AUDITER LA SÉCURITÉ D'UNE APPLICATION iOS AVEC NEEDLE

RandoriSec - Davy DOUHINE (@ddouhine)

mots-clés : NEEDLE / iOS / AUDIT / APPLICATION / MOBILE

Needle [needle] (aiguille en anglais) est un cadriciel (framework) open source qui accélère considérablement les analyses orientées sécurité des applications iOS. Conçu par Marco Lancini de la société MWR et présenté lors de l'édition 2016 de Black Hat Vegas, il prend une place laissée vacante jusqu'à maintenant. Sans lui les testeurs étaient obligés de s'armer de tout un tas d'outils disposant chacun de sa syntaxe, de son mode opératoire et de ses limitations. Avec Needle de très nombreuses actions peuvent maintenant être lancées depuis une interface unique, avec une syntaxe et un esprit modulaire « à la metasploit » qui plus est !

1 Pas de bonne trousse à outils sans aiguille

L'installation très bien documentée **[installation]** se fait rapidement et sans douleur sur Kali et OS X et ne mérite donc pas de s'étendre dessus. Elle consiste à installer les quelques dépendances comme python, sqlite, frida ou sshtunnel puis à faire un git clone. Un docker devrait même être bientôt disponible, ce qui laisse peu d'excuses pour ne pas l'essayer.

Les habitués de Metasploit ne seront pas dépaysés, la structure modulaire reprend le même principe : on sélectionne un module (**use xxx**), on spécifie une cible (ici une application plutôt qu'un hôte dans Metasploit), on fournit éventuellement une ou plusieurs options (**set OPTION1 xxx**), on supprime une option (**unset OPTION2**) et on lance le tout (**run**). Il est également possible de lister les tâches qui tournent en arrière-plan (**job**) puis de les arrêter (**kill**).

Certaines commandes basiques, mais essentielles empruntent la même syntaxe : **pull/push** pour rapatrier ou déposer des fichiers, **shell** pour obtenir un shell sur le terminal, **search** pour chercher une chaîne de caractères dans les modules (pratique pour retrouver rapidement un module) et les commandes sont historisées depuis la version 0.2.0 : la flèche du haut rappellera les dernières commandes.

```
needle — sshpass ◂ Python — 80×24
.github          CHANGELOG.md     README.md
Davys-MBP:needle root# cd needle/
Davys-MBP:needle root# python needle.py

                Needle v0.2.0 [mwr.to/needle]
   [MWR InfoSecurity (@MWRLabs) - Marco Lancini (@LanciniMarco)]

[needle] >
[needle] >
[needle] >
[needle] > shell
[*] Spawning a shell...
[*] Checking connection with device...
[V] Connection not present, creating a new instance
[V] Setting up USB port forwarding on port 2222
[V] Setting up SSH connection...
[+] Connected to: 127.0.0.1
Warning: Permanently added '[127.0.0.1]:2222' (RSA) to the list of known hosts
iPad-de-davy:~ root#
iPad-de-davy:~ root#
```

Figure 1 : Lancement de Needle !

2 Les étapes de l'analyse d'une application

Avant de commencer à aiguillonner, rappelons les grandes étapes d'une analyse d'application iOS : analyse de code (encore faut-il en disposer), analyse des données (les préférences utilisateur, cookies, caches, etc.), contournement des fonctions de sécurité s'il y en a (détection de jailbreak, code pin, etc.) puis finalement interception des communications réseau.

Chaque étape demandera l'utilisation de techniques et d'outils particuliers. La plupart de ces opérations ne seront donc réalisables que sur un terminal jailbreaké de manière à pouvoir bénéficier d'allègement des fonctions de sécurité ainsi que des logiciels nécessaires qui ne sont pas disponibles sur le store officiel.

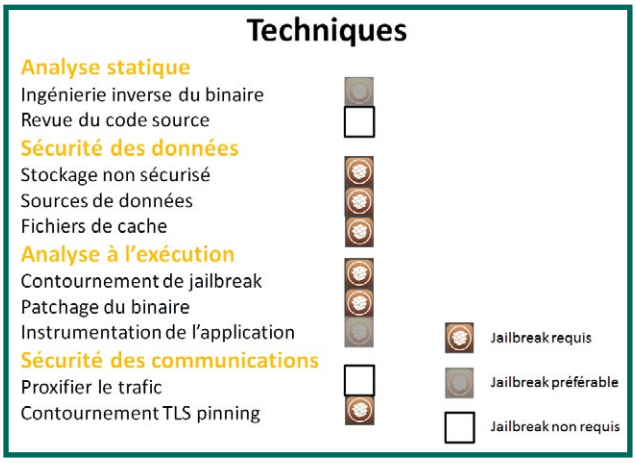

Figure 2 : Techniques et leur besoin de jailbreak.

Sur la figure 2, les étapes principales sont listées et pour chacune est indiqué le besoin de disposer d'un terminal jailbreaké ou non.

On voit que c'est surtout ce qui concerne l'analyse des communications qui est réalisable sans jailbreak et encore cela est impossible si l'application utilise le *certificate pinning*, ce qui est de plus en plus fréquent.

En effet, cette analyse va généralement commencer par la mise en place d'un proxy intercepteur (comme Burp Suite **[burpsuite]**) qui introduira fatalement une rupture SSL/TLS que l'application auditée ne tolèrera pas, car elle s'attendra à recevoir un certificat bien particulier et sans lui elle stoppera la communication.

3 Environnement de test

On vient de voir qu'il était préférable d'avoir sous la main un terminal jailbreaké pour réaliser un audit de sécurité. Pour Needle, c'est indispensable et c'est un prérequis parfois difficile à remplir : tous les terminaux (tablettes ou smartphones) et toutes les versions d'iOS ne sont pas débridables.

