Security Testing
Google Glass- Getting Started

David Brown
Senior Security Engineer
dbrown securityinnovation com

ABSTRACT

Wearable computers have been pursued for decades as an inevitable and transformative technology, one with the plausible capability of changing our relationships with each other and the world around us. The "Explorer Program" was introduced in early 2013 and was unceremoniously cancelled in January 2015 after finding trouble gaining widespread adoption or social acceptance [1]. Due to various testing constraints described in more detail below, along with the high retail price of the device when it was available for purchase, little security analysis of the device's attack surface, architecture, or custom software was made available to the general public. In the hope of making this type of analysis easier for future wearable embedded devices, this paper details many of the steps and processes needed to configure Google Glass for security testing.

1 INTRODUCTION

With the first generation of wearable computers now entering mainstream availability, questions have arisen as to whether the technology is ready and/or capable of serving as a replacement for mobile phones. While limitations of current mobile phones (usability constraints, unreliable networks, and low battery lives) all call for a fix, the promises of wearable computing are grander in nature, representing ideas that even a decade ago would have been considered science fiction. Among other things, the widespread adoption of wearable devices could enable seamless and persistent connections to the Internet, allowing for true augmented reality, the sharing of experiences in real time, and more exotic non-incremental improvements in existing technology. Imagine the ability to experience an event collectively (e.g. a wedding recorded from the perspective of every participant) or the creation of a "*life log*" containing a complete history of an individual's experiences.

With the potential to introduce many new social and technological concepts, wearable computing also brings with it the potential for altering aspects of basic social concepts including individual privacy, computer security, criminal liability, and many other aspects of our digital lives. Although privacy researchers and science fiction authors have been investigating these implications for a long time [2] [3] [4], little formal security analysis has been published regarding the first generation devices such as Google Glass. For the most part, it remains to be seen whether these initial devices introduce a new attack surface with unique technological concerns or whether they are simply an iteration of current mobile phone technology.

The Android platform has a relatively large number of tools built by developers and security researchers that can be leveraged when testing mobile devices and applications. Google Glass, although based on the Android kernel, is different enough in design and implementation that a number of relatively unique testing constraints are introduced. In particular, because Glass makes use of a nontraditional UI, many of the tools typically used to test or analyze Android devices or applications are not usable. While investigating various aspects of Google Glass from a security perspective, testing constraints and a lack of guidance necessitated the collection of various techniques, scripts, and tools into a single document which you will find below.

Challenges testing glass:

- Because of the lack of a traditional screen, many applications designed for testing are unusable. This includes tools to proxy device traffic, modify and add certificates to the device's trust store, or manipulate the device's file system.
- The device firmware includes few common command line tools necessary for testing, requiring them to be cross compiled for the device.
- The lack of an official emulator necessitates testing on actual Glass hardware which was prohibitively expensive when available to the public.
- Glass makes heavy use of multiple wireless network stacks associated with both physical and virtual network devices. (e.g. WIFI, Bluetooth, VPN's, loopback, etc)
- The transition from Eclipse (w/ Android SDK) to Android Studio for Glass App development rendered many of the existing Glass tutorials, forum threads, and help articles on the Internet outdated.

Assumptions:

- The Tester has a Google Glass headset and is willing to root the device, voiding any remaining warranties.
- The Glass device and the computer used to facilitate testing of Glass are members of the same wireless network allowing traffic between peers.
- The Glass device is configured with appropriate firmware. XE22, based on Android API Platform 19 was used in the work described in this document.
- The Tester is familiar with the Linux command line and is comfortable compiling command line tools.
- The computer used for testing the device is a modern Linux distribution with commonly available command line tools.

2 TEST ACCOUNTS

2.1 Google Accounts

While Glass retains most of its functionality without needing to be paired to a mobile device, the primary benefits from wearing Glass come when pairing the device with an Android or iOS device. This integration is necessary in order for Glass to make use of contacts present on the paired device, maintain a persistent Internet connection, display notifications from the device in the Glass HUD, and enable specific features associated with the MyGlass application (e.g. locating the Glass device or screen-casting the Glass HUD). If you have already performed any testing or development with Glass, you will have noticed that the MyGlass app requires a relatively large number of permissions.

