Triton: A Carrier-based Approach for Detecting and Mitigating Mobile Malware

Arati Baliga[1], Jeffrey Bickford[2]
and Neil Daswani[3]

[1] *NYU Polytechnic School of Engineering*
[2] *AT&T Security Research Center*
[3] *Twitter Inc.*

Received 28 February 2014; Accepted 28 February 2014;
Publication 2 July 2014

Abstract

The ubiquity of mobile devices and their evolution as computing platforms has made them lucrative targets for malware. Malware, such as spyware, trojans, rootkits and botnets that have traditionally plagued PCs are now increasingly targeting mobile devices and are also referred to as mobile malware. Cybercriminal attacks have used mobile malware trojans to steal and transmit users' personal information, including financial credentials, to bot master servers as well as abuse the capabilities of the device (e.g., send premium SMS messages) to generate fraudulent revenue streams.

In this paper, we describe Triton, a new, network-based architecture, and a prototype implementation of it, for detecting and mitigating mobile malware. Our implementation of Triton for both Android and Linux environments was built in our 3G UMTS lab network, and was found to efficiently detect and neutralize mobile malware when tested using real malware samples from the wild. Triton employs a defense-in-depth approach and features: 1) in-the- network malware detectors to identify and prevent the spread of malware and 2) a server-side mitigation engine that sends threat profiles to an on-the-phone trusted software component to neutralize and perform fine-grained remediation of malware on mobile devices.

* This work was conducted while Arati Baliga was with the AT&T Security Research Center. Neil Daswani conducted this research while employed at Dasient. Twitter acquired Dasient in January 2012.

Journal of Cyber Security, Vol. 3 No. 2, 181–212.
doi: 10.13052/jcsm2245-1439.324

1 Introduction

Mobile devices have become an integral part of our daily lives; we rely on them to send and receive email, communicate with family and friends, perform financial transactions, and much more. Due to the inherent trust users place in these devices, as well as the availability and frequent download of hundreds of thousands of apps, it is no coincidence that mobile devices are now targets of complex malware attacks. According to a threat report by F-Secure Labs, 5,033 malicious Android applications were discovered in the second quarter of 2012, a 64% increase compared to the previous quarter, including the first Android malware to use a drive-by-download vector for infection [6].

From a mobility network provider point of view, the mobile malware threat not only impacts its individual customers, but also impacts the security and reliability of the mobility network as a whole. Researchers have shown the feasibility of denying mobility network services using specially targeted SMS messages, control channel vulnerabilities, and mobile botnets [28, 42, 51]. In fact, in the recent outbreak of SpamSoldier [14], attackers formed a botnet of SMS spammers, making what was once a research problem now a reality. As with many other security problems, the problem of mobile malware needs to be addressed holistically via a defense-in-depth approach that includes prevention, detection, containment, and recovery techniques. Various solutions have been proposed to tackle the increasing number of mobile threats, though from the perspective of a network provider, no optimal solution exists today.

Many app stores are beginning to use security APIs to scan apps before they list them [9] [1]. While this will limit some malicious apps from being downloaded by the user, mobile malware can be delivered via other attack vectors, such as visiting infected websites (drive-by-downloads), downloading apps from unsafe app stores, spam email or SMS/MMS, or simply downloading unsafe content from unrated or malicious web sites. Recent work has also shown the feasibility of subverting the app store review process, thereby compromising the integrity of the app store itself [47, 8]. Though most companies provide mobile variants of their signature-based anti-virus software; these schemes typically require an exhaustive set of signatures and can be easily thwarted by malware that use techniques such as encryption and packing [24, 34]. In fact, Google's own App Verification Service, introduced in Android 4.2, only detects 15% of 1,200 malware samples previously released to the public [36]. Alternatively, host-based behavioral detection engines, which can detect these sophisticated threats, are simply infeasible to deploy on current mobile devices due to their heavy resource requirements and limited energy constraints [46, 20].

To protect both their customers and network infrastructure, network providers frequently deploy network-based anomaly detectors capable of detecting malicious traffic patterns, such as botnet communication patterns, worm traffic, and DDoS attacks [40]. Within the mobility network, these same security services exist, though they are expanded to include mobility specific attacks, such as SMS spamming campaigns and premium number fraud. Traffic characteristics and malicious payloads are typically analyzed using inhouse analysis environments and third-party cloud services without resource constraints. Traffic characteristics from millions of users are analyzed every day, giving a network provider visibility into a large set of attacks. However, when the network detects a misbehaving device and determines that its activity is harmful to both other customers and the network, the only current possible mitigation strategy is deactivating the device, resulting in dissatisfied customers and calls to customer service.

In this paper, we describe Triton, a new, network-based architecture and a prototype implementation of it for detecting and mitigating mobile malware. Triton combines the strength of network-based detection with the abilities of a trusted device component to identify the malicious app and mitigate the infection. Triton employs a defense-in-depth approach where in-the-network malware detectors communicate network threats to an on-the-phone trusted software component to identify and neutralize malware on the device. Combining network based detection with the ability to identify malware on the device allows Triton to provide protection against threats even in the absence of an anti-virus signature, provide faster response to ongoing threats, operate at lower costs, and leads to a minimal increase in battery consumption on the end device.

The contributions of this work are as follows:

- The Triton architecture detects and renders malware ineffective. Triton derives its effectiveness by placing network level components that detect and communicate threat profiles to a trusted software component running on the device that can identify and mitigate malware.
- A prototype implementation of Triton that we built in our 3G UMTS lab, and discuss some of the real-world trade-offs that we encountered in building it.

The remainder of the paper is organized as follows. In Section **1**, we provide a quick primer of the 3G mobility network to provide the reader an understanding of how Triton fits in. In Section 3, we describe our defense-in-depth approach including Triton's design and implementation. We present our experimental results in Section 4. We address counter attacks, scalability and limitations in Section 5. Related work is covered in Section 6 and we finally conclude in Section 7.

2 Background

This section describes the basic elements involved in the 3G UMTS network [16]. We use this type of network to design, develop and test our architecture. The architecture that we propose is generic enough and can be deployed with other types of 3G networks as well as 4G LTE networks.

