Taming Mr Hayes: Mitigating Signaling Based Attacks on Smartphones

Collin Mulliner, Steffen Liebergeld, Matthias Lange, and Jean-Pierre Seifert
Technische Universität Berlin and Deutsche Telekom Laboratories
D-10587, Berlin, Germany
{collin,steffen,mlange,jpseifert}@sec.t-labs.tu-berlin.de

***Abstract*—Malicious injection of cellular signaling traffic from mobile phones is an emerging security issue. The respective attacks can be performed by hijacked smartphones and by malware resident on mobile phones. Until today there are no protection mechanisms in place to prevent signaling based attacks other than implementing expensive additions to the cellular core network. In this work we present a protection system that resides on the mobile phone. Our solution works by partitioning the phone software stack into the application operating system and the communication partition. The application system is a standard fully featured Android system. On the other side, communication to the cellular network is mediated by a flexible monitoring and enforcement system running on the communication partition. We implemented and evaluated our protection system on a real smartphone. Our evaluation shows that it can mitigate all currently known signaling based attacks and in addition can protect users from cellular Trojans.**

***Keywords*-Smartphones, Cellular Signaling, Attack Mitigation, Operating Systems, System Virtualization**

I. Introduction

In the past years a lot of effort has gone into securing smartphones. There are academic contributions [20], [8], [35] and work performed by smartphone operating system (OS) vendors such as Apple, Google, Symbian, RIM or Microsoft. However, the efforts concentrated on the OS, to protect users from attacks and to mitigate malware such as Trojans.

Despite recent attacks, which target the cellular core network, few methods of defense are known. These attacks are based on hijacked mobile phones (mobile botnets) that produce signaling traffic sent from mobile phones to the cellular network core conducting Denial-of-Service attacks. These attacks demonstrated that current security improvements seek to protect the actual device and not the environment in which they operate, namely, the cellular core network.

Related security and reliability problems are caused by rooted (fully user controlled) smartphones. The problem is that rooting disables protection mechanisms of the OS, allowing the user to install arbitrary applications to his device. Such applications might leverage extended access privileges and may use them for intentional malicious activity and accidental harmful operations.

In this paper we present our novel solution for protecting the cellular network infrastructure from malicious smartphones. Our protection system is called the *virtual modem*. It secures the baseband, the entity that communicates with the cellular network. To the best of our knowledge nobody has yet attempted this path for securing cellular communication.

In contrast to a network side solution our protection system is designed to run on the mobile phone. Changes to the cellular network equipment are very expensive, and time consuming which would result in a slow adoption of any newly proposed protection mechanism. On the other hand, smartphone development cycles are very short. New smartphones are brought to the market every 6 months. Thus, we believe a device-side protection system has a significantly higher chance to be adopted.

Instead of implementing our protection system directly on the cellular connectivity hardware, we achieve protection by controlling the communication channel between the OS and the baseband. The smartphone is partitioned and the OS is separated from the baseband. The separation is implemented through virtualization. The actual core of our protection system is comprised of an AT command filter. With the implementation of our protection system based on the Android [10] platform we show that our approach is feasible for real-world smartphones. Still we think its design is general enough to be used for other smartphone OSes as well.

The main contributions of this paper are:

- **Categorization of Signaling Issues:** We categorize different security and reliability issues that are caused by signaling traffic related to smartphone use and abuse. The issues can be separated into intentional actions (attacks), and side effects that can be abused for attacks.

- **Cellular Signaling Filter:** We introduce a novel mechanism to protect cellular network infrastructure against overloading from smartphones. This is achieved by filtering the signaling channel directly on the smartphone. This avoids expensive changes on the cellular core network. We further show that our novel security mechanism can be used to protect the user from Trojans that cause premium rate charges via SMS.

- **Safe-to-root virtualized Android:** We designed and built a safe-to-root virtualized Android. Our virtualized

Android can be rooted and modified as the owner of the device wishes. The device manufacturer together with the operator retain full control over the cellular network interface (the baseband) and thus can prevent the device from being abused for launching attacks.

The rest of this paper is organized as follows. We first provide some background information on smartphones and cellular networks in Section II. In Section III we give a detailed overview of the threats to both the network and the phone owner that are related to the baseband of a modern smartphone. The design of our protection system to mitigate these threats is described in Section IV. Implementation details of our prototype system are described in Section V. In Section VI, we discuss our actual mitigation technique in great detail. The evaluation of our protection system is presented in Section VII. In Section VIII, we discuss related work before we conclude and outline future work in Section IX.

II. Background

In this section we provide the reader with background information on cellular communication. First, we briefly introduce the cellular network components and cellular signaling. These are targeted by the attacks we describe later in Section III. Second, we provide some details on the common hardware design that is found in almost every modern smartphone today.

A. Cellular Network Architecture

The basic architecture of a cellular network is shown in Figure 1. The architecture leans towards a GSM network but is very similar to a 3G network at least for the scope of this paper. The network consists of the Base Stations (BS), the connected Base Station Controller (BSC) (not shown in the figure), the Mobile Switching Center (MSC), the packet-data infrastructure consisting of the Serving GPRS Support Node (SGSN), the Gateway GPRS Support Node (GGSN), and the central user database the Home Location Register (HLR). The HLR keeps track of all users and their accounting information.

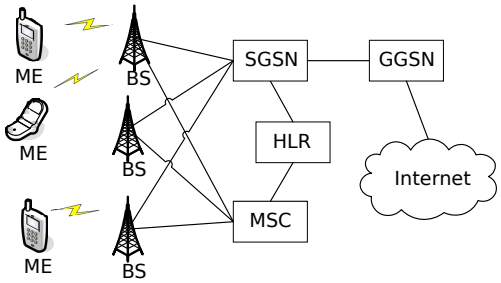

Figure 1. Basic setup of a cellular network.

B. Cellular Signaling

Signaling traffic generated by the Mobile Equipment (ME) is sent to the MSC and HLR in case of voice calls, SMS, and updating account settings (such as call-forwarding). Packet-data related signaling is mainly directed towards the SGSN, the GGSN, and of course the HLR.

Packet Data Protocol (PDP) setup in order to establish IP connectivity is a complex process. When a ME wishes to establish a PDP context it sends a GPRS-attach message to the SGSN. The SGSN authenticates the ME using the HLR. Next, the PDP context is established and stored at the SGSN and GGSN. This includes records and parameters for billing, quality of service information, and the IP address assigned to the specific PDP context. Maintenance and distribution of the PDP context information across the different network components is a costly process as it involves many components across the cellular network.

