UNIVERSITÀ DEGLI STUDI DI BRESCIA

Analisi statica del malware AlinaPOS

Relazione progetto Sicurezza Informatica

Arici Nicola - 715253
Giustolisi Riccardo - 715230

A.A. 2021/2022

Indice

Capitolo 1

Introduzione

La presente relazione ha lo scopo di descrivere tutte le fasi del nostro lavoro, specificando anche i concetti teorici sui quali questo progetto si basa. In particolare, l'obiettivo da raggiungere è quello di provare ad applicare un attacco informatico (nel nostro caso, il *Fast Gradient Sign Method*) ad un modello di Machine Learning per la classificazione di malware, esplorandone l'efficacia e valutando eventuali contromisure.

Prima di tutto verranno esposti i concetti base, ossia il malware impiegato, le informazioni ricavate per poterlo analizzare (header del file PE), il dataset utilizzato e il modello costruito per la classificazione. Dopo aver riportato i risultati di questa fase preliminare, si procederà con la descrizione dell'attacco e la nostra implementazione.
Per finire con una riflessione sulle conseguenze dell'attacco applicato e su eventuali contromisure per rendere il classificatore più robusto.

Per realizzare tutto ci siamo affidati a Google Colab, software che permette di creare ed eseguire i Python Notebook. Il virus, scritto in C++ è stato modificato in una macchina virtuale con sistema operativo Windows 10, istanziata con VirtualBox, attraverso l'ambiente di sviluppo Microsoft Visual Studio Code.

Capitolo 2

Concetti preliminari

In questo capitolo saranno descritti gli elementi che stanno alla base del lavoro svolto: il malware impiegato, il formato PE, il modello costruito per la classificazione di eseguibili malevoli e il dataset impiegato per l'addestramento.

2.1 Il malware: Alina POS

Il malware impiegato è Alina, in circolazione dal 2012 (scaricabile al seguente link), specializzato nell'attaccare sia terminali POS che computer con software POS dedicato, entrambi con sistema operativo Windows, con l'intento è di rubare informazioni su carte di credito e di debito dal sistema del punto vendita infettato.

Durante le transazioni con carta di credito, i dati vengono decrittografati e temporaneamente tenuti in memoria dal software POS.In questo intervallo temporale il malware Alina preleva dalla RAM del dispositivo infetto le informazioni sulla carta di credito e, solo dopo averne verificato la correttezza, le trasmette al server C&C (Command & Control) attraverso un tunneling DNS.

Per carpire tali informazioni, Alina ispeziona i processi utente con l'aiuto delle chiamate API di Windows:

- **CreateToolhelp32Snapshot()**: acquisisce un'istantanea di tutti i processi in esecuzione;
- **Process32First()/Process32Next()**: recupera le informazioni sul primo/successivo processo rilevato in uno snapshot del sistema.

Alina mantiene una blacklist di processi che non devono essere aperti per cercare dati da estrapolare. Se un processo non fa parte di questa lista, viene usata la funzione **OpenProcess()** per leggere ed elaborare i contenuti in memoria: se non sono contenuti dati utili, il processo viene aggiunto alla blacklist, altrimenti si procede con l'estrapolazione delle informazioni. Una volta che i dati vengono rubati, Alina li invia ai server C&C utilizzando un comando HTTP POST codificato in binario.

Il lavoro da noi svolto in questa fase si concentra sull'analisi delle informazioni contenute nell'eseguibile del malware, il quale segue il formato standard PE.

2.2 Formato PE

Il formato Portable Executable (PE) è un formato standard per gli eseguibili, per i file oggetto, librerie condivise e device drivers di Win32. Questo formato è usato nelle versioni a 32-bit e 64-bit del sistema operativo Microsoft Windows, e può essere considerato come una struttura dati che incapsula tutte le informazioni necessarie al loader di Windows per caricare ed eseguire il codice macchina di un file eseguibile. Il termine "portable" si riferisce alla versatilità del formato che può essere adattato a numerose architetture. Le informazioni contenute nel PE sono divise in headers, sezioni e campi che contengono le informazioni utilizzate dal linker per mappare il file in memoria.

L'obiettivo di questa prima fase è quello di costruire una rete neurale in grado di classificare un eseguibile come malware o meno analizzando le informazioni contenute nell'header.

2.3 Dataset

Il dataset utilizzato per l'addestramento della rete neurale è composto da un insieme di header di vari file PE, malevoli e non. In particolare sono presenti 19611 record, suddivisi in 14599 malware e 5012 software non pericolosi, ognuno dei quali composto da settantanove campi:

- Il primo campo contiene il nome del software in formato stringa;
- Settantasette valori riferiti all'header dei file PE;
- Un campo per la classificazione dell'eseguibile come malware. Se il software è un malware avrà valore uno, altrimenti avrà valore zero.

