Quarkslab

Reverse engineering sous Android et iOS
JSSI 2013

Sébastien Kaczmarek
skaczmarek@quarkslab.com

Plan

Plan

iOS et Android

iOS

- Utilisé uniquement sur iPhone, iPad, Apple TV et iPad
- Noyau `XNU`, historiquement basé sur `BSD`, architecture `ARMv7`
- Projet fermé même si `XNU` est open source

Android

- Système d'exploitation développé actuellement par Google suite au rachat de Android
- Noyau linux pour architecture `ARM` principalement
- Version actuelle : 4.2.2 Jelly Bean
- Le projet est open source

Plan

Développement sous iOS

Principe

- iPhoneSDK, iOS Dev Center
- Environnement de développement officiel XCode
- Compilateur `GCC / LLVM`

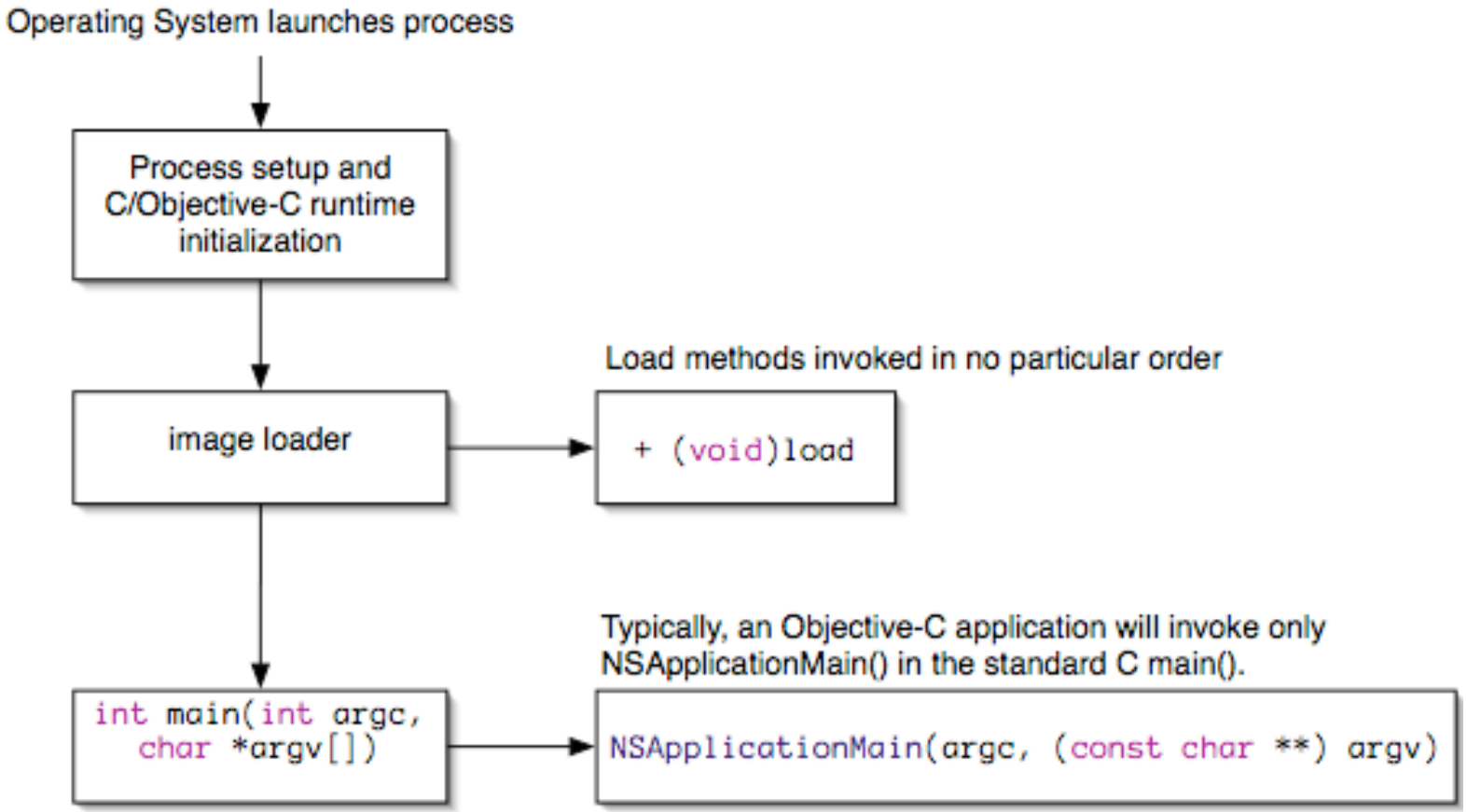
Langages

- Langage `C/C++` ou `Objective-C`
- Objective-C peut être mélangé avec du code écrit en `C`
- Cocoa est l'API native pour `Objective-C`
- La majorité des applications utilisent `Obj-C/Cocoa`