C. Smartphone Architecture

Modern smartphones consist of two individual subsystems, the application processor and the baseband processor. Together with the peripheral hardware such as the touch screen, audio input and output, and the GPS receiver these two systems form the actual smartphone. Figure 2 depicts the conceptual system design of a smartphone. This is how an iPhone, Android, and Windows Phone 7 device looks on the inside.

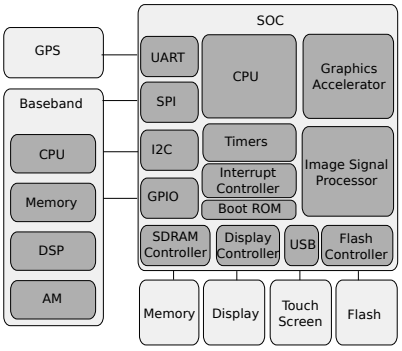

Figure 2. The basic design of a modern smartphone.

The application processor usually comes in form of a System on a Chip (SoC) design. The CPU and many of the controllers for connected peripherals shown in Figure 2 are included on the same chip. The application processor runs the smartphone OS such as Android or iOS and all the applications (e.g. email client, telephone).

The baseband processor is the communication interface to the cellular network. It consists of a general purpose CPU, a Digital Signal Processor (DSP), and the necessary radio components such as a signal amplifier. The baseband processor runs a specialized real-time operating system. Baseband chipsets are a highly specialized field since they have to be certified by multiple institutions before they are allowed to operate on public cellular networks. Because the process of development and certification is very costly, there are few baseband manufacturers [27].

The application processor and the baseband processor are connected at few points. This allows for better flexibility

for the various phone manufacturers. The connections are for digital audio input and output (voice calls) and for controlling the baseband's functionality. The control channel is conceptually a serial connection that can be implemented using buses such as SPI or USB. Over this serial connection, the application processor uses an extended version of the Hayes *Attention* (AT) command set[1] to interact with the baseband. In Section VI-D we will explain the communication between application processor and baseband in more detail.

III. Threats

In this section we introduce the different classes of threats that we address with the protection system presented in this work. There are three basic classes: threats that hijacked smartphones pose to the cellular core network, malware residing on the smartphone – with and without system privileges, and rooted devices.

A. Hijacked Phones and Mobile Botnets

The threats that hijacked phones and mobile botnets pose to the cellular network infrastructure and mobile customers is an emerging trend. A good example is the ikee.B [24] iPhone botnet. The bot infected about 22,000 devices and contained a HTTP-based Command and Control system.

Traynor et al. show in [30] that smartphone-based botnets can pose a serious threat to the cellular core network. They demonstrated that mobile botnets can overload backend systems such as the HLR and thus bring down the cellular network itself. Their attack is based on AT commands issued by zombie phones, which cause a high load on the HLR. Specifically, they issue the AT command to configure and enable call-forwarding settings. We discuss the actual details of the attack in Section VII where we evaluate our protection system against the various attacks.

The second issue with mobile botnets is their use of SMS messages for their Command and Control (C&C) communication, as demonstrated by Mulliner et al. in [19]. Their proof-of-concept bot uses SMS messages for delivering the C&C messages between the nodes of the botnet. A similar proof-of-concept SMS controlled botnet was created in [33].

The SMS messages must be blocked to prevent botnet communication and to ensure that the subscriber (owner) does not incur any additional charges related to the increased SMS traffic.

B. PDP Context Change

Fast PDP context activation and de-activation leads to high network load on the GGSN and SGSN infrastructure. This is performed by either malicious applications or badly configured mobile phones. This is possible because on smartphone platforms such as Android any application has access to the network configuration and thus is able to change the packet-data settings.

On Android it is possible to force an PDP context change every 2 seconds. This will result in roughly 43,200 PDP activations per day (24 hours). A rogue application can easily carry out a Denial-of-Service attack against an operator's packet-data infrastructure, if it is installed on enough devices.

The GSM Association (GSMA) points out a similar problem [11] through the use of pre-paid SIM cards. Travelers who do not want to pay high roaming costs often buy pre-paid SIM cards. A flood of PDP context activation attempts can occur under two conditions: First, the pre-paid SIM card does not match the configured packet-data settings (the one of the home operator), but the phone keeps trying to activate packet-data every few seconds. Second, the pre-paid account is below the number of credits that are required to establish a PDP context. In both cases the PDP context creation is rejected by the network, but for the phone it looks like a technical error, and thus it repeatedly attempts to reconnect to the network.

C. Premium Rate SMS Trojans

Fraud caused by SMS Trojans such as `FakePlayer-A` [9] is a long standing problem in the mobile phone world costing consumers a considerable amount of money every year [21]. This kind of fraud is possible since on modern smartphones any application has access to the cellular API and is therefore able to send SMS messages. The same problem applies to voice calls to premium numbers.

Smartphone platforms such as Android or Symbian implement mandatory access control to restrict arbitrary access to system resources such as location, Internet, or cellular access. These permissions are hard coded into the application. At installation time of an application the user is shown a list of required permissions. The user can accept these or cancel the installation process. It is not possible to selectively accept or deny access privileges. Thus, many users simply accept such permission requests without considering their implications.

For example, on Android the permission required to send SMS messages is called *android.permission.SEND_SMS*.

D. Rooted Phones

Rooted or jailbroken smartphones are a serious security risk. Once a device is rooted, many security features of the operating system, such as network and cellular access restrictions as well as data-caging, are gone. Thus, the entry barrier for malware such as Trojans or botnets is much lower on rooted phones.

Rooting can happen in two ways. First, voluntarily by the owner who wants to be able to install additional, potentially unauthorized, applications. This type of rooting is often done by simply installing a modified firmware on the device. Thus, no security flaws are actually exploited.

Second, by malware such as DroidDream [17] in order to gain maximum privileges on the infected system. This type of rooting is achieved by exploiting known security flaws in the respective smartphone OS.

IV. Design

Our aim is to mitigate Denial-of-Service attacks based on signaling traffic sent from mobile phones. As described

[1] http://en.wikipedia.org/wiki/Hayes_command_set

in Section II-C the baseband is a phone's gateway to the cellular network. Consequently, our protection system must have exclusive control over the baseband hardware. For clarification, we define the following criteria for our protection system.

Integrity Our protection infrastructure must withstand attacks from the smartphone OS. Even a rooted phone must not be able to directly tamper with the baseband. This can only be achieved, if our protection system is spatially *isolated* from the smartphone OS, e.g. it must not depend on its correct operation.

Completeness All cellular network access must be mediated and controlled by our trustworthy components.

Universality Our solution must be applicable to all cellular networks without requiring modifications to the operator's equipment.

Portability Our solution shall be usable on commercial off-the-shelf smartphones. It must not require additional hardware or hardware modifications. Our system has to support different baseband chips as well as popular smartphone CPUs. The solution must not depend on a certain smartphone OS. However, for practical reasons (open source, popularity) we chose the Android OS for this work.