2.1 3G UMTS Network Primer

Figure 1 shows the basic components of a UMTS network. In a UMTS network, a mobile device connects to the network via a radio link to the nearest base station, also referred to as the Node B. Multiple base stations are connected to a Radio Network Controller (RNC). For access to the circuit switched services, such as phone calls and SMS messages, multiple RNCs are connected to a Mobile Switching Center (MSC). SMS messages are sent to the nearest SMS Center (SMSC) from the MSC over the control channel. For access to the data services, multiple RNCs are connected to the Serving GPRS Support Node (SGSN). The MSC, SGSN and the Visitor Location Register (VLR) track devices that are connected to the network that they are visiting. Every subscriber in the UMTS network is identified with an International Mobile Subscriber Identity (IMSI) number. Every device is identified with an

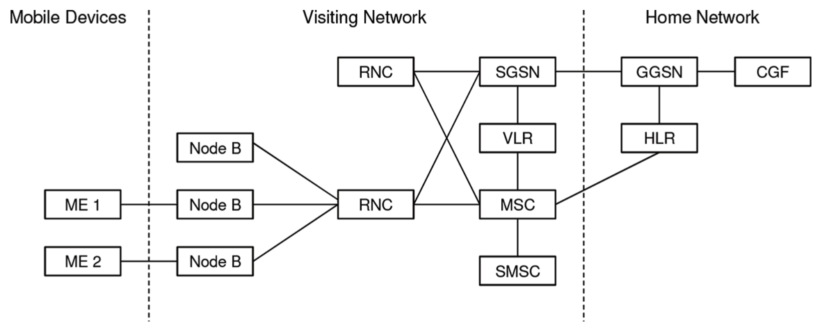


Figure 1 3G UMTS Network Architecture

International Mobile Equipment Identity (IMEI) number. Every subscriber also has a home network that stores the subscriber profile in the Home Location Register (HLR), including the IMSI and the IMEI numbers. Mutual authentication between a mobile device and a visited network is carried out with the support of the serving SGSN or the MSC/VLR. The Gateway GPRS Support Node (GGSN) acts as an anchor for all data traffic originating from the mobile device irrespective of its location. For simplicity and clarity, we have only described elements that are sufficient for basic understanding of the UMTS network. The GGSN is the component that our architecture interfaces with within the UMTS network and therefore is described in more detail below.

2.2 Gateway GPRS Support Node (GGSN)

The GGSN is a node that acts as a gateway between the mobility network and the Internet. The mobile device connects to the local SGSN, which in turn builds a tunnel to the GGSN using the GPRS Tunneling Protocol (GTP). When the device moves to a different location, it switches SGSNs while the GGSN serves as the anchor point of the tunnel that routes data traffic to the Internet. The GGSN covers a very large part of the mobility network for data services as it resides within the user's home network. This placement allows the GGSN to serve as a central point of observation for data traffic and therefore is ideal for placement of network based malware detectors.

The GGSN assigns an IP address to every single outgoing data connection originating from the mobile device. This IP address is randomly picked from a pool of IP addresses owned by the GGSN. Therefore, a single IP address from this pool might represent different mobile devices within the same mobility network at different points in time. Alternatively, different IP addresses might correspond to the same device at different time instances. The GGSN contains all information about user's data usage. It feeds this information into a Charging Gateway Function (CGF). The CGF accounts for data usage and generates billing information based on the usage and the type of data plan subscription. The GGSN naturally acts like a Network Address Translation (NAT) device and has information to correlate the IP address assigned to the device to its identity at any given point in time. This structure of the mobility network provides the following advantages that we leverage.

- **Centralized view of data traffic:** The GGSN acts as a gateway to the data traffic and has visibility into data generated by all the mobile devices that it serves. This centralized view enables detection of large scale attacks, such as command and control traffic originating from mobile botnets and fast spreading worms.

- **Mapping IP to the device identity:** When a new mobile device initiates a data connection, the GGSN creates a Packet Data Protocol (PDP) context for the device. The PDP context is maintained at both ends of the GTP tunnel between the SGSN and the GGSN. It contains all information about the device that has requested the data service including its IMSI/IMEI numbers. The GGSN picks a free IP address from its pool and assigns it to the new PDP context. Because the GGSN maintains this information, at any given point in time, it is able to identify the device with the given IP address. The architecture we propose utilizes this mapping to accurately identify infected devices sending out malicious traffic.

3 Design and Implementation

Triton employs a defense-in-depth approach in combating mobile malware, which broadly comprises of the following steps.

- **Infection prevention:** Triton is able to prevent infections by blocking mobile devices from either visiting or downloading known "bad" apps and content. It can push infection information of new and ongoing threats to non-infected devices, which will prevent malware from running on these devices altogether.
- **Effective detection:** Mobile devices are heavily network centric as most content is delivered from the Internet via apps or websites. Triton places malware detectors within the mobility network and monitors for signs of infection. It also interfaces with third-party cloud services for specialized offline analysis of apps and content, which it uses for detection and prevention.
- **Immediate containment:** Triton employs a trusted component on the mobile device that receives threat profiles from the network. Threat profiles characterize the traffic that was flagged by the network as malicious. The trusted component can identify the application that generated the malicious traffic and immediately stop it from executing. It can also notify the user and remove the application from the device once it has been neutralized.
- **Fine grained response:** Triton's fine-grained response of only containing malware without hampering execution of other legitimate applications, allows the user safe continued access to his device.

3.1 Architecture

Figure 2 shows the end-to-end architecture of Triton and how it interfaces with the GGSN within the 3G UMTS network. Triton places several new components within the mobility network - the Mitigation Engine (MiE), the Network Malware Detector (NMD), the Filter, the Packet Inspector (PI) and the database. It also places a Trusted Host Component (THC) on the mobile device itself to assist with containment.

As explained in Section 2.2, mobile devices send data traffic to the Internet via the GGSN. When a mobile device attaches to the network, it establishes a new PDP context and is assigned an IP address from a pool owned by the GGSN. We instrument the GGSN to log the IP address assigned to the device and its IMSI/IMEI numbers into the database. It also logs the time duration for which the current IP belonged to the specific device, i.e. the duration of the PDP context. This information is required to map the IP address assigned to the device with its device identity (IMSI/IMEI).