Il suffit de consulter la figure 3 (provenant du site **[iphonewiki]**), qui liste les jailbreaks disponibles pour iOS 9, pour s'en rendre compte.

Pour corser le tout, certaines dépendances de l'outil ne sont pas compatibles avec iOS 9 ou supérieur et d'autres encore ne fonctionnent pas avec les plateformes 64bits. La version `0.2.0` de Needle, sortie en février 2017, introduit un premier support d'iOS 10 et la version `1.0.0` sortie en mars 2017 l'améliore encore en intégrant NeedleAgent, une application iOS à installer

iOS	Jailbreak Tool	Tool Version	Device																						
			iPad 2	iPad 3	iPad 4	iPad Air	iPad Air 2	iPad Pro (12.9 inch)	iPad Pro (9.7 inch)	iPad mini	iPad mini 2	iPad mini 3	iPad mini 4	iPhone 4S	iPhone 5	iPhone 5c	iPhone 5s	iPhone 6	iPhone 6 Plus	iPhone 6s	iPhone 6s Plus	iPhone SE	iPod touch 5G	iPod touch 6G
9.0	Pangu9 for 9.0-9.1	1.0.0-1.3.2 (Windows)/1.0.0-1.1.1 (Mac)	Yes							Yes													Yes	
9.0.1	Pangu9 for 9.0-9.1	1.0.0-1.3.2 (Windows)/1.0.0-1.1.1 (Mac)	Yes							Yes													Yes	
9.0.2	Pangu9 for 9.0-9.1	1.0.0-1.3.2 (Windows)/1.0.0-1.1.1 (Mac)	Yes							Yes													Yes	
9.1	Pangu9 for 9.0-9.1	1.3.0-1.3.2 (Windows)/1.1.0-1.1.1 (Mac)	No			Yes				No	Yes			No			Yes						No	Yes
	Home Depot	Rev 1 - Rev 7	Yes	No					N/A	No				Partial[9.x 1]		No						N/A	Yes	No
9.2	Pangu9 for 9.2-9.3.3[9.x 7]	1.0.0-1.1.0	No			Yes				No	Yes			No			Yes						No	Yes
	Home Depot	Rev 1 - Rev 7	Yes	No						No				Yes	Partial[9.x 1]	No							No	
9.2.1	Pangu9 for 9.2-9.3.3[9.x 2]	1.0.0-1.1.0	No			Yes				No	Yes			No			Yes						No	Yes
	Home Depot	Rev 1 - Rev 7	Yes	No	Yes	No				Partial[9.x 3]	No			Yes	Partial[9.x 1]	Partial[9.x 4]	No						No	
9.3	Pangu9 for 9.2-9.3.3[9.x 2]	1.0.0-1.1.0	No			Yes				No	Yes			No			Yes						No	Yes
	Home Depot	Rev 1 - Rev 7	Yes	No										Yes	No									
9.3.1	Pangu9 for 9.2-9.3.3[9.x 2]	1.0.0-1.1.0	No			Yes				No	Yes			No			Yes						No	Yes
	Home Depot	Rev 1 - Rev 7	Yes	Partial[9.x 5]		No								Yes	No									
9.3.2	Pangu9 for 9.2-9.3.3[9.x 7]	1.0.0-1.1.0	No			Yes				No	Yes			No			Yes						No	Yes
	Home Depot	Rev 1 - Rev 7	Yes	Partial[9.x 6]		No				Partial[9.x 3]	No			Yes	Partial[9.x 1]	Yes	No						Yes	No
9.3.3	Pangu9 for 9.2-9.3.3[9.x 2]	1.0.0-1.1.0	No			Yes				No	Yes			No			Yes						No	Yes
	Home Depot	Rev 1 - Rev 7	Yes	Partial[9.x 6]		No				Partial[9.x 3]	No			Yes			No						Yes	No
9.3.4	Home Depot	Rev 1 - Rev 7	Yes	Partial[9.x 7]		No								Yes	Partial[9.x 4]		No						Yes	No
9.3.5	N/A	N/A	No																					

Figure 3 : Jailbreaks pour iOS9.

sur le terminal qui permet de s'affranchir des différents problèmes de compatibilité et de dépendances en intégrant directement certaines fonctionnalités comme celles de class_dump. C'est toutefois la version 0.2.0, plus stable, qui a été utilisée pour cet article.

Mais le casse-tête ne s'arrête pas là, il faut également prendre en compte la version d'iOS minimale requise pour faire fonctionner l'application à auditer, souvent iOS 8 aujourd'hui. Il faut donc se tourner vers un terminal et une version d'iOS suffisamment récents pour qu'ils puissent lancer l'application à auditer, mais pas trop non plus pour pouvoir disposer d'un jailbreak correct et des dépendances qui vont bien. Si vous trouvez un terminal dans une version débridable d'iOS 8 vous serez donc dans la meilleure configuration possible, jusqu'à ce que les applications requièrent iOS 9 minimum, soit probablement d'ici quelques mois.

Une fois Needle installé et le terminal jailbreaké relié à l'ordinateur (par USB ou alors connecté au même réseau Wifi), Needle va pouvoir commencer à picoter.