Security Our protection system must not pose additional threats to the smartphone OS. This criterion is similar to the integrity criterion, but also includes availability and confidentiality of the whole system.

Upgrades and modifications to the cellular operator's equipment are very expensive and take a lot of time. In contrast, the smartphone market is advancing rapidly, with frequent releases of new smartphone generations. Each smartphone generation might bring new issues that would require new measures on the operator's side. We opted for a solution that addresses the signaling problem directly at its root, the smartphone itself (*Universality Criterion*).

A reasonable place for our protection system is the baseband as this is the smartphone's gateway to the cellular network (*Completeness Criterion*). The baseband processor has its own memory and is physically isolated from the application processor (*Integrity Criterion* and *Security Criterion*).

Baseband chipsets are under tight control by their manufacturers. Hardware details and the software stack are kept as trade secrets. Thus, no SDK or developer documentation is available. The *Portability Criterion* requires us to implement our protection system on commonly used basebands, which could turn out to be inherently difficult as the basebands might vary vastly. Also, a modified baseband would probably require re-certification, which due to time and cost constraints is infeasible.

Instead we chose to build our protection system on the application processor. The *Completeness Criterion* requires that the smartphone OS cannot directly access the baseband hardware. All cellular network access needs to be mediated by a custom proxy component. We call this component the *virtual modem*. The virtual modem runs as a separate task. We ensure spatial isolation between the smartphone OS, in our case Android, and the virtual modem by running Android in a virtual machine (VM). Figure 3 shows this setup. Direct hardware access of the Android VM to the baseband is denied. Instead we present the Android VM with an interface to the virtual modem. This ensures that even in the event of a rooted Android, the network operator cannot be adversely affected. For all other hardware, e.g. wireless LAN and graphics, we allow the VM exclusive access to the underlying hardware interfaces. We assume that the device's DMA feature can be restricted to safe memory locations[2]

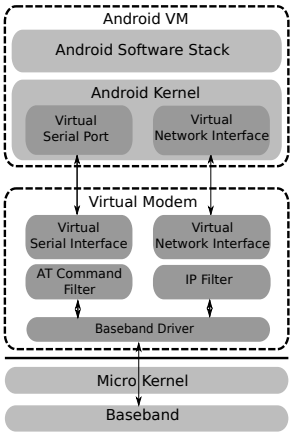

Figure 3. The architecture of our protection system. All interaction with the baseband is mediated by the *virtual modem*. Android runs inside its own virtual machine.

A. Micro Kernel as Secure Foundation

In contrast to a monolithic kernel such as Linux a micro kernel merely implements essential mechanisms. This dramatically reduces the complexity of the kernel. Components such as device drivers or protocol stacks are implemented as user-level tasks [16]. Isolation between user-level tasks is enforced with address spaces. All communication between tasks is done via efficient explicit kernel-mediated inter-process communication (IPC).

Modern third-generation micro kernels implement object-capabilities. This access control scheme makes it possible to build systems that implement the principle of least authority (POLA). POLA states that each component is equipped with the minimum set of permissions necessary to fulfill its task.

The micro kernel ensures spatial and temporal isolation of its user-level tasks. It guarantees safe object access via object-capabilities. This ensures the *Integrity Criterion*.

[2]Technology such as IO-MMUs is already available in personal computers. Similar technology is likely to be implemented in future smartphone CPUs.

B. Virtualized Android

As outlined in the previous section the micro kernel partitions the system in a secure way. The partition running Android is implemented as a virtual machine.

Virtualization requires the Android OS to run with less privileges than the micro kernel. On the other hand, the Android kernel expects to have exclusive control of the hardware. Unfortunately, today's smartphone CPUs are not natively virtualizable, which prevents virtualization in the form of trap and emulate [23].

Fortunately, it is possible to run monolithic OS kernels such as Linux as a user-level task on top of a micro kernel. Härtig et al. [12] showed that the overhead of running a monolithic OS on top of a micro kernel is between 5 and 10 percent. We believe that this is acceptable on modern smartphones, given the merits it brings in terms of security.

In our setup the Android kernel is modified to run as an application on top of the micro kernel. As such, it can only access memory pages that we present it with. By granting a predefined set of IO memory pages, we can restrict the hardware that the Android kernel can access. We enforce that Android cannot directly access the baseband by not giving it access to the baseband's IO memory. Instead, we present it with an interface to our virtual modem. This ensures that all cellular communication is mediated by our protection system (*Completeness Criterion*).

Whereas we slightly modify the Android kernel, its user-level software stack remains unmodified. We designed the Android VM to be safe-to-root, be it voluntarily by the user, or by malware. If the user wants to flash his device, he is free to exchange the content of the Android partition. A commercial version of our protection system requires a bootloader that is capable of restricting updates to the Android partition. Required adjustments of currently used bootloaders are minimal.

C. Virtual Modem

The virtual modem is the only software that is allowed direct access to the baseband hardware. As such, it mediates all cellular network access of Android. It consists of the following components.

Baseband Driver The baseband driver contains all the logic needed to communicate with the baseband hardware. The actual implementation is specific to the device, and often contains numerous dependencies such as drivers for I2C or SPI buses. The baseband driver also contains the logic to tunnel IP data packets through the cellular data network.

Virtual Serial Interface Our virtual modem provides its client (the virtualized Android) with a virtual serial interface for sending and receiving the AT command stream.

AT Command Filter All AT commands are mediated and filtered by our AT command filter. The AT command filter is the central component that enforces our policies on the baseband. It will be explained in detail in Section VI.

Virtual Network Interface Once a data connection is established, all data packets are transfered between the baseband driver and Android via a virtual network interface.

IP Filter The virtual modem includes the infrastructure for network address translation (NAT).

V. Implementation

We built our prototype around an Intel x86-based smartphone. However, the design described in Section IV applies equally well to the widely used ARM architecture.

We picked the *Fiasco.OC* [32] micro kernel as the foundation of our system. Fiasco.OC is a modern third-generation micro kernel, which provides the features outlined in Section IV.

A. Hardware

We developed our prototype on an Aava [4] development phone. The phone hardware is built around the Intel Moorestown [13] platform. It consists of a SoC that contains a graphics accelerator (GPU) and a low voltage Atom core. The Atom CPU is clocked at 1.5Ghz and supports hyperthreading. The board is equipped with 512MB RAM. For debugging purposes a UART is connected via SPI. The phone contains a ST-Ericsson U300 series baseband.

A picture of one of our development phones is included in the Appendix.

