iOS Security

Kieran Twidale-Smith

~/Whoami

Some information of what I do

What I Do

My name is Kieran, I study Computer Information Security at Sheffield Hallam University currently on my second year. In my spare time I research Penetration Testing, iOS Security, Reverse Engineering, Programming, Malware Analysis and help run the SHU Hack Society.


B6011111@my.shu.ac.uk


@TwidsDev


Kieran Twidale-Smith

iOS History

Features And Security Implementations

iPhone OS 1

Released: June 2007

No AppStore | No security what so ever; the wallpaper, which couldn't be changed, was exploited and changed using JavaScript

iPhone OS 2

Released: July 2008

Introduced the AppStore | Introduced Encrypted Filesystem, Bootloader etc..

iPhone OS/iOS 3

Released: June 2009

Introduction to Find My iPhone and introducing remote wipe to everyone not just corporate businesses

iOS 4

Released: June 2010

Dropped support for original iPhone & iPod Touch. Introduced long Alphanumeric passwords rather than 4-digit passcode

iOS History

Features And Security Implementations

iOS 5

Released: October 2011

Introduced Find my Friends and Fixed over 96 vulnerabilities. Ranging from Data leaking to logging AppleID and password

iOS 6

Released: September 2012

Introduced more settings for Privacy. Integrated Facebook and Twitter and fixed a total of 197 Security Flaws.

iOS 7

Released: September 2013

Introduced an SSL Bug called goto Fail bug bypassing SSL Checks. Also introduced Activation Lock and TouchID

iOS 8

Released: September 2014

Introduced more secure Wifi Scan, spoofing the MAC Address when scanning for new networks and fixing 56 security vulnerabilities

iOS History

And Security Implementations

iOS 9

Released: September 2015

Introduced 6-digit passcode along side a range of content blockers for Safari and other web apps. Fixed 105 Vulnerabilities.

iOS 10

Released: September 2016

Dropped Bootloader and kernel encryption??! Separate Code Execution and Introduced Kernel Patch Protection

iOS 11

Released: September 2017

Introduction to FaceID and patches the Kernel Patch Protection bypass vulnerability

iOS Security Features

1

Secure Boot Chain

The bootloaders, kernel, kernel extensions, and baseband firmware is cryptographically checked at boot to ensure its integrity. Booting the device only after verifying the chain of trust.

2

Code Signing

All apps require to be signed by Apple to run. This ensures all executable code has been checked and verified and has come from the original source without being tampered with. This also includes stock applications.

3

Application Sandbox

All third party applications are restricted from accessing or modifying files stored outside the original application.

4

Data Protection

While locked, the device is encrypted with AES-256 bit from a dedicated AES-256 cryptographic engine. Using the devices Unique ID (UID) from the Secure Enclave Processor and the Group ID (GID) fused to make the AES-256 key.

5

Secure Enclave Processor

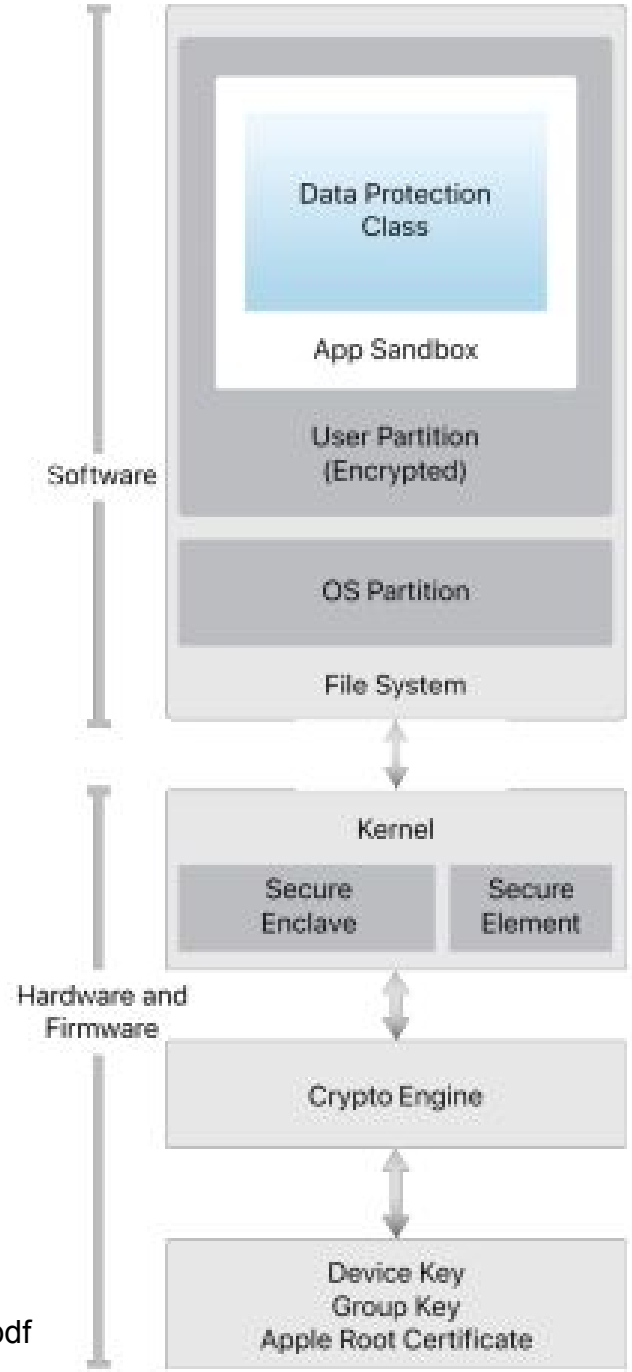
A separate processor is used to provide all cryptographic operations while being in encrypted memory. This processor includes a true random, hardware random number generator. This ensures data integrity even if everything down to the kernel has been compromised.