4 Analyses

4.1 L'analyse statique

Si le code source de l'application est fourni, la recherche de chaînes de caractères spécifiques peut s'avérer très utile, car elle peut dévoiler de nombreuses informations permettant d'aider l'auditeur à découvrir certaines vulnérabilités. On peut citer par exemple :

- la mise en cache accidentelle de ressources ou de frappes clavier ;
- le stockage d'identifiants en clair ou dans des fichiers non protégés ;
- la présence de serveurs « back-end » non détectés lors d'une analyse dynamique de l'application ;
- la mauvaise gestion du presse-papier (est-il vidé lorsque l'application est fermée ? est-il de type privé pour ne pas être accessible par les autres applications ?) ;
- l'utilisation d'une base SQL, pouvant mener à des injections SQL ;
- l'utilisation de la classe **UIWebView** (permettant d'inclure du contenu web) pouvant mener à des injections de code de type XSS ;
- la journalisation trop verbeuse ;
- l'appel à des fonctions C comme strcat, strcpy, sprintf, fopen, etc.

Le module **static/code_checks** facilite la tâche de l'auditeur et automatise toutes ces recherches.

Bien sûr il est assez rare de disposer du code source de l'application, heureusement dans ce cas quelques modules peuvent, à partir de l'application installée sur le terminal, aider l'auditeur dans la compréhension du fonctionnement et de la configuration de l'application.

Le module **binary/info/metadata** est sans doute le premier à exécuter lorsque l'on se lance dans l'analyse d'une application. Il va récupérer plusieurs informations essentielles comme la version, l' **UUID**, le chemin du binaire, les répertoires d'installation du bundle et des données, les « url handlers », etc.

Au premier lancement d'un module, si aucune application cible n'a été préalablement sélectionnée (avec **set APP xxx**), un « wizard » va se charger de rapatrier la liste des applications installées par l'utilisateur (grâce au fichier **/private/var/installd/Library/MobileInstallation/LastLaunchServicesMap.plist** pour iOS 9) et l'afficher à l'écran pour sélection. Une fois la sélection faite, les informations sont rapatriées puis affichées à l'écran.

Il est utilisé ici sur l'application Skype for Business (voir figure 4).

Ensuite l'ensemble des autres actions sera réalisé par défaut sur la même application. Pour changer de cible, il suffit de remonter d'un niveau (**back**) puis de choisir l'identifiant de l'application (**Bundle ID**) avec **set APP xxx** et de repartir sur le module de son choix comme par exemple **binary/info/compilation_checks** qui va vérifier si quelques options de sécurité (chiffrement du binaire, stack canaries, **ARC**, **PIE**) sont positionnées pour l'application **[iicontact]**.

```
[needle][metadata] > use binary/info/compilation_checks
[+] Resource file successfully loaded
[needle][compilation_checks] > run
[*] Checking connection with device...
[+] Already connected to: 127.0.0.1
[V] Creating temp folder: /var/root/needle/
[+] Target app:  net.iicontact.mobileapp
[V] Analyzing binary...
[+] arm64
[+]             Encrypted: OK
[+]        Stack Canaries: OK
[+]                   ARC: OK
[+]                   PIE: OK
[needle][compilation_checks] > [needle][metadata] > run
[*] Executing Local Command: [needle][metadata] > run
```

Le module **binary/reversing/shared_libraries** va lister les bibliothèques utilisées.

```
[needle][shared_libraries] > run
[*] Checking connection with device...
[+] Already connected to: 127.0.0.1
[V] Creating temp folder: /var/root/needle/
[+] Target app: com.microsoft.lync2013.iphone
[V] Analyzing binary for dynamic dependencies...
  /private/var/containers/Bundle/Application/96842139-F5C8-4F38-
B2D1-8DC7EAFFD446/SfB.app/SfB:
  @rpath/Model.framework/Model (compatibility version 1.0.0,
current version 1.0.0)
  @rpath/HockeySDK.framework/HockeySDK (compatibility version
1.0.0, current version 1.0.0)
  @rpath/IPA.framework/IPA (compatibility version 1.0.0, current
version 1.0.0)
```