È possibile scaricare questo dataset al seguente link.

2.4 Modello

Come già anticipato, l'obiettivo di questa parte del lavoro è costruire una rete neurale in grado di classificare il file PE di un software come malware o no dall'analisi dell'header. A questo proposito, il primo passo è quello di estrarre i valori che saranno poi analizzati dall'eseguibile del malware Alina.

A questo punto, prima di procedere con la costruzione effettiva della rete neurale, abbiamo preparato i dati contenuti nel dataset per effettuare l'addestramento. Dopo aver eliminato la colonna dei nomi, abbiamo separato quella relativa alla classificazione dell'eseguibile come malware dagli altri campi. Successivamente, utilizzando la funzione **Sequential()** della libreria *keras*, abbiamo inizializzato un modello sequenziale. A quest'ultimo vengono aggiunti 120 neuroni e viene specificato un tasso di *Dropout* del 20%. Dopo di che viene aumentato lo spazio delle features per ottimizzare le performance passando a 300 neuroni, riportandoli successivamente a 120. Infine, volendo avere come output del modello la classificazione del software, abbiamo impostato un'unica uscita con la funzione di attivazione sigmoidea. L'ultimo passo prima di procedere con l'addestramento del modello è

stato quello di definire la funzione di loss binaria *Cross Entropy*, l'ottimizzatore *Adam* e le metriche per la valutazione delle performance. In particolare, la valutazione del modello è stata effettuata calcolando le seguenti metriche:

- *Accuracy*: misura la percentuale di previsioni corrette;
- *F1*: media armonica tra precisione e recall;
- *Precision*: misura la percentuale di positivi effettivi tra gli esempi previsti come positivi;
- *Recall*: misura quanti positivi effettivi sono stati previsti come positivi.

```
model = Sequential()
model.add(Dense(120, input_dim=X_train.shape[1]))
model.add(Dropout(0.2))
model.add(Dense(300))
model.add(Dropout(0.2))
model.add(Dense(120))
model.add(Dropout(0.2))
model.add(Dense(units=1))
model.add(Activation('sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam',
              metrics=['acc',f1_m,precision_m, recall_m])
```

Figura 2.1: Struttura modello sequenziale

Dopo di che, abbiamo iniziato l'addestramento utilizzando la funzione **fit()** e specificando il numero di epoche a 30 e il *batch* a 64. Di seguito viene mostrata la valutazione della rete neurale:

Accuracy	96%
Precision	97%
Recall	98%
F1	97%

Il modello così definito classifica l'eseguibile del malware Alina POS come malevolo al 99,8616%.

Capitolo 3

Attacco

In questo capitolo sarà descritto l'attacco apportato al modello presentato nel capitolo precedente. Lo scopo ultimo sarà quello di ottenere una classificazione errata, ovvero no malware, per il nostro eseguibile.

3.1 Fast Gradient Sign Method

La tecnica di attacco implementata è il *Fast Gradient Sign Method*, abbreviato in FSGM. FGSM è una tecnica *white box*, ovvero una tecnica che ha conoscenza completa del modello di machine learning utilizzato per la classificazione, che punta a generare degli *adversarial examples*, campioni ben progettati con l'intenzione di causare un errore di classificazione.

L'attacco è straordinariamente potente e tuttavia intuitivo. È progettato per attaccare le reti neurali sfruttando il modo in cui apprendono. Piuttosto che lavorare per ridurre al minimo la loss regolando i pesi in base ai gradienti, l'attacco regola i dati di input per massimizzare la loss in base agli stessi gradienti. Nel dettaglio, partendo dall'esempio di input normalizzato X l'adversarial example X_* è così calcolato:

$$X_* = X + \epsilon \cdot sign(\nabla_x J(X, y_{true}))$$

con:

- ϵ: parametro che controlla l'ampiezza della perturbazione;
- $sign$: funzione segno;
- ∇: gradiente del modello rispetto al campione X;
- y_{true}: etichetta del campione X

3.1.1 Implementazione

L'attacco è stato eseguito grazie alla libreria Python *adversarial-robustness-toolbox*, che fornisce strumenti che consentono a sviluppatori e ricercatori di difendere e valutare modelli di Machine Learning contro gli attacchi di evasione, avvelenamento, estrazione e inferenza.