6

Other Exploit Mitigations

Mitigations like Kernel Patch Protection (KPP) or Kernel Address Space Layout Randomisation (KASLR) to make exploiting difficult.

iOS Diagram



https://www.apple.com/business/docs/iOS_Security_Guide.pdf

What makes a jailbreak?

•••

- ✓ Disable iOS Restrictions
- ✓ Escalated Privileges to root
- ✓ Full R/W access to file system
- ✓ Code Signing Bypass
- ✓ Escaped Sandbox
- ✓ Potentially KPP Bypass

So lets escape from that Jail!

This jailbreak is made possible from CoalFire-Research and all the source-code can be found at https://github.com/Coalfire-Research/iOS-11.1.2-15B202-Jailbreak

To start escaping from this Jail we'll need a vulnerability along with an exploit for that vulnerability. That's where we look at Ian beers async_wake_iOS project which returns tfp0 (tfp stands for task_for_port).

The task port is a mechanism that allows access to read and write to memory in a process which allows us to use offsets directly access code in the kernel. This gives us root! But not access to run unsigned code.

```
build_id: 15B202
sysname: Darwin
nodename: nokia-388
release: 17.2.0
version: Darwin Kernel Version 17.2.0: Fri Sep 29 18:14:51 PDT
2017; root:xnu-4570.20.62~4/RELEASE_ARM64_S8000
machine: iPhone8,1
this is iPhone 6s, should work!
message size for kalloc.4096: 2956
got user client: 0x6207
 [+] prepared kqueue
task self: 0xffffffff11095ef40
our task port is at 0xffffffff11095ef40
found target port with suitable allocation page offset:
0xffffffff112b9fa68
replacer_body_size: 0xb74
message_body_offset: 0x448
0
e00002c9
0
0
1
2
```

Xcode console of async_wake exploit in progress (truncated)

```
198
199
got replaced with replacer port 45
found kernel vm_map: 0xffffffff10a55de80
second time got replaced with replacer port 0
will try to read from second port (fake kernel)
kernel read via fake kernel task port worked?
0x0000000000420000
0x0000000000000000
0xffffffff10a5863c0
0xffffffff10a586410
about to build safer tfp0
message buffer: ffffffff110ab8000
fake_kernel_task_kaddr: ffffffff110ab8000
read fake_task_refs: d00d
about to test new tfp0
kernel read via second tfp0 port worked?
0x0000000000420000
0x0000000000000000
0xffffffff10a5863c0
0xffffffff10a586410
built safer tfp0
about to clear up
cleared up
tfp0: 1888b0b
```

Obtaining tfp0 using async_wake exploit (truncated)

Vulnerabilities used: CVE-2017-13865, CVE-2018-13861

What makes a jailbreak?

...

- Disable iOS Restrictions
- Potentially KPP Bypass
- Escalated Privileges to root
- Full R/W access to file system
- Code Signing Bypass
- Escaped Sandbox

So lets escape from that Jail!

This jailbreak is made possible from CoalFire-Research and all the source-code can be found at https://github.com/Coalfire-Research/iOS-11.1.2-15B202-Jailbreak

To run unsigned code we need to run processes with PID 0 Credentials (Process ID). In other words run the code or process as root. So how do we do this?

Find PID0 and use the offset 0x100 to find the credentials. Find my PID and overwrite my credentials with the ones we found. Loop through all active threads and copy the credentials and Done!

```
build_id: 15B202
sysname: Darwin
nodename: nokia-388
release: 17.2.0
version: Darwin Kernel Version 17.2.0: Fri Sep 29 18:14:51 PDT
2017; root:xnu-4570.20.62~4/RELEASE_ARM64_S8000
machine: iPhone8,1
this is iPhone 6s, should work!
message size for kalloc.4096: 2956
got user client: 0x6207
 [+] prepared kqueue
task self: 0xffffffff11095ef40
our task port is at 0xffffffff11095ef40
found target port with suitable allocation page offset:
0xffffffff112b9fa68
replacer_body_size: 0xb74
message_body_offset: 0x448
0
e00002c9
0
0
1
2
```