B. L4Android

The L4Android project [15] is based on L4Linux [31], a version of Linux that was ported from the machine interface to the micro kernel interface of Fiasco.OC. In addition to L4Linux, which is derived from the mainline Linux kernel, L4Android incorporates Google's Linux kernel modifications to support the Android software stack.

L4Android runs as an application in its own address space on top of the micro kernel. Each of the Android processes runs in its own address space and benefits of the same isolation capabilities as on the stock Android kernel. As the L4Android kernel ABI is compatible with Android, we can run all Android applications without modifications, even those containing native code.

L4Android supports the Android user-level software stack in versions 2.1 (Eclair) up to 2.3 (Gingerbread) and enables us to even run multiple instances of Android in parallel on one device.

C. System Setup

Our setup is depicted in Figure 4. It is made up of two logical partitions: The *Android VM* and the *virtual modem* partition. The former runs the L4Android kernel and the Android user-level software stack (including applications).

The virtual modem partition consists of a L4Linux instance, the *Forwarder* and our *AT command filter*. We grant L4Linux exclusive access to the baseband. This has the benefit of allowing the use of the vendor supplied native Linux driver, and we do not need to implement our own one.

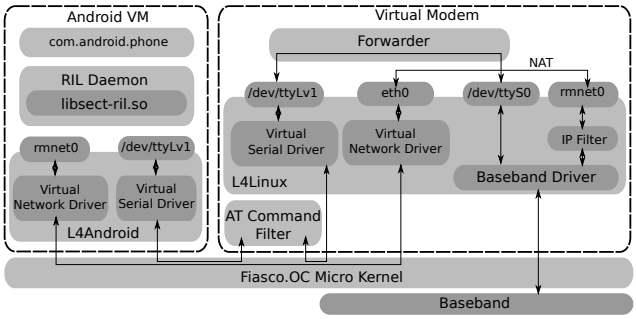

Figure 4. Our implementation consists of two components running on the Fiasco.OC micro kernel, the L4Android running the Android software stack and the virtual modem. The virtual modem is composed of three parts, the AT command filter, a Linux kernel that contains drivers and the Forwarder.

L4Linux is responsible for:

- Booting and initializing the baseband. This potentially includes loading a firmware to the baseband.
- Running the baseband driver. This includes the driver for the serial line that connects the baseband to the application CPU, and all the logic needed to demultiplex the serial stream into data packets and commands. Furthermore, it implements the protocol stacks needed to tunnel IP packets over the cellular network.

L4Linux implements advanced functionality such as IP filtering or Network Address Translation (NAT). It does not present a user interface as it does not require user interaction.

Communication between Android and the virtual modem partition is established via two channels. One is the virtual serial line to transmit the AT command stream. It is proxied by the AT command filter to implement the filtering. The other channel is the virtual network for IP-based data connections.

Virtual Serial Device The virtual serial device is used for all baseband commands. Both the L4Android and L4Linux kernel contain a custom driver that presents a serial device to applications. The custom driver establishes a virtual bidirectional serial line and sends all data via IPC.

Virtual Network Interface For data connections, we employ a shared memory based virtual Ethernet driver. Packets are written into a shared memory region, and the receiver is notified of incoming packets via IPC.

The L4Linux instance in the virtual modem partition forwards data received on the virtual devices to the corresponding physical device and vice versa. For the serial devices this task is performed by the Forwarder, which is implemented as a Linux application. It routes commands between the virtual serial line and the serial control channel of the baseband. In addition, the Forwarder parses the PDP context activation reply from the baseband (see Figure 5), extracts the parameters and applies them to the network interface presented by the baseband driver. The original values are replaced with the parameters necessary to configure the virtual network interface in the Android partition.

The forwarding of IP packets between the virtual network interface and the one provided by the baseband driver is performed using the Linux *netfilter* infrastructure. We setup a simple IP masquerading rule, but more advanced firewall rules can be added.

D. Modifications to the Android RIL

The Radio Interface Layer (RIL) daemon in Android abstracts details of the baseband implementation for upper layers in the Android stack. This includes voice calls, SMS messages, creation of PDP contexts, and configuration of the baseband. Specifics of the baseband are implemented by the baseband manufacturers in separate libraries, such as `libreference-ril.so`. The libraries are loaded by the RIL daemon to access the baseband functionality. Each vendor has to develop such a library when adopting Android to a new baseband.

From Android's perspective our virtual modem behaves like a specific baseband implementation. Consequently we built our own abstraction library (`libsect-ril.so`) for the RIL daemon. The rest of the Android user-level software stack remains unmodified.

The Android RIL configures the network interface used for data connections. As shown in Figure 5 the connection parameters of a PDP context are transfered as XML. Our library extracts these parameters and applies them to the virtual network interface.

VI. The AT Command Filter

The baseband takes care of all interaction between the smartphone OS and the cellular network. The interface between the OS and the baseband is a serial character stream. The serial stream carries commands (signaling) and data (packet-data; IP packets). Voice is handled through other interfaces. Our focus is the signaling. Signaling is done via the GSM extensions to the AT command set as standardized in [2].

The curious reader might also think about the so-called GSM-codes that one can enter into a phone's dialer application (e.g. `##002#` to clear call-forwarding settings). These GSM-codes are part of the Man-Machine Interface (MMI) standard [1] and are simply translated into AT commands by the user-level phone dialer application.

In the rest of this section we first characterize the signaling relevant AT commands and give some brief insights of our specific baseband. Then we discuss special issues with filtering AT commands and how we solved them. In the remaining part of the section we present our implementation, how we block commands, and how we profiled the AT commands to determine the baseline for configuring our filter.

A. AT Command Characterization

We analyzed the AT communication to determine what commands and command sequences are used to perform critical operations such as changing call-forwarding or packet-data settings. Below we briefly discuss the relevant commands.

AT+CGDCONT Configure a PDP context. This sets the connection parameters such as the Access Point Name (APN), user name and password, and other optional parameters. We provide an example of this command later in this section.

AT+CGACT Activate a configured PDP context. However, this standardized command, is not used on the ST-Ericsson baseband that our hardware comes with. There, activation of the PDP context works differently and is described below. There are other commands to activate and de-activate a PDP context, but these are not considered within the scope of this work.

AT*EPPSD PDP context control for our ST-Ericsson baseband. The command takes the PDP context index and the new state (1 = up or 0 = down) as arguments. In the next section we provide more details on the PDP context setup and activation.

AT+CMGS Send an SMS message. The SMS message is provided as hex encoded Protocol Data Unit (PDU). The command below sends an SMS message in PDU mode, the message consists of 17 bytes.

```
AT+CMGS=17
>
0001000c811015214365870000 04d4f29c0e
```

ATD+4930835358585; Initiates a voice call to the given number. The semicolon signals the baseband that the call is actually a voice call. Without semicolon the baseband tries to establish a data call.

