6.857: Pebble Smartwatch Security Assessment

Thomas Boning *tboning@mit.edu*

Ceres Lee *cereslee@mit.edu*

Steven Valdez *dvorak42@mit.edu*

May 12, 2014

ABSTRACT

The Pebble Watch is a popular device in a new breed of technology. It is one of the first consumer smartwatches able to run arbitrary code and pair with an Android or iPhone mobile phone in order to get text alerts, weather data, and other application data. The Pebble Watch provides an interface for the Watch applications to have access to persistant storage and phone application communication; however, these features also make it a potential attack vector for attackers to acquire sensitive data such as position information, text messages, and other private data from the phone. For our 6.857 Final Project, we performed a security analysis on the device to attempt to find any vulnerabilities or exploitable attack vectors to access this private data or to break the implicit security assumptions of the Pebble Watch.

1 Introduction

The goal of our project was to perform a security analysis on the Pebble Watch, released by Pebble Technology. The Pebble Watch is one of the first consumer smartwatches that can run arbitrary applications. The Watch also pairs with an Android or iPhone device in order send and receive data with phone applications. This information, such as text alerts or weatherdata, can be displayed to the user or used by an application running on the Pebble Watch. As the number of accessories and "smart" gadgets increases each year, the number of devices running custom software and containing potentially private data increases, which widens the attack surface to a users confidential data. Devices like the Pebble and Google Glass provide new attack vectors that have not been thoroughly analyzed for security vulnerabilities. Such devices have the potential to leak any sensitive information stored on a paired phone, be it personal contact information, text messages, or other data. As these accessories get smarter and become networked as part of the "Internet of Things", users should take into account the potential for attacks to use the accessories as carriers for malicious code or potential Trojan horses that can invade the secure sandbox that keeps your phone and data safe from prying eyes and hijacking, just to name a few potential dangers.

2 Attack Vectors

In our survey of the Pebble Watch, we found a number of potential attack vectors that we examined and attempted to exploit in order to access the private and sensitive data that is on the paired device. The primary attack vectors we

looked into were:

- **The Bluetooth stack on the Pebble.** If it were possible to hijack the connection between the pebble and the device, we would have full access to both the pebble and the phone.
- **The capabilities of Pebble Watch applications (Watchapps).** We wanted to explore the possiblities of using apps to access restricted memory on the watch or the phone.
- **Applications on the phone that connect to the Pebble Watch.** Pebble Watchapps can bundle javascript that runs inside the main Pebble phone application. Seperate phone applications can also interface with the Pebble phone app to communicate with the Pebble Watchapps.
- **Hardware exploits.** We considered the possiblity of physical hardware exploits on the watch, but in order to do so, we would have to physically break open our Pebble watch. Since we only had one watch and we did not want to pay for a second one, we decided not to explore these possibilities for our project.
- **Malicious apps on the phone.** It may have been possible to access resources of Pebble connected applications through malicious apps on the phone or device itself. However, we decided that since this involved hacking the mobile device applications without involving the Pebble, malicious applications fell outside the scope of our project. These were thus not explored.

3 Bluetooth

One of the attack vectors we initially looked into was the Bluetooth connection between the Pebble and a paired Android or iPhone mobile device. The Pebble uses a Bluetooth stack that primarily supports Bluetooth 2.1 with Simple Secure Pairing as the default and secondarily supports Bluetooth 4.0 with EDR for iOS7 iPhone devices. This connection is used by the Pebble to communicate with the phone to upload new applications and also to transmit data such as text messages and weather alerts.

Since there are no permission requirements or even confirmations on the side of the Pebble Watch to upload new Watchapps and because the Pebble receives most of the private information over the Bluetooth channel, this was one of the first attack vectors that we looked into. We found little transparency on the code running on the Pebble and likewise could not find the details about the Bluetooth stack that was being used. Coupled with the fact that the Pebble uses a non-standard OS, we though that it would be a viable target for exploitation. The main areas of the Bluetooth stack that we hoped to exploit were either causing the watch to communicate using an older and more insecure version of the Bluetooth protocol or finding issues with the way the Bluetooth 2.1 protocol was implemented. Due to the nature of Bluetooth 4.0 and lack of many 4.0 capable Bluetooth devices, we did not focus on that protocol as part of our security analysis.