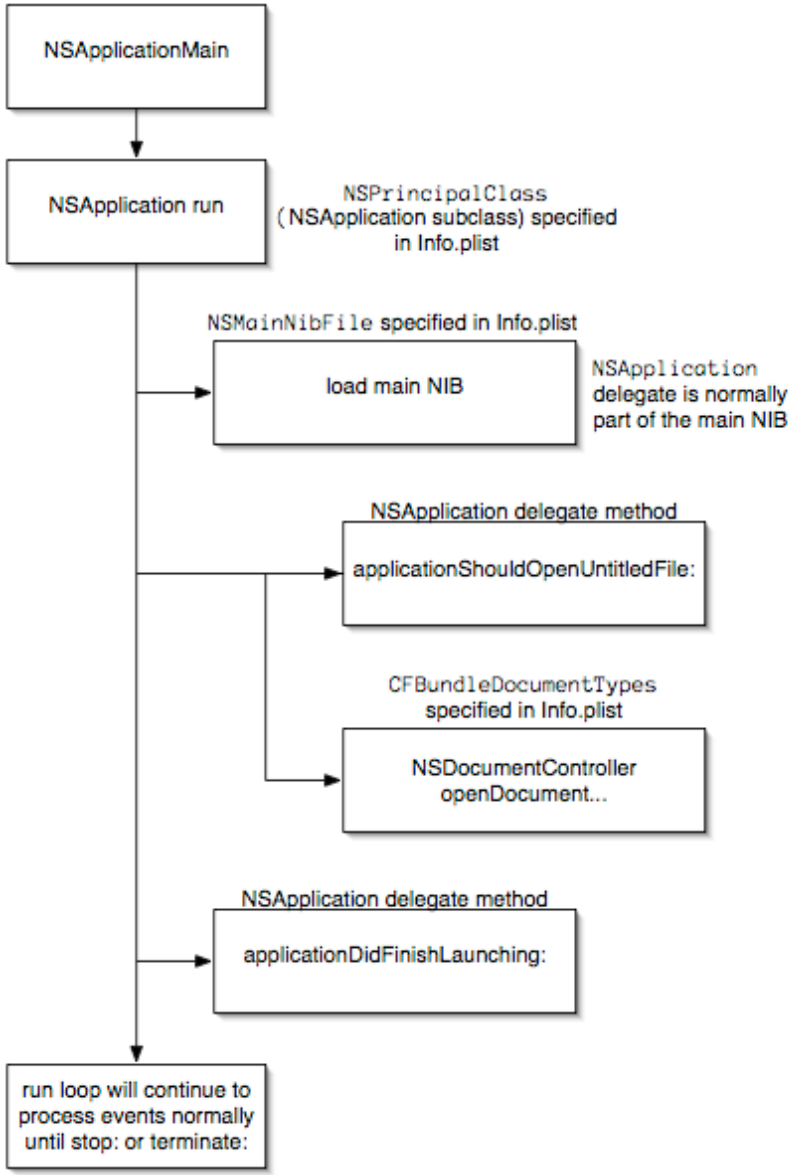
Développement sous iOS

Objective-C

- Point d'entrée : `NSApplicationMain()`
- Appelé dans `main()`



Développement sous iOS



Développement sous iOS

Contenu d'une application

- Une application = un répertoire .app
  - Binaires au format `MACH-O`
  - Info.plist : la configuration du bundle (type, nom, version,...)
  - Fichiers *.nib : ressources graphiques
  - _CodeSignature\CodeResources : dictionnaire de hashs
  - embedded.mobileprovision : fichier provisionning
- Restriction de l'exécution à certains UDID

Publier sur AppStore

- La signature des applications est obligatoire
- Les applications sont théoriquement "testées" avant d'être publiées

Plan

Développement sous Android

Framework

- Android SDK
- Android Developement Tools (`ADT`) pour Eclipse
- Compilateur Java et conversion en `Dalvik`
- Émulateur Android inclus, plutôt lent cependant.
- Possibilité de simuler des appels et envois de SMS

Langages

- Machine virtuelle `Dalvik`, Java-like mais avec un bytecode différent
- Il est possible d'utiliser du code natif compilé avec gcc par exemple

Développement sous Android

Contenu d'un fichier .apk

- `AndroidManifest.xml` (description de l'application)
  - Nom et version
  - Droits d'accès et dépendances
- `/META-INF` (`manifest.inf`, `cert.rsa`, `cert.sf`)
- `/lib`, librairies et code natif
- `classes.dex`, le code `Dalvik` de l'application
- Des ressouces dans `/res` et resources.arsc

Signature

- Signature de `cert.sf` (la liste des condensats SHA-1 de l'archive)
- Signature au format `DER` dans `cert.rsa`

Plan

Plan

Debugging

Objectifs

- Comprendre les logiques non observables
- Analyser les bugs
- Retranscrire des algorithmes
- Déprotéger

Techniquement?

- Tracer en mode pas à pas
- Instrumentation

Plan

iOS fermé au debugging?

À première vue oui

- Debugging possible seulement depuis `XCode`? Et pour le noyau ?
- `OpenSSH` et `gdb` sur l'AppStore ? :)
- Pas de debugging sans jailbreak

Cependant

- `gdb` et `debug_server` sont fournis avec `XCode`, légèrement cachés...
- De nombreuses parties de l'API Cocoa sont open source
- De nombreux outils existent: `class-dump-z`, `cyscript`, `MobileSubstrate`, `...`
- Les sources de gdb pour iOS sont disponibles:
  http://www.opensource.apple.com/release/iphone-20