MyGlass 3.5 Permissions (from AndroidManifest.xml)

```
<uses-feature android:name="android.hardware.telephony"
<uses-feature android:name="android.hardware.location"
<uses-feature android:name="android.hardware.location.gps"
<uses-feature android:name="android.hardware.location.network"
<uses-feature android:name="android.hardware.wifi"
android:name="android.permission.ACCESS_WIFI_STATE"/>
android:name="android.permission.ACCESS_NETWORK_STATE"/>
android:name="android.permission.CHANGE_WIFI_STATE"/>
android:name="android.permission.BLUETOOTH"/>
android:name="android.permission.BLUETOOTH_ADMIN"/>
android:name="android.permission.GET_ACCOUNTS"/>
android:name="android.permission.USE_CREDENTIALS"/>
android:name="android.permission.MANAGE_ACCOUNTS"/>
android:name="android.permission.INTERNET"/>
android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
android:name="android.permission.RECEIVE_SMS"/>
android:name="android.permission.RECEIVE_MMS"/>
android:name="android.permission.SEND_SMS"/>
android:name="android.permission.READ_SMS"/>
android:name="android.permission.READ_PHONE_STATE"/>
android:name="android.permission.WRITE_SMS"/>
android:name="android.permission.ACCESS_FINE_LOCATION"/>
android:name="android.permission.ACCESS_COARSE_LOCATION"/>
android:name="android.permission.READ_CONTACTS"/>
android:name="android.permission.WAKE_LOCK"/>
android:name="android.permission.DISABLE_KEYGUARD"/>
android:name="android.permission.BROADCAST_STICKY"/>
android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
android:name="com.google.glass.companion.GLASS_INFO"
android:name="com.google.glass.companion.GLASS_INFO"/>
```

This level of integration with a user's mobile device (and user data) results in Glass being restricted to a single user device. There is no way to "switch users" in Glass nor can it be associated with more than one identity without wiping the device and starting from scratch. Because a goal during security testing is to observe the behavior of the device under various scenarios, it is desirable to create a new Google identity that will be used only during testing. This will aid in isolating test data within a defined silo and help prevent inadvertent leaking of personal or professional data (e.g. automatic posting of the tester's workstation screen to a public Google+ account).

If there is a desire to test Glass interactions taking place directly between Google identities, it will be necessary to set up additional Google identities in order to provide contacts to the primary Google testing identity. This may be especially true if you intend to test during hours when your peers may be asleep.

2.2 WIFI Access Point

Although instructions to do so are not contained in this paper, it may be worthwhile to consider setting up a dedicated wireless access point to use during testing [5]. In addition to the network traffic collection methods described below, having a dedicated access point allows for router level interception of Glass transactions and provides further segregation between potentially sensitive personal/professional data, environments, and test data.

3 ANDROID DEBUG BRIDGE (ADB)

Android Debug Bridge (ADB) is the primary mechanism used to communicate with Android devices during testing [6]. Since Glass is based on the Android platform, we can leverage many of the tools provided by ADB [7]. Before we will be able to capture network traffic, record device logs, investigate the file system, or root the device, we will need to install and configure ADB [8]. During testing, we will use Android Studio, a development environment that provides a number of useful tools referenced throughout this document.

Glass, like other Android devices, also uses *logcat* as a central device wide logger. The behavior of *logcat* can be configured on the device to helpfully increase the verbosity of the logs, set up a rotation schedule [9] (useful when the logs began to grow in size), and the generated logs can be consumed by a number of other Android development tools. During testing logging output may be correlated with heap dumps and method *trace logs* providing a valuable source of information into the behavior of Glass or an application deployed to the device.

3.1 Installing & Configuring ADB for Glass

1. Ensure that the Java JDK 7 or greater is installed: http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html
2. Install Android Studio: https://developer.android.com/sdk/installing/studio.html
3. Add the installed Android Studio '*platform-tools*' subdirectory to the PATH environment variable
4. Create a Google account and pair it with the Glass headset which will be used during testing
5. On Glass, enable ADB by navigating to *Settings -> Device Info -> Turn on Debug*

Test the ADB -> Glass connection

1. List all of the devices ADB is able to detect. If everything is working correctly, the serial number for the device will be shown.

```
$ adb devices
List of devices attached
015DA2021200701B     device
```

2. If no devices are listed, ensure that the device is connected via USB
3. If a device is listed, but only as a series of question marks followed by the phrase "*no permissions*", ADB will require elevated privileges. This can be resolved by starting the ADB daemon as an administrator.

```
$ adb devices
List of devices attached
???????????   no permissions
$ adb kill-server
$ sudo adb start-server
$ adb devices
List of devices attached
015DA2021200701B     device
```

4. If additional troubleshooting is necessary, see the following page: https://stackoverflow.com/questions/5510284/adb-devices-command-not-working

Configure ADB to use WIFI

This is needed only if you want to disconnect USB during testing [10].