AT+CCFC Configure, activate, and de-activate call-forwarding settings. The command takes the type of call-forwarding such as when *busy* or *unreachable*, and the destination number as arguments. The example below sets call-forwarding for the *busy* state to the given number.

```
AT+CCFC=1,1,"4915112345678",129,0
```

AT+CFUN Configuration of the baseband state. The most common states are: Flight mode (stop all radio transmissions), GSM only, 3G only, and GSM+3G (prefer 3G) mode. The command below switches the baseband to Flight mode.

```
AT+CFUN=4
```

B. PDP Context Setup on the STE Baseband

First, the PDP context is configured using the standardized command `AT+CGDCONT`. Activation is performed by the custom `AT*EPPSD` command. The baseband replies with a XML text block containing the IP address, subnet mask, MTU, and DNS server IP addresses. Figure 5 shows an example of the whole process including the context configuration.

```
AT+CGDCONT=1,"ip","internet.t-mobile","",0,0
OK
AT*EPPSD=1,1,1
<?xml version="1.0"?>
<connection_parameters>
<ip_address>10.165.132.86</ip_address>
  <subnet_mask>255.255.255.255</subnet_mask>
  <mtu>1500</mtu>
  <dns_server>193.189.244.225</dns_server>
  <dns_server>193.189.244.206</dns_server>
</connection_parameters>
OK
*EPSB
```

Figure 5. Configuration and activation of a PDP context on our ST-Ericsson baseband hardware.

C. Special Problems

While analyzing the AT command interface and experimenting with our device we identified some additional issues with the AT commands.

Special case APN. Some operators have an additional APN for MMS, therefore, one has to take care of additional legal APN activate-deactivate sequences. Our implementation includes additional checks which ensure that deliverability of MMS messages is not restricted.

Command side effects. Certain AT commands have side effects that need to be taken into account by our filter. We determined that the baseband state switch command (AT+CFUN) is such a case. If the baseband is switched between 2G and 3G the PDP context is disconnected and reconnected.

D. Filtering AT Commands

As shown in Figure 3 and 4, the AT command filter sits between the Android user-level telephony stack and the baseband.

The filter parses commands issued by the RIL (the RIL daemon runs in the Android partition) and enforces the configured filter rules. Commands that are not relevant are forwarded to the baseband without applying any parsing. Results are passed back to the RIL.

We implemented filters for all commands we discussed earlier in Section VI-A. These are: packet-data configuration and activation (AT+CGDMNT and AT*EPPSD), call-forwarding (AT+CCFC), modem control (AT+CFUN), SMS (AT+CMGS), and calls (ATD). The filter works as an intelligent rate limiter.

It counts how often a command is issued within a period of time (the *interval*). If the count reaches the *threshold* all further commands issued within the *interval* are blocked. The rule below will allow issuing 5 AT+CCFC commands within 60 seconds.

```
AT_CCFC_interval = 60 (seconds)
AT_CCFC_threshold = 5 (# commands)
```

Certain commands have to be combined (see special issues Section VI-C). The core logic of our filter is shown in Figure 6.

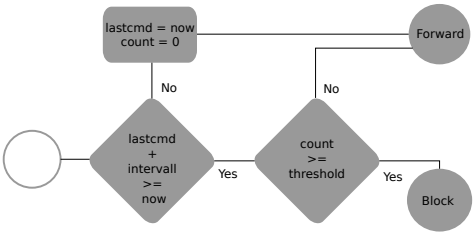

Figure 6. For the commands of interest we track each instance of a command within the configured interval. If the configured threshold is reached the command is blocked. When the interval expires the counter is reset.

E. SMS Filter

We implemented additional filters to inspect the PDU of SMS messages [3] sent from Android to the virtual modem, more closely. We implemented two features. First, a premium rate number detector. Second, a binary payload detector.

The **Short Code Detector** inspects the destination number of every SMS message that is sent. Premium rate numbers are mostly implemented using so-called *short codes*, telephone numbers as short as 4-6 digits. If a short code is detected and short code blocking is activated the command is blocked.

Our prototype blocks all SMS messages sent to short codes. To allow sending *legit* SMS to short codes we need to complete the implementation of the *secure GUI* we describe in the future work in Section IX-A.

The **Binary Message Payload Detector** inspects the header and payload field of every SMS message that is sent through the filter. It uses a simple heuristic to determine if the message is binary. The heuristic checks one flag in the header and checks if the message body mainly consists of non-printable characters. The ratio of printable to non-printable can be configured. It further checks for base64 encoding and flags the message as binary if this is detected. If the message is determined to be binary it will be subjected to the rate limiting rule for binary SMS messages.

F. Blocking Commands

Commands are blocked by simply not forwarding them to the baseband. To not confuse the application logic in the RIL our filter issues an appropriate error message for each blocked command. The error message is injected into the stream that otherwise carries the responses from the baseband to the RIL. Special commands are never blocked due to various reasons. These are:

Switch to flight mode (AT+CFUN=4). This is necessary since flight mode is a required functionality that must always work. Even if the *threshold* for the CFUN command is reached a switch to flight mode is always permitted. In the worst case the phone will remain in flight mode until the *interval* expires.

PDP context deactivation (AT*EPPSD). This is necessary to prevent excessive data costs. For example, when the phone is roaming and the user wants to deactivate packet-data.

Emergency calls (ATD 911;) must always work due to regulations.

G. Profiling benign AT Command Usage

To determine useful and working intervals and thresholds for configuring our filter we monitored the AT commands we are interested in. To determine how often and when these commands are issued we set the intervals to 86,400 seconds (one day). Thus, the filter only counts the number of commands but never actually blocks anything. Table I shows AT command usage in general. Note, that the call-forwarding command (AT+CCFC) is issued multiple times at the point when the user opens the call-forwarding settings screen. This is because the phone always queries the network for the settings since these can be changed from multiple places. When the user changes a setting an additional command is issued. Followed by querying the state again. Therefore, the call-forwarding filter has to take into account that at a certain time multiple commands are executed in a row.

In the Appendix we included an example output of our filter log right after the phone booted.

Command	#	When	Why
AT+CFUN	2	Boot	Flight mode. Normal mode.
AT+CFUN	1	Use	Switch to GSM-only.
AT+CDGMNT	1	Boot	Set PDP configuration.
AT*EPPSD	1	Boot	Activate PDP context.
AT+CMGS	1	Use	Send a SMS message.
ATD	1	Use	Issue a voice call.
AT+CCFC	3	Use	Query forwarding settings.
AT+CCFC	2	Use	Set a call-forwarding.

Table I
AT COMMANDS ISSUED DURING RUNTIME.