Unfortunately, even after running most of the common exploits against the watch, we were unable to get any positive results that did not require the user

to approve a Bluetooth connection from the attacker. We found that, by default, the watch uses Bluetooth 2.1 with numeric comparison. In this mode, a number appears on both the watch and the device the watch pairing with. The user must make sure that the numbers that appear on both screens are the same and then approve the connection. The bluetooth connection cannot be formed without the user's approval. This mode was proven secure in 2008, so we found it was not possible to middle-man the connection between the watch and the device.[2] Though, as a result of the experimentation, we discovered that once a connection is established, the Pebble Watch completely trusts the device, and arbitrary code can be pushed onto the Pebble Watch from the approved device. While the lack of any hardware or software exploits on the Bluetooth stack prevent an attacker from uploading arbitrary executables to the device without the user, it does provide a smaller attack vector. The user of the Pebble smartwatch may accidently accept the malicious connection, given that the UI design of the accept screen on the Pebble is small and rather unremarkable. At that point, an attacker can exploit potential vulnerabilities using malicious Pebble Watchapps.

4 Pebble Watchapps

The meat of the Pebble that sets it apart from other watches on the market is its ability to run arbitrary programs on the hardware. These programs tend to be called Watchapps or Watchfaces, since a majority of them tend to be different visualizations for the current time. These Watchfaces have access to a limited API that Pebble provides, seemingly preventing people from running any sort of

harmful code on the device. Fortunately, the code that is being run on the Pebble is compiled C code, which allows a programmer to access some parts of memory that the API doesn't provide access to by using "invalid" pointers that allow the user to dump various parts of memory.

The Pebble only allows one Watchapp at a time to run, though up to eight Watchapps can be loaded onto the Pebble. The Pebble uses the UUID to distinguish between Watchapps. Additionally, the watch has a limited amount persistant storage, accessed per Watchapp as a key-value store. Both the loaded Watchapps and the persistant memory are stored in the Flash Memory of the Pebble Watch.

One attack that we discovered is to fake the UUID of a Watchapp to be the UUID of another Watchapp that uses persistant storage (Figure 1). This is fairly easy, since Watchapps have a json manifest that lists the UUID in string form. Then, when the malicious app is installed, it causes the previous app to be unloaded from the Pebble Watch. However, the persistant storage is not cleared by this, unlike normal Watchapp unloading. Thus, the new app has access to the older app's storage data. However, there is no authentication that the apps have the same author or are even similar in nature. In practice, the user would notice one of their watchapps being deleted for no apparent reason, especially since this requires the user to install the malicious app. This would allow malicious access to a targetted watchapp's storage based on the UUID. Although the key value store is based on a 32 bit key, this could also be extracted from the targetted Watchapp.

