Consequences of a Jailbroken iDevice

Part of the Reverse Engineering and Code Modification Prevention Umbrella
Agenda

• A Brief History of iOS Attacks
• iOS Security Basics
• Jailbreaking 101
A BRIEF HISTORY OF IOS ATTACKS
iPhone 2007

- Security... what's that?
  - Stripped-down OS
  - No privilege separation:
    All processes ran as root.
  - No code-signing enforcement
  - No DEP
  - No ASLR
  - No sandboxing
  - No app store
Libtiff Vulnerability (iOS 1.1.2)

Victim surfs malicious web site and attacker gets remote ‘root’ access to device

1. Get user’s Safari Browser to view a malicious TIFF image;
2. Browser’s *libtiff* library attempts to render image;
3. TIFF contains malicious input that renderer places into heap memory and executes
CommCentre Vulnerability (iOS 2)

An attacker can remotely eavesdrop on conversations, monitor user’s location, or force victim’s phone to join a botnet by sending an SMS to the victim.

1. Attacker sends victim malicious incoming SMS that contains code
2. CommCentre parses incoming SMS and buffer overflow occurs
3. CommCentre executes code as root user
Rick Astley is “Never Gonna Give You Up” *Ikee Worm* (iOS 2)

Jailbroken devices typically contained an SSH server with default root passwords...

1. Worm spread between Jailbroken devices that used the same default passwords.
2. Later incarnations performed more malicious acts than simply changing wallpapers
3. Lock phone for ransom, stealing content, botnet enrollment
Storm 8 (2009)


1. Apps collected cell phone numbers of the devices on which users were playing; 20 million downloads!
2. Uploaded data to Storm 8 servers
3. Resulted in class-action lawsuit; Storm8 claims it was a simple mistake...
SpyPhone (2010)

SpyPhone accesses every possible piece of information exposed through the sandbox

- Cell-phone number
- Read / write access to address book
- Safari / YouTube search terms
- E-mail account information
- Keyboard cache
- Geotagged photos
- GPS Information
- WiFi access point names

Even inside a ‘safe’ environment, a malicious app can extract a frightening amount of information
Pwn2Own (2010)

Annual competition in which contestants exploit mobile devices with unknown vulnerabilities for cash, their device, and a ‘masters’ jacket...

• Winners of 2010 discovered vulnerabilities in MobileSafari of iPhone 3GS that allow remote execution of code;

• Their test code opened up the SMS database and sent the entire database’s contents to a remote server;

• They won $15,000 and an iPhone 3GS
Attacker can silently jailbreak victim’s device by tricking victim into visiting a malicious web site

1. User visits malicious site;
2. Attacker exploits MobileSafari stack-overflow vulnerability in its font rendering code to execute foreign code within the browser;
3. Malicious payload exploited second vulnerability (integer overflow) that allowed privilege escalation to get increased access to device;
4. Attacker can now execute code within kernel space
5. Attacker disables code-signing
6. Attacker downloads remote libraries and silently jailbreaks device.
Apple quickly patched iOS to mitigate ‘Star’ vulnerability. Apple also introduced ‘ASLR’ around the same time.

Attacker achieved ‘drive-by’ Jailbreaking by exploiting critical vulnerabilities:

- Ability to execute foreign code
- Ability to raise privilege escalation to disable code signing (within kernel space)
- Ability to bypass ASLR
iOS Security Controls Strategy

- Historical vulnerabilities illustrate the importance of particular security iPhone security controls:
  - App Encryption
  - Data Execution Prevention
  - Code Signing
  - Address Space Layout Randomization
  - Sandboxing
iOS Security Controls
App encryption
iOS Security Controls

Code signing
iOS Security Controls

- Code execution policies
  - ASLR
    - Address Space Layout Randomization
  - \( W^X \) Memory pages
    - No self-modifying code
  - Stack canaries
iOS Security Controls
Sandboxing
Circumventing iOS Controls

• Jailbreaking
  – Remove iOS controls
  – Gain root access
  – Custom kernel
  – Privilege escalation
Jailbreaking Motivation

• Jailbreaking removes critical security controls from the iOS and allows an attacker to return to these earlier vulnerabilities we’ve already seen...

• Why jailbreak?!
  – Adding features
  – Carrier independence
  – OS customization
  – Security auditing
  – Piracy
  – Espionage/Forensics
Jailbreak History

- iPhone 1.0 (released June 29th 2007)
  Broken July 10th 2007
- 4.3.2
  redsn0w 0.9.11x (Broken April 2011)
- 4.3.3
  jailbreakme.com remote jailbreak (Broken July 2011)
- 5.1.1
  absinthe 2.0.x (Broken May 2012)
- 6.1
  evasion (Broken Jan 30 2013)
- 7.0
  evasion (Broken December 22 2013)
Jailbreak History

- Time to jailbreak increases when:
  - New OS versions
  - New hardware versions
- Apple continually patches known exploits
Ramifications of Jailbreaking

• If a device is Jailbroken, all bets are off...
  – Application encryption is not enforced
  – Code-signing is disabled
  – Arbitrary remote code execution is possible
  – Exploitation of other applications on the device is entirely possible
• The attack is really limited by the imagination of the attacker
Ramifications of Non-Jailbreaking

Even if you are running on a non-Jailbroken device, there are plenty of remote-execution vulnerabilities out there...
Ramifications of Non-Jailbreaking

• SpyPhone illustrates the problems of running things on a non-Jailbroken device:
  – AppStore approval process is not transparent or rigorous;
  – Sandboxing is far too permissive and allows access to all sorts of things apps shouldn’t have access to;
  – Information disclosure to third-parties is probably the greatest risk a user will face

• iOS Security controls are not bug-free
  – Check out the CVE iOS Security Vulnerabilities database to see the latest and greatest security exploits
Conclusions

• Jailbreaking teaches us a lot about the pitfalls of iOS security

• There are plenty of bad things that can happen in non-jailbroken environments

• Jailbreaking is not going away anytime soon

• In your code, always test for the presence of a jailbroken environment. There’s a lot of risk in these environments

• Follow xcon for more information about how to reliably detect jailbroken environments