1. Ensure that both the device and computer are connected to the same WIFI network
2. Connect the device via USB as described in the steps above
3. Obtain the IP address of the Glass device. Note the IP Address of the *wlan0* device
```
$ adb shell netcfg
```
4. Configure ADB to use TCP/IP instead of USB
```
$ adb tcpip 5555
```
5. Connect ADB to the Glass device over WIFI using the address obtained in *Step 3* above
```
$ adb connect <WLAN0_IP_ADDRESS>
```
6. Disconnect the Glass USB cable

3.2 Capturing Device Logs using ADB and *logcat*

The simplest way to observe the logging output while logged into a device is to use the built in *logcat* utility [11]. Although the instructions below provide simple access to the raw *logcat* output, log data can be filtered and displayed in a variety of ways [12].

1. Log into a remote shell on the device
```
$ adb shell
$ logcat
```
2. Alternatively, logging data can be easily redirected to the testing environment for collection
```
$ adb logcat
```

4 ELEVATED PRIVILEGES

Although *ADB* can be used to connect to non-rooted devices, most of the test configuration below depends on the Glass device being rooted. By rooting the device with *ADB* and *fastboot*, we gain the ability to inspect and modify the device's configuration, file system, and network connections. The ability to root an Android device is dependent on the device manufacturer and, helpfully, Google continues to provide rooted boot images for most versions of the Glass firmware.

4.1 Rooting Glass

Note: Rooting a Glass device voids all device warranties provided by Google.

Obtain a rooted boot image

1. On the device, navigate to *Settings -> Device info*
2. Take note of the Firmware Version displayed

3. Download the *boot.img* corresponding to the firmware version observed in the "*Device info*" pane: https://developers.google.com/glass/tools-downloads/system

Root the Device

1. Reboot the device into the locked bootloader using the following command: [13]

```
$ adb reboot bootloader
```

2. Unlock the current bootloader (*NOTE: This will wipe the device*).

```
$ fastboot oem unlock
```

3. Flash the obtained *boot.img* file to the Glass device

```
$ fastboot flash boot <PATH_TO_boot.img>
```

4. Reboot the device normally

```
$ fastboot reboot
```

5. Once the device has completed rebooting, confirm that root access is now possible

```
$ adb root
$ adb shell
```

4.2 Un-Rooting Glass

In general, Android relies on the operating system for many security controls. This includes preventing an attacker from leveraging an application layer vulnerability to gain access to system or user data. Enabling root access on any Android device increases the attack surface of the device by providing malicious applications or users elevated privileges. Because there is no application for Glass similar to Android's *SuperSU,* restricting root access to applications unless specifically granted by a user, it is recommended that once testing is complete, Glass be restored to factory conditions [14].

NOTE: It is unlikely that this will restore the warranty.

Obtain an appropriate Factory System Image

1. Download the desired *Signed Glass Image* archive: https://developers.google.com/glass/tools-downloads/system
2. Expand the downloaded archive and locate the following three files:

```
-rw-r--r--  1 tester tester   5607424 Jan 1 2009 boot.img
-rw-r--r--  1 tester tester   6340608 Jan 1 2009 recovery.img
-rw-r--r--  1 tester tester 667674008 Jan 1 2009 system.img
```

Un-Root the Device [13]

1. Using *ADB*, reboot the Device into the bootloader

```
$ adb reboot bootloader
```

2. Flash the factory images obtained from Step 1 above

```
$ fastboot flash boot <PATH_TO_boot.img>
$ fastboot flash system <PATH_TO_system.img>
$ fastboot flash recovery <PATH_TO_recovery.img>
```

3. Erase the Device Cache

```
$ fastboot erase cache
```

4. Erase the User Data

```
$ fastboot erase userdata
```

5. Re-lock the OEM

```
$ fastboot oem lock
```

6. Reboot the device

```
$ fastboot reboot
```

5 THE GLASS FILE SYSTEM

Although some components of the Glass environment are unique to the platform, many of the file system locations, permissions, and security controls are inherited directly from Android. As with Android, ADB provides a simple mechanism for transferring files (or entire directories) between a Glass device and test environment. Many of the more Glass specific instructions below rely on this transfer mechanism to retrieve, manipulate, and deploy configuration files or Glass applications.