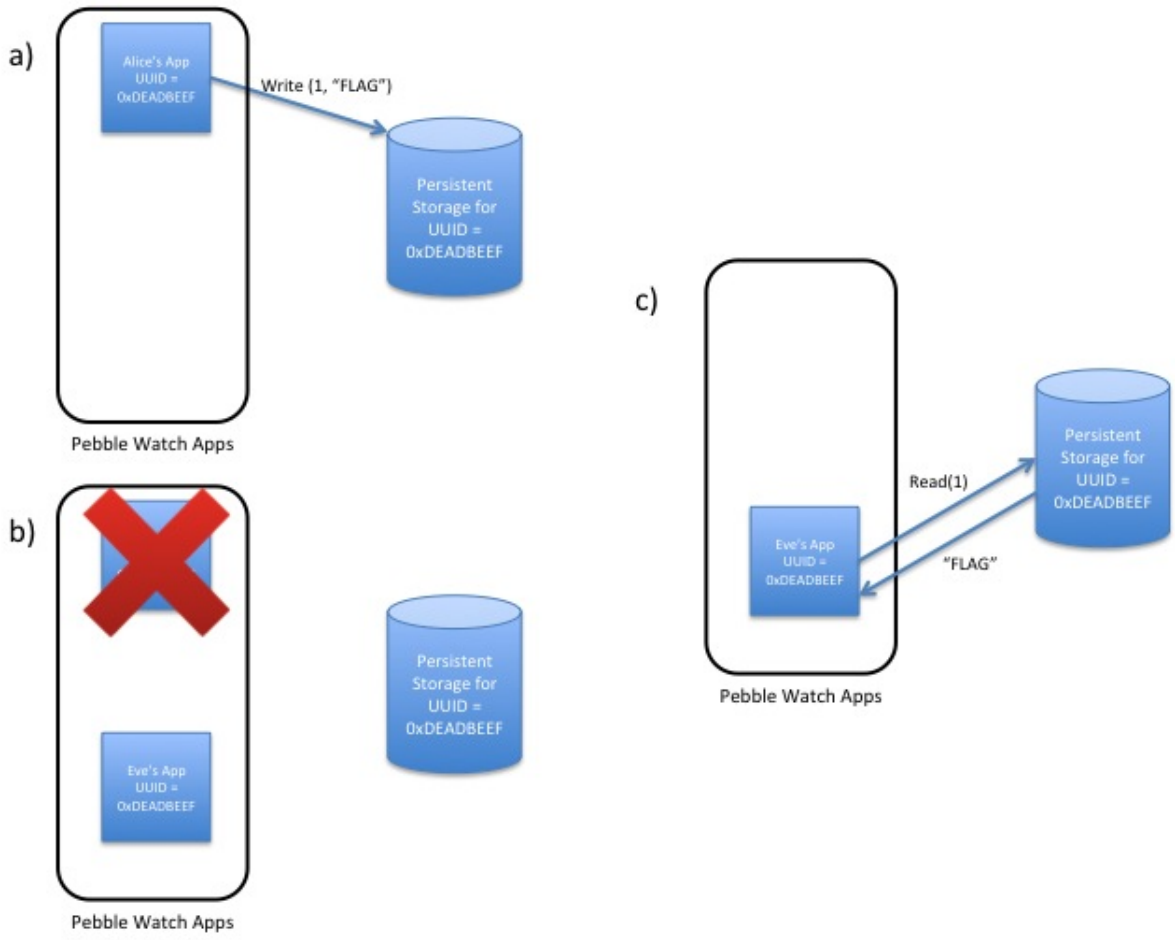

Figure 1: An example of how to read another application's memory. **a)** Alice loads an app onto the watch with `UUID = 0xDEADBEEF`. The watch creates persistent storage for the application and only allows an application with `UUID = 0xDEADBEEF` to read from this persistent storage. **b)** Eve deletes Alice's app on the watch, and Eve uploads her own app also with `UUID = 0xDEADBEEF`. **c)** Eve's app reads from persistent storage. Since Eve's app shares a `UUID` with Alice's now-deleted app, the watch allows Eve's app to read from the persistent storage from Alice's app. The persistent storage only checks the `UUID`, so Eve is able to read the memory, as well as anything written to memory by Alice's app.
(Note: The actual UUID structure is longer and takes the form of "6aaaaaaa-0000-0000-0000-000000000000". In this example, we used a shorter UUID, but the same example still holds.)

4.1 Arbitrary Memory Access

The ability to dump memory using "invalid" pointers allowed us to loop through the flash memory and dump about half of it. Unfortunately, we discovered that what we had originally believed was arbitrary memory access was only arbitrary up until halfway through the Flash Memory. We believe that beyond that memory address holds the application specific code and data that is protected from read or write access.