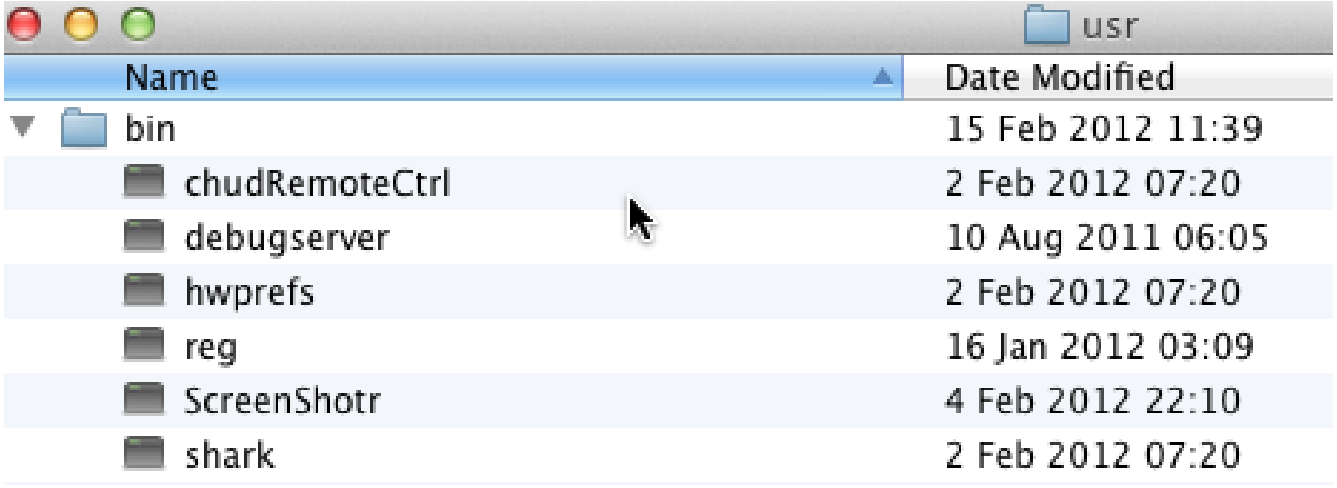
Debugging avec gdb

Où est gdb?

- La version ARM est fourni avec `XCode`
- Renommé en `gdb-arm-apple-darwin`
  /Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/Developer

Remote debugging

- Dans l'image `DeveloperDiskImage.dmg`
  /Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/DeviceSupport



Debugging avec gdb

Utiliser gdb

- Des droits spécifiques doivent être ajoutés à gdb

```
- <plist version="1.0">
  - <dict>
      <key>com.apple.springboard.debugapplications</key>
      <true/>
      <key>get-task-allow</key>
      <true/>
      <key>task_for_pid-allow</key>
      <true/>
    </dict>
  </plist>
```

Remote debugging

```
ldid -Sentitlements.xml gdb
codesign -s "iPhone developer" --entitlements gdb.xcent -f gdb
```

Debugging avec gdb

Limites

- Les applications multi-threadées sont mal supportées
- Pas de support natif pour des plugins
- Parfois lents (ex: breakpoints)

gdb ou lldb?

- LLDB repose sur le projet `LLVM`
- Disponible pour `Mach-O` (i386+x86-64), `ELF` 32-bit et `ARM/Thumb`
- Remote debugging (i386 et x86-64)
- Scripting avec Python
- Résoud plusieurs problèmes liés à l'usage de gdb

Plan

Debugging avec gdb

Debugging avec gdb

- Local ou remote debugging
- Difficultés à debugger les applications multithreadées
- Debug server de `IDA Pro` depuis la version 6.1, problèmes de stabilité

Et pour Dalvik?

- Plugin DDMS pour Eclipse
- Une analyse statique peut suffire dans la majorité des cas

Plan

Plan

Debugging Android

Outillage

- `apktool` pour extraire les fichiers de l'application
- `dex2jar` pour convertir le Dalvik en bytecode Java
- `JAD` ou `JD-GUI` pour récupérer les code en Java
- `Androguard`, + analyse de similarités

Outillage

- Les outils de debug/reverse ne sont pas aussi fiables que sur x86
- De nombreux outils pratiques n'existent pas

Debugging Android

Décompilation

- Extraire le fichier apk avec `apktool`
- Conversion des fichiers .dex en bytecode Dalvik avec `Baskmali`
- Conversion des fichiers .dex en jar avec `dex2jar`

Edition

- Le bytecode Dalvik peut être édité (fichiers *.smali) avant d'être réintégré dans le code de l'application finale

Debugging Android - exemple d'injection statique

Compilation du payload

```
#:~/src/Android/flash/src/com/xxx/player>> javac -classpath ~/src/Android/sdk/
platforms/android-10/android.jar Payload.java
```

Convertion du bytecode java en .dex

```
#r:~/src/Android/flash/src>> dx --dex --keep-classes --output=Payload.dex
com/xxx/player/Payload.class
```

Désassemblage du code au format Dalvik

```
#:~/src/Android/flash/src>> baksmali Payload.dex
```

Debugging Android - exemple d'injection statique

Compilation du payload

```
// Code injecté
.method public static injectedFunction(Ljava/lang/String;)V
...
.end method

// Code modifié
.method static aRandomFunction()Ljava/lang/String;
.registers 6
.prologue
.line 71
const-string v0, ""


invoke-static v0, Lcom/xxx/player/Random;->aRandomFunction(Ljava/lang/String;)V


...
.end method
```