Glass applications are packaged as Android APKs which allows the use of standard APK testing tools during security testing. Security testing of a Glass application will rely on being able to manipulate an application's structure, permissions, resources, and signature. Instructions on reversing Glass packages downloaded via ADB are provided below. In addition, a list of file and directory locations is included which represent potentially interesting areas. These locations either contain a local device's configuration, sensitive assets, or are used during the execution of sensitive operations.

5.1 File Transfers using ADB

1. Copying Files from Device to Host
```
$ adb pull -p <REMOTE_PATH> <LOCAL_DIRECTORY>
```
2. Copying Files from Host to Device
```
$ adb push -p <LOCAL_PATH> <REMOTE_DIRECTORY>
```

5.2 Decompiling Glass Packages

Obtain the necessary tools

1. apktool: https://bitbucket.org/iBotPeaches/apktool/downloads
2. dex2jar: https://code.google.com/p/dex2jar/downloads/list
3. jd-gui: http://jd.benow.ca/#jd-gui-download

Obtain a sample application

1. Obtain a root shell on the device
```
$ adb root
```
2. Download the package from the Device
```
$ adb pull /system/app/<APK_NAME>.apk
```
3. Unpack the APK
```
$ java -jar apktool_2.0.0rc2.jar d <APK_NAME>.apk
```
4. Decompile the APK's Java classes
```
$ dex2jar <APK_NAME>.apk
```
5. View the APK Manifest File
```
$ cat <APK_NAME>/AndroidManifest.xml
```
6. View the decompiled Java code
```
$ jd-gui <APK_NAME>_dex2jar.jar
```

5.3 Potential Areas of Interest

1. User Data and Authentication Tokens
```
/data/system/users/0/accounts.db
```
2. Enabled Package Components
```
/data/system/users/0/package-restrictions.xml
```
3. List of Installed Package Names and Locations
```
/data/system/packages.list
```
4. Device Shared Storage *(/sdcard/ is emulated*)
```
/sdcard/
```

5. Application Data

```
/data/data/<package_name>
```

6. System Process PID's

```
/data/system/registered_services
```

7. Google Sync Authorizations

```
/data/system/sync/accounts.xml
```

8. System Logs

```
/data/media/0/logs
```

9. Bluetooth Configuration

```
/data/misc/bluedroid/bt config.conf
/data/misc/bluedroid/btaddr
```

10. WIFI Configuration

```
/data/misc/wifi/wpa_supplicant.conf
/data/misc/wifi/softap.conf
/data/misc/wifi/ipconfig.txt
```

11. Default Glass Packages (*Note: this is a subset of the default packages*)

```
package:com.google.glass.voice
package:com.google.glass.logging
package:com.google.glass.gesture
package:com.google.glass.companion
package:com.google.glass.maps
package:com.google.glass.browser
package:com.google.glass.lockrecovery
package:com.google.glass.deviceadmin
package:com.google.glass.home
package:com.google.glass.setup
package:com.google.glass.camera
package:com.google.glass.settings
package:com.google.glass.phone
package:com.google.glass.boutique
package:com.google.glass.bluetooth
package:com.google.glass.frameworkstubs
package:com.google.glass.search
package:com.google.glass.sync
package:com.google.glass.update
package:com.android.inputdevices
package:com.android.providers.userdictionary
package:com.android.sharedstoragebackup
package:com.android.vpndialogs
package:com.android.certinstaller
package:com.android.providers.contacts
package:com.android.keychain
package:com.android.location.fused
package:com.android.keyguard
package:com.android.shell
package:com.android.providers.downloads
package:com.google.android.gsf.login
package:com.google.android.glass.internal
package:com.google.android.gms
package:com.google.android.backuptransport
package:com.google.android.tts
package:com.google.android.gsf
package:com.google.android.glass.systemui
package:board_id.com.ti
```

6 NETWORK TRAFFIC

In order to proxy, capture, and analyze network traffic to and from Glass persistently a few tools are required. As is the case in other areas, many of the steps will be similar to Android (including the use of the Android NDK). Glass imposes a number of testing restrictions that result in additional work being necessary in order to monitor network traffic persistently.

