Security Evaluation of Apple's iPhone

Charlie Miller
Jake Honoroff
Joshua Mason

Independent Security Evaluators

{cmiller, jake, josh} AT securityevaluators.com

July 19, 2007

Executive Summary

The Apple iPhone was released to much fanfare on June 29, 2007. Because of the large amount of personal information stored on these mobile devices, we decided to conduct a security analysis of the iPhone. The iPhone's applications for surfing the web and checking emails are potentially at risk to remote attacks. We wanted to determine exactly how well the software on the iPhone was designed to resist such attacks.

The iPhone runs a stripped down and customized version of Mac OS X on an ARM processor. Much of the device's claimed security is reliant on its restrictions against running third party applications. Only Javascipt code can be executed in the Safari web browser, ensuring that all such code executes in a “sandbox” environment. Many of the features of Safari have also been removed, such as the ability to use plug-ins such as Flash. Likewise, many filetypes cannot be downloaded. These actions serve to reduce the attack surface of the device.

However, there are serious problems with the design and implementation of security on the iPhone. The most glaring is that all processes of interest run with administrative privileges. This implies that a compromise of any application gives an attacker full access to the device. Like the desktop versions of Mac OS X on which its operating system is based, the iPhone also does not utilize widely accepted practices, such as using address randomization or non-executable heaps, to make exploitation more difficult. These weaknesses allow for the easy development of stable exploit code once a vulnerability is discovered.

To demonstrate these security weaknesses, we created an exploit for the Safari browser on the iPhone. We used an unmodified iPhone to surf to a malicious HTML document that we created. When this page was viewed, the payload of the exploit forced the iPhone to make an outbound connection to a server we controlled. The compromised iPhone then sent personal data including SMS text messages, contact information, call history, and voice mail information over this connection. All of this data was collected automatically and surreptitiously. After examination of the filesystem, it is clear that other personal data such as passwords, emails, and browsing history could be obtained from the device. We only retrieved some of the personal data but could just as easily have retrieved any information off the device.

Additionally, we wrote a second exploit that performs physical actions on the phone. When we viewed a second HTML page in our iPhone, it ran the second exploit payload which forced it to make a system sound and vibrate the phone for a second. Alternatively, by using other API functions we discovered, the exploit could have dialed phone numbers, sent text messages, or recorded audio (as a bugging device) and transmitted it over the network for later collection by a malicious party.

Apple was notified of these findings, including detailed technical documentation, on July 17th. While this paper serves to highlight our findings, we will not release the remaining technical details until August 2nd. This delay is provided in order to give Apple sufficient time to produce patches so that attackers cannot take advantage of these vulnerabilities.

Introduction

After months of anticipation, the iPhone was finally released to long lines of customers on June 29, 2007. Touted by some articles as the most "successful product introduction of the 21st century", we felt it was important to examine this device for its overall security posture.

Mobile devices, such as the iPhone, are becoming ubiquitous in society. Individuals talk to their friends, conduct business, shop online, and listen to music, all from these small portable computers. Some of these devices, including the iPhone, provide data transmission over cellular communications and also through WiFi connections when accessible. These mobile devices are often used in corporate settings and travel in and out of the physical building, passing from outside the firewall into the heart of the protected network of enterprises. People store their address books, passwords, email, and other personal data on these devices. For these reasons, it is important that these devices are designed and implemented in as secure a manner as possible.

In this paper, we examine the security of the Apple iPhone. We highlight some strengths in the design and point out some significant problems. We demonstrate the security deficiencies by producing an exploit which takes advantage of a flaw in the MobileSafari application to gain complete control over the device.