The PI sniffs the Internet facing interface of the GGSN and logs all network data flows, originating from and to the mobile devices, into the database. It also logs information from some application level protocols, such as HTTP and DNS. The NMD operates on logged network flows and detects malicious traffic patterns e.g., the NMD might find a mobile device conducting an IP scan or a port scan, or sending traffic to blacklisted command and control servers. Since the NMD has a large scale view of the mobility network, it is well placed to detect spread of worms or other types of large scale attacks, such as DDoS or botnet command and control communication patterns. It is also well-placed to detect other kinds of application level accesses, such as access to malicious websites or malicious app downloads. Along with traditional network-based detectors, the NMD can also rely on external cloud-based

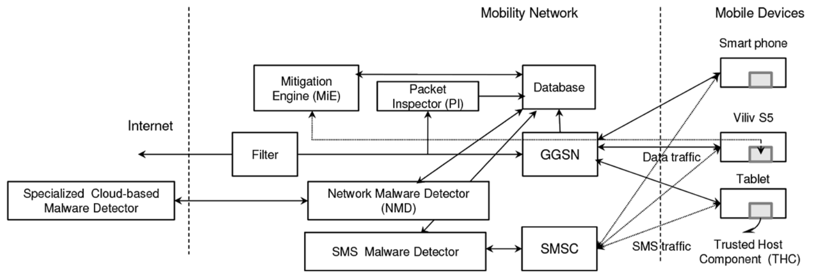


Figure 2 Mobile malware mitigation architecture

services for specialized analysis, e.g., analysis of specific URLs for drive-by-downloads or conducting behavioral analysis of an app that was accessed by the end mobile device. Analysis results from cloud services are used as network signatures for detection both in the NMD and the Filter. The Filter component blocks future access from mobile devices to websites or apps that are identified as malicious. The MiE uses the alert information generated by the NMD to generate a threat profile and communicates the threat profile to the THC on the end mobile device. The mobile device identifies the application that is infected and immediately stops it from executing on the device.

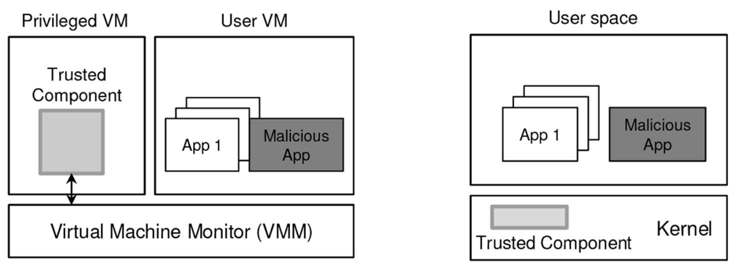
Below, we explain in detail the role of each of the components of Triton and how they communicate with each other to effectively combat mobile malware.

3.1.1 Trusted Host Component (THC)

Triton places the THC on the mobile device in two different ways; primarily based on whether the underlying platform supports virtualization.

Figure 3(a) shows a THC on the device that supports virtualization. On a virtualized platform, the THC runs as an application inside a privileged virtual machine (VM) and all other user applications run inside a User VM. Placing the THC in this fashion allows us to leverage certain well-known security properties of the virtual machine architectures [29, 27]. By placing the THC in the privileged VM, it is effectively isolated from other applications running on the device yet is able to inspect on the state of the applications and the operating system running inside the User VM. This architecture can protect against malware that resides in user, as well as kernel, space inside the User VM.

Figure 3(b) shows the THC running inside the operating system kernel for platforms that do not support virtualization, which is the case with current



(a) Trusted component in the privileged VM (b) Trusted component inside the operating system.

Figure 3 Mobile device architecture and placement of the trusted component

commercial mobile phones. We use this architecture on smart phones where the THC runs inside the kernel. By running in kernel space, the THC is able to intercept system calls from user space applications and inspect application state. However, this model has the limitation that it is unable to detect or defend against attacks that compromise the kernel, e.g., kernel-level rootkits. While using this architecture, we assume that the threat to the device is only in user space.

In both cases, the THC listens for connections from the MiE on a special TCP port. It also maintains a rolling log of network connections both incoming and outgoing generated by user applications running on the device. This log maps each network flow to the application that generated the flow or is the recipient of the flow. When the network identifies that a device is infected with malware, it receives infection information from the MiE about the network flows that were found to be malicious. The THC refers to its log to find the application that generated the malicious flow.

3.1.2 Packet Inspector (PI)

The PI component sniffs the outgoing physical interface of the GGSN, also known as the Gi interface. It logs into the database, all bidirectional network flows from the mobile devices. Network flows comprise of source and destination IP addresses, source and destination ports and the protocol used. The PI also logs some relevant information from certain application level protocols of interest, such as domain names for DNS and URLs for HTTP. DNS information allows the NMD to identify applications trying to contact malicious domains. Logging HTTP URLs helps identify access to malicious websites from mobile devices.

3.1.3 Network Malware Detector (NMD)

The NMD operates on logged network flows and identifies malicious network traffic. The NMD is designed with extensibility in mind and can leverage any number of well known tools or techniques to identify malicious traffic patterns [2, 15, 23, 31–33, 41 52–53]. In order to improve Triton's effectiveness over time, future network-based detection algorithms can be easily incorporated into the NMD. At the IP/TCP/UDP layer, the detector might find a device scanning other IP addresses or ports. A high scan rate indicates a worm trying to spread to new devices, such as the iKee.B worm which targeted jailbroken iPhones [49]. Often port scans look for specific open ports running services with vulnerabilities. Other types of network layer detection might involve using the amount of data or connections to detect Denial of Service (DoS) attempts, including DoS attempts on the mobility network itself [42]. The NMD can also maintain an IP blacklist, obtained from 3rd party sources, network based detection algorithms and internal observations, to identify malicious connections from devices to such servers. This often indicates infection, e.g., a bot trying to connect to its command and control server or a malicious application that tries to send stolen information back to the attack server.

Apart from detection at the network layer, detection can be incorporated by scanning headers of higher level protocols contained within the network traffic, such as DNS. With DNS information, the network detector can identify malicious connections to blacklisted domains. In addition to scanning higher level protocol headers, the detector can also perform deep packet inspection on suspicious traffic to match for known malware signatures. This approach can be used selectively as it increases the storage load on the server significantly.

The NMD logs all alerts into the database. The alert contains the IP address of the device and the time at which the flow was recorded to be malicious.