La classe utilizzata per implementare l'attacco, contenuta nel modulo *art.attacks.evasion*, è *FastGradientMethod*. L'attacco è un'estensione di quello spiegato sopra, ma nel nostro lavoro viene utilizzato nella versione standard FSGM attraverso l'utilizzo del parametro *norm = inf*, che indica al metodo di utilizzare la norma infinita per generare le perturbazioni.

```
classifier = TensorFlowV2Classifier(model=model,
                                    nb_classes=2,
                                    input_shape=X_train.shape[1],
                                    loss_object=tf.keras.losses.binary_crossentropy)

attacker = FastGradientMethod(estimator=classifier,
                              minimal=True,
                              eps=0.5)
```

Figura 3.1: Implementazione Python FSGM

I parametri utilizzati per definire l'attacco sono i seguenti:

- estimator: classificatore oggetto dell'attacco. Nel dettaglio è stato utilizzata la classe *TensorflowV2Classifier* per rappresentare il modello descritto in precedenza, passando i seguenti parametri:
  - model: il modello Keras definito al capitolo precedente;
  - nb_classes: il numero di classi in uscita. Essendo un classificatore binario (malware si/no) il parametro è stato impostato a due;
  - input_shape: grandezza dei dati di input;
  - loss_object: la funzione di loss per cui calcolare i gradienti. Questo parametro viene applicato per addestrare il modello e calcolare i gradienti rispetto all'input.
- minimal: impostato a vero indica all'attacco di calcolare la perturbazione minima
- eps: valore che controlla l'ampiezza della perturbazione.

Una volta definito l'oggetto *attacker* è possibile passargli il record normalizzato per generarne una versione elusiva. A causa dei limiti imposti dal formato PE l'attacco tende a generare dei record con valori prevalentemente negativi dato che non essendo presenti nel dataset di training, portano facilmente la rete neurale a sbagliare la classificazione del malware. A causa di questo limite, abbiamo analizzato solo il segno della variazione del valore, ovvero se questo aumenta o diminuisce nel record classificato male.

Di tutte le settantasette feature presenti nell'header, solo alcune sono state considerate per la modifica. Le prime feature, quelle facenti parte del DOS header, sono state scartate perché non modificabili (come *e_magic*) o perché difficilmente modificabili. Altre feature, facenti parte di altre sezioni dell'header, non sono state prese in considerazione perché difficilmetne impostabili attraverso modifiche al codice o al compilatore.

Di seguito sono riportati i valori del record originale, il valore suggerito dall'attacco e il segno della variazione solamente per le feature che abbiamo volontariamente modificato. Per un'analisi completa rifarsi all' Appendice A.

Feature	Malware	FSGM	Segno
MinorSubsystemVersion	0	-466.106	-
MajorSubsystemVersion	6	6.4059	+
MinorImageVersion	0	531.595	+
MajorImageVersion	0	-629.594	-
SizeOfStackCommit	4096	-33903.5	-
SizeOfHeapReserve	1048576	752001	-
SizeOfHeapCommit	4096	-282483	-
SizeOfCode	4096	$2.66879e{+}07$	+
SizeOfInitializedData	4096	$-3.00365e{+}07$	-
SizeOfUninitializedData	4096	$-1.53424e{+}07$	-

3.2 Modifiche al codice e al compilatore

Per modificare le feature individuate nella sezione precedente è stato deciso di apportare modifiche sia al codice che alle opzioni del compilatore.

Le prime riguardano l'introduzione di una classe di *Padding* il cui scopo era quello di introdurre delle stringhe casuali e delle semplici operazioni sulle stringhe per aumentare il valore delle feature *SizeOfCode* e *SizeOfInizialedData*.

```
#include <string>
class Padding{
    public:
        Padding(){
            std::string padding;
            padding = "J1BZBXVD=FRC63XHK<FXXQ664Y8V:4X>VXW4DYYMEFN0;5NVXF7L@DKW@5LJFJ6F;9Z4V6=8=3>7=IZ0<DIZW?X0=ZMF
            padding = "PJNXJLTLLO537OWIP?2RPS5R81N3B688@@>R516G;KB7LDE5V4I@@DS@JAL9:9@9QZ=>E6XESA@GG?=U5O1ONX5M63G<
            padding = "2K9QV==ICRVU>XJ156@Z?ULW@H<5GG5AGNXIH24;:UOIJ5VZ==<D9I6I?:UF:Z1Q>67WOVS32TGULK=5MDM06AS9=DN0

            std::string padding0;
            padding0 = "Proin laoreet cursus odio a placerat. Suspendisse nisl lectus, egestas ac magna ac, ornare
                "imperdiet condimentum risus, tincidunt aliquam massa placerat convallis. Aliquam fringilla vitae e
            padding = padding0;

            padding = padding0 + "123";
            padding0 = padding0 + "123";
            padding = padding + "123";
            padding0 = padding0 + "123";
        }
};
```

