INDIVIDUAZIONE, ACQUISIZIONE E REVERSE ENGINEERING DI UN CAPTATORE INFORMATICO (I PARTE)

In questo numero: 1. L'obiettivo, 2. Premessa metodologica, 3. La modalità di inoculazione, 4. L'individuazione del trojan.
Nel prossimo numero: 5. Acquisizione ed estrazione del captatore, 6.1 Attività di reverse engineering, 6.2 Analisti statica del trojan, 6.3 Analisi in modalità di esecuzione del trojan.

Paolo REALE, consulente nell'ambito dell'ICT ed esperto in Digital Forensics, Presidente dell'Osservatorio Nazionale di Informatica Forense e Presidente della commissione ICT dell'Ordine Ingegneri di Roma.

Fabio ZITO, laureato in Giurisprudenza, attualmente iscritto al Master Cybersecurity, Digital Forensics and Data Protection, da alcuni anni svolge attività di consulente informatico, lavorando su diversi casi, alcuni anche mediaticamente rilevanti, sia nel campo civile che penale.

1. L'obiettivo

In un recente caso giudiziario, l'elemento accusatorio principale era costituito da intercettazioni ambientali effettuate utilizzando un captatore informatico, inoculato su un iPhone di proprietà di un imputato. Nell'ambito delle indagini difensive, ci è stato chiesto - se possibile - di individuare le modalità di inoculazione di tale captatore, l'applicazione stessa utilizzata come trojan e le informazioni di connessione per indentificare i server con il quale scambiava i dati.

Benché l'argomento sia ampiamente trattato sotto il profilo giuridico, nella letteratura tecnica non sono molti i riferimenti bibliografici in merito all'individuazione e al funzionamento dei captatori informatici. Per questo abbiamo deciso di proporre in questo articolo più specialistico, in ottica forense, il metodo e i risultati del lavoro svolto. Per dovere di riservatezza, ogni elemento di possibile identificazione di persone, luoghi o aziende è stato omesso o oscurato.

2. Premessa metodologica

Le metodologie e gli strumenti che consentono la ricostruzione del funzionamento di un software attraverso la tecnica del *reverse engineering* possono essere molteplici e spesso dipendono sia dalla tipologia del software in esame (es. programma compilato vs programma interpretato), sia dal sistema operativo, dal tipo di architettura, dal linguaggio di programmazione, etc.

Per non appesantire la lettura con eccessivi dettagli tecnici, si elencheranno qui di seguito le metodologie e gli strumenti utilizzati durante questa attività. In particolare, essendo il captatore un file eseguibile (compilato per processori con architettura ARM[1] a 64 bit) in cui le istruzioni sono codificate in codice binario è stato necessario prima di tutto convertire tali istruzioni in *assembly*, che è un linguaggio di programmazione molto simile al linguaggio macchina, ma che risulta comunque più comprensibile all'uomo. Per fare questo si sono utilizzati i seguenti strumenti: Otool, Class-dump, MachOView e Hopper[2].

Inoltre, per meglio comprendere sia il funzionamento che il valore di alcune variabili individuate nel codice disassemblato, è stato necessario "eseguire" direttamente il trojan con l'ausilio del software di debugging *lldb*[3]. Tale applicativo, semplificando molto, consente - tra le altre cose - di:

1) "lanciare" un'applicazione è "bloccarla" in un punto specifico, impostando quelli che tecnicamente vengono definiti *breakpoint*;
2) eseguire l'applicazione "bloccata" un'istruzione alla volta (passo-passo);
3) leggere il contenuto memorizzato all'interno dei registri del processore[4], ogni qualvolta viene eseguita un'istruzione.

Attraverso queste operazioni è possibile ricostruire "passo passo" l'esecuzione dell'applicativo e verificare quali sono le operazioni che compie, in modo da comprenderne la logica e gli effetti. Si tratta, come facilmente intuibile, di un procedimento laborioso e *time-comsuming*, ma in totale assenza di informazioni in merito al funzionamento dell'applicativo, si è ritenuto che questo approccio costituisse l'unico utile alla ricostruzione del funzionamento effettivo del trojan.