Debugging Android

Utilisation de setpropex

- Modification des "property" ro.*
- Objectif: positionner ro.debuggable à 1
- Lancement des applications en mode debug
- Attachement d'un debugger tel JDWP

En pratique

- Utilisation de ptrace (`PTRACE_ATTACH`)
- Utilisation de /proc/pid/...
- Ecrasement des property avec `PTRACE_POKEDATA`

Plan

Debugging iOS

Objective-C sous IDA

- IDA 6.2 supporte le parsing des applications `Objective-C`
- La vue globale est très pertinente pour un langage de haut niveau

Function name	Segment	Start
-[PlayVideoTab setPlayerView:]	__text	000073D4
-[PlayVideoTab setUrlTextField:]	__text	00007424
-[PlayVideoTab urlTextField]	__text	00007254
-[PlayVideoTab viewDidUnload]	__text	000072B0
-[PlayerAppDelegate application:didFinishLaunchingWithOptions:]	__text	00002BE8
-[PlayerAppDelegate applicationDidBecomeActive:]	__text	00002BBC
-[PlayerAppDelegate applicationDidEnterBackground:]	__text	00002BB4
-[PlayerAppDelegate applicationDidReceiveMemoryWarning:]	__text	00002BC4

```
v6 = objc_msgSend(v5, "getPolicies");
v7 = objc_msgSend(v6, "objectAtIndex:", 0);
if ( objc_msgSend(v7, "getAuthenticationMethod") == (void *)2 )
{
  v8 = objc_msgSend(*(void **)(v4 + 156), "getServerUrl");
  objc_msgSend((void *)v4, "faxsLog:", CFSTR("Authenticating to License server: %@"), v8);
  *(_DWORD *)(v4 + 152) = objc_msgSend(&OBJC_CLASS___DRMManager, "sharedManager");
```

Debugging iOS

Objective-C avec Class-dump

- class-dump-z par KennyTM
- Retour aux sources :)

```
@interface AuthenticateTab : UIViewController {
UITextField* authUserTextField;
UITextField* authPasswordTextField;
PlayerViewController* m_parent;
}
@property(retain, nonato[fragile]mic) UITextField*
    authUserTextField;
@property(retain, nonatomic) UITextField* authPasswordTextField
    ;
@property(retain, nonatomic) PlayerViewController* m_parent;
-(id)initWithParent:(id)parent;
-(void)didReceiveMemoryWarning;
-(void)viewDidUnload;
-(void)dealloc;
-(void)authenticate;
-(void)unauthenticate;
-(void)authenticateToDomainServer;
-(void)unauthenticateToDomainServer;
@end
```

Debugging iOS

Objective-C et instrumentation

- Presque "full objects"
- Bonne nomenclature et hiérarchie de classes
- Les objets envoient et recoivent des messages
- Équivalence à une pompe à messages (cf. Windows)
- Tout repose sur quelques fonctions : mainly `objc_msgSend`, `objc_setProperty`, ...

Difficile à debugger?

- En comparaison à C++, non :)
- Tous les noms de classes et méthodes apparaissent en clair
- Les membres de classes sont également visibles

Debugging iOS

Objective-C et objc_msgSend()

- le code contient principalement des apples à `objc_msgSend()`
  - 1er argument : pointeur objet (ISA)
  - 2ème argument : sélecteur (nom de méthode)
  - Et une liste d'arguments de taille variable

Exemple d'instrumentation avec gdb

```
break objc_msgSend
commands
printf "-[%s %s]\n", (char *)class_getName(*(long *)$r0, $r1), $r1
c
end
```

Debugging iOS

Analyse dynamique?

- Cycript - http://www.cycript.org/
- Alternative à `JSCocoa`
- Appel natif à du code Objective-C code depuis un langage de script

Caractéristiques

- Attachement à un processus
- Instanciation d'objets à partir d'une adresse virtuelle
- Appel aux méthodes des classes
- Manipulation d'objets / parsing

Debugging iOS

ApplicationDelegate et hiérarchie

- Généralement toutes les applications utilisent une délégation de `UIApplication`
- Les délégués recoivent des messages de leurs parents
- Le membre `sharedApplication` de `UIApplication` est une mine d'information

Bénéfices

- Pas besoin de sous-classer
- Il est facile d'étendre le framework existant
- Plus simple à débugger

Debugging iOS

Exemple de trace avec Cycript

```
cy# var app = [UIApplication sharedApplication]
"<UIApplication: 0xde751d0>"

cy# app.delegate
"<PlayerAppDelegate: 0xde87750>"

cy# var delegate = new Instance(0xde87750)
"<PlayerAppDelegate: 0xde87750>"

cy# delegate.viewController.player
"<AVPlayer: 0x4e0970>"

cy# var player = new Instance(0xdedbd70)
"<AVPlayer: 0xdedbd70>"

cy# [player play]

cy# function ls(a) {var x={}; for(i in *a) { try{ x[i] = (*a)[i];} catch(e){}}
return x;}
cy# ls(player)
{isa:"NSKVONotifying_AVPlayer",_player:"<AVPlayerInternal: 0xded8160>"}

cy# player_int = new Instance(0xded8160)
"<AVPlayerInternal: 0xded8160>"
```