Figura 3.2: Classe di Padding C++

Mentre per le modifiche riguardanti il compilatore siamo andati ad intervenire sul linker attraverso due direttive. La prima, /SUBSYSTEM:WINDOWS,9.0, specifica l'ambiente per il file eseguibile. La scelta del sottosistema influisce sul simbolo del punto di ingresso (o funzione del punto di ingresso) che verrà selezionato dal linker. Nel dettaglio i numeri nove e zero rappresentano rispettivamente la versione massima e la versione minima del sottosistema. Seguendo i risultati dell'attacco FSGM il valore minimo sarebbe dovuto diminuire ulteriormente, ma a causa dei limiti imposti dallo standard PE è impossibile avere un valore negativo per un attributo. Al contrario, come suggerito dall'attacco, la versione massima è stata incrementata il più possibile. Un valore superiore di nove però impedisce al malware di essere eseguito su Windows 10.

La seconda direttiva introdotta, /VERSION:11.3, indica al linker di inserire un numero di versione nel PE Header. Al contrario del caso precedente l'attacco suggerisce di incrementare la versione minima e diminuire la versione massima. Per evitare problemi di consistenza (dove la versione minima supera quella massima), è stato deciso di impostare la versione minima a tre e la versione massima a undici.

3.3 Risultati

Dopo le modifiche sopra descritte il PE header viene modificato come segue:

Feature	Malware	Elusive
NumberOfSections	9	6
TimeDateStamp	1632748257	1634218530
SizeOfCode	876544	654848
SizeOfInitializedData	241152	432128
AddressOfEntryPoint	409775	118016
ImageBase	4194304	65536
MajorOperatingSystemVersion	6	9
MajorImageVersion	0	11
MinorImageVersion	0	3
MajorSubsystemVersion	6	9
CheckSum	0	1147526
SizeOfImage	1544192	1105920
SizeOfStackCommit	4096	124
SizeOfHeapReserve	1048576	792
SizeOfHeapCommit	4096	456

SectionsLength	9	6
SectionMinEntropy	0	1
SectionMinRawsize	0	512
SectionMinVirtualsize	265	480
SectionMaxPhysical	876274	654623
SectionMaxVirtual	1511424	1077248
SectionMaxPointerData	1081856	1057280
SectionMaxChar	3758096544	3221225536
ImageDirectoryEntryImport	1491856	1045260
ImageDirectoryEntryResource	1507328	1073152

L'obiettivo dell'attacco era ingannare il modello di classificazione utilizzato. Ciò è perfettamente riuscito dato che la versione elusiva del malware è classificata come tale con una probabilità del 2,03% rispetto alla versione base classificata al 99,86%.

Capitolo 4

Contromisure

In questo breve capitolo verranno trattate alcune contromisure all'attacco precedentemente descritto che permettono di rendere più robusto il modello ad attacchi di questo tipo.

4.1 Riduzione spazio delle feature

L'obiettivo di questa contromisura è duplice. In primo luogo è possibile ridurre la complessità della rappresentazione dei dati per limitare l'efficacia delle perturbazioni; in secondo luogo, nel nostro caso particolare, è possibile rimuovere le feature che più favoriscono la classificazione errata della versione elusiva. Sebbene queste tecniche funzionino bene nella prevenzione degli attacchi contraddittori, hanno l'effetto collaterale di peggiorare l'accuratezza del modello su esempi reali.

Nel dettaglio abbiamo testato la seconda tecnica partendo dall'insieme di feature per le quali la versione base e la versione elusiva del malware differiscono. A turno, abbiamo rimosso una feature dal dataset, addestrato un nuovo modello da zero e valutato le nuove predizioni. Per ogni feature la versione base è valutata sempre come malware con un'altissima confidenza (valore minimo 0.948 con la rimozione di *SizeOfHeapCommit*). Mentre, la versione elusiva è predetta sempre come non malware, con una confidenza simile al modello con tutte le feature, eccetto nel caso della rimozione di *MajorSubsystemVersion*. Con la rimozione di questa feature sia la versione base che la versione elusiva sono predette malware dal modello con confidenza pari a 1.00. Questa sicurezza ha portato qualche dubbio sulla correttezza del modello, ma osservando le metriche sopra descritte, tutte riportano valori maggiori di 0.95.