Specialized detectors. Network providers typically rely on specialized detectors for certain services that are outside their area of expertise. In Triton, the NMD employs specialized cloud-based detectors for a more comprehensive, time-consuming analysis of unknown URLs and apps seen in the network. When a user downloads an unknown app or visits an unknown URL, the Filter optimistically grants access to the resource, while initiating a URL/app scanning request to the cloud service in parallel. Though some simple, signature-based checks can be done to determine if an application or URL is already previously known to be malicious, scanning applications and URLs for previously unknown threats using dynamic or behavioral analysis can detect malware variants for which no signature exists yet. Analysis results returned by the cloud service are fed back into the NMD to enhance its algorithms and to prevent future accesses to the malicious URLs/apps. If a device gets infected before the results are received, an alert is generated by the NMD, and the mitigation steps to contain the effects of the malware are executed by the MiE and THC working together in tandem.

SMS Malware Detector. If the cloud-based analysis reports malware sending premium SMS messages or messages that have been previously identified as spam [3], the message or premium number is added to an SMS blacklist within our database. The SMS Malware Detector, running on the SMSC has access to this database and can match all outgoing SMS messages with this blacklist and generate an SMS alert.

3.1.4 Filter

The Filter component blocks access to malicious content/connections e.g., it can check for future accesses to URLs or apps that are flagged as malicious. In such a case, the Filter component drops the request and returns a stub page to the user, which informs him of the website being malicious. The Filter component can also filter generic netflows identified as malicious, such as communication with blacklisted IPs or domain names.

3.1.5 Mitigation Engine (MiE)

The MiE processes the alerts generated by the NMD. An alert contains the IP address of the infected device and the time at which the alert was generated for the given network flow. The MiE has to first identify the correct device based on this information as the device might have disconnected from the network or might have acquired a new IP address. The MiE first obtains the IMSI/IMEI of the device that the IP address belonged to at the time the alert was generated. This information can be obtained by correlating the time for which the PDP context was valid and had the aforementioned IP address. From the IMSI/IMEI number, it checks if the device is connected to the network and has a valid PDP context. It obtains the new IP address of the device in this case and makes a connection to the THC on the device. If the device is offline, this alert is ignored and processed later when the device connects back to the network.

With the valid IP address, the MiE connects to the THC on a special TCP port. The THC listens on this special port for commands from the MiE and receives threat profiles in order to identify malicious apps.

3.2 Network-Device Communication

When a device is uninfected, the network passively scans for signs of malicious traffic generated by the device. Communication is initiated by the MiE, only when signs of infection are found via the NMD. In all cases, the MiE initiates the connection to the THC on the device. We assume that the MiE and the THC have a unique preshared key that they use for secure communication. The preshared key can be securely stored on the device and distributed along with the THC.

3.2.1 Threat profile communication

The MiE identifies the infected device in response to an alert generated by the network detector or as a result of offline analysis. For example, the NMD might have identified that the device is infected because it found the device initiating an HTTP connection to a bot controller. After establishing a secure channel with the THC on the device, the MiE sends a threat profile to it. Figure 4 shows an example threat profile generated by the MiE for a device infected with the DroidDream malware [7]. In this case, the network identifies a device which has contacted a blacklisted IP of the botnet command and control server. The profile includes the details of the malicious flow as identified by the network, which includes the source and destination IP and port pairs, the protocol and the time the connection was made. This profile also commands the THC to terminate the application that generated the malicious flow.

The host component also maintains a log of network connections from the device. This log comprises of the eight tuple <Source IP, Dest IP, Source Port, Dest Port, Protocol, Application, Start Time, End Time>. Compared to the network, the host component additionally stores the application identity that is sending/receiving packets. By matching the tuples with the threat profile, the host component is able to find the application that generated the malicious flow.

The MiE responds with possible remediation actions to be performed depending on the severity of the threat. If the flow, identified by the network, is not found in the log, the THC stores this information in a watch list and looks for future matching packets that might be sent from the device. For example, if a bot on the device initiated HTTP connections to its controlling server, this event is likely to repeat within the near future and will be detected at that point in time.

```
<threat>
  <description>
      Blacklisted IP Connection
   </description>
  <profile>
    <flow>
      <sip>172.16.23.68</sip>
      <sport>42965</sport>
      <dip>184.105.245.17</dip>
      <dport>8080</dport>
      <protocol>TCP</protocol>
      <time>2011-08-01 15:52:35</time>
     </flow>
    </profile>
    <mitigation action="kill_app"/>
 </threat>
```

Figure 4 Threat profile for the DroidDream app connecting the botnet server

3.2.2 Mitigation Actions

Mitigation actions can be explicitly requested by the MiE or performed by the host component after receiving infection information. In both cases, the mitigation actions effect only the malware program running on the device without hampering the user from using other applications or functionality. In some cases, where the network has enough information about an ongoing threat, it can request the THC to perform prevention in order to contain the threat. For example, if Triton has identified both the network activity and application of a fast spreading worm, it might preemptively share this threat profile with other devices before they are infected, thus preventing infection.

Android smart phones today allow Google to remotely send commands to either install or remove applications from their smart phones via their GTalk service. Google has been using this feature to revoke applications found to be malicious or violating their Android market developer distribution agreement or content policy [13, 5]. Our architecture on the other hand allows the mobility carrier to invoke such functionality on all types of heterogeneous devices that are allowed to connect to its mobility network, upon discovering that those devices are infected. As opposed to the kill switch that Google can invoke to remove applications only downloaded from the Android market [5], Triton can protect end users from malicious apps that might have been installed from other alternative app stores, malware that gets installed as a result of a drive by download, malware that installs with the user permission as a result of email or SMS/MMS spam or other types of malicious attacks, such as worms, etc.

3.3 Implementation

We have a fully operational 3G UMTS instance that implements the Triton architecture. Below, we describe the implementation details of prototyping Triton within our 3G UMTS lab network.

3.3.1 Network components

We use the OpenGGSN software running on a Linux server as the GGSN node [12]. The OpenGGSN node interfaces with the SGSN within our 3G wireless lab. We modified OpenGGSN to log information about PDP contexts assigned to the mobile devices within the mobility network into a MySQL database that runs on the same server. This information consists of the device identity, i.e. the IMSI/IMEI numbers and its corresponding IP address for a valid PDP context time duration. The PI, NMD, MiE and the Filter are all applications that run on the same Linux server and can query the MySQL database.