Debugging iOS

Le cas d'un player VOD

```
cy# var d = [VODLoader vodLoader]
@"<VODLoader: 0x2effb10>"
cy# d.getObj[0].name
@"XYZ VOD"
cy# d.getObj[0].su
subcategories  superclass
cy# d.getObj[0].subcategories
@["<VCategory: 0x8f91e30>","<VCategory: 0x8fd5ee0>","<VCategory: 0x8f264f0>","<VCategory: (
cy# d.getObj[0].subcategories[0].subcategories[0].name
@"Ma chaine TV - PrOn"
cy# d.getObj[0].subcategories[0].subcategories[0].subcategories[0].vods[0].free
0
cy# d.getObj[0].subcategories[0].subcategories[0].subcategories[0].vods[0].free = 1
cy# d.getObj[0].subcategories[0].subcategories[0].subcategories[0].vods[0].free
1
cy# d.getChannel[32].startFree = [[NSDate alloc] initWithString:@"2009-09-16 12:40:00 +0000
@"2009-09-16 12:40:00 +0000"
cy# Clock.messages["hasPriv:"] = function() { return true; }
cy# Clock.messages["time:"] = function() { return true; }
cy# Clock.messages["time"] = function() { return true; }
cy# [UIApp.windows[0].rootViewController->loginObject_ allowLogIn:1]
```

Debugging iOS

Et gdb?

- Difficulté dans la résolution des symboles Obj-C
- La pile d'appels est souvent juste une suite d'offsets
- La gestion des threads est parfois problématique

Mais...

- Certaines méthodes sont toutefois reconnues (static)
- `lldb` tend à remplacer `gdb`, de même `llvm` pour `gcc`
- `lldb` peut-il tourner en dehors de XCode?

Debugging avec lldb

Mise en pace

```
lipo -thin armv7 debugserver -output debugserver7
ldid -Sent.xml
```

Ajout des privilèges

```
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyLis
<plist version="1.0">
<dict>
<key>com.apple.springboard.debugapplications</key>
<true/>
<key>get-task-allow</key>
<true/>
<key>task_for_pid-allow</key>
<true/>
<key>run-unsigned-code</key>
<true/>
</dict>
</plist>
```

Debugging avec lldb

Se connecter au serveur de debug

```
$quarkspad>debugserver localhost:6789 -x spring ./SomeApps

$my_computer> (lldb) platform select remote-ios
$my_computer> (lldb) process connect connect://x.x.x.x:6789
Process X stopped
...
(lldb)
...
```

Debugging iOS

lldb et stackframes

```
(lldb) frame variable
self = (SKTGraphicView *) 0x0000000100208b40
_cmd = (struct objc_selector *) 0x000000010001bae1
sender = (id) 0x00000001001264e0
selection = (NSArray *) 0x00000001001264e0
i = (NSUInteger) 0x00000001001264e0
(lldb) frame variable self.isa
(struct objc_class *) self.isa = 0x0000000100023730
```

Scripts Python

```
breakpoint command add -s Python
> thread = frame.GetThread()
> thread.StepInstruction(1)
> newFrame = thread.GetFrameAtIndex(0)
> print " " * thread.GetNumFrames() + newFrame.GetFunctionName()
> process = thread.GetProcess().Continue()
> DONE
```

Debugging iOS

Backtrace and désassemblage à chaque arrêt

```
(lldb) target stop-hook add
Enter your stop hook command(s). Type 'DONE' to end.
> bt
> disassemble --pc
> DONE
Stop hook #1 added.
```

Breakpoint sur un sélecteur

- gdb ne différencie pas sélecteurs et fonctions C/C++

```
(lldb) breakpoint set --selector count
(lldb) br s -S count
```

Debugging iOS

Watchpoint conditionnel

```
(lldb) watch set var global
(lldb) watchpoint modify -c '(global==5)'
(lldb) c
...
(lldb) bt
* thread #1: tid = 0x1c03, 0x0000000100000ef5 a.out`modify + 21 at
main.cpp:16, stop reason = watchpoint 1
frame #0: 0x0000000100000ef5 a.out`modify + 21 at main.cpp:16
frame #1: 0x0000000100000eac a.out`main + 108 at main.cpp:25
frame #2: 0x00007fff8ac9c7e1 libdyld.dylib`start + 1
(lldb) frame var global
(int32_t) global = 5
```

Debugging iOS

Arrêt sur classe

- Affichage de la variable `this` lors d'un arrêt sur la classe `MyClass`

```
(lldb) target stop-hook add --classname MyClass --one-liner
"frame variable *this"
(lldb) ta st a -c MyClass -o "fr v *this"
```

Dumping des sections

```
(lldb) image dump sections a.out
```

Plan

Instrumenter SSL

Objectifs

- Les proxy SSL (ex Burp Proxy) ne permettent pas toujours de "casser" la session SSL et réaliser un MITM en cas de vérification du certificat serveur

Exemple avec gdb manuellement

```
(gdb)break CFHTTPMessageAppendBytes
Breakpoint 3 at 0x39aee945
(gdb) c
Continuing.
Breakpoint 3, 0x39aee945 in CFHTTPMessageAppendBytes ()
(gdb) x/2s $r1
0x243cf7c:
"HTTP/1.1 200 OK\r\nServer: Apache-Coyote/1.1\r\nExpires: Fri,
30 Mar 2012 08:25:07 GMT\r\nContent-Type: application/octet-stream\r\n
Transfer-Encoding: chunked\r\nDate: Fri, 30
```