The reason for this is that the Pebble Watch uses the STM32F205RE Cortex M3 CPU[1]. This comes with a Memory Protection Unit (MPU) that allows the Watch to manage memory access and writes. According to the data sheet for the CPU the "memory area is organized into up to 8 protected areas that can in turn be divided up into 8 subareas. The protection area sizes are between 32 bytes and the whole 4 gigabytes of addressable memory." [4]. This MPU is interfaced into the real-time operating system of the Pebble Watch, allowing it to override the protections for the legitimate interface into the persistent storage.

The memory dump consisted of a number of memory segments, including a segment that with firmware of the Pebble smartwatch that was uploaded to it by the Pebble Android app or through a connected development computer. The segment appears to be offset from the beginning of the flash memory (Figure 2), indicating that the first segment is likely a custom bootloader that is meant to load the firmware and then handle control over to the Pebble OS. One potential avenue to look into if we had more experience with disassembling code would be to

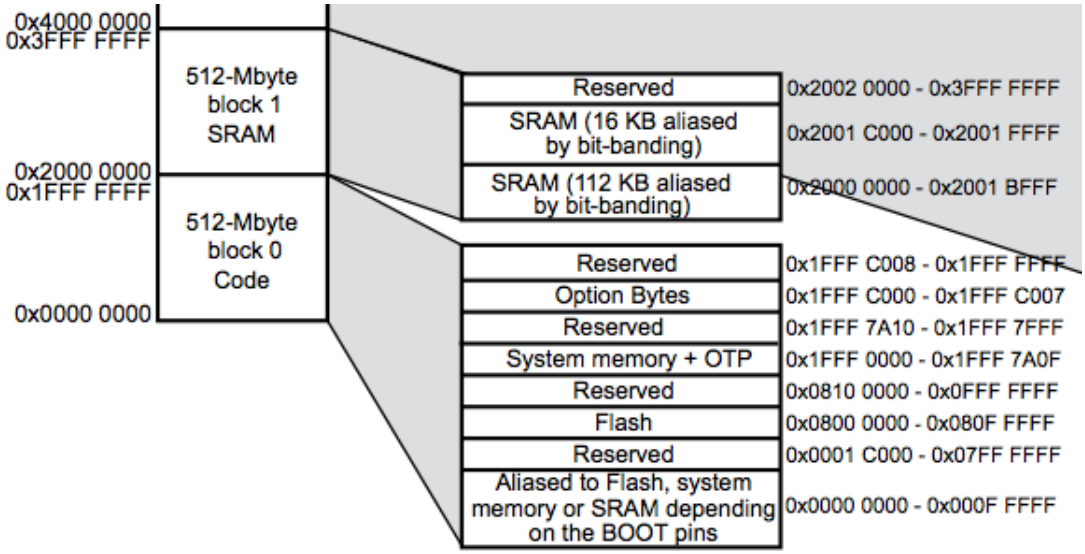

Figure 2: A memory map for the Pebble Watch. The Pebble has a STM32F205RE Cortex M3 CPU microprocessor. There are 1024KB of flash memory, which exists from 0x08000000 to 0x080FFFFF.[4]

attempt to write data to critical parts of the flash memory to change the behavior of the Pebble. Unfortunately, without a clear understanding of how the firmware is structured, attempts to overwrite parts of memory could completely brick the Pebble, making it completely unusable for any further security analysis.

One idea that we had initially considered when starting this security analysis was to could create a malicious program that could propogate itself to other Pebble devices given an insecure Bluetooth stack, and the ability to make arbitrary system calls to the firmware. Unfortunately, with the results of the Bluetooth analysis and the difficulty in reversing the firmware, we were unable to further explore these avenues of attack.

