach] hook_func_addr=%p, new_func_addr=%p
[DetourAttach] after mmap=%p
[DetourAttach] copy_func(%x) %02X %02X %02X %02X %02X %02X
    %02X %02X %02X %02X %02X %02X
[DetourAttach] hook_func_addr(%x) %02X %02X %02X %02X %02X
    %02X %02X %02X %02X %02X %02X %02X
[DetourDetach] hook_func_addr(%x) %02X %02X %02X %02X %02X
    %02X %02X %02X %02X %02X %02X %02X
```

Come dimostrato dal frammento riportato in 2.17 e dall'analisi di alcuni frammenti chiave del codice assembly in 2.2.2.7, la libreria svolge un'**intensa attività di logging** scrivendo direttamente in memoria di massa attraverso una procedura chiamata `out_dump_log`. Dal listato 2.17 possiamo osservare un'interessante riferimento al percorso `/tmp/.ICE-unix/`: in accordo alla docu-

mentazione relativa alla versione R6.8.2 di X11[36] (la famosa implementazione del X Window System[37]) 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 all'autenticazione o al *byte order negotiation*[38]. 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.

Listing 2.18: Stringhe estratte dal file `2.so` (4)

```
545 GenerateRandAmount
546 GenerateResponseTransaction1
547 GenerateResponseTransaction2
548 GenerateResponseInquiry1
549 Crypt
```

Osservando infine il frammento riportato in 2.18 possiamo comprendere l'esistenza di alcuni metodi utilizzati per rispondere alle transazioni finanziarie generate dai sistemi bancari sotto attacco.

2.2.2.2 Analisi del codice assembly

Benché naturalmente sprovvista di una procedura `main`, trattandosi di una shared library, il file `2.so` assume un ruolo centrale per il corretto svolgimento dell'attacco poiché esporta tutte le procedure necessarie per manipolare le transazioni elettroniche dei sistemi finanziari attaccati. I metodi esportati dal file `so.2`, una parte dei quali sono mostrati nel listato **??**, sono molto numerosi e riguardano principalmente l'ispezione e la manipolazione dei messaggi usati dal protocollo ISO8583 a cui si aggiungono altre procedure di supporto, tra cui quelle usate per implementare un meccanismo di logging (`out_dump_log`) e altre usate per gestire una tabella hash (`hashmap_new`, `hashmap_init`, `hashmap_get` ecc.). Di seguito riportiamo l'analisi di alcune delle procedure principali presenti nel file.

2.2.2.3 La procedura `DL_ISO8583_MSG_GetField_Bin`

Analizzando le prime righe di codice della procedura, riportate in parte nel listato 2.19, troviamo molte istruzioni `std` e `mr` utilizzanti i registri `r0` e `r31` come sorgenti per popolare un altro insieme di registri; perciò si può supporre che i registri `r0` e `r31` siano stati usati per contenere i dati passati come argomento alla funzione, presumibilmente l'indirizzo dell'area di memoria del messaggio da ispezionare e un riferimento al campo da estrarre (probabilmente una stringa o un identificatore numerico).

Listing 2.19: Codice assembly estratto dal file `2.so`

```
347 std     r31,-8(r1)
348 stdu    r1,-80(r1)
349 mr      r31,r1
350 mr      r0,r3
```

[36] Cfr. `https://www.x.org/releases/X11R6.8.2/doc/RELNOTES5.html`
[37] Cfr. `https://www.x.org/wiki/`
[38] Cfr. `https://www.x.org/releases/X11R7.7/doc/libICE/ICElib.html#Overview_of_ICE`

```
351 std      r4,136(r31)
352 std      r5,144(r31)
353 std      r6,152(r31)
354 stw      r0,128(r31)
```

Un altro frammento della stessa procedura, riportato nel listato 2.20, mostra come l'ispezione del campo di interesse appartenente al messaggio ISO8583 avvenga per mezzo di un ciclo; notiamo infatti diverse istruzioni di `beq` (*Branch On Equal*) aventi come argomento uno stesso indirizzo target e altrettante istruzioni `cmpdi` (*Compare Doubleword Immediate*). E' probabile che tale ciclo sia stato usato per attraversare il flusso di byte che compone un certo messaggio fino al raggiungimento dell'indirizzo corrispondente al campo di intessere che pare venga restituito alla funzione chiamante per mezzo di un'apposita istruzione `mr` coinvolgendo il registro `r3` come output (riga 14010).

Listing 2.20: Codice assembly estratto dal file `2.so`