4.2 Data augmentation e adversarial training

Visti i risultati ottenuti dall'attacco FSGM al modello costruito, è bene pensare a dei metodi da utilizzare per rendere questo classificatore più resistente ad attacchi di questo tipo. Una prima possibilità potrebbe essere quella di ampliare il dataset per l'addestramento. In particolare, si potrebbe pensare di simulare l'attacco FSGM con tutti i malware contenuti nel dataset e aggiungere tutti i record così generati ad esso classificandoli come malevoli.

Come visto nel capitolo 3.1.1, l'attacco preso in considerazione porterà a costruire dei record contenenti valori negativi non verosimili in quanto si stanno trattando degli header di file PE. A questo proposito, per avere dei valori plausibili da poter inserire nel dataset di training, si può scegliere di sostituirli col valore 0, oppure ripristinare quelli antecedenti l'attacco.

Appendice A

Risultati FSGM

Feature	Malware	FSGM	Segno
e_magic	23117	23117.5	+
e_cblp	144	637.6	+
e_cb	3	-719.6	-
e_cric	0	-606.1	-
e_cparhdr	4	-428.2	-
e_minalloc	0	-457.9	-
e_maxalloc	65535	60979.7	-
e_ss	0	318.6	+
e_sp	184	-440.8	-
e_csum	0	507.6	+
e_ip	0	-913.5	-
e_cs	0	-571.3	-
e_lfarlc	64	922.6	+
e_ovno	0	-822.4	-
e_oemid	0	-1909.6	-
e_oeminfo	0	-2034.6	-
e_lfanew	272	307.1	+
Machine	332	6111.1	+
NumberOfSections	9	10.0	+

TimeDateStamp	1632748257	$1.8e+09$	+
PointerToSymbolTable	0	$6.7e+07$	+
NumberOfSymbols	0	$-6.0e+07$	-
SizeOfOptionalHeader	224	227.0	+
Characteristics	258	5009.1	+
Magic	267	223.6	-
MajorLinkerVersion	14	18.9	+
MinorLinkerVersion	29	20.1	-
SizeOfCode	876544	$2.7e+07$	+
SizeOfInitializedData	241152	$-3.0e+07$	-
SizeOfUninitializedData	0	$-1.5e+07$	-
AddressOfEntryPoint	409775	$-5.4e+07$	-
BaseOfCode	4096	$-2.6e+07$	-
ImageBase	4194304	$2.3e+11$	+
SectionAlignment	4096	4647.6	+
FileAlignment	512	-28.6	-
MajorOperatingSystemVersion	6	-457.0	-
MinorOperatingSystemVersion	0	406.3	+
MajorImageVersion	0	-629.6	-
MinorImageVersion	0	531.6	+
MajorSubsystemVersion	6	6.4	+
MinorSubsystemVersion	0	-466.1	-
SizeOfHeaders	1024	-5964.7	-
CheckSum	0	$-2.5e+08$	-
SizeOfImage	1544192	$-1.5e+07$	-
Subsystem	2	2.2	+
DllCharacteristics	33088	24889.3	-
SizeOfStackReserve	1048576	359825.0	-
SizeOfStackCommit	4096	-33903.5	-

SizeOfHeapReserve	1048576	752001.0	-
SizeOfHeapCommit	4096	-282483.0	-
LoaderFlags	0	$-3.1e+07$	-
NumberOfRvaAndSizes	16	$-7.4e+06$	-
SuspiciousImportFunctions	17	13.6	-
SuspiciousNameSection	0	-0.1	-
SectionsLength	9	8.0	-
SectionMinEntropy	0	-0.90	-
SectionMaxEntropy	0	0.5	+
SectionMinRawsize	0	252793.0	+
SectionMaxRawsize	0	-0.5	-
SectionMinVirtualsize	265	253126.0	+
SectionMaxVirtualsize	0	-0.5	-
SectionMaxPhysical	876274	$5.4e+06$	+
SectionMinPhysical	0	0.5	+
SectionMaxVirtual	1511424	$-9.0e+06$	-
SectionMinVirtual	0	-0.5	-
SectionMaxPointerData	1081856	$1.5e+08$	+
SectionMinPointerData	0	-0.5	-
SectionMaxChar	3758096544	$3.5e+094$	-
SectionMainChar	0	0.5	+
DirectoryEntryImport	5	8.8	+
DirectoryEntryImportSize	145	81.5	-
DirectoryEntryExport	0	77.5	+
ImageDirectoryEntryExport	0	$-1.1e+07$	-
ImageDirectoryEntryImport	1491856	$3.8e+06$	+
ImageDirectoryEntryResource	1507328	$-1.9e+06$	-
ImageDirectoryEntryException	0	$2.9e+07$	+
ImageDirectoryEntrySecurity	0	$1.1e+07$	+