5 Pebble Operating System

Pebble's firmware opeerating system, Pebble OS, is based on FreeRTOS, a real time operating system. While Pebble OS is closed source, FreeRTOS is not. We

looked at FreeRTOS as a possible attack vector. Unfortunately, we were unable to find any existing attacks on or exploits for FreeRTOS. Through our reserach, we discovered a FreeRTOS is under GPL with an optional exception which allows closed source commercial products to used FreeRTOS. The website claims that FreeRTOS receives over 100,000 downloads a year and that a number of commercial products use FreeRTOS.[3]

Of interesting note is that some other smart watches also used FreeRTOS. Among these are inPulse Watch and MetaWatch. We concluded that FreeRTOS likely provides reasonable security, since it has been used in a number of commercial products. However, since these products are closed source, there are likely a number of security vulnerabilities that are currently undiscovered, either in the propreitary code, or in the FreeRTOS basis. Due to the difficulty of finding these, we decided to devote our time researching other attack vectors.

6 Phone Applications

The Pebble Watch is designed communicate with a phone over Bluetooth, and has two types for phone applications that can be connected to it. The first is a separately created phone application that uses the Pebble Framework and interfaces with the main Pebble app to send data. The second option is a javascript application that is bundled with the Watchapp. This javascript is run inside the official Pebble phone application with a limited interface for messaging the Pebble Watchapps as well as a persistant storage on the phone.

Both of these options offer Internet access via the phone and for Pebble Watchapp

communication. In particular, the separate Phone App option lets the writer of the app subscribe to any Watchapps UUID without authentication. This means that a phone app could passively eavesdrop on the watch to phone messages of the messaging interface provided by Pebble. Additionally, this means that any phone app using the Pebble interface can send messages to any Watchapp it knows the UUID for without the Watchapp being able to authenticate this data. This is somewhat worrying, as Watchapps could use the messages they receive to execute code or reply with sensative data.

The javascript application interface also provides a way to get an "account token" that uniquely identifies a user across phone and Pebble devices. This is based on the user's Pebble account and Watchapp's UUID. The token is created using a static 'salt' and the MD5 hashing function. However, phone applications cannot manually construct the "account token" for other apps without spoofing their UUID, as the user identifing information is not provided to the javascript application.

7 Future Work

In addition to the attack vectors we explored, there are several more that we believe could be analyzed in the future. These include:

- Internal hardware. As mentioned above, we chose not to open the Pebble to analyze the hardware, as this would certainly break the device; however, we conceived several useful points to analyze. For instance, by analyzing the flash controller unit, we could check the protection level of the flash memory.

This protection is linked to 8 bits written in the flash, and it determines what sort of protection the flash memory has. Such information would help craft future attacks on the device.

- Overriding the Memory Processor Unit (MPU). The MPU allows the watch to manage memory access and writes. By the unit, it could be possible to gain arbitrary read and write access on the Pebble.

- Firmware analysis. By analyzing and attaining a better understanding of the firmware of the device, we could attempt to overwrite the flash memory of the device to change the behavior of the Pebble device. Additionally, knowledge of the firmware would generally be useful for analyzing the software components of the Pebble.

8 Conclusion

While we were unable to find any major exploitable issues with the Pebble smartwatch, we were able to analyze a number of attack vectors on the watch and identify some potential avenues that could use further research and analysis. The primary takeaway is that users of the Pebble Watch should exercise caution when installing applications they have not written themselves to their watch.

References

[1] Wikimedia Foundation. Pebble (watch), May 2014. URL `https://en.wikipedia.org/wiki/Pebble_(watch)`.

[2] Ulrike Meyer Johannes Barnickel, Jian Wang. Implementing an attack on bluetooth 2.1+ secure simple pairing in passkey entry mode, 2012. URL `https://itsec.rwth-aachen.de/publications/draft.pdf`.

[3] Real Time Engineers Ltd. Freertos. URL `http://www.freertos.org/RTOS-contact-and-support.html`.

[4] STMicroelectronics. Stm32f205xx —— stm32f207xx, 2013. URL http://www.st.com/st-web-ui/static/active/en/resource/technical/document/datasheet/CD00237391.pdf.