Public Developments

Prior to this work, there was a large amount of work by independent researchers in an effort to access and unlock the iPhone. Much of this work was reported on the #iphone-dev IRC channel. One of the first discoveries was made by a noted researcher Jon Lech Johansen, known as "DVD Jon". He figured out a way to access the WiFi capabilities of the iPhone without activating it. Next, a tool was developed called *iPhoneInterface*. This tool, using the same API functions used by the iTunes player, was able to access the small portion of the filesystem accessible to iTunes. The user of this program could navigate through the file system and retrieve and place files on the device. The next step was a program developed called *jailbreak*. This program modified the iPhone in such a way that the entire filesystem could be accessed through the iPhoneInterface application. This did have the drawback that syncing was impossible after using the tool. One of the latest developments includes the ability to activate the phone by only using AT&T prepaid cards. Only in the last couple of days did this group of researchers develop the ability to produce binaries for the iPhone. At the writing of this document, no one has been able to unlock the phone.

Overall Security Architecture

The Apple iPhone runs a stripped down version of the Mac OS X operating system commonly found on Apple desktops and servers. The major difference is that it runs on an ARM processor instead of x86 or PowerPC processor. ARM processors are common in mobile devices due to their lower power consumption. The iPhone comes with a version of the Safari web browser called MobileSafari. This version of the browser is very similar to the typical Safari application and they share much of the same code base. However, many of the features of Safari have been removed in MobileSafari. Likewise, MobileMail is the iPhone mail client and is a stripped down version of the Mail program found on Mac OS X. There are also programs to send SMS text messages, view YouTube videos, and check stocks, among others. The iPhone is capable of sending and collecting data using the EDGE network provided by AT&T wireless. Alternatively, it can send data via WiFi connections and also has Bluetooth capabilities. An iPhone is activating by communicating with the iTunes

program running on a computer over a USB connection. This USB connection is also the way the iPhone retrieves and syncs data from desktop computers, allowing syncing with Calendars, address books, mail settings, etc. It also is the mechanism whereby music and videos can be placed on the iPhone.

The security architecture of the iPhone can best be described as one of reducing the attack surface. This is accomplished by limiting the number of applications on the device and limiting the functionality of the existing applications. The device does not even contain common binaries such as *bash*, *ssh*, or even *ls*. For an example of the reduced functionality of the applications actually present, consider MobileSafari. On a normal Mac OS X computer, Safari comes with the ability to display Flash, while MobileSafari does not have this capability. Likewise, when downloading archives like dmg or zip files, Safari will launch the appropriate helper applications, save them to disk, and then mount or uncompress them. MobileSafari simply refuses to download these file types. Similar limitations are placed on MobileMail as many attachment types supported on standard Mac OS X installs are not supported by MobileMail. Likewise, the iPhone does not listen on any TCP or UDP ports for incoming data. This reduction of the type of data processed by the device is an effective way to reduce the exposure to potential vulnerabilities.

Continuing on this trend of reducing exposure, the Apple iPhone does not allow third party applications to run on the device. This is accomplished by denying access to the file system to the USB connection and not supplying an appropriate SDK. Similarly, a technical difficulty exists in that there is no "toolchain" in which to build applications for Mac OS X running on an ARM processor. That is, no compiler exists outside of Apple which can build applications which would run on the iPhone, even if the file could be placed in the file system. (Some of these considerations have been changed by the progress of researchers attempting to open these devices, as discussed in the previous section). By not allowing 3rd party applications, again the number of opportunities for attackers is diminished.

Unfortunately, once an iPhone application *is* breached by an attacker, very little prevents an attacker from obtaining complete control of the system. All the processes which handle network data run with the effective user id of 0, i.e. the superuser. This means that a compromise of any application gives the ability to run code in the context of that application which has the highest possible privilege level. Additionally, no address randomization was used in by the operating system. This means that each time a process runs, the stack, heap, and executable code is located at precisely the same spot in memory. This helps attackers write reliable exploit code by allowing them to guess the layout of memory from run to run of an application and even from device to device. Most modern operating systems incorporate some sort of address randomization. Additionally, the heap (and possibly the stack) is executable. Again, this has the effect of making exploit development easier for an attacker as it allows them to simply place their code on the heap and jump to it once they have control of the program. Had these precaution been taken, it would have forced attackers to use more sophisticated methods of exploitation such as return-to-libc. Therefore, while precautions were made to reduce the amount of code available to a remote attacker, once a vulnerability is located it is relatively easy for them to successfully exploit and obtain complete control of the device.