VII. EVALUATION

We developed a set of test applications to simulate rogue behavior such as updating call-forwarding settings or changing the PDP context. We further acquired a sample of an actual Premium Rate SMS Trojan for the Android platform to test against a real-world malware. Below we first describe our

evaluation environment – our small GSM network, followed by the evaluation of our protection system against the threats we described in Section III.

A. Our GSM Test Network

Our setup consists of a small GSM network that is based on an ip.access nanoBTS. The nanoBTS is managed by OpenBSC [34]. Our network is operated in a Faraday cage, where we conduct all our experiments safely. OpenBSC comes with additional components that provide a SGSN and a GGSN, which allows to operate a packet-data network in addition to the voice and SMS services. The setup allows us to monitor all relevant aspects of the cellular network. Such as PDP context establishment and incoming and outgoing SMS traffic.

Through the use of this environment we can test and verify our implementation.

B. Limiting the Call-forwarding Attack

The call-forwarding attack as described by [30] is based on insertion of call-forwarding settings by hijacked phones. Their attack requires 2,500 Transactions Per Seconds (TPS) for low traffic networks and up to 30,000 TPS for high traffic networks.

The victim phones issue AT+CCFC commands to configure and enable call-forwarding. The authors of [30] calculated that on average a command takes 4.7 seconds to complete, meaning one can issue up to 12 commands per minute. Thus, they require 11,750 bots to perform the attack on a low traffic network.

$$4.7\ seconds * 2,500\ TPS = 11,750\ hosts$$

For our initial experiment we configured the filter to allow 5 commands per minute. We chose this configuration because the Android call-forwarding configuration panel issues 3 commands when it is started. These commands query the network for the current state as shown in Table I. The command that causes high load (enable call-forwarding) is only issued when the user changes a setting. After changing the setting the network is queried again. We, therefore, set the *threshold = 5*. With this setting the botnet's size already has to more than double in order to successfully perform the attack. Figure 7 shows the necessary size increase of the botnet described in [30] to perform the attack if the zombie phones are equipped with our protection system.

To further improve the protection provided by our system we can increase the *interval* of the call-forwarding filter, resulting in an even lower number of commands per minute. For example, allowing just 10 call-forwarding commands over a period of 10 minutes of time. Such a threshold results in 1 command per minute on average, which is reasonable for normal usage.

C. Limiting PDP Context Changes

To limit the number of PDP context changes we have to mediate two different commands. The commands are described in Section VI-C where the side effects of these commands are presented. The side effect, which must be detected by our system, is switching the baseband mode between GSM-only, 3G-only, and GSM+3G. Calculating the *threshold* for PDP context changes is straightforward.

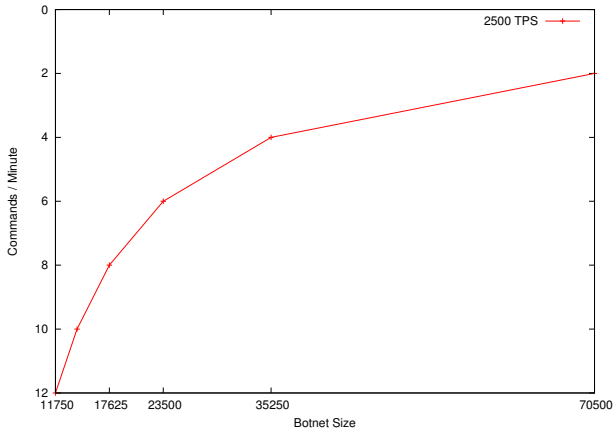

Figure 7. The increase in size of the botnet necessary to maintain the 2,500 TPS with our protection system in place.

Defining p_t as the threshold for PDP context changes, e_t as the threshold for AT*EPPSD commands, and c_t as the threshold for AT+CFUN commands, yields

$$p_t = e_t + c_t.$$

The graph in Figure 8 shows the number of possible PDP context changes depending on the settings of p_t. Without any rate limiting applied, 30 changes per minute is the maximum possible.

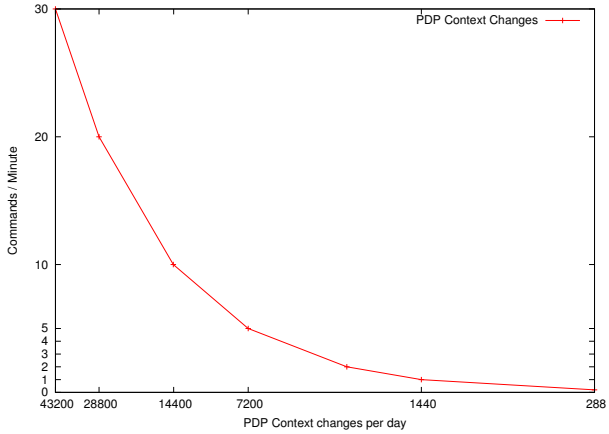

Figure 8. The graph shows the number of PDP context changes possible based on how many commands are allowed per minute. The last point at 288 PDP context per day is based on 0.2 commands per minute.

D. SMS Trojan

We installed the `FakePlayer-A` [9] premium SMS Trojan to a test phone equipped with our protection system. The Trojan is built to send a SMS message to a premium rate

number to steal money from the victim. We closely monitor the log output of our filter to determine what happens. In addition we also monitor the phone's behavior on our private GSM network, to see if it actually sends any SMS message or not.

The Trojan tries to send an SMS message to the number 3353. As this number is short (only 4 digits) it is detected by our *Short Code Detector*. Extensive analysis of this Trojan [9] determined that the Trojan sends a SMS to either 3353 or 798657. However, we only observed attempts to 3353 in our lab. Below is an excerpt from our filter log.

```
AT+CMGS=15
0001000481333500006b71cce56bb01
number: 3353
short number >3353< detected, could be premium
filterd: filtered returned: 0
filterd: blocking
 0001000481333500006b71cce56bb01
```

E. SMS Controlled Botnets

SMS controlled botnets such as [19], [33] send and receive SMS messages for their command and control channel. Since this work focuses on outgoing signaling traffic we decided to only look at outgoing traffic, from the phone to the network.

To prevent botnet communication we enabled our *Binary Payload Detector* together with the rate limiting for the AT+CMGS command (the command to send SMS messages). The rate limiter will only prevent the phone from sending binary SMS messages at a high rate. Binary SMS messages are rarely sent by the user since these are mostly used by applications. Furthermore, the most common usage of binary SMS are messages that are received by the phone (e.g. as part of MMS). Text messages on the other side are often used in an instant messaging scheme with a high rate of outgoing and incoming messages. Therefore, blocking text messages will be more complicated since they would need to be analyzed thoroughly before one is able to safely block them.