Instrumenter SSL

Capture de certificats

```
(gdb) b *(SecCertificateCreateWithData+90)
Breakpoint 6 at 0x343d14c2
(gdb) c
Continuing
[Switching to process 366 thread 0x3307]
Breakpoint 6, 0x343d14c2 in SecCertificateCreateWithData ()
(gdb) x/4b $r0
0xd90c020: 0x30 0x82 0x05 0xc2
```

Le cas d'un protocole quelconque

```
(gdb)break CCCryptorCreateFromData
Breakpoint 3 at 0x30aee836
(gdb) c
Continuing.
Breakpoint 4, 0x30aee836 in CCCryptorCreateFromData ()
(gdb) x/s $r3
0xe677ba0:
"c6b9LNtA0ni6Vw=="
```

Instrumenter SSL

MobileSubstrate

- Hooking des fonctions natives et Cocoa
- Définition de règles par processus

Makefile

```
SDK=$(XCODE)/Platforms/iPhoneOS.platform/Developer/SDKs/
    iPhoneOS5.1.sdk/
CC:=$(wildcard $(XCODE)/Platforms/iPhoneOS.platform/Developer/
    usr/bin/arm-apple-darwin10-*gcc*)
CFLAGS=-Wall -isysroot $(SDK)  -framework CoreFoundation -
    framework Security  -framework CFNetwork -L $(XCODE)/
    Platforms/iPhoneOS.platform/Developer/usr/lib/ -lsubstrate
LD=$(CC)

https_snif.dylib: https_snif.c
        $(CC) $(CFLAGS) -dynamiclib -fPIC -o $@ $^
        ldid -S $@
```

Instrumenter SSL

Exemple avec SSL

```
CFIndex HOOK_CFReadStreamRead(CFReadStreamRef stream,UInt8 *
    buffer,CFIndex bufferLength) {
static char buf[4096];
CFIndex ret = CFReadStreamRead_old(stream,buffer, bufferLength)
    ;
if(ret>0) {
if(strlen((char *)buffer)<4096) {
    syslog(LOG_NOTICE,"[+] Stream read");
    DumpBinary(buffer,ret);
}
    return ret;
}
__attribute__((constructor))
static void initialize() {
syslog(LOG_NOTICE, "HTTPS_SNIF: hooking :)");
MSHookFunction(CFReadStreamRead,HOOK_CFReadStreamRead,&
    CFReadStreamRead_old);
}
```

Instrumenter SSL

D'autres idées

- `CFStringCreateWithCString`
- `SecTrustEvaluate`
- `SecTrustServerEvaluateAsync`
- `CFHTTPMessageSetHeaderFieldValue`
- `CFDictionaryGetValue`
- ...

Instrumenter SSL

In certificates we trust!

```
const void *HhookCFDictionaryGetValue(CFDictionaryRef dic,
     const void *key)
{
CFArrayRef* no_policy;
no_policy = CFArrayCreate(NULL,NULL,NULL,NULL);

   [...]

        if(CFEqual(key, CFSTR("policies")))
        {
            return no_policy;
        }

     }
     return CFDictionaryGetValue(dic, key);
}
```

Mise en place

Injection dans le démon securityd

```
launchctl unload /System/Library/LaunchDaemons/com.apple.
    securityd.plist
launchctl load com.apple.securityd.plist
```

Utilisation de DYLD_INSERT_LIBRARIES

```
<key>EnvironmentVariables</key>
<dict>
<key>DYLD_INSERT_LIBRARIES</key>
<string>/tmp/code.dylib</string>
</dict>
```

Instrumentation de SSL sous Android

Contournement de la validation des certificats

- `android-ssl-bypass`
- https://github.com/iSECPartners/android-ssl-bypass

En pratique

- Scripting de `JDWP` (breakpoint, écriture en mémoire)
- Utilisation de l'API `JDI`
- Instrumentation de `TrustManager` et `HttpsURLConnection`

En complément

Quelques outils

- Sniffer pirni en local pour une connexion 3G
- Burp Proxy
- ...

Plan

Plan

Chiffrement des exécutables

Le cas de iOS

- Applications chiffrées via l'API Fairplay (DRM Apple)
- Whitebox AES
- Obfuscation de code (code flattening)

Contournement

- run
- dump
- what else?

Chiffrement des exécutables

Les outils

- https://github.com/stefanesser/dumpdecrypted/blob/master/dumpdecrypted.c

Usage

```
iPod:~ root# DYLD_INSERT_LIBRARIES=dumpdecrypted.dylib /var/mobile/Applications/xxxx/Scan.a
mach-o decryption dumper
```

Déchiffrement

```
[+] Found encrypted data at address 00002000 of length 5745816 bytes - type 1.
[+] Opening /private/var/mobile/Applications/xxxx/MyApp.app/MyApp for reading.
[+] Reading header
[+] Detecting header type
[+] Executable is a FAT image - searching for right architecture
[+] Correct arch is at offset 9826187 in the file
[+] Opening MyApp.decrypted for writing.
[-] Failed opening. Most probably a sandbox issue. Trying something different.
[+] Opening /private/var/mobile/Applications/xxxx/tmp/MyApp.decrypted for writing.
[+] Copying the not encrypted start of the file
[+] Dumping the decrypted data into the file
[+] Copying the not encrypted remainder of the file
[+] Closing original file
[+] Closing dump file
```