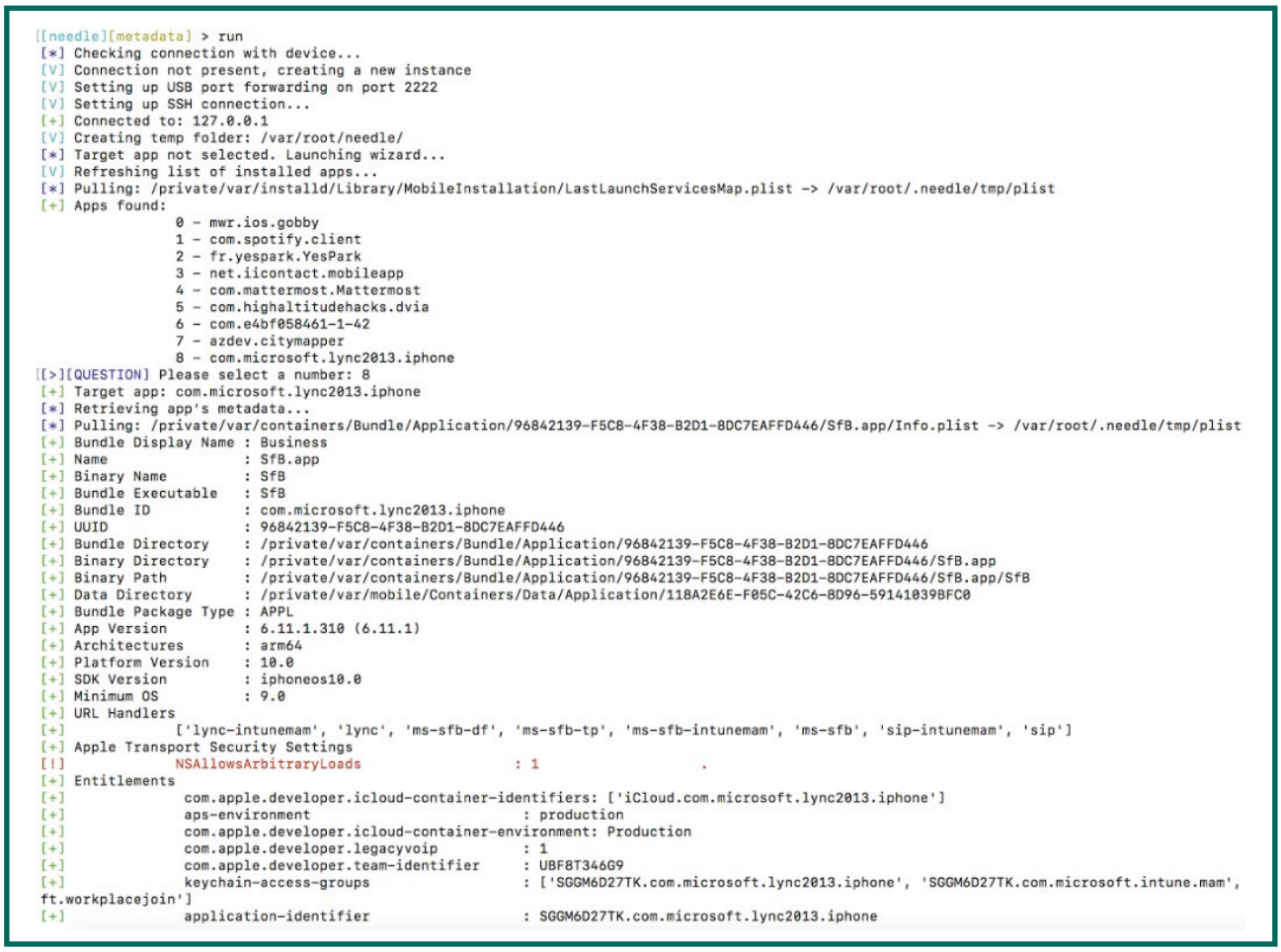

Figure 4 : Extrait de la sortie du module binary/info/metadata.

```
  @rpath/ADAL.framework/ADAL (compatibility version 1.0.0, current
version 1.0.0)
  /System/Library/Frameworks/PushKit.framework/PushKit
(compatibility version 1.0.0, current version 1.0.0)
  /usr/lib/libz.1.dylib (compatibility version 1.0.0, current
version 1.2.8)
  /usr/lib/libresolv.9.dylib (compatibility version 1.0.0, current
version 1.0.0)
  /usr/lib/libsqlite3.dylib (compatibility version 9.0.0, current
version 252.0.0)
  /usr/lib/libxml2.2.dylib (compatibility version 10.0.0, current
version 10.9.0)
  /System/Library/Frameworks/CoreSpotlight.framework/CoreSpotlight
(compatibility version 1.0.0, current version 1.0.0)
  /System/Library/Frameworks/LocalAuthentication.framework/
LocalAuthentication (compatibility version 1.0.0, current version
240.1.5)
```

Le module **binary/reversing/strings** va chercher dans les ressources de l'application ainsi que le binaire, préalablement déchiffré, les chaînes de caractères ainsi que les URI.

Une différence notable par rapport à un **strings** Unix/Linux: ici, par défaut, seules les chaînes de dix caractères ou plus seront affichées (contrairement à quatre), mais le nombre de caractères minimum est configurable avec l'option **LENGTH**.

L'option **FILTER** permet, comme son nom l'indique, de filtrer la sortie. Ici nous cherchons à titre d'exemple la chaîne « jailb » pour déterminer si l'application procède à une vérification de l'intégrité du terminal.

```
[+] Resource file successfully loaded
[needle][strings] > set FILTER jailb
FILTER => jailb
[needle][strings] > run
[*] Checking connection with device...
[+] Already connected to: 127.0.0.1
[V] Creating temp folder: /var/root/needle/
[+] Target app: com.microsoft.lync2013.iphone
[*] Decrypting the binary...
[?] The app might be already decrypted. Trying to retrieve the IPA...
[*] Unpacking the IPA...
[V] Analyzing binary...
[V] Analyzing resources...
[+] The following strings have been found:
  gThis app cannot be used because you are using a jailbroken device.
Contact your IT department for help._
  Impossible d'utiliser cette application, car vous utilisez un
appareil jailbroken. Contactez votre service informatique pour
obtenir de l'aide._
[*] Saving output to file: /var/root/.needle/output/strings
[*] Analyzing strings (press any key to continue)...
[needle][strings] >
[needle][metadata] > use binary/reversing/strings
```

Le module **binary/reversing/class_dump** va simplement lancer l'outil class_dump sur le binaire pour retrouver les interfaces (voir encart). Ce qui peut aider dès lors que l'on s'intéresse de près au fonctionnement d'une application et/ou que l'on souhaite contourner certaines de ses fonctions de sécurité comme la détection de jailbreak ou la saisie d'un code PIN.

Malheureusement la version de class_dump fournie ne supporte pas les applications 64bits. Un contournement possible est d'utiliser le module **hooking/frida/script_enum_all-methods** ou tout simplement de récupérer le fichier ipa (l'archive de l'application) avec **binary/installation/pull_ipa** pour ensuite utiliser une version plus récente de class_dump sur sa machine.

Ici l'opération est lancée sur Damn Vulnerable iOS Application **[dvia]**, une application à visée éducative.