There are a variety of ways to capture Glass traffic, but this paper will provide instructions on using *tcpdump* and *netcat*. This testing configuration will mirror all traffic passing through the Glass device's WIFI adapter to a remote Wireshark instance where it can be inspected, analyzed, or passed to any other desired tool.

Because Factory Glass images do not include many common command line utilities present in a typical Android or Linux environment, it is necessary to cross compile some tools in order to capture device wide traffic. The instructions below explain how to use the Android NDK to build and deploy *libpcap*, *tcpdump*, and *netcat* for the ARM environment currently used by Glass [15]. Although the Android NDK is a relatively large package, in this section we will primarily be interested in the ability to generate a compiler toolchain specific to the Android platform used by the Glass firmware. This toolchain will subsequently be used to compile the ARM binaries that will be deployed to the device. Once the ARM binaries have been compiled and deployed, a set of traffic forwarders and listeners will be configured on the device, allowing for the remote capture of the device's network traffic.

Before proceeding to the next step, the following is assumed:

1. Java, GCC, and ADB are installed on the computer facilitating the test
2. The Glass device has been rooted
3. A root shell on the device has been obtained

6.1 Configure Glass and Deploy Necessary Tools

Install Wireshark

Note: This will be used to observe captured traffic to and from the device

1. Download and Install Wireshark according to the instructions here: https://www.wireshark.org/download.html
2. Ensure that Wireshark has access to the available network interfaces: (*Wireshark should never be run with elevated privileges* [16])
    - http://wiki.wireshark.org/CaptureSetup/CapturePrivileges

Install the Android Native Development Kit (NDK)

1. Download the Android NDK package (*Note: r10c was used during testing*): https://developer.android.com/tools/sdk/ndk/index.html#Downloads
2. Install according to the instructions here: https://developer.android.com/tools/sdk/ndk/index.html#Installing
3. Make a note of the location of the *make-standalone-toolchain.sh* script

```
/android-ndk-r10c/build/tools/make-standalone-toolchain.sh
```

Build the Compiler Toolchain

1. Using the Android NDK, generate the toolchain [17]

```
sh <Path_to_NDK>/build/tools/make-standalone-toolchain.sh --
platform=<Android_Platform_Version> --install-
dir=<Path_to_Toolchain>
```

2. Set the necessary Environment Variables

```
export CC=arm-linux-androideabi-gcc
export RANLIB=arm-linux-androideabi-ranlib
export AR=arm-linux-androideabi-ar
export LD=arm-linux-androideabi-ld
export PATH=<Path_to_Toolchain>/bin:$PATH
```

Build libpcap

Note: Although the libpcap library is required in order to build tcpdump in the following step, it will not be deployed.

1. Obtain the *libpcap* source: http://www.tcpdump.org/release/libpcap-1.4.0.tar.gz
2. Unpack the source

```
tar -xvzf libpcap-1.4.0.tar.gz
```

3. Configure ARM compilation [18]

```
./configure --host=arm-linux --with-pcap=linux
ac_cv_linux_vers=2
Chmod +x runlex.sh
```

4. Build *libpcap*

```
make
```

Build tcpdump

1. Obtain the *tcpdump* source: http://www.tcpdump.org/release/tcpdump-4.4.0.tar.gz
2. Unpack the source

```
tar -xvzf tcpdump-4.4.0.tar.gz
```

3. Configure ARM compilation [19]

```
$ ./configure --host=arm-linux --with-pcap=linux
ac_cv_linux_vers=2 --with-crypto=no
$ sed -i".bak" "s/setprotoent/\/\/setprotoent/g" print-
isakmp.c
$ sed -i".bak" "s/endprotoent/\/\/endprotoent/g" print-
isakmp.c
```