3.3.2 Trusted component on Android

On Android, the THC runs within the Android kernel, implemented as a kernel module as shown in Figure 3(b). We use this architecture to demonstrate the implementation on smart phones as they exist today. Because the THC runs within the kernel, it can only provide complete protection against attacks that operate in user space, which is largely the case for smart phone malware today. This implementation does not take into account attacks that already have obtained kernel level control, such as kernel-level rootkits. We defer discussion about how malware might subvert the host component if it obtains kernel level control and our countermeasures against such attacks to Section 5.1.

The THC obtains information about Android applications by intercepting system calls. It intercepts all socket related calls to create a log of all TCP and UDP traffic that originates from Android applications. It buffers and logs network activity within a small rolling log. This log is comprised of the network end point information, such as source and destination IP addresses and port numbers, the application that generated or received the traffic and the start and end times of the network flow. The application name that sends traffic on a specific socket is obtained by internally walking the various kernel data structures. A significant amount of malware on Android also generates malicious SMS messages. To keep track of SMS messages, the THC also intercepts all writes to the modem file descriptor and looks for AT commands. By using this technique, it also logs all SMS messages that are generated by Android applications.

To receive threat profiles from the network MiE, the THC runs a TCP server within kernel space as a kernel thread. This thread listens to commands from the network component and can decode the threat profiles that it receives. Upon receiving a threat profile, it decodes the malicious flow, refers to its log to find the application that generated the flow and kills the application by posting a kill signal to the appropriate process. The application is further blacklisted and prevented from running on the device thereafter.

3.3.3 Trusted component on Xen/Linux

Current mobile virtualization solutions are limited in their availability and cannot be installed on any commercially available devices today. To demonstrate how Triton works with virtualizable platforms, which are expected to be available on smart phones in the near future [11, 17, 35], we built our prototype on the Viliv S5 mobile device due to its functional equivalence of a smart phone. The Viliv S5 is equipped with an Intel Atom Z520 1.33 GHz processor and runs the Xen VMM [27] to achieve isolation between the user VM and the THC executing within a privileged VM, both running the Linux operating system as shown in Figure 3(a).

The THC inside the privileged VM must identify network activity generated by the user VM and identify the application that generated this traffic. By using a hypervisor-based system, the THC can remain isolated and secure even if the user VM's kernel is compromised by malware. Because we assume the kernel in the user VM can be compromised, the THC does not rely on any information within the user VM kernel and therefore must bridge the semantic gap between the hypervisor and the user VM.

In the hypervisor, we add functionality for intercepting and forwarding process and networking related system calls to the THC in the privileged VM. We use a technique published in [19], to pass all system calls to the hypervisor. Relevant system calls are forwarded to the THC via shared memory pages, which are accessible to both the hypervisor and the THC. The THC waits on a Xen event channel in order to be notified when new information arrives on the shared pages. Below, we describe how this implementation tracks applications running on the system and how it accounts for the network traffic that they generate.

Process tracking. In Linux, processes are executed using the execve system call. The execve system call takes an argument, which is the full path name of the application to be executed. The THC obtains the name of the application executing by using this information from the hypervisor. To identify the process currently running, without relying on the kernel data structures of the user VM, we rely on the characteristics of the x86 memory management system.

In the x86 architecture, each process is separated within its own virtual address space using multi-level pages tables. Each address space is indexed by a high level page directory table, which is stored in the process control register (CR3) during process execution and can be used to uniquely identify a process [39, 18]. When a process is created using the the execve system call, we capture the address of the page directory and pass the address on to the THC. This gives us a mapping between a binary name and the current process running, identified by the CR3 value.

Network activity tracking. When an application generates network ac tivity, it makes a sequence socketcall system calls. We trace various socket calls, such as connect and sendto, and pass their arguments to the THC in order to log network activity of the user VM. Since these system calls occur within the context of the currently executing process, the CR3 register is also passed to the THC in order to identify the application which caused this network activity. This generates a mapping between the application and the network activity that it is involved in.

Mitigation. Because killing a process relies on various kernel data structures of the user VM, the THC cannot directly kill a running application. Instead, once a malicious application has been identified, the binary name and any currently executing CR3 values corresponding to this binary are blacklisted. If the malicious application attempts to make any further socket system calls, the hypervisor returns an error resulting in a system call failure. This forces the kernel to return an error, effectively denying service to the application.

4 Evaluation

In this section, we describe the experimental setup and the specific experi ments that we used to illustrate the defense-in-depth approach of our prototype Triton.

4.1 Experimental setup

Mobile devices equipped with specialized SIM cards can connect to the base station in our 3G wireless lab. The devices that we use in our experiments are the HTC Aria smart phone and the Viliv S5. The HTC Aria phone is equipped with a Qualcomm MSM 7227 Chipset, a ARMv6 600 MHz processor, and 291 MB of memory. We run the CyanogenMod 7 distribution [4] and Android version 2.3.4, API version 10 on the phone. The trusted component runs as a kernel module on this phone. The Viliv S5 is equipped with an Intel Atom Z520 1.33 GHz processor, 4.8” touch screen, 32GB hard drive, 1GB of memory, WiFi, Bluetooth, a 3G modem, and GPS. We use the Xen 4.0.1 hypervisor in paravirtualized mode. The privileged VM runs Fedora 12 with Linux 2.6.27.42 patched for Xen dom0 support while the User VM runs CentOS 5.5 with Linux 2.6.27.5 including Xen paravirtualization support.

4.2 Malware Prevention

We describe here the experimental setup that we have to prevent mobile devices from downloading known bad content and apps. Once resources are identified as malicious, future access is blocked, effectively protecting end users from downloading malicious content.

4.2.1 Behavioral Scanning of Apps and URLs

The Triton architecture uses a cloud-based malware analysis service to behaviorally scan applications and URLs offline. When a user downloads an application through an app store and/or has an application transmitted to his device, the download request URL, containing an application identifier, is observed by the packet inspector in the network. Triton uses this identifier to download the application through the normal app store channel and does not expose the user that has downloaded the application in any way. Due to privacy concerns, we do not rely on any sensitive information of the user and do not obtain applications from user devices. Triton currently only intercepts apps downloaded from the official Android market, but can be easily extended to include other app stores as well.