Le fichier ipa est récupéré.

```
[needle] > use binary/installation/pull_ipa
[needle][pull_ipa] > run
[+] Target app: com.highaltitudehacks.dvia
[*] Retrieving app's metadata...
[*] Pulling: /private/var/containers/Bundle/Application/45ED9C82-
8F09-416A-836F-F40128F41FEF/DamnVulnerableIOSApp.app/Info.plist ->
/var/root/.needle/tmp/plist
[*] Decrypting the binary...
[?] The app might be already decrypted. Trying to retrieve the
IPA...
[*] Unpacking the IPA...
[*] Pulling: /var/root/needle/decrypted.ipa -> /var/root/.needle/
output/app.ipa
[needle][pull_ipa] >
```

Puis class_dump est lancé.

```
Davys-MacBook-Pro:~ root# ./class-dump -H -o DVIA_class_dump/
DamnVulnerableIOSApp
2017-03-02 19:56:33.030 class-dump[13350:6432470] Warning: Parsing
method types failed, setTable:
2017-03-02 19:56:33.031 class-dump[13350:6432470] Warning: Parsing
method types failed, table
2017-03-02 19:56:33.034 class-dump[13350:6432470] Warning: Parsing
method types failed, getOrCreateGroup
2017-03-02 19:56:33.035 class-dump[13350:6432470] Warning: Parsing
instance variable type failed, _group
2017-03-02 19:56:34.077 class-dump[13350:6432470] Warning: Parsing
method types failed, getOrCreateGroup
Davys-MacBook-Pro:~ root# ls DVIA_class_dump/ | more
AppDelegate.h
AppleWatchFirstChallengeViewController.h
AppleWatchViewController.h
ApplicationPatchingDetailsVC.h
ApplicationPatchingVC.h
AttackingFlurryViewController.h
AttackingGAViewController.h
AttackingParseViewController.h
AttackingThirdPartyLibrariesTableViewController.h
BFAppLink.h
BFAppLinkNavigation.h
BFAppLinkResolving-Protocol.h
BFAppLinkReturnToRefererController.h
BFAppLinkReturnToRefererView.h
BFAppLinkReturnToRefererViewDelegate-Protocol.h
BFAppLinkTarget.h
BFExecutor-Background.h
BFExecutor.h
```

Pour finalement analyser les en-têtes récupérées.

```
Davys-MacBook-Pro:~ root# grep --color -iE "jailb"  DVIA_class_
dump/*
DVIA_class_dump/ApplicationPatchingDetailsVC.h:- (void)
jailbreakTestTapped:(id)arg1;
DVIA_class_dump/DamnVulnerableAppUtilities.h:+ (void)
showAlertForJailbreakTestIsJailbroken:(_Bool)arg1;
DVIA_class_dump/FlurryUtil.h:+ (BOOL)deviceIsJailbroken;
DVIA_class_dump/JailbreakDetectionVC.h:@interface
JailbreakDetectionVC : UIViewController
DVIA_class_dump/JailbreakDetectionVC.h:- (_Bool)isJailbroken;
DVIA_class_dump/JailbreakDetectionVC.h:- (void)
jailbreakTest2Tapped:(id)arg1;
DVIA_class_dump/JailbreakDetectionVC.h:- (void)
jailbreakTest1Tapped:(id)arg1;
DVIA_class_dump/PFDevice.h:@property(readonly, nonatomic,
getter=isJailbroken) _Bool jailbroken;
DVIA_class_dump/SFAntiPiracy.h:+ (_Bool)isTheDeviceJailbroken;
DVIA_class_dump/SFAntiPiracy.h:+ (int)isJailbroken;
```

Ces en-têtes (voir encart « Interfaces/méthodes ») seront utiles pour identifier les objets utilisés, en déduire les fonctions de sécurité et finalement tenter de les contourner.

Interfaces/méthodes

Objective-C, un langage orienté objet qui date des années 80 créé à l'époque de la société NeXT (la parenthèse hors Apple de Steve Jobs) est utilisé pour développer les applications natives sur iOS.

Dans ce langage qui est une évolution du C, le code est découpé en deux parties :

- **les fichiers `.h` qui sont les en-têtes et qui contiennent les interfaces de classes définissant la manière dont les objets doivent être utilisés ;**
- **les fichiers `.m` qui contiennent les méthodes et les implémentations, le code proprement dit.**

4.2 Sécurité des données

De par leur nature, les terminaux peuvent facilement être volés. Les développeurs d'applications sensibles à la sécurité doivent donc être vigilants avec les données que leurs applications manipuleront et qui seront persistantes sur les terminaux. L'auditeur peut donc vérifier avec quelles précautions les données sensibles sont stockées en essayant d'identifier le contenu stocké en clair, de manière chiffrée, mais en utilisant un algorithme de chiffrement maison ou encore avec une classe de protection des données inappropriée (voir encart « Data Protection Class »).

Les modules dédiés à ce travail sont très logiquement classés sous l'arborescence **storage**.

esiea GROUPE

FORMATIONS OPÉRATIONNELLES EN SÉCURITÉ

/ Formations présentielles - Campus Paris V^e

formations-securite@esiea.fr / esiea.fr/formations-securite

/ Candidatures MS-SIS : en cours

FORMATION À PLEIN TEMPS

6 mois de pédagogie, puis 6 mois en entreprise

Prochaine rentrée : octobre 2017

MASTÈRE SPÉCIALISÉ SÉCURITÉ DE L'INFORMATION ET DES SYSTÈMES

(MS-SIS : 740 heures de cours)

Accrédité par la Conférence des Grandes Écoles

- Réseaux
- Sécurité des réseaux, des systèmes d'information et des applications
- Modèles et Politiques de sécurité
- Cryptologie

Mastère Spécialisé labellisé SecNumedu par l'ANSSI

ASM / C / crypto / firewalling / forensic / GPU / Java / malware / OSINT / pentest / python / reverse / SCADA / scapy / SDR / suricata / vlro / vuln / web...