```
347 cmpdi    cr7,r0,0
348 beq      cr7,0x10002048
349 lwz      r0,128(r31)
350 cmplwi   cr7,r0,128
351 bgt      cr7,0x10002048
352 lwz      r0,128(r31)
353 clrldi   r9,r0,32
354 ld       r11,136(r31)
355 addi     r0,r9,1
356 rldicr   r0,r0,4,59
357 add      r9,r0,r11
358 addi     r9,r9,8
359 ld       r0,0(r9)
360 cmpdi    cr7,r0,0
361 beq      cr7,0x10002048
```

2.2.2.4 La procedura `DL_ISO8583_MSG_GetField_Str`

La funzione `DL_ISO8583_MSG_GetField_Str` è sostanzialmente identica a quella precedentemente descritta sebbene il nome suggerisca che tale funzione restituisca indubbiamente una stringa piuttosto che dati binari.

2.2.2.5 La procedura `DL_ISO8583_MSG_SetField_Bin`

L'esistenza di tale procedura dimostra che il malware non si limita solo all'ispezione dei messaggi ma che è in grado di manipolarne i contenuti. Come mostrato nel listato 2.20, in modo simile alle altre procedure, il messaggio viene dapprima ispezionato per mezzo di istruzioni di salto incondizionato (b) e condizionato (ble) insieme ad opportune istruzioni di comparazione (cmplwi) fino al raggiungimento dell'indirizzo corrispondente al campo da modificare. Una successiva istruzione di salto alla procedura `DL_ISO8583_MSG_AllocField`, attraverso la quale viene presumibilmente allocata un'opportuna area di memoria atta ad ospitare il campo con i nuovi valori, è seguita infine dall'invocazione della procedura `memmove` completando così l'operazione di modifica del messaggio.

Listing 2.21: Codice assembly estratto dal file `2.so`

```
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
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>
```

2.2.2.6 La procedura `DL_ISO8583_MSG_RemoveField`

La procedura `DL_ISO8583_MSG_RemoveField` presenta a una struttura sostanzialmente identica a quelle viste finora. L'unica differenza degna di nota, come mostrato nel listato 2.22, riguarda l'eliminazione del campo specificato che avviene attraverso una chiamata alla procedura `free`.

Listing 2.22: Codice assembly estratto dal file `2.so`

```
347 mr      r3,r0
348 bl      0x100002dc <.free>
349 ld      r2,40(r1)
```

2.2.2.7 La procedura `out_dump_log`

La procedura denominata `out_dump_log`, usata, come suggerisce il nome, per ovvie finalità legate al **debugging** dell'applicazione e al **logging** di tutte le informazioni di interesse raccolte durante l'attacco, rappresenta quella invocata il maggior numero di volte all'interno del codice della libreria: ben 32 volte. Una delle procedure esportate, chiamata `NewRead`, invoca la suddetta funzione ben 11 volte e, in generale, l'invocazione è preceduta quasi sempre da un'altra nei confronti della funzione `ReadRecv`; quest'ultima considerazione dimostra come

quasi certamente gli attaccanti ispezionavano il contenuto dei messaggi ricevuti salvando tutti i dati in **memoria di massa**, affinché fosse accessibile di seguito per un qualche scopo.

rappresenta quella invocata più frequentemente all'interno del codice; ben 32 volte il che dimostra la sua importanza dato il volume di dati registrato dagli attaccanti.

Tale procedura, richiamata molte volte nel codice, ha come scopo quello di scrivere messaggi di log opportunamente formattati in un file esterno, forse per motivi di debug o per tener traccia dello stato di avanzamento dell'attacco. La prima porzione del codice assembly, mostrata nel listato 2.23, è dominata da una grande quantità di istruzioni `std` usate per popolare tutti i registri dalla numero 3 alla 10 e dalla numero 23 alla 31, che probabilmente conterranno i dati da stampare nel file di log. Dal momento che la totalità di queste istruzioni usano i registri `r0` e `r31` come sorgenti quest'ultimi conterranno i dati passati come argomento alla funzione.

Listing 2.23: Codice assembly estratto dal file `2.so`