If an app has never been downloaded before or its status has expired, either the application identifier or application itself is sent to the cloud-based service for static and behavioral analysis. The static analysis phase involves quick checks against previously known malware signatures and identifies an application's possible characteristics based on permissions, API calls and app metadata. On the other hand, behavioral analysis of the application is conducted by emulating simulated user actions within a mobile emulator sandbox to determine what the application does. The analysis platform records various characteristics of an application such as system calls, bandwidth utilization and connections to domains/IPs. If the application sends SMS spam, attempts to gain root access, connects to a bot master, etc., the application is flagged as malware. Network characteristics, such as command and control servers or SMS spam messages, are logged in the database for use in the NMD.

In order to trigger malicious features of an app, user behavior is dynamically generated while the app is running. Depending on the complexity of the malware, multiple levels of scanning may be needed such as behavior after reboot, directed user-like behavior, and random aggressive crash testing. Though some apps may take up to 2 hours to analyze, in general we found that most malicious apps currently trigger their malicious functionality right after execution, in which case will be detected and generate a network signature within three minutes. The app analysis platform currently incurs a false positive rate of 1 in 10,000 and is continuously improving over time.

The Filter blocks an app download if the app was previously flagged as malicious. If the app has been downloaded before and was reported as benign, no new request is sent to the cloud service and the app can be downloaded as normal.

Similarly, when a user visits a URL on the mobile phone's browser, the URL is forwarded to the cloud service to scan for drive-by-downloads that might be initiated by that URL. The cloud service uses its malware analysis platform to conduct both static and behavioral checks to determine what the URL does, and the URL is flagged if it attempts to conduct a drive- by-download or is identified to be a phishing URL. The malware analysis platform used for URL detection had a low false positive rate, approximately 1 in 10,000,000. A similar caching mechanism with a set timeout is used to avoid redundant requests to the cloud service.

4.2.2 Scan Frequency and Caching

In order to put an upper bound on the number of users that can be infected in between scans of a resource, the frequency of scanning is chosen to be proportional to the frequency of accesses. Such an approach allows for an elegant trade-off between the scanning cost and level of security desired. If a resource is very popular, it is scanned more frequently as an infection of that resource could affect many users quickly, whereas if a resource is less popular, the cost of scanning it is kept in proportion with its low usage.

In our experiments, we send every new request that has not been refreshed within the past four hours to the cloud service for scanning. Upon the first visit of a malicious website or attempt to download a malicious app, we found that subsequent users are protected from visiting the website by the Filter component, which blocks access to the URLs found to be malicious.

4.3 Malware Detection and Mitigation

This set of experiments illustrates how Triton effectively detects and mitigates malware on the end mobile device. In all the experiments, we run the malware on the device and the network detector generates an alert as soon as it detects the first malicious flow. The MiE generates the threat profile and sends it to the THC on the device, which accurately identifies the malicious application and stops it from executing on the device. It is further blacklisted and prevented from running in the future.

4.3.1 Android malware

Table 1 shows the malicious apps tested, some of which were uploaded into the official Android market and were downloaded by users. We use these in our experiments as they form a good representative set of malware that exists today for Android phones. Column 1 shows the malware name. Column 2

Table 1 This table shows the malware that we use to run experiments on the HTC Aria smartphone running Android. All attacks were successfully detected by Triton and the malware was disabled from executing on the device.

Malware Name/Android Package Name	**Malware Type**	**Malicious Network Behavior**
Trojan: Android/Geinimi.A	Trojan, Bot	Connects to several malicious domains
Exploit:Linux/DroidRooter.A	Trojan, Bot	Connects to a malicious IP address
Trojan:Android/Twalktupi.A	Trojan, SMS spam	Connects to a malicious domain Sends spam SMS
Android.Trojan.Bgserv.A	Trojan, SMS spam	Sends SMS
Android.Trojan.FakePlayer.A	Trojan	Sends SMS to premium number
Trojan:Android/HippoSms.A	Trojan	Connects to malicious domains Sends SMS to premium numbers
Android.Trojan.GGTracker.A	Trojan	Sends SMS to premium number
Golddream.A	Trojan, Spyware	Connects to a malicious domain
Plankton	Spyware	Connects to a malicious domain

shows the type of malware that we obtained from our forensic analysis and documentation publicly available about the malware. Column 3 shows the network behavior of the application that was flagged as malicious.

All the malware samples were detectable in the network because they performed one or more of the following actions - (a) contacted malicious IP addresses (b) contacted malicious domains (c) Sent SMS spam or (d) Sent SMS to premium numbers.

The Trojan:Android/Geinimi.A is a repackaged sex positions application that connects to a botnet. The Exploit:Linux/DroidRooter.A, also popularly known as DroidDream, is a repackaged version of a bowling game application that also connects to a botnet and awaits additional commands.

The Trojan:Android/Twalktupi.A, popularly known as the "Walk&Text" application, sends SMS spam to all the contacts available in the contact list of the infected device. Android.Trojan.Bgserv.A also sends SMS spam. Android.Trojan.FakePlayer.A, Trojan:Android/Hippo-Sms.A and Android.Trojan.GGTracker.A all send SMS messages to premium numbers.

Plankton and GoldDream.A are spyware applications that connect to malicious domains and leak sensitive information, such as IMEI numbers. Gold- Dream.A logs incoming SMS messages to a local file and sends this over to the attacker. Plankton 1 was added to about ten other applications on the official Android market from three different developers. Its stealthy design also explains why some earlier variants have resided on the market for more than two months without being detected by current mobile anti-virus software [10]. Since Plankton[1] runs a background service that connects to a remote server, identified by a scanning request to the cloud service, all apps that include Plankton would be automatically detected by the NMD and disabled via the THC.

Triton successfully detected and mitigated all malware attacks on the end device.

4.3.2 Linux malware

On Linux, we chose malware that had some network footprint, such as worms, malware that launches Denial of Service (DoS) attacks, and bot software. Although the Linux platform is not a popular target for malware writers, we use real Linux samples and find that they do illustrate the capabilities of Triton.