/ Candidatures BADGE-RE et BADGE-SO : à partir d'octobre 2017

2 FORMATIONS EN COURS DU SOIR ET WEEK-ENDS (sur 6 mois)

Prochaine rentrée : février 2018

BADGE REVERSE ENGINEERING

(BADGE-RE : 230 heures de cours)

- Analyse de codes malveillants
- Reverse et reconstruction de protocoles réseau
- Protections logiciels et unpacking
- Analyse d'implémentations de cryptographie

Malware / ASM / IDA-Pro / x86 / ARM / debugging / crypto / packer / kernel / mlasm...

BADGE SÉCURITÉ OFFENSIVE

(BADGE-SO : 230 heures de cours)

- Détournement des protocoles réseaux non sécurisés
- Exploitation des corruptions mémoires et vulnérabilités web
- Escalade de privilèges sur un système compromis
- Intrusion, progression et prise de contrôle d'un réseau

Crypto / scan / OS / sniffing / OSINT / wifi / reverse / pentest / scapy / réseau IP / web / metasploit...

En partenariat avec quarkslab

Accrédité par la Conférence des Grandes Écoles

BADGE Bilan d'Aptitude Délivré par les Grandes Écoles

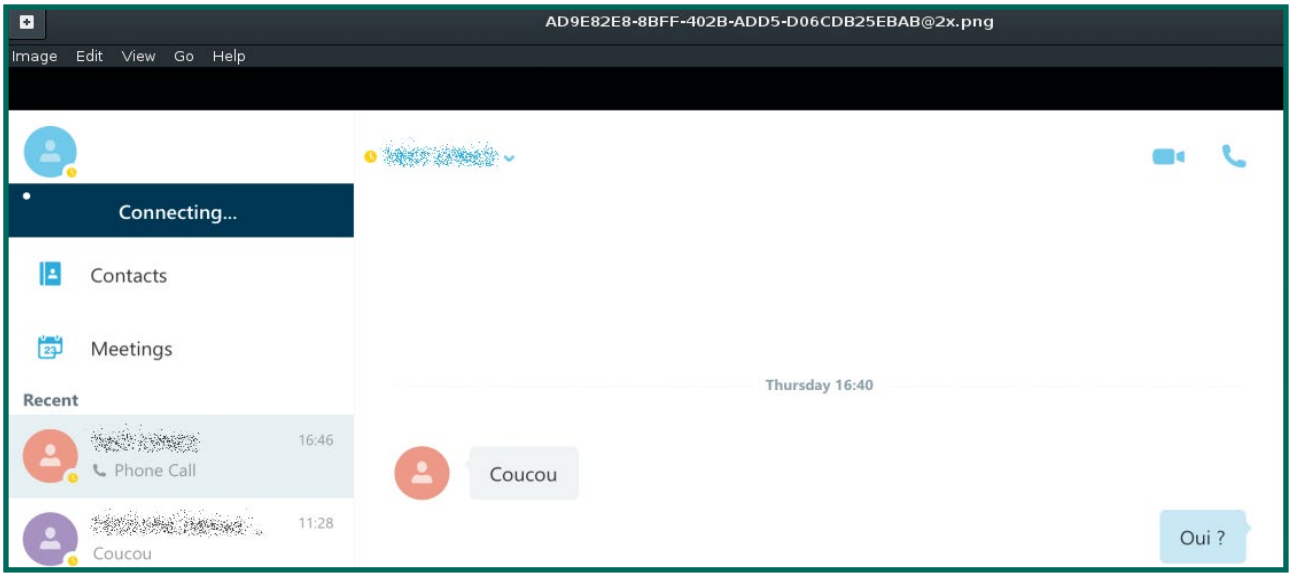

Figure 5 : Capture d'écran récupérée avec le module « storage/caching/screenshot ».

La mise en cache de frappes clavier sensibles (ex. : identifiants) est un cas d'école. Le module **storage/caching/keyboard_autocomplete** qui va extraire toutes les frappes clavier mises en cache (automatiquement par le système pour personnaliser l'autocorrection) permet de procéder à cette vérification.

```
[needle][keyboard_autocomplete] > run
[*] Checking connection with device...
[+] Already connected to: 127.0.0.1
[V] Creating temp folder: /var/root/needle/
[*] Running strings over keyboard autocomplete databases...
[+] The following content has been found:
DynamicDictionary-5
again
Clio
findme
France
hello
Hello
mirabeau
Paris
Renault
```

Aucun identifiant ne semble avoir été mis en cache ici, mais si c'était le cas les développeurs pourraient se prémunir de ce comportement en spécifiant la déclaration **UITextAutocorrectionTypeNo**.

À noter que seul le clavier standard (blanc) est concerné par cette mise en cache, le noir, utilisé notamment par iTunes, ne l'est pas.

Sur iOS lorsque l'utilisateur passe d'une application à une autre, une capture d'écran est réalisée par défaut, ce qui permet de les afficher toutes en appuyant deux fois sur le bouton principal (Home).

Dans le cas où des informations sensibles sont manipulées (application bancaire par exemple ou gestionnaire de mots de passe), cette fonctionnalité peut toutefois être désactivée (le développeur peut, par exemple, désigner une image statique qui sera utilisée systématiquement pour la capture d'écran).

Data Protection Class

Sur iOS, l'ensemble des fichiers est chiffré. Les classes de protection de données définissent les conditions de déchiffrement des fichiers. Les fichiers appartenant à la classe la plus restrictive, NSProtectionComplete, ne sont accessibles que lorsque le terminal est déverrouillé, en effet ils sont chiffrés avec une clé dérivée du code PIN de l'utilisateur et la clé AES du terminal. Ceux appartenant à la classe la moins restrictive, No Protection, sont accessibles même si le terminal est verrouillé. NSFileProtectionCompleteUntifFirstUser Authentication offre la même protection que NSProtectionComplete à ceci près que la clé de chiffrement n'est pas supprimée lorsque le terminal est verrouillé. C'est-à-dire que les fichiers sont inaccessibles uniquement tant que l'utilisateur n'a pas démarré son terminal puis tapé au moins une fois son code PIN. Finalement, Protected Unless Open laisse accès aux fichiers tant qu'ils sont ouverts.