3. La modalità di inoculazione

Per individuare il captatore si è partiti, *in primis*, dalla ricerca dell'elemento di innesco dello stesso, qui costituito da un 'SMS spoofing[5]' contenente la 'trappola' informatica destinata a consentire lo scaricamento e l'attivazione del trojan. Individuato l'SMS in questione, oltre alla data e l'ora è stato indentificato anche il link (da cui con ogni probabilità e stato poi scaricato il captatore). Da una successiva ricerca di quest'ultimo sulla *Web History* del dispositivo analizzato, si è potuto constatare che tale link è stato effettivamente cliccato qualche se-

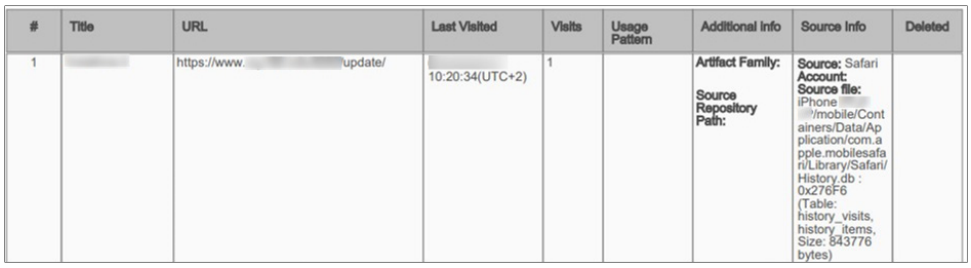
1 L'architettura ARM, in elettronica e informatica, indica una famiglia di microprocessori RISC a 32-bit e 64-bit sviluppata da ARM

2 Nella versione demo

3 https://lldb.llvm.org/

4 Piccola parte di memoria utilizzata per velocizzare l'esecuzione dei programmi fornendo un accesso rapido ai valori usati più frequentemente e/o tipicamente (https://it.wikipedia.org/wiki/Registro_(informatica))

5 https://en.wikipedia.org/wiki/SMS_spoofing



#	Title	URL	Last Visited	Visits	Usage Pattern	Additional Info	Source Info	Deleted
1		https://www. update/	10:20:34(UTC+2)	1		Artifact Family: Source Repository Path:	Source: Safari Account: Source file: iPhone /mobile/Containers/Data/Application/com.apple.mobilesafari/Library/Safari/History.db : 0x276F6 (Table: history_visits, history_items, Size: 843776 bytes)	

Figura 1 - Estratto del web history da cui si evince il click del link

condo dopo la ricezione dell'SMS (**figura 1**) e - come vedremo anche nel prossimo paragrafo - la consequenzialità temporale è uno degli elementi che ha permesso di confermare che l'applicazione individuata fosse il trojan utilizzato per le intercettazioni ambientali.

4. L'individuazione del trojan

Al fine di meglio comprendere i passaggi tecnici successivi, risulta utile una breve digressione sulle modalità di installazione di un'app su un dispositivo iOS, nel caso in cui l'app in questione non risieda sul noto "Apple Store". Apple, infatti, limita l'installazione diretta delle applicazioni al di fuori del suo *store* ufficiale attraverso due modalità:

1. la prima è generalmente utilizzata dagli sviluppatori per testare le proprie applicazioni e consiste sostanzialmente nella generazione con il proprio account "sviluppatore" -attraverso Xcode[6]- di un file con estensione *.mobileprovisioning*, conosciuto come *provisioning profile* o più semplicemente *provisioning*, e un certificato digitale che permette l'installazione dell'applicazione su un dispositivo Apple, superando i controlli di sicurezza. La procedura appena descritta ha però due limiti. Il primo è la necessità di disporre fisicamente del dispositivo, in quanto per procedere con l'installazione è necessario collegarlo direttamente al PC, il secondo nella versione *free*, è la breve durata del certificato, infatti dopo una settimana scade, e l'applicazione non è più disponibile sul dispositivo.
1. La seconda, denominata "*Apple Developer Enterprise Program*[7]", è utilizzata dalle aziende per distribuire le proprie applicazioni "*in-house*" (p.es. direttamente ai propri dipendenti, clienti, agenti, etc.) in modo che le stesse non siano di dominio pubblico, quindi senza dover utilizzare l'App Store. Questa seconda modalità non "soffre" dei limiti[8] visti in quella precedente, ed è proprio la violazione di una della policy prevista nelle norme contrattuali del "*Apple Developer Enterprise Program*" - cioè quella che proibisce espressamente di distribuire app iOS aziendali proprietarie ai normali utenti - che viene utilizzata come 'escamotage' per installare applicazioni non desiderate (malware, virus, trojan, etc.) su dispositivi di soggetti terzi. Lo "strumento" che consente di superare i controlli di sicurezza necessari per procedere con l'installazione di una applicazione - attraverso la modalità appena descritta - è sempre l'utilizzo di un file di *provisioning*, e un certificato, ma quest'ultimo - invece di essere generato da un account "sviluppatore"- è "rilasciato" dalla Apple agli account di tipo *Enterprise* (generalmente di proprietà delle aziende medio/grandi) che ne fanno richiesta in modo oneroso. Questo consente di "firmare" il pacchetto di installazione rendendolo valido, quindi scaricabile dalla rete e installabile[9].