As briefly mentioned above, the iPhone designers restricted access to the filesystem on the device. While a USB connection to the device is established for syncing with iTunes, the filesystem cannot be mounted. Even more, the filesystem accessible to iTunes is chroot'ed such that only a small set of the filesystem is visible over this connection. This set of the filesystem does not include the binaries or libraries available. This lim-

ited access to the filesystem doesn't particularly serve a security role from the perspective of a remote attacker. Instead, this serves as an example of design intended to protect the exclusivity of the iPhone to AT&T. If more thought had gone into protecting the applications from remote attack and less on preventing the unlocking of the device, the overall security of the device might have been improved.

Suggested Improvements

The security design of the Apple iPhone concentrates all of its defenses on reducing the device's expose to vulnerabilities. However, it does not provide any defense once an attack has occurred. We recommend using a *layered* approach to security. Based on the findings of the security design and implementation of the device discovered, we recommend the following changes to be made to the iPhone.

- Install applications such that they run as an unprivileged user. This would result in a successful attacker only gaining the rights of this unprivileged user.

- *chroot* the applications such that they can not access the personal data needed exclusively by the other applications. For example, MobileSafari would not have access to email or SMS messages. Likewise, MobileMail would not have access to browsing history.

- Add heap and stack address randomization. This will serve to make the development of exploits for vulnerabilities more difficult.

- Use a memory protection scheme such that no memory pages are both writable and executable. Again, this serves to make exploit development much more difficult.

Vulnerability Analysis

In order to find vulnerabilities on the iPhone, a few options are available to a researcher. Using jailbreak and iPhoneInterface, the binaries can be extracted from the device and statically analyzed, using a disassembler. Additionally, since the MobileSafari and MobileMail applications are based on the open source WebKit project, a source code audit of that package can be performed. Finally, dynamic analysis, or fuzzing, can be executed against the device. This involves sending malformed data to the device in an effort to cause a fault and make it crash. Such fuzzing can be performed against applications such as MobileSafari or against the WiFi or BlueTooth stack.

The vulnerability we discovered and exploited was found in MobileSafari using fuzzing.

Attack Scenarios

There are at least a couple of vectors whereby a MobileSafari exploit can be used against the iPhone. The first is via email. A link to a malicious site can be included in an email sent to the victim. When the victim clicks the link, they will be taken to the webserver containing the malicious HTML and the exploit will take control of their device.

A more subtle approach involves a man in the middle attack. An attacker could set up and advertise a free WiFi hotspot in a heavily populated area. The iPhone will automatically seek these out and ask the user to connect to them. Once connected, all traffic from the victim will pass through the attacker controlled wireless router. The attacker can intercept and change any HTTP traffic intended for the victim. This traffic can invisibly be modified to contain the iPhone exploit code. Again, complete control will be obtained over the iPhone. This time the only actions performed by the victim include using an unsafe WiFi connection and surfing to any website. This last scenario is aided by the fact that iPhones advertise their existence via HTTP

headers. In this manner the exploit code can be delivered only to iPhones and not other devices and browsers.

A similar attack can occur if an attacker controls the DNS used by an iPhone user. In this case, the attacker can point the iPhone to the page containing the exploit whenever the the iPhone attempts to go to a new page. This method of attack requires no interaction from the user, only that they are using the web browser.

Blackbox Exploitation

Once a vulnerability has been identified, the next step is developing a functioning exploit. This is very difficult for a device such as the iPhone since there is no way to run a debugger against the target application in order to view memory and discover the way the execution flows in the application. However, in this case we were able to utilize the Mac OS X crash reporter. This daemon runs and monitors any programs for crashes. When one is detected it records a log of the crash, including relevant register values. These reports can then be transported to a desktop computer when syncing. The crash reports can also be downloaded directly off the iPhone using jailbreak and iPhoneInterface.