VIII. Related Work

Related work falls into four categories. First, security enhancements for smartphones. Second, virtualization on smartphones. Third, Android specific security extensions. Forth, infrastructure-based security enhancements for cellular networks.

Traynor et al. summarize in [28] the lack of security features on mobile and smartphones and discuss possible solutions. Part of their work presents SELinux [18] as means of access control of system resources. But the authors come to the conclusion that such an approach is infeasible. In our work, we directly address the specific problems of signaling attacks. We not only propose a solution, but fully implemented a prototype and evaluated it. Mulliner et al. [20] build a label based tracking system that tracks a process' access to network interfaces to limit future access to other network resources such as the cellular modem. The SEIP [35] architecture uses D-Bus in combination with SELinux to enforce access policies for applications accessing various system resources on a smartphone.

Selhorst et al. [26] describe a Trusted Mobile Desktop prototype that, similar to our approach, uses a micro kernel together with multiple virtualized Linux instances. In their setup, a so-called *User Linux* partition drives the baseband and runs the user's applications. A separate component signs and encrypts SMS. The encrypted SMS is then sent via the User Linux's baseband driver. They do not provide means for protecting the cellular network from malicious behavior of the User Linux partition. Schmidt et al. [25] describe how a trusted mobile platform can be built on a trustworthy platform. The authors employ the Turaya security kernel to run a virtualized legacy operating system (Linux) side-by-side with multiple trusted engines. They do not propose to control network interfaces. Klein et al. [14] design and implement seL4. In their work the authors demonstrate that the implementation of a modern third-generation micro kernel can be formally verified to match its specification. VMware [22] ported their virtualization software to the Android platform. However, this port runs a virtualized guest version of Android on top of a host Android. Additionally, our solution is based on a micro kernel and has a significantly smaller trusted computing base.

Enck et al. build TaintDroid [8] a taint tracking based security and privacy enhancement for Android. TaintDroid is able to track which data an application accessed. The MockDroid [6] Android enhancement adds the possibility to selectively mock specific features such as Internet connectivity. Thus applications cannot use specific functionalities even if they actually are available. In [7], [5] the authors build user land domain isolation systems for Android in order to protect against malware attacks. These systems do not protect against malware that has root level access, in comparison, our solution specifically protects against this threat.

Previous work on protecting cellular phone networks has targeted other attack vectors such as [29] that investigates countermeasures for preventing resource exhaustion attacks against cellular phone networks carried out over the Internet.

IX. Conclusions and Future work

We designed and implemented our protection system called the *virtual modem* to protect cellular network infrastructure from hijacked smartphones. The virtual modem mediates all signaling traffic from the smartphone OS to the baseband and thus protects the cellular network. The implementation is based on running Android and our virtual modem in isolated partitions on top of a micro kernel. Our solution is independent from the baseband and thus supports wide adoption.

We evaluated our implementation using real mobile phone hardware that we connect to our own GSM network. Our GSM network allows us to monitor all relevant activities of the phone. Part of the evaluation was installing a real-world Trojan on the device. The Trojan was successfully launched, but our virtual modem prevented the fraudulent access to the cellular network.

Signaling attacks are a serious threat and recognized as such by the GSMA. The evaluation of our protection system showed that it can effectively prevent these attacks and thus protect cellular core networks. In addition it protects the end-user. We believe that protecting the baseband from the smartphone OS is the necessary next step in the evolution of securing smartphones. Thus leading towards creating more responsible devices that participate in the growing mobile communication world.

A. Future Work

Our virtual modem can be enhanced with the following functionality.

VPN Gateway The modem can establish access to VPNs in a way that even a rooted Android cannot access the key material.

Advanced Intrusion Detection/Prevention We can enhance our IP filter with logic to detect and prevent attacks against the smartphone as well as against the operator.

Policy Update Infrastructure The virtual modem can include an update infrastructure to allow the operator to update the filtering policies. Such an update would be performed transparently to the user.

Secure GUI With the addition of a *secure graphical user interface*, we can implement a dialog that enables the virtual modem to ask the user for admission of premium SMS and calls. To make sure that Android malware cannot mimic the admission dialog, or automatically send the confirmation input, the dialog must be presented in a way that does not depend on Android for input. Doing so requires virtualization of the graphics and input hardware.

Hardware Virtualization Porting the Android kernel to our micro kernel requires a significant amount of work. When hardware support for CPU virtualization becomes available on smartphones, it can both reduce the amount of modifications to the Android kernel, and may improve the performance of our Android VM.

Acknowledgements

The authors would like to thank Axelle Apvrille for supplying us with a sample of the AndroidOS.FakePlayer-A Trojan for our evaluation. We would also like to thank Dmitry Nedospasov, Borgaonkar Ravishankar, and Patrick Stewin for reviewing the paper.

References

[1] 3GPP/ETSI. 3GPP TS 02.30 - Man-Machine Interface (MMI) of the Mobile Station (MS). http://www.3gpp.org/ftp/Specs/html-info/0230.htm, June 2002.

[2] 3GPP/ETSI. 3GPP TS 27.007 - Technical Specification Group Terminals; AT command set for User Equipment (UE). http://www.3gpp.org/ftp/Specs/html-info/27007.htm, June 2003.

[3] 3GPP/ETSI. 3GPP TS 23.040 - Technical realization of the Short Message Service (SMS). http://www.3gpp.org/ftp/Specs/html-info/23040.htm, September 2004.

[4] Aava Mobile Oy. Aava Mobile. http://www.aavamobile.com/.

[5] J. Andrus, C. Dall, A. V. Hof, O. Laadan, and J. Nieh. Cells: A Virtual Mobile Smartphone Architecture. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles*, October 2011.

[6] A. R. Beresford, A. Rice, N. Skehin, and R. Sohan. MockDroid: trading privacy for application functionality on smartphones. In *12th Workshop on Mobile Computing Systems and Applications*, March 2011.

[7] S. Bugiel, L. Davi, A. Dmitrienko, S. Heuser, A.-R. Sadeghi, and B. Shastry. Practical and lightweight domain isolation on android. In *Proceedings of the 1st ACM CCS Workshop on Security and Privacy in Mobile Devices (SPSM)*. ACM Press, Oct 2011.

[8] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI'10, pages 1–6, Berkeley, CA, USA, 2010. USENIX Association.

[9] Fortinet. Trojan-SMS.AndroidOS.FakePlayer-A. http://www.fortiguard.com/encyclopedia/virus/android_fakeplayer.a!tr.html, August 2010.

[10] Google Inc. Android. http://www.android.com/.

[11] GSM Association (GSMA). Network Efficiency Threats v0.4a, May 2010.