Nel nostro caso, nella directory *ProvisioningProfiles* (**figura 2**) del report generato da *Physical Analyzer* (*Cellebrite*) del dispositivo analizzato, è stato individuato il file di *provisioning profile*[10] installato in corrispondenza

6 Ambiente di sviluppo integrato, completamente sviluppato e mantenuto da Apple, contenente una suite di strumenti utili allo sviluppo di software per i sistemi macOS, iOS, watchOS e tvOS (https://it.wikipedia.org/wiki/Xcode)

7 https://developer.apple.com/programs/enterprise/

8 Per il certificato di tipo Enterprise è prevista una scadenza annuale.

9 Per completezza di informazione, si aggiunge che entrambe le modalità prevedono, alla prima apertura dell'app, la necessità di autorizzare lo sviluppatore e/o l'azienda indicati nel file di provisioning (fig. 12). Questa operazione deve essere condotta manualmente, tramite alcune operazioni tipicamente non di semplice attuazione per l'utente medio, per cui -al fine di garantire l'effettiva inoculazione- l'attività di installazione può essere "guidata" in modo puntuale da un "finto" operatore.

10 Lo stesso file, come vedremo nel pros-

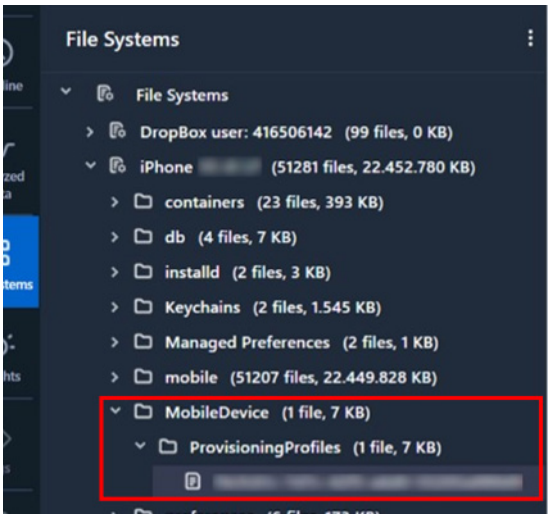


Figura 2 - Directory "*ProvisioningProfiles*"

della stessa data e ora (**figura 3**) di ricezione dell'SMS e della successiva apertura del link così come descritto nel precedente paragrafo.

Detto file è stato quindi analizzato con l'ausilio di un semplice editor di testo, potendo conoscere quindi il nome dell'App, il gruppo di sviluppo, il certificato, l'azienda a cui è assegnato il certificato stesso etc. Come si vede chiaramente dagli estratti riportati di seguito (**figura 4**) il file di *provisioning* non nient'altro che un file XML dove le informazioni racchiuse tra il *root tag plist*, sono organizzate attraverso coppie di elementi formati da chiave-valore (quest'ultimo è anche tipizzato), ad esempio la chiave *TimeToLive* è valorizzata con l'intero *365*, a significare che tale file avrà una durata di 365 giorni.