Table 2 shows the representative samples that we used in our experiments. Column 1 shows the name of the malware. Column 2 shows the type and Column 3 shows the network behavior that the Malware Detector used to flag the flow as malicious and generate an alert. For IP and port scanners, the Malware Detector detects them based on the fact that the number of connections to a diverse set of IP addresses/ports exceeds a set threshold. Blacklisted IP addresses and ports are detected in a similar fashion as the Android malware, where the blacklists are constantly updated from third party cloud services doing malware analysis. DoS attempts are detected when malware tries to open connections to a given IP address or a set of IP addresses within a specified period of time that exceeds the normal threshold. In all the experiments with Linux malware from the set above, the malware detector successfully detected the malicious flows and the malware on the device was correctly identified from the threat profile generated by the MiE. The malware was subsequently disabled and blacklisted from running on the device in the future.

4.4 Performance

Below, we discuss the response time and the performance overhead of Triton.

[1]Prof. Jiang's research group at NC State Univ identified Plankton as spyware, but the authors of Plankton claim to be a legitimate ad network. Google suspended Plankton from its Android Market.

Table 2 This table shows the malware that we use to run experiments on the ViliV S5 running Linux inside a Xen VMM. The network behavior was flagged as malicious, and the malware was disabled from executing on the device.

Malware Name	Malware Type	Malicious Network Behavior
Trojan-DDoS.Linux.Fork	Bot	Connects to a blacklisted destination server and port number
Trojan-Spy.Linux.XKeyLogger.b	Port scanner, Keylogger	Conducts high rate TCP port scan
Net-Worm.Linux.Cheese	Worm	Connects to random IP addresses on a specific port
Net-Worm.Linux.Mworm.a	Worm	IP/Port scan over specific IP ranges
Trojan-DDoS.Linux.Blowfish	Worm	Connects to a blacklisted port
DoS.Linux.Arang	DoS	Creates a denial of service attacks to a specified victim
FTP AnoScan	Port Scanner	Scans for open FTP ports
FTPNullSearch02	Port Scanner	Scans for open FTP ports
Flooder.Linux.Alcohol.a	DoS	Creates a denial of service attacks to a specified victim

4.4.1 Response time

Automated mitigation has the potential to protect many users as well as a mobility provider's network from the negative effects of malware. For instance, in the DroidDream attack that took place in March 2011 in which over 260,000 users had downloaded malware to their phones, automated mitigation could have protected most users.

DroidDream was injected into re-packaged versions of over 50 of the most popular applications on Google's Android Market, and once downloaded, the malware shipped the user's IMSI/IMEI numbers and other personally identifiable information (PII) to a bot master. The attack was first noticed on March 1, 2011 [7], and Google started rolling out a fix in the form of the Android Market Security Tool on March 5, 2011. While data is not available regarding the download rate of DroidDream applications, by assuming a uniform download rate we can do a back-of-the-envelope calculation as to how many users could have been protected using automated mitigation.

As per the forensics for DroidDream, identified by Triton as shown in Figure 4, an infected phone can be identified by a connection to the DroidDream bot master, 184.105.245.17. Once DroidDream and its bot master are identified as malicious, Triton can contain malicious effects on all phones which download DroidDream thereafter. In particular, the mitigation engine would instruct the trusted component on the phone to kill the DroidDream process, and firewall off communication with the bot master. If 260,000 phones were infected by March 5, and the infections started on March 1, then approximately 2,200 phones were infected per hour. If a threat profile were to have been deployed by the mitigation engine within the first hour, then 99.2% of attempted DroidDream infections that occurred afterwards would not result in PII leakage or compromise of the phone. In addition, for the 0.8% of phones that were already infected, the DroidDream process could be killed and communication with the bot master could be cut off to prevent further damage.

4.4.2 Performance overhead

In this section, we run the following three workloads to record the overhead generated by running the THC on the device.

Browsing Workload. We use an automatic browsing script to measure our overhead against a typical mobile browsing experience. During the experiments we visited google.com, gmail.com with an account opened, and cnn.com. The script also watches a 60 second YouTube clip, within the browser on Linux and within the YouTube app on Android. Lastly the script checks an email account using the Thunderbird application on Linux and the default email application on Android.

To measure the overhead, we execute the workload on each platform both with and without the host component. We measure the time the workload executes using the wall clock time and average the results over five runs of the experiment. The results are summarized in Table 3. The in-kernel implementation on Android incurs a minimal overhead of 1.2%, while the VMM based implementation incurs a 9% overhead. The higher overhead is caused by the increased complexity of the hypervisor-based host component compared to the in-kernel version.

File Download. Mobile device users frequently download small files, such as music MP3s or applications from various app stores. To measure the overhead of a similar file download, we downloaded a 5 megabyte file from [21]. We performed this experiment on both the Linux and Android platforms by using the wget command. Because the file download initiates a single TCP connection we observe minimal overhead and report the download time overhead for both the kernel and hypervisor-based host component as 1.7% and 0.7% respectively. In each case the overhead is minimal and within the standard deviation of the workload.

CPU Intensive Workload. For completeness we include a CPU intensive workload for the hypervisor-based component. Because we trap every system call within the hypervisor, we wanted to observe the overhead of the system

Table 3 This table shows the performance overhead recorded for three different workloads resembling user actions on the device with and without the trusted host component (THC)

	Android THC		Linux THC	
Workload	Without THC	With THC	Without THC	With THC
Web Browsing	150.2 + –1.6*s*	152 + –2*s*(1.2%)	198.2 + –2.9*s*	216.4 + –8*s*(9.2%)
File download	114 + –0.7*s*	114.2 + –1.6*s*(1.7%)	114.4 + –3.2*s*	115.2 + –2.9*s*(0.7%)
LMBench	N/A	N/A	198.3 + –0.6*s*	201.3 + –2*s*(1.5%)

in general. We chose a CPU intensive workload designed to measure OS performance called lmbench [44]. Lmbench exercises multiple OS interfaces and calls numerous system calls. We report an average overhead of (1.5%) and a standard deviation within both experiments, with and without the host component.

5 Discussion

In this section, we discuss counter attacks, scalability issues and the limitations of our approach.

5.1 Subverting the Host Component

The attacker might be able to subvert the host component in one of the following ways:

Root the device. On non-virtualized platforms, such as smart phones available today, the trusted host component runs as part of the operating system kernel. Several instances of malware on the Android platform have been known to root the device and install a rootkit [13], potentially compromising the integrity of our host component. We recommend that the ultimate deployment of this architecture only be made on platforms that support our virtualization-based trusted host component.