While the CrashReporter provides register values and basic memory mapping information, it does not include direct access to the memory. In order to obtain this crucial information, it is possible to modify the iPhone in such a way that the applications will dump core files when they crash. This is accomplished by adding the file /etc/lauchd.conf containing the line

```
limit core unlimited
```

to the iPhone using iPhoneInterface. Core files can be retrieved off the iPhone from the /cores directory, again using iPhoneInterface. These core files can be read by a debugger like *gdb*. While standard gdb cannot display the context information contained in the core files, it can correctly access the virtual memory contained within it. Between the crash reports and the core files, a complete snapshot of the the application when it crashed is obtained.

In order to obtain data from the application when it wasn't crashing, such as at a particular instruction of interest, we needed to modify the application. To do this, we first obtained the application being exploited off the iPhone filesystem, using iPhoneInterface. We then disassembled it, and modified it such that it would crash at the instruction we wanted to investigate. For example, we could modify the instruction to set the program counter to zero at that time. Placing the binary back on the iPhone, sending data to it, and finally collecting the crash report and core file, gave all the information needed. While tedious, this technique was effective.

Of course, this is a far cry from having a debugger on the system, but it was enough to successfully develop a successful attack against this platform. Had the addresses been randomized, it would have made a hard job even more difficult.

Blackbox Shellcode Development

In order to generate valid opcodes for the iPhone, we first installed a Linux x86 to ARM cross compiler. This would compile our ARM assembly to bytecode which we could then extract into shellcode.

Besides not having a debugger, developing iPhone shellcode also presented other challenges. Since we didn't have access to an ARM processor with a debugger, we had absolutely no real way to test the shellcode besides trying it and using the core files obtained. This greatly increased our development time.

Vulnerability Details

This section is temporarily omitted. Full details will be released on August 2nd. This delay is provided so that Apple has time to develop appropriate patches to mitigate these vulnerabilities.

Conclusions

Due to the widespread attention paid to the iPhone and the security implications of mobile devices in general, we performed a security evaluation of the new Apple iPhone. While Apple takes some precautions to minimize the amount of code accessible to remote attackers, it did not take other basic precautions in designing a robust security solution for the device. While made more difficult due to the closed nature of the device, with little effort we were able to find a vulnerability in the iPhone. We were then able to leverage this vulnerability and use it to write an exploit which could extract personal information off the device without the user ever knowing.

Acknowledgments

We'd like to thank the ISE management for allowing us some time to look at these devices. We'd also like to thank some of our coworkers for their help on this project: Sam Small, Adam Stubblefield, and Matt Green.

References

Mac OS X internals: A Systems Approach, Amit Singh, Addison Wesley, 2006

Mac OS X Debugging Magic: http://developer.apple.com/technotes/tn2004/tn2124.html

CrashReporter: http://developer.apple.com/technotes/tn2004/tn2123.html

DTrace to be included in Next Mac OS X: http://sun.systemnews.com/articles/102/2/news/16842

OS X Heap Exploitation Techniques http://felinemenace.org/papers/p63-0x05_OSX_Heap_Exploitation_Technqiues.txt

Hack a Mac, get $10,000 http://news.com.com/8301-10784_3-9710845-7.html

Safari for Windows: Released and hacked in a day http://www.infoworld.com/article/07/06/11/Safari-for-Windows-released-and-hacked-in-a-day_1.html

With Windows port, a bug-hunting Safari for Apple http://www.infoworld.com/article/07/06/12/With-Windows-port-a-bug-hunting-Safari-for-Apple_1.html

Mac OS X PPC Shellcode Tricks http://uninformed.org/?v=1&a=1&t=pdf

Building and Testing gcc/glibc cross toolchains http://www.kegel.com/crosstool/

"Into my ARMS" Developing StrongArm/Linux shellcode http://isec.pl/papers/into_my_arms_dsls.pdf

Main Page - The iPhone Dev Wiki http://iphone.fiveforty.net/wiki/index.php?title=Main_Page