Tra le chiavi è stata evidenziata in rosso la chiave *ProvisionsAllDevices* (valorizzata a *true*), in quanto tale chiave è presente solo nei file di provisioning generati con account di tipo *Enterprise*, cioè quelli in cui il certificato è rilasciato dal "*Apple Developer Enterprise Program*", mentre nei file di *provisioning* generati con

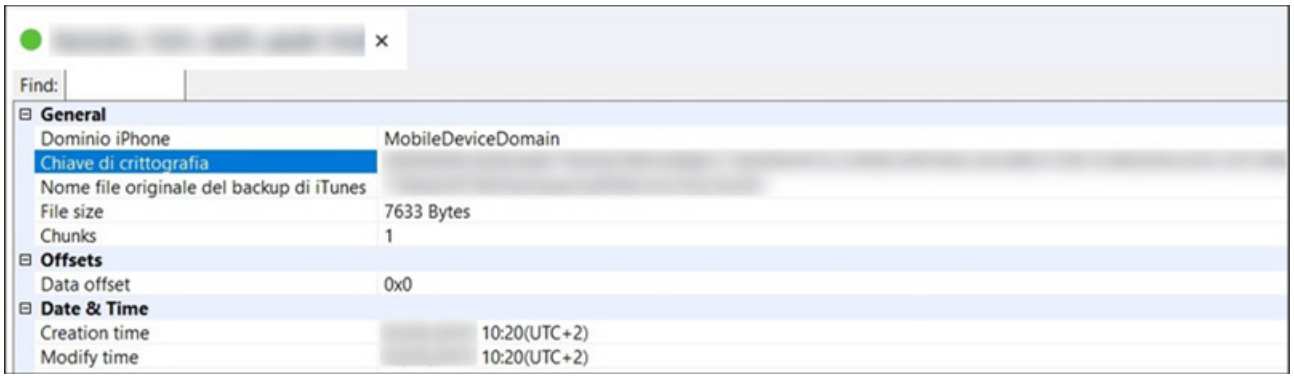


Figura 3 - Estratto del mobile provisioning dove si evidenzia la data e l'ora di installazione

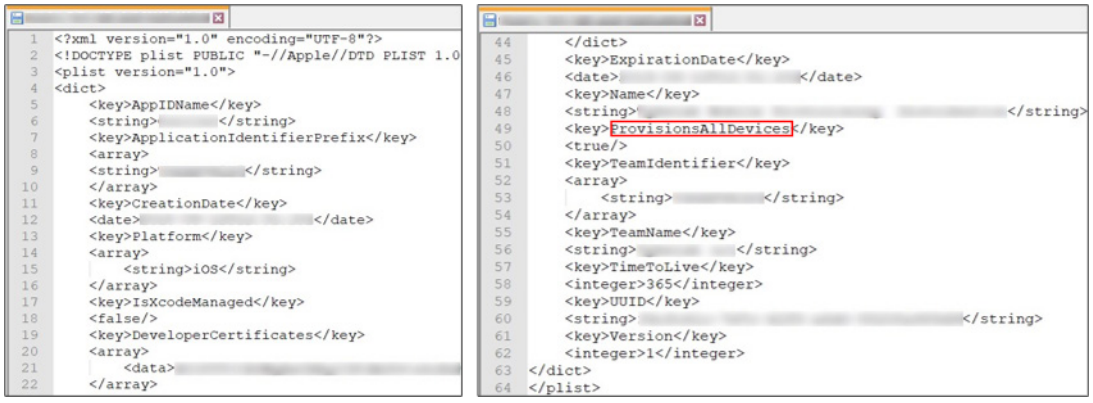


Figura 4 - Dettagli del certificato di provisioning dell'app individuata come probabile trojan

simo paragrafo, è presente nei pacchetti delle applicazioni iOS (.app o se compressi .ipa) come embedded.mobileprovision.

account "sviluppatore" tale chiave non è presente ed è "sostituita" da *ProvisionedDevices* la quale è valorizzata con un array contenete la