Plan

Antidebug avec ptrace() (iOS)

```
if ( !ptp )
{
  v0 = ptps;
  v1 = *(_DWORD *)ptps;
  *(_DWORD *)(ptps + 4) ^= 0x1AC55AA2u;
  *(_DWORD *)v0 = v1 ^ 0x54E6CE2F;
  v2 = *(_DWORD *)(v0 + 8) ^ 0x54E6CE2F;
  *(_DWORD *)(v0 + 12) ^= 0x1AC55AA2u;
  *(_DWORD *)(v0 + 8) = v2;
  v3 = dlopen(0, 10);
  ptp = (int (__fastcall *)(_DWORD, _DWORD, _DWORD, _DWORD))((unsigned int)dlsym(v3, (cons
  v4 = ptps;
  v5 = *(_DWORD *)ptps;
  *(_DWORD *)(ptps + 4) ^= 0x1AC55AA2u;
  *(_DWORD *)v4 = v5 ^ 0x54E6CE2F;
  v6 = *(_DWORD *)(v4 + 8) ^ 0x54E6CE2F;
  *(_DWORD *)(v4 + 12) ^= 0x1AC55AA2u;
  *(_DWORD *)(v4 + 8) = v6;
}
ptp = (int (__fastcall *)(_DWORD, _DWORD, _DWORD, _DWORD))((unsigned int)ptp ^ 0x293D5234)
ptp(31, 0, 0, 0);
result = (unsigned int)ptp ^ 0x293D5234;
ptp = (int (__fastcall *)(_DWORD, _DWORD, _DWORD, _DWORD))((unsigned int)ptp ^ 0x293D5234)
return result;
```

Antidebug avec ptrace() (iOS)

Appel avant main()

```
void constructor(void) __attribute__ ((constructor));
void constructor(void) { Appel à ptrace... }
```

Antidebug avec ptrace() (iOS)

Pour simplifier

```
#define PT_DENY_ATTACH 31
ptrace(PT_DENY_ATTACH, 0, 0, 0);
```

Caractéristiques

- Appel obfusqué avec des opérations inutiles
- Anti-debug + anti-attach avec gdb

Contournement

- Utilisation de "waitfor" dans gdb pour un attachement anticipé
- Patch à la volée, passage d'un argument invalide

Antidebug avec gettime()

```
*getimeofday ^= v144;
((void (__fastcall *)(int *, _DWORD))*getimeofday)(&v537, 0);
*getimeofday ^= v144;
memcpy(&v542, max_bitsb, 0x10u);
LODWORD(v159) = -1000000 * v145 - WB_TABLE_MAX_INDEXb;
HIDWORD(v159) = (-1000000LL * (unsigned __int64)(unsigned int)v145 >> 32)
              + -1000000 * (v145 >> 31)
              - ((-1000000 * v145 >= WB_TABLE_MAX_INDEXb)
               + v496);
v160 = (unsigned __int64)(v159 + v538 + 1000000LL * v537) >> 32;
is_debugged = v159 + v538 + 1000000 * v537;
is_not_debugged = v160;
if ( v160 )
  is_not_debugged = 1;
_CF = is_debugged >= 0x1E8480;
_ZF = is_debugged == 0x1E8480;
v163 = 0;
if ( !_ZF & _CF )
  v163 = 1;
if ( !v160 )
  is_not_debugged = v163;
if ( !is_not_debugged )
{
```

Antidebug avec gettime()

Caractéristiques

- Mesurer le temps d'exécution d'une portion de code
- Détection du debugging pas à pas
- Fiabilité moyenne

Contournement

- Patcher l'écart de temps maximal codé en dur
- Ne pas poser de point d'arrêt entre 2 appels à `gettimeofday()`

Plan

Obfuscations : chiffrement des chaines de caractères

```
01 00 00 00    _addHeadersToMessage.enc DCD 1          ; DATA XREF: _addHeadersToMessage
                                                       ; _addHeadersToMessage+42↑o ...
00 00 00 00    padding          DCD 0
F8 62 F1 3E    func_key         DCD 0x3EF162F8
65 98 A8 E0    xor_key          DCD 0xE0A89865
09 00 00 00                     DCD 9
B7 F7 01 F6 27+xor_string       DCB 0xB7, 0xF7, 1, 0xF6, 0x27, 0xDA, 0x3A, 0xA2, 0x82
D2 FC C7 BA B6+xor_string1      DCB 0xD2, 0xFC, 0xC7, 0xBA, 0xB6, 0xF8, 0xDB, 0xDF, 0x22
```

Propriétés

- Complexifier l'analyse statitque
- Éviter les attaques "simples"

Obfuscations : mélanges de blocs

```
_SKRedirFun_asm                         ; CODE XREF: _decrypt+3A↓p
                STMFD           SP!, {R4-R11,LR}
                B               .spaghetti_label_1
; End of function _SKRedirFun_asm


; =============== S U B R O U T I N E =======================================


.spaghetti_label_87                     ; CODE XREF: .spaghetti_label_86+10↓j
                MVNS            R4, R4
                SUB             R4, R4, R7
                SUB             R11, R11, R7
                B               .spaghetti_label_88
; End of function .spaghetti_label_87


; =============== S U B R O U T I N E =======================================


.spaghetti_label_241                    ; CODE XREF: .spaghetti_label_240+8↓j
                MOVS            R11, R11
                MVNS            R11, R11
                MVN             R4, R4
                EORS            R4, R4, #0x51
                ADDS            R4, R4, R7
                B               .spaghetti_label_242
```