Xcode console of async_wake exploit in progress (truncated)

```
198
199
got replaced with replacer port 45
found kernel vm_map: 0xffffffff10a55de80
second time got replaced with replacer port 0
will try to read from second port (fake kernel)
kernel read via fake kernel task port worked?
0x0000000000420000
0x0000000000000000
0xffffffff10a5863c0
0xffffffff10a586410
about to build safer tfp0
message buffer: ffffffff110ab8000
fake_kernel_task_kaddr: ffffffff110ab8000
read fake_task_refs: d00d
about to test new tfp0
kernel read via second tfp0 port worked?
0x0000000000420000
0x0000000000000000
0xffffffff10a5863c0
0xffffffff10a586410
built safer tfp0
about to clear up
cleared up
tfp0: 1888b0b
```

Obtaining tfp0 using async_wake exploit (truncated)

```
tfp0: 1888d0b
[+] Attempting to obtain root
[i]     old:
[i]     uid=501 gid=501 euid=501 geuid=501
[d] Patching ourselves with pid(0) at 0xffffffff013801cd0
[d] Old creds: 0xffffffff1154aff00
[d] Patching our creds with 0xffffffff112325ef0
[i]     new:
[i]     uid=0 gid=0 euid=0 geuid=0
[+] Got ROOT!!!!!
[+] Reverting privs to avoid a crash...(getuid=501)
```

Vulnerabilities used: CVE-2017-13865, CVE-2018-13861

What makes a jailbreak?

...

- Disable iOS Restrictions
- Potentially KPP Bypass
- Escalated Privileges to root
- Full R/W access to file system
- Code Signing Bypass (We're running self signed code not unsigned code)
- Escaped Sandbox (We escaped the sandbox for the exploit application but not the full sandbox)

So lets escape from that Jail!

This jailbreak is made possible from CoalFire-Research and all the source-code can be found at https://github.com/Coalfire-Research/iOS-11.1.2-15B202-Jailbreak

While running as root is cool, we don't have access to the root filesystem. This is due to the file system partition being mounted as READ ONLY.

Because of sandbox protection we also can't write to /tmp, /var or /private; along with a huge amount of other files. For any binaries we run to get root filesystem access, we'll have to run them from a directory we created. So we'll call this /Jailbreak. This will be the directory that stores all our unsigned/selfsigned code.

We can't remount the filesystem because the Kernel forces these protections which could result in an unsuccessful attempt of mounting the filesystem or just a kernel panic.

A known solution for this is to turn the flag MNT_ROOTFS off, quickly remount the file system, then turn the flag back on before a check runs from the kernel.

What makes a jailbreak?

...

- Disable iOS Restrictions
- Potentially KPP Bypass
- Escalated Privileges to root
- Full R/W access to file system
- Code Signing Bypass (We're running self signed code not unsigned code)
- Escaped Sandbox

So lets escape from that Jail!

This jailbreak is made possible from CoalFire-Research and all the source-code can be found at https://github.com/Coalfire-Research/iOS-11.1.2-15B202-Jailbreak

The kernel has multiple mechanisms to combat and hijack unsigned code. Apples new filesystem type AMFI can make running unsigned code easier as it runs a Daemon called AMFID. This Daemon is a weak link for us to use. When sent a request to run a binary, the kernel captures the request and calls a function in AMFID. This is used to validate whether the code is signed and return with if it is signed or not.

Using an exploit called mach_portal with a few tweaks, again by Ian Beer, you can register a exception for AMFID that gives us the ability to run the function that returns “Signed”. We send a bad address to AMFID which it can’t take, it runs our exception which tells the kernel “Yep, its signed by Apple. Go run the code”.

```
Pseudo code:
Declare signed as Boolean
If signed = false then
          DON’T RUN CODE, ITS UNSIGNED
If signed = true then
          RUN THE CODE, ITS VERIFIED
     Exception //We add this Exception to tell it what to do if I can’t handle the data
          RUN THE CODE, ITS VERIFIED

Lets say the user runs: signed=1234 into the code above
```

What makes a jailbreak?

...

- Disable iOS Restrictions
- Potentially KPP Bypass
- Escalated Privileges to root
- Full R/W access to file system
- Code Signing Bypass
- Escaped Sandbox

So lets escape from that Jail!

This jailbreak is made possible from CoalFire-Research and all the source-code can be found at https://github.com/Coalfire-Research/iOS-11.1.2-15B202-Jailbreak

●●●

And the last but by no means least part of the jailbreak, Disabling iOS Restrictions.

When Jailbreaking the worst thing to happen is for your device to update by its self using OTA (Over the Air) Updates. This can brick a jailbroken device forcing the user to restore or remove the jailbreak.

To combat this you can just edit the hosts file, because, you know… you're root. So all you'd have to do is run the command

```
iPhone: ~ root# echo '127.0.0.1  mesu.apple.com' >> hosts
```

This adds 127.0.0.1 mesu.apple.com to the end of the hosts file which will point all requests wanting to go to mesu.apple.com to the loopback address at 127.0.0.1. Stopping your iDevice from updating.

What makes a jailbreak?

...

- ✓ Disable iOS Restrictions
- ✓ Escalated Privileges to root
- ✓ Full R/W access to file system
- ✓ Code Signing Bypass
- ✓ Escaped Sandbox
- ✓ Potentially KPP Bypass

Thank you

Any Questions?

Kieran Twidale-Smith

@TwidsDev

Email address
B6011111@My.shu.ac.uk