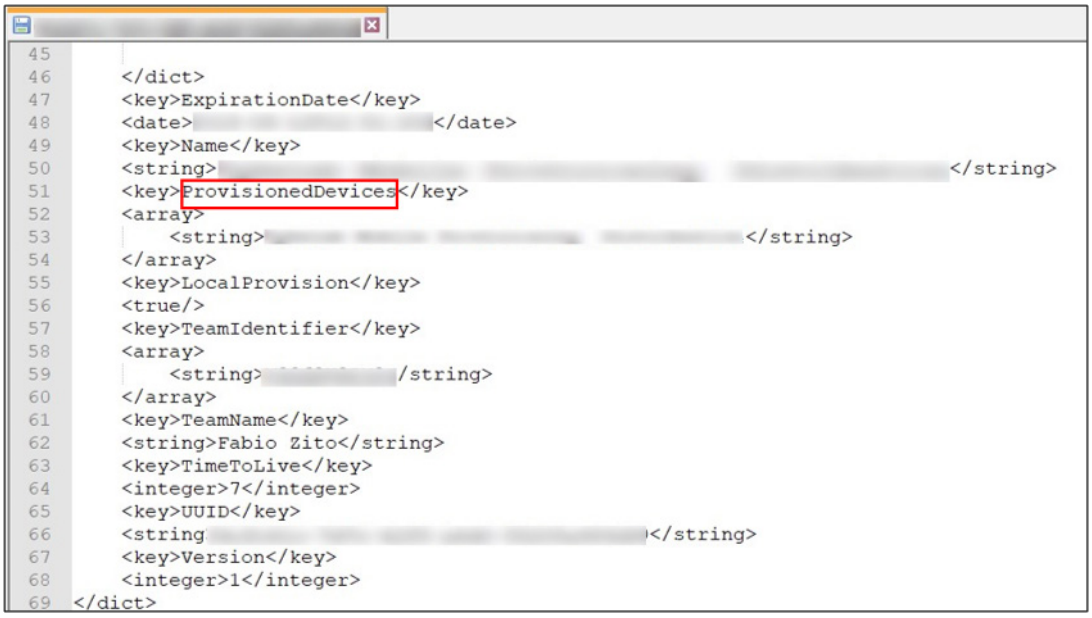


Figura 5 - Dettaglio di file di provisioning sviluppatore dove è presente la chiave ProvisionedDevices

lista degli UUID dove l'applicazione può essere installata (vedi **figura 5**).

Successivamente, è stata effettuata una ricerca tra le applicazioni installate sullo smartphone in esame utilizzando come chiave di ricerca il nome dell'applicazione individuato nel file di *provisioning*. Il risultato di tale ricerca è quello mostrato in **figura 6** nella quale è stato evidenziato in rosso l'autorizzazione dell'accesso al microfono del cellulare da parte della suddetta applicazione.

Con tale autorizzazione è stata concessa la gestione concreta del microfono, rendendolo 'attivabile' in modo indipendente dalla consapevolezza dell'utilizzatore del cellulare.

Gli artefatti così individuati in questa prima parte di analisi, in particolare:

1. l'individuazione di una applicazione installata non utilizzando lo Store ufficiale di Apple, ma attraverso l'utilizzo del "*Apple Developer Enterprise Program*";
2. la data e l'ora di memorizzazione del file di *provisionig profile* associato alla suddetta applicazione coincidente sia con la ricezione dell'SMS, che con l'apertura del link da cui probabilmente e stato effettuato il download;
3. L'autorizzazione all'accesso al microfono da parte dell'applicazione che di fatto lo rendeva attivabile in modo indipendente dalla volontà dell'utilizzatore del cellulare;

hanno fatto ritenere di aver individuato con ragionevole certezza l'App come "captatore". ©

#	Name	Description	Timestamp	Copyright	Permissions	Alias names	App usage	App categories	Deleted
1	Application ID: com.ios. Source:	Version: Operation Mode: Foreground Application Size (bytes): 0 Source file: TarArchive/Backup/Manifest.plist : 0x125C7 (Size: 111830 bytes) iPhone /mobile/Library/TCC/TCC.db : 0x1151B (Table: access, Size: 98304 bytes) Artifact Family: Source Repository Path:	Purchase Date Install Date Last Modified Deleted Date		Microphone		Launches Activations Last Launch Active Time 00:00:00 Background Time 00:00:00	Categories	

Figura 6 - Autorizzazione al microfono concessa all'app individuata sul dispositivo analizzato