4. Build *tcpdump*

```
$ make CFLAGS=-DNBBY=8
```

Deploy tcpdump

1. Obtain a root shell on the device
```
$ adb root
```
2. Push the binary to the device
```
$ adb push tcpdump /sdcard/
```
3. Mount the */system* partition with write privileges
```
$ adb shell "mount -o remount,rw /system"
```
4. Copy the binary to the */system/bin* directory
```
$ adb shell "cp /sdcard/tcpdump /system/bin/"
```
5. Ensure that the binary is executable
```
$ adb shell "chmod 744 /system/bin/tcpdump"
```
6. Remount the */system* partition as readonly
```
$ adb shell "mount -o remount,ro /system"
```
7. Delete the temporary *tcpdump* binary
```
$ adb shell "rm /sdcard/tcpdump"
```
8. On the Glass device, confirm that *tcpdump* is successfully deployed
```
$ adb shell tcpdump --version
tcpdump version 4.4.0
libpcap version 1.4.0
```

Build netcat

1. Obtain the *netcat* source:
http://sourceforge.net/projects/netcat/files/netcat/0.7.1/netcat-0.7.1.tar.gz
2. Unpack the source
```
tar -xvzf netcat-0.7.1.tar.gz
```
3. Configure ARM compilation
```
$ ./configure --host=arm-linux
```
4. Build *netcat*
```
$ make
```

Deploy netcat

1. Obtain a root shell on the device
```
$ adb root
```
2. Push the binary to the device
```
$ adb push netcat /sdcard/
```
3. Mount the */system* partition with write privileges
```
$ adb shell "mount -o remount,rw /system"
```
4. Copy the binary to the */system/bin* directory
```
$ adb shell "cp /sdcard/netcat /system/bin/"
```
5. Ensure that the binary is executable
```
$ adb shell "chmod 744 /system/bin/netcat"
```
6. Remount the */system* partition as readonly
```
$ adb shell "mount -o remount,ro /system"
```
7. Delete the temporary *netcat* binary
```
$ adb shell "rm /sdcard/netcat"
```
8. On the Glass device, confirm that *netcat* is successfully deployed
```
$ adb shell netcat --version
netcat (The GNU Netcat) 0.7.1
Copyright (C) 2002 - 2003  Giovanni Giacobbi
```

6.2 Capture Glass WIFI Traffic

Once the steps above are completed and *tcpdump* and *netcat* have been deployed to the device, capturing device wide traffic is possible by using *tcpdump* to redirect all WIFI traffic to *stdout*, which is then piped directly to *netcat*. *Netcat* is in turn configured to route all received packets to a remote listener. Note that while the technique below uses *ADB* to facilitate a port forward there are also a number of alternatives for redirecting traffic from the device.

1. Remotely run *tcpdump* on redirect the Glass WIFI adapter to a local port using *netcat*

```
adb shell "tcpdump -i 2 -n -s 0 -w - | netcat -l -p 11233"
```

2. Configure *ADB* to forward the traffic to *netcat* and route to Wireshark [20]

```
adb forward tcp:11233 tcp:11233 && nc 127.0.0.1 11233 |
wireshark -k -S -i -
```

6.3 Configuring a Browser Proxy

Glass has a number of global OS settings, one of which is *http_proxy*. Any proxy aware Glass application on the device will attempt to reference this value when building network connections and route traffic accordingly. The only application observed using this setting during testing is the native Glass Web Browser which can be used to navigate web pages, albeit in a limited way. Security testing of the browser can be achieved by setting *http*_proxy to point towards a web proxy of choice. Note that this setting is persistent across reboots and, because it relies on manipulating a local *sqlite* database, requires *sqlite3* (as shown below) or an alternative such as *sqlitebrowser* [21].

1. Obtain a root shell on the device

```
$ adb root
```

2. Download the *settings.db* file from the device

```
$ adb pull
/data/data/com.android.providers.settings/databases/
```

3. Determine the IP address and Port of the desired web proxy
4. Using *sqlite3*, add a row to the global table with the following values [22]

```
$ sqlite3 <path_to_database_file>/settings.db "INSERT INTO
global VALUES (99, 'http_proxy', '<PROXY_IP>:<PROXY_PORT>');"
```

5. Using *ADB*, push the modified *settings.db* file to the device and reset the device

```
$ adb push settings.db
/data/data/com.android.providers.settings/databases/
$ adb reboot
```

7 CAPTURING THE GLASS HUD

When testing, it is useful to be able to document the state of the device under a specific test case or scenario. Taking screen shots of the Glass "heads up display" (HUD) is an important part of this.