Obfuscations : EPO

Propriétés

- Technique `EPO` (Entry Point Obscuring)
- Bruteforce du point d'entrée de la fonction à appeler à partir d'une table
- Obfuscation du flot d'exécution par mélange de blocs

Contournement

- Script IDA Python
- Reproduction du bruteforce

Obfuscations : prédicats opaques

Définition

- Un prédicat opaque est un saut conditionnel paraissant vérifier une condition, cette condition étant en réalité toujours vraie ou fausse

Constats

- Les prédicats opaques peuvent être pénibles pour l'analyse statique
- Un prédicat opaque ne résiste pas à l'analyse dynamique
- Cependant, l'usage de modèles mathématiques plus complexes peut rendre l'analyse dynamique très fastidieuse

Obfuscations : prédicats opaques

En pratique: exemple 1

```
double d1 = Math . pow(162 , 3D) ;
double d2 = 162;
if ( ( d1 - d2 ) % 3D == 0D) // Always True {
for ( int i = 0 ;  i<16 ; i++)
abyte2 [ i ] = 0 ;
}
else {
for ( int i = 0 ; i<16 ; i++)
abyte2 [ i ] = 1 ;
}
```

Remarque

- Un émulateur de code permet de passer outre ce type de prédicat

Obfuscations : prédicats opaques

Exemple 2: calcul d'une somme

```
int cpt = 0,sum = 0;
while((cpt < 241) && (333*(((cpt+2)*(cpt+2)) + (cpt+1)*(cpt+1))
 < (666*((cpt+2)*(cpt+1))) )) {
sum += data[cpt++];
}

cpt = 0

while(cpt < 1000) {
sum += data[cpt++];
}
```

Analyse

- Le condition $x^2 + y^2 < 2 * x * y$ est toujours fause

Plan

Restricition d'accès

Restriction réseau

```
CTTelephonyNetworkInfo *netinfo
CTCarrier *carrier = [netinfo subscriberCellularProvider];
[netinfo mobileNetworkCode];
```

Propriétés

- Restriction par rapport au nom de l'opérateur
- Restriction par `MNC` (Mobile Network Code)

Contournement

- Patching de l'application

Restricition d'accès

Restriction réseau - Android

```
if (!telephonymanager.getSimOperatorName().toLowerCase( ).nstartsWith("operateur"))
        return ERROR;
```

Contournement

- Patching

Cependant

- L'opérateur peut effectuer des contrôle en amont, au niveau des BTS
- L'attaquant peut facilement être détecté si la ligne est associée à son nom

Plan

Jailbreak detection sous iOS

Evidemment

- Vérifications réalisées client-side et en userland

Quelques cas classiques

- Recherche de "Cydia", "OpenSSH" ou d'autres applications non approuvées par Apple
- Tester si le port 22 est ouvert
- Tester la présence de fichiers présents sur un périphérique jailbreaké :
  - System/Library/LaunchDaemons/com.apple.absinthed.K93.plist
  - /usr/sbin/absinthed.K93;
  - Tous les fichiers dans /private/var/mobile/Media/corona/
- Options de montage dans /etc/fstab

Plan

Apple HTTP Live Streaming Protocol

Apple HTTP Live Streaming Protocol

- MPEG-2 Transport Stream
- Fichier d'index au format `M3U8` (segmentation, timing, clefs,...)
- Protection du contenu avec `AES-128`
- Livraison des clefs par `HTTP/HTTPS` (hmm...)
- Accélération hardware

Spécifications

http://tools.ietf.org/html/draft-pantos-http-live-streaming-10

Apple HTTP Live Streaming Protocol

Les fichiers d'index M3U8

- URL du serveur qui délivre la PKE
- Un simple accès à l'URL permet d'obtenir la clef
- L'utilisation d'un certificat client nécessite cependant d'instrumenter l'application

```
#EXTM3U
#EXT-X-ALLOW-CACHE:NO
#EXT-X-TARGETDURATION:10
#EXT-X-MEDIA-SEQUENCE:193
#EXT-X-KEY:METHOD=AES-128,URI="http://m3u8.wilmaa.com/php/myStream.php"
#EXTINF:11,
20110521090004_media_193.ts
#EXTINF:10,
20110521090004_media_194.ts
#EXTINF:10,
20110521090004_media_195.ts
#EXTINF:10,
20110521090004_media_196.ts
#EXTINF:10,
20110521090004_media_197.ts
```

Apple HTTP Live Streaming Protocol

Fail

- `wget http://fail.myapp.com/getkey.php`

Solution partielle

- Sous classer `NSURLProtocol`
- Récupérer la clef à l'URL indiquée dans le fichier `M3U8` depuis l'application et non par le service `mediaservd` de Apple
- Utiliser un algorithme symétrique en boîte blanche

Risques

- Rip du code de la boite blanche sans l'analyser

Exemple de whitebox "light"

Comment attaquer?

- Rip du code
- Analyse toutefois nécessaire afin de réaliser une attaque MITM et reconstruire la fonction de déchiffrement
- Réutilisation des clefs fixes et inversion du processus de chiffrement

Questions?

www.quarkslab.com

contact@quarkslab.com | @quarkslab.com