```
347 mflr     r0
348 std      r23,-72(r1)
349 std      r24,-64(r1)
350 std      r25,-56(r1)
351 std      r26,-48(r1)
352 std      r27,-40(r1)
353 std      r28,-32(r1)
354 std      r29,-24(r1)
355 std      r31,-8(r1)
356 std      r0,16(r1)
357 stdu     r1,-4624(r1)
358 mr       r31,r1
359 std      r4,4680(r31)
360 std      r5,4688(r31)
361 std      r6,4696(r31)
362 std      r7,4704(r31)
363 std      r8,4712(r31)
364 std      r9,4720(r31)
365 std      r10,4728(r31)
366 std      r3,4672(r31)
```

La parte centrale della procedura, mostrata invece nel listato 2.24, contiene un insieme di istruzione il cui scopo evidentemente è quello di scrivere tutti i dati precedentemente raccolti su un file. Come si può facilmente notare dal listato 2.24, è facile intuire che ogni stringhe venga dapprima realizzata facendo ricorso alla funzione standard `snprintf` e poi, dopo l'apertura del file di log attraverso la chiamata di sistema `fopen`, vengano scritti aggiungendo ulteriori informazioni come data e ora locale, come dimostrano le istruzioni di salto verso le procedure `gettimeofday` e `localtime`. La procedura si conclude con una chiamata alla procedura `close` per poi chiudersi definitivamente con l'istruzione `blr` che permette la ritornare alla procedura chiamante.

Listing 2.24: Codice assembly estratto dal file `2.so`

```
347 bl       0x10000748 <.snprintf>
348 ld       r2,40(r1)
```

```
addi     r0,r31,4280
mr       r3,r0
ld       r4,856(r2)
bl       0x10000770 <.fopen>
ld       r2,40(r1)
mr       r0,r3
std      r0,152(r31)
ld       r0,152(r31)
cmpdi    cr7,r0,0
beq      cr7,0x1000a0ec
addi     r0,r31,4264
mr       r3,r0
li       r4,0
bl       0x10000798 <.gettimeofday>
ld       r2,40(r1)
addi     r0,r31,4264
mr       r3,r0
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 un 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.

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, in virtù delle caratteristiche tecniche del malware e della forma di attacco perpetrata dai cyber-criminali, tale attività poco efficace contro FASTCash.

Come abbiamo avuto modo di notare durante l'analisi, l'attacco effettuato dal malware FASTCash è per sua natura molto difficile da contrastare attraverso gli aggiornamenti di sicurezza perché basa il proprio funzionamento sull'uso (sarebbe meglio dire *abuso*) dei servizi essenziali offerti dal kernel del sistema operativo, in particolare la gestione dei processi/thread e i 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 non avrebbe contrastato efficacemente il malware FASTCash.

Supponendo anche rilascio di aggiornamenti di sicurezza che impongano restrizioni sull'uso delle *syscall* per l'accesso ai servizi del SO, il malware FAST-

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

Cash continuerebbe ad agire incontrastato essendo nascosto all'interno di un processo legittimo; oltre ad essere poco efficace, restrizioni sull'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

In base a quanto detto finora, riteniamo che gli sforzi finalizzati a reagire a questa forma di attacchi dovrebbero essere rivolti nell'impedire ai cyber-criminali di poter intraprendere l'attacco fin dal principio.

Le contromisure da adottare 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 funzionare un accesso in lettura/scrittura al pseudo-file system `/proc` per compiere l'attacco, riteniamo che l'uso delle ACL avrebbe potuto contrastare efficacemente FASTCash ad esempio impedendo a priori l'accesso alla suddetta directory.

Teoricamente, configurando opportunamente il sistema rendendo non eseguibili i file binari presenti nelle partizioni più vulnerabili, ad esempio 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 attività di aggiornamento e di configurazione del software.

4.2.2 Meccanismi di Whitelisting

Oltre a regolare le autorizzazioni di accesso alle risorse, un'altra efficace contromisura consiste nell'adottare meccanismi di **Whitelisting** che consentano l'accesso alle risorse del sistema *solo ed esclusivamente* ai 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 deter-

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

minate risorse del sistema. Utilizzando un approccio denominato *default deny*, che si contrappone all'approccio *default allow* adottato dalla maggioranza degli 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.

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

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**.

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 e coordinati attraverso algoritmi di consenso come Raft o Paxos, sarebbe stato possibile contrastare in modo molto efficace l'attività dei cyber-criminali costretti ad attaccare un numero maggiore di sistemi.

Ricorrendo a strumenti di virtualizzazione è possibile, oltre ad aumentare il grado di isolamento dei sistemi, aumentare il grado di ridondanza per le funzionalità critiche

Se si utilizzano sistemi operativi minimali come gli unikernel, in virtù delle loro caratteristiche, è possible migliorare le prestazioni ottenute attraverso la virtualizzazione e aumentando maggiormente la sicurezza del sistema grazie i vantaggi in termini di sicurezza dell'uso degli unikernel

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.

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

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.

4.5 Regole generali

Elenco delle tabelle

Listings