7.1 Taking a Screenshot

Often it is only necessary to take a quick screenshot of the Glass display. This can be done easily with the *screencap* utility and the ADB *pull* command as shown below. This method is also easily scriptable as shown below.

```
$ adb shell screencap -p /sdcard/<file_name>.png
$ adb pull /sdcard/<file_name>.png <screenshots_directory>
$ adb shell rm /sdcard/<file_name>.png
```

7.2 Screen-casting

There are two primary options for mirroring the Glass display in real time to a computer, both of which rely on *ADB* and the Android SDK. Additionally, the *MyGlass* mobile app allows a paired Glass device's HUD to be mirrored real-time onto the screen of the Android or iOS device.

Casting to a Computer

- https://code.google.com/p/android-screen-monitor/
- http://droid-at-screen.ribomation.com/installation/
- https://support.google.com/glass/answer/3068035?hl=en (uses the MyGlass Android App)

8 CONCLUSION

8.1 Bringing Everything Together

Google Glass is a first attempt at combining a set of advanced sensors, wireless protocols, and a new UI paradigm to existing Internet and mobile network infrastructure. As a result of Glass' relative complexity (forked kernel, stand-alone app-store, multiple permission models, etc) the overall attack surface for the device is larger than most single applications or mobile devices. At a high level, Glass is intended to function as a seamless front-end for a much larger collection of services, drawing together Internet endpoints, contextual personal data, and the local environment which surrounds a wearer. Determining whether this goal has been accomplished is both interesting and exciting, a sign of things to come, but it is equally important to determine whether the goal is being accomplished in a way that protects both current Glass wearers and the people around them.

8.2 Additional Testing Opportunities

A number of areas which are likely interesting from a security perspective were not included in this paper. Testing of these areas will likely include some Glass specific configuration and include:

- Bluetooth & WIFI Pairing
- Interactions between Glass devices
- Permission Models (Glass OS, Glass Apps, and MyGlass)
- Certificate Trust Store and Keychain Access

8.3 Additional Resources

Application Analysis

- http://blog.apkudo.com/tag/baksmali/
- https://code.google.com/p/android-apktool/wiki/FAQ
- https://code.google.com/p/dex2jar/wiki/Faq

Capturing Network Traffic

- https://github.com/chatch/tcpdump-android
- https://gadgetcat.wordpress.com/2011/09/11/tcpdump-on-android/
- http://codeseekah.com/2012/08/07/port-forwarding-an-android-local-port/#more-1296
- http://mitmproxy.org/doc/transparent/linux.html
- http://www.cyberciti.biz/faq/linux-setup-default-gateway-with-route-command/
- www.kandroid.org/online-pdk/guide/tcpdump.html

Installing Custom Certificates

- http://wiki.cacert.org/FAQ/ImportRootCert
- http://forum.cyanogenmod.org/topic/82875-installing-cacert-certificates-on-android-as-system-credentials-without-lockscreen/
- http://forum.cyanogenmod.org/topic/82875-installing-cacert-certificates-on-android-as-system-credentials-without-lockscreen/
- http://mitmproxy.org/doc/ssl.html
- http://blog.philippheckel.com/2013/07/01/how-to-use-mitmproxy-to-read-and-modify-https-traffic-of-your-phone/#Attacking-HTTPS-connections

Browser Proxy Configuration

- http://muzso.hu/2013/04/07/changing-android-system-settings-eg.-airplane-mode-radios-via-commandline-on-rooted-phone
- https://www.debian-administration.org/article/595/Need_a_generic_iptables_tcp_proxy

9 REFERENCES

[1] Google, "We're graduating from Google[x] labs," 15 January 2015. [Online]. Available: https://plus.google.com/+GoogleGlass/posts/9uiwXY42tvc. [Accessed 10 March 2015].

[2] A. Allesandro, R. Gross and F. Stutzman, "Face Recognition Study - FAQ," CMU, 4 August 2011. [Online]. Available: http://www.heinz.cmu.edu/~acquisti/face-recognition-study-FAQ/. [Accessed 3 March 2015].

[3] D. Eggers, The Circle, New York City: Knopf Doubleday Publishing Group, 2013.