Le module **storage/caching/screenshot** permet de vérifier le comportement de l'application en récupérant l'image qui a été stockée.

Les modules **storage/data/files_xxx** vont rapatrier puis afficher les fichiers de cache, les cookies, les fichiers plist ou les bases sqlite.

L'application Skype stocke les conversations dans une base sqlite (**DataStore.sqlite**).

```
[needle][files_sql] > run
[*] Checking connection with device...
[+] Already connected to: 127.0.0.1
[V] Creating temp folder: /var/root/needle/
```

```
[+] Target app: com.microsoft.lync2013.iphone
[*] Looking for SQL files...
[*] Retrieving data protection classes...
[*] The following SQL files have been found:
        0   -
[NSFileProtectionCompleteUntilFirstUserAuthentication] /private/
var/mobile/Containers/Data/Application/F9134464-6151-4F91-8891-
92310C8F064E/Documents/.localytics/462f81e6d6b18fd930bdf01-
aaf73ccc-f746-11e4-3944-00eadbc03855/analytics.sqlite
        1   -
[NSFileProtectionCompleteUntilFirstUserAuthentication] /private/
var/mobile/Containers/Data/Application/F9134464-6151-4F91-8891-
92310C8F064E/Documents/.localytics/462f81e6d6b18fd930bdf01-
aaf73ccc-f746-11e4-3944-00eadbc03855/marketing.sqlite
        2   -
[NSFileProtectionCompleteUntilFirstUserAuthentication] /private/
var/mobile/Containers/Data/Application/F9134464-6151-4F91-8891-
92310C8F064E/Documents/.localytics/462f81e6d6b18fd930bdf01-
aaf73ccc-f746-11e4-3944-00eadbc03855/profiles.sqlite
        3   -
[NSFileProtectionCompleteUntilFirstUserAuthentication] /private/
var/mobile/Containers/Data/Application/F9134464-6151-4F91-8891-
92310C8F064E/Documents/DataStore.sqlite
Please select a number: 3
[*] Pulling: /private/var/mobile/Containers/Data/Application/
F9134464-6151-4F91-8891-92310C8F064E/Documents/DataStore.sqlite ->
/root/.needle/output/sql_datadir_Documents_DataStore.sqlite
[*] Spawning SQLite3 console...
```

On peut constater que le fichier utilise la classe de protection **NSFileProtectionCompleteUntifFirstUserAuthentication** et qu'au sein de la base SQLite les conversations ne sont pas chiffrées (voir le dump SQLite et la capture d'écran du terminal). Par ailleurs, les conversations sont également sauvegardées par iTunes, mais c'est un autre sujet.

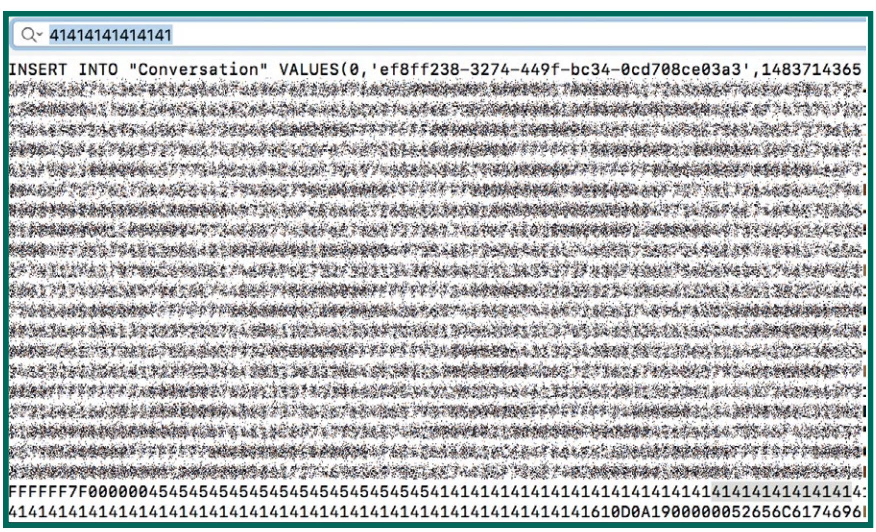

Figure 6 : Extrait d'une conversation récupérée avec le module « storage/caching/files_sql ».

Finalement, le module **storage/data/keychain_dump** va récupérer le trousseau d'accès (bien veiller à l'utiliser avec un terminal déverrouillé, car le trousseau n'est pas accessible lorsque le terminal est verrouillé).

L'auditeur pourra y trouver des identifiants, des jetons de session et d'autres informations sensibles.

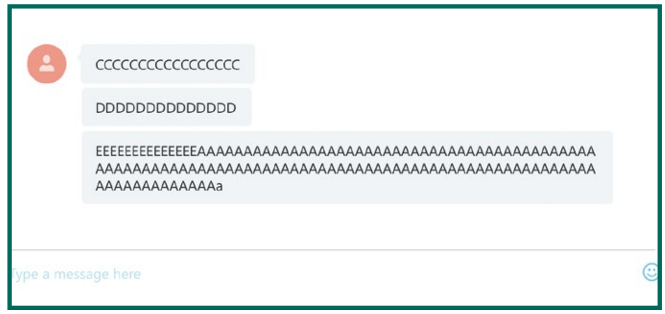

Figure 7 : Capture d'écran d'une conversation ».