[12] H. Härtig, M. Hohmuth, J. Liedtke, J. Wolter, and S. Schönberg. The performance of µ-kernel-based systems. In *Proceedings of the sixteenth ACM symposium on Operating systems principles*, SOSP '97, pages 66–77, New York, NY, USA, 1997. ACM.

[13] Intel Corporation. Introducing the Next-Generation Intel® Atom™ Processor-based Platform. http://download.intel.com/pressroom/kits/atom/z6xx/pdf/Fact_Sheet_Intel_Atom_Processor_Platform.pdf, 2010.

[14] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood. seL4: Formal Verification of an OS Kernel. In *ACM Symposium on Operating System Principles*, pages 207–220. ACM, 2009.

[15] M. Lange and S. Liebergeld. L4Android: Android on top of L4. http://www.l4android.org, February 2011.

[16] J. Liedtke. On micro-kernel construction. In *Proceedings of the fifteenth ACM symposium on Operating systems principles*, SOSP '95, pages 237–250, New York, NY, USA, 1995. ACM.

[17] Lookout Inc. DroidDream. http://blog.mylookout.com/2011/03/security-alert-malware-found-in-official-android-market-droiddream/, March 2011.

[18] P. Loscocco and S. Smalley. Integrating Flexible Support For Security Policies Into The Linux Operating System. In *Proceedings of the FREENIX Track of the 2001 USENIX Annual Technical Conference*, 2001.

[19] C. Mulliner and J.-P. Seifert. Rise of the iBots: 0wning a telco network. In *Proceedings of the 5th IEEE International Conference on Malicious and Unwanted Software (Malware)*, Nancy, France, October 2010.

[20] C. Mulliner, G. Vigna, D. Dagon, and W. Lee. Using Labeling to Prevent Cross-Service Attacks Against Smart Phones. In *Proceedings of the Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, volume 4064 of *LNCS*, pages 91–108, Berlin, Germany, July 2006. Springer.

[21] J. Niemelä. Mobile Malware And Monetizing 2011. https://noppa.tkk.fi/noppa/kurssi/t-110.6220/luennot/T-110_6220_mobile_maleware.pdf, 2011.

[22] PC World. VMWare Shows off Mobile Virtualization on Android. http://www.pcworld.com/article/219671/vmware_shows_off_mobile_virtualization_on_android.html, 2011.

[23] G. J. Popek and R. P. Goldberg. Formal requirements for virtualizable third generation architectures. *Commun. ACM*, 17:412–421, July 1974.

[24] P. A. Porras, H. Saidi, and V. Yegneswaran. An Analysis of the iKee.B iPhone Botnet. In *Proceedings of the 2nd International ICST Conference on Security and Privacy on Mobile Information and Communications Systems (Mobisec)*, May 2010.

[25] A. U. Schmidt, N. Kuntze, and M. Kasper. On the deployment of mobile trusted modules. In *Wireless Communications and Networking Conference, 2008. WCNC 2008. IEEE*, pages 3169–3174. IEEE, 2008.

[26] M. Selhorst, C. Stüble, F. Feldmann, and U. Gnaida. Towards a trusted mobile desktop. In *Proceedings of the 3rd international conference on Trust and trustworthy computing*, TRUST'10, pages 78–94, Berlin, Heidelberg, 2010. Springer-Verlag.

[27] Strategy Analytics. Q3 2009 Cellular Baseband Market Review. http://blogs.strategyanalytics.com/HCT/post/2010/03/09/Q3-2009-Cellular-Baseband-Market-Review.aspx, March 2010.
[28] P. Traynor, C. Amrutkar, V. Rao, T. Jaeger, P. McDaniel, and T. La Porta. From Mobile Phones to Responsible Devices. *Journal of Security and Communication Networks (SCN)*, 2010.
[29] P. Traynor, W. Enck, P. McDaniel, and T. La Porta. Mitigating Attacks on Open Functionality in SMS-Capable Cellular Networks. *IEEE/ACM Transactions on Networking (TON)*, 2009.
[30] P. Traynor, M. Lin, M. Ongtang, V. Rao, T. Jaeger, T. La Porta, and P. McDaniel. On Cellular Botnets: Measuring the Impact of Malicious Devices on a Cellular Network Core. In *ACM Conference on Computer and Communications Security (CCS)*, November 2009.
[31] TU Dresden. L4Linux - Running Linux on top of L4. http://os.inf.tu-dresden.de/L4/LinuxOnL4/, January 2011.
[32] TU Dresden. The Fiasco microkernel. http://os.inf.tu-dresden.de/fiasco/, January 2011.
[33] G. Weidman. Transparent Botnet Control for Smartphones over SMS. http://www.grmn00bs.com/GeorgiaW_Smartphone_Bots_SLIDES_Shmoocon2011.pdf, January 2011.
[34] H. Welte. OpenBSC. http://openbsc.osmocom.org/trac/, 2008.
[35] X. Zhang, J.-P. Seifert, and O. Acicmez. SEIP: Simple and Efficient Integrity Protection for Open Mobile Platforms. In *Information and Communications Security*, volume 6476 of *Lecture Notes in Computer Science*, pages 107–125. Springer Berlin / Heidelberg, 2010.

APPENDIX

Figure 9. One of our Aava devices. A debug board is attached to the right and provides a serial line.

```
current time: 3793

APN[ 1]      : "internet.t-mobile"
       state : 1
       count : 1
       last  : 3793
       cfg   : 3793
APN current                    : 1
APN switch   count             : 1
APN switch   interval (policy) : 86400
APN switch   threshold (policy): 6
APN switch   last              : 3793

CALLFWD[0]   interval (policy) : 86400
CALLFWD[0]   threshold (policy): 5
CALLFWD[0]   last              : 0
CALLFWD[0]   count             : 0
CALLFWD[1]   interval (policy) : 86400
CALLFWD[1]   threshold (policy): 5
CALLFWD[1]   last              : 0
CALLFWD[1]   count             : 0
CALLFWD[2]   interval (policy) : 86400
CALLFWD[2]   threshold (policy): 5
CALLFWD[2]   last              : 0
CALLFWD[2]   count             : 0
CALLFWD[3]   interval (policy) : 86400
CALLFWD[3]   threshold (policy): 5
CALLFWD[3]   last              : 0
CALLFWD[3]   count             : 0
...          ...
CALLFWD[5]   count             : 0

GSMONLY      interval (policy) : 86400
GSMONLY      threshold (policy): 4
GSMONLY      last              : 3778
GSMONLY      count             : 2
GSMONLY      mode              : 1

BINSMS       interval (policy) : 86400
BINSMS       threshold (policy): 1
BINSMS       last              : 0
BINSMS       count             : 0

SMSSHORT     count             : 0
```

Figure 10. The status of our AT command filter after booting the device for AT command profiling.