[4] A. Miller, "Personal Privacy in the Computer Age: The Challenge of a New Technology in an Information-Oriented Society," *Michigan Law Review,* vol. 67, no. 6, pp. 1089-1246, 1969.

[5] HakShop, "WiFi Pineapple Mark V Standard," [Online]. Available: http://hakshop.myshopify.com/products/wifi-pineapple. [Accessed 10 March 2015].

[6] Google, "Android Debug Bridge," [Online]. Available: https://developer.android.com/tools/help/adb.html. [Accessed 10 March 2015].

[7] Unknown, "Google Glass Developer Jump Start," 25 May 2013. [Online]. Available: http://glassdev.blogspot.com/. [Accessed 10 March 2015].

[8] Google, "Setting up the development environment," 9 January 2015. [Online]. Available: https://developers.google.com/glass/develop/gdk/quick-start#setting_up_the_development_environment. [Accessed 10 March 2015].

[9] H. Yang, ""adb logcat" Command Options and Log Buffers," 2012. [Online]. Available: http://www.herongyang.com/Android/Debug-adb-logcat-Command-Option-Log-Buffer.html. [Accessed 10 March 2015].

[10] F. Gallego, "Wireless ADB connection to Google Glass," coderwall, 21 August 2013. [Online]. Available: https://coderwall.com/p/vq-eaw/wireless-adb-connection-to-google-glass. [Accessed 10 March 2015].

[11] Google, "logcat," [Online]. Available: https://developer.android.com/tools/help/logcat.html. [Accessed 10 March 2015].

[12] Google, "Reading and Writing Logs," [Online]. Available: https://developer.android.com/tools/debugging/debugging-log.html. [Accessed 10 March 2015].

[13] Google, "System and Kernel Downloads," Google Developers, 27 February 2015. [Online]. Available: https://developers.google.com/glass/tools-downloads/system. [Accessed 10 March 2015].

[14] Chainfire, "SuperSU v1.00 has been released !," 28 January 2013. [Online]. Available: http://www.chainfire.eu/articles/129/SuperSU_v1_00_has_been_released_/. [Accessed 10 March 2015].

[15] A. Barr and D. Clark, "Google Glass Deal Thrusts Intel Deeper Into Wearable Tech," Wall Street Journal, 30 November 2014. [Online]. Available: http://www.wsj.com/news/article_email/google-glass-deal-thrusts-intel-deeper-into-wearable-devices-1417395598-lMyQjAxMTA0MjMwMDEzMDA4Wj. [Accessed 10 March 2015].

[16] WireShark, "Security," 6 May 2013. [Online]. Available: https://wiki.wireshark.org/Security. [Accessed 10 March 2015].

[17] V. Kornacki, "Monitoring Android Network Traffic Part I: Installing The Toolchain," Symantec Connect, 10 February 2014. [Online]. Available: http://www.symantec.com/connect/blogs/monitoring-android-network-traffic-part-i-installing-toolchain. [Accessed 10 March 2015].

[18] V. Kornacki, "Monitoring Android Network Traffic Part II: Cross Compiling TCPDUMP," Symantec Connect, 10 February 2014. [Online]. Available: http://www.symantec.com/connect/blogs/monitoring-android-network-traffic-part-ii-cross-compiling-tcpdump. [Accessed 10 2015 March].

[19] Whisper Bytes, "Cross-Compiling for Android - Installing tcpdump," WordPress, 28 June 2014. [Online]. Available: https://whisperbytes.wordpress.com/2014/06/28/cross-compile-for-android-installing-tcpdump/. [Accessed 10 March 2015].

[20] D. Miller, "Debug Network Activity in Android Using Wireshark," WordPress, 3 February 2011. [Online]. Available: https://dennismillerdev.wordpress.com/2011/02/03/debug-network-activity-in-android-using-wireshark/. [Accessed 10 March 2015].

[21] rp-, "DB Browser for SQLite 3.5.1," 8 February 2015. [Online]. Available: https://github.com/sqlitebrowser/sqlitebrowser/releases/tag/v3.5.1. [Accessed 10 March 2015].

[22] rtm, "Set proxy for android web browser," Let's Talk about Google Android, 17 January 2008. [Online]. Available: http://discuz-android.blogspot.com/2008/01/set-proxy-for-android-web-browser.html. [Accessed 10 March 2015].