4.3 Analyse à l'exécution

Le principal intérêt de cette partie de l'analyse est de comprendre l'impact de l'application sur le système et de contourner certaines fonctions de sécurité comme la détection de jailbreak ou d'un code PIN.

Concernant le contournement de fonctions de sécurité, trois approches sont possibles :

- instrumenter les méthodes avec Cycript, Frida, Theos ou Snoop-It (ce dernier ne supporte pas les applications 64 bits) ;
- patch du binaire avec Hopper ou IDA ;
- patch du runtime avec gdb.

Needle présente un intérêt uniquement pour la première approche. La deuxième consiste à modifier le binaire à la main avec un désassembleur et la troisième a modifier directement l'application en mémoire avec un débogueur.

L'instrumentation consiste aussi à déboguer l'application, comme pour le patch d'un binaire, mais sans pour autant descendre au niveau assembleur. Un autre avantage est qu'il est possible de travailler avec un terminal non débridé. D'autres prérequis sont toutefois nécessaires : disposer du logiciel Xcode, l'environnement de développement d'Apple et donc... d'un Mac.

Les outils Cycript et Frida, permettant d'instrumenter les applications iOS (mais pas seulement) en JavaScript (ainsi qu'en Objective-C pour Cycript) sont intégrés dans Needle sous les modules **hooking/cycript/cycript_shell** et **hooking/frida/friday_shell**.

Ils fonctionnent en s'injectant dans le processus de l'application et en fournissant une console dans laquelle il est possible d'interagir en appelant des méthodes voire en les modifiant à la volée.

Exemple d'appel à une méthode de validation d'un code PIN :

```
[*] Spawning a Cycript shell...
Warning: Permanently added '[127.0.0.1]:2222' (RSA) to the list of
known hosts.
cy# UIApp
#"<UIApplication: 0x15752e730>"
cy# choose(RuntimeManipulateDetailsVC)
[#"<RuntimeManipulateDetailsVC: 0x15756a510>"]
cy# [#0x15756a510 validateCode:@"000"]
false
cy# [#0x15756a510 validateCode:@"123"]
true
cy#
```

Après avoir injecté Cycript, il est possible d'effectuer un appel à la méthode **validateCode** de l'interface **RuntimeManipulateDetailsVC** pour vérifier si le code PIN est correct.

Avec quelques lignes de code, il est possible d'effectuer une attaque par force-brute pour le découvrir.

```
// localisation de l'adresse de la classe
RuntimeManipulateDetailsVC
choose(RuntimeManipulateDetailsVC)
[#"<RuntimeManipulateDetailsVC: 0x136e94af0>"]

// appel de la méthode validateCode()
[#0x136e94af0 validateCode:@"000"]
false

var pin=0;
0

// bruteforce
function brute() {
    for(var i=0; i<=999; i++) {
        var result = [#0x136e94af0 validateCode:i.toString()];
        if(result=="1") {pin=i;}
            }
        }

brute()

print:pin;
123
```

Attention, si un mécanisme de blocage au bout de plusieurs tentatives infructueuses est en place, il faudra préalablement l'identifier et le désactiver.

D'autres modules sont utiles pour surveiller le presse-papier **dynamic/monitor/pasteboard** ou les journaux du système **dynamic/monitor/syslog**, qui parfois contiennent des informations sensibles comme des identifiants, des jetons de sessions ou encore des informations de débogage.

Finalement, le module **dynamic/monitor/files** servira à identifier les fichiers qui sont créés sur le système, par exemple suite au renseignement d'un formulaire d'authentification.

4.4 Analyse des communications

Cette partie de l'analyse se résume en grande partie par la configuration d'un proxy, comme Burp Suite, sur le terminal préalablement libéré des contraintes du *certificate pinning* avec SSL Kill Switch 2 **[sslkillswitch2]** puis par l'examen des communications.

Il peut être utile de modifier les requêtes et/ou les réponses et d'étudier le comportement de l'application.

Par exemple, le client Skype tolère bien l'interception et la modification de messages. Ainsi il est possible de modifier un message reçu par un destinataire sans que l'émetteur soit averti de sa modification, ce qui peut aboutir à quelques quiproquos.

Seuls quelques modules Needle, la plupart traitant des certificats (pour les lister, les supprimer, les installer), aident le travail de l'analyste sur cette partie.

Conclusion

Il est impossible de couvrir l'ensemble de l'outil dans ce seul article, mais j'espère avoir éveillé votre curiosité avec ces quelques cas pratiques et pour ceux qui veulent approfondir le sujet, l'excellent Mobile Application Hacker's Handbook **[mahh]** est l'ouvrage de référence à avoir sous la main, qui, même s'il n'aborde pas Needle (l'outil n'existait pas au moment de sa rédaction) couvre tous les sujets du spectre de la sécurité des applications mobiles. ■

Remerciements

Merci à Marco Lancini pour l'outil et le workshop à Deepsec.

Références

[needle] https://github.com/mwrlabs/needle

[installation] https://github.com/mwrlabs/needle/wiki/Installation-Guide

[burpsuite] https://portswigger.net/burp/

[iphonewiki] https://www.theiphonewiki.com/wiki/Jailbreak

[skype] https://itunes.apple.com/us/app/skype-for-business-formerly-lync-2013/id605841731

[iicontact] https://itunes.apple.com/fr/app/iicontact/id1072100138

[dvia] http://damnvulnerableiosapp.com/

[frida] https://www.frida.re/docs/ios/#without-jailbreak

[sslkillswitch2] https://github.com/nabla-c0d3/ssl-kill-switch2

[mahh] http://eu.wiley.com/WileyCDA/WileyTitle/productCd-1118958500.html