Send commands via 3G. An attacker may try to subvert the trusted host component when it is connected to the Internet via 3G/4G. For example, by sending commands to the trusted component, an attacker might attempt to remove their app from the blacklist or disable other benign apps for malicious intent. This involves the attacker sending commands to the special port that the trusted component on the device listens to. Apart from the fact that the attacker does not possess the keys for communicating with the device, the GGSN can be configured to block all communication to this port from other devices on the Internet as well as devices within the mobility network itself. The only communication that happens on this secure port is between the Mitigation Engine and the device.

Parasitic Malware. In some cases, malware can inject itself into other benign system processes. This is an especially popular technique used with malware that infects PCs running the Windows operating system [50]. In such cases, the trusted host component will identify the infected system process as malicious and kill it. While it is not clear if malware on mobile devices are following a similar trend, the host component can be trivially extended to identify the correct malware program by using techniques discussed in [50].

5.2 Scalability

When deployed in a real network, Triton should be able to handle millions of devices and traffic sent by them. While Triton is a prototype, it can lever age several well-known techniques to handle scale in a real network. The packet inspector can be a passive sniffer that can sniff traffic at very high speeds [25–26, 38]. Databases can handle queries and very fast lookups of terabytes of data [30, 22]. The workload of other components, such as the mitigation engine, the filter and the malware detectors can be split by using large clusters of machines and load balancing techniques already used in large data environments.

5.3 WiFi Offload

Due to both data cap and bandwidth limitations, mobile device users typically offload traffic to WiFi hotspots when available. Though Triton's network component does not have visibility into traffic that is offloaded via WiFi, each device still logs its own network activity in the host component. Since there are still millions of other customers currently connected to the mobility network at any given time, Triton's network detection algorithms maintain their global visibility and effectiveness. If a significant threat is detected, Triton can proactively communicate with devices that are currently connected to WiFi via an out of band channel, such as SMS or WAP push, to ensure that the device is not effected by the threat.

5.4 Limitations

The approach that we propose offloads detection to the network, while the host component does much of the mitigation and prevention. This is well suited to attacks that manifest themselves in the network, which the network provider has a good view of, such as, botnet command and control patterns, spread of worms, DDoS activities, malicious applications phoning home and so on. These activities are generally characteristic to mobile malware where attack activity is driven by economic incentives. However, malware can be designed to be largely focused on the device and operate without a network footprint. Our approach cannot detect malware that is purely device centric. In such cases, we assume the presence of other anti-virus software on the device that can detect such malware.

With network level detection techniques catching up, malware can become more stealthy and piggyback on genuine network traffic for communication with its controlling servers or spread to other devices. Stealthy malware is a challenge to detect in the network and would be challenging for our approach as well.

6 Related Work

We can classify related work in three main categories: network-based detec tion techniques and host based detection techniques that we leverage, and work that uses a combination of network and host based techniques.

There is a large body of work on network-based anomaly detection. Recent work has focused on detecting botnet command and control communication patterns based on protocols and heuristics [40], presence of a botnet infection cycle visible from the network [32], spatio-temporal correlations in network traffic [33] and clustering of network traffic [31] for detection of botnets. A similar body of work exists for other types of malware such as worms [52–53, 41], exploit code inside network flows [23], etc. Triton relies on efficient network-based detection techniques and can leverage them to improve its detection accuracy and efficacy.

We leverage techniques for securely inspecting the state of a user VM from a privileged VM as discussed in prior work [29, 18, 48]. Many VMM- based techniques have been proposed to detect malware running inside the user VM [43, 37]. Triton uses similar inspection techniques to find a malicious application but relies on network-based detectors to flag malicious traffic.

Zeng et al. developed a technique to improve the accuracy of botnet detection by using additional information on PCs, such as CPU and memory, via a component installed on the PC [54]. Srivastava et al. used a detector in the network to identify anomalous traffic, while a host component attributed this traffic to identify parasitic malware on the end device [50]. Our work differs in several ways from the above two works. We focus primarily on mobile devices as opposed to devices on the wireline network. We propose a defense in-depth approach that can detect, prevent and contain malware as opposed to only detecting it. The Triton architecture is designed to be integrated in a real mobility network and uses the on-host component only to prevent or mitigate an infection and therefore operates with a very low overhead on the mobile device.

Airmid [45] proposes a similar idea of remote repair where a network and host component act in tandem to recover from mobile malware infections. At the high level, our work shares the same insight but differs in the following ways: (a) We have a complete end to end working instance on a real 3G UMTS network where we address the design and implementation challenges encountered in realizing the architecture within the carrier. Airmid simulates the carrier side and assumes the existence of a server that can handle malicious traffic originating from mobile devices. (b) We report performance overheads on mobile devices supporting two different architectures - an inkernel component on HTC Aria phones and VMM based architecture running Xen on Viliv S5 devices. Airmid on the other hand only considers an in-kernel implementation. (c) We perform experiments with real mobile malware sam ples that have posed real threats on the Android market and show that our architecture can automatically identify malware and recover the end devices from malware infections. Airmid on the other hand uses prototype malware samples. Finally, Triton is built to interoperate with existing carrier equipment and does not require any changes within the carrier network elements, and therefore lends itself to easy adoption by carriers.

7 Conclusions

In this paper, we have described Triton, a new architecture for detection and containment of mobile malware. Triton employs a defense-in-depth approach where in-the-network malware detectors identify and prevent the spread of malware and communicate the threat to an on-the-phone trusted software component to identify and neutralize malware on the device. This allows mobility service providers to quickly respond to ongoing malware threats and contain malware on mobile devices, even in the absence of an anti-virus signature.

We reported on our experience and learnings from design and implementation of Triton in our 3G wireless lab for its server-side infrastructure components, and two prototype implementations for its client-side components namely, a kernel-level implementation on Android smart-phones, and a

VMM-based implementation on Linux Viliv devices. The prototype implementation of Triton successfully achieved infection prevention, effective detection, immediate containment, and fine-grained response on a diverse, representative set of real Android and Linux malware with a very low performance overhead. While our research described in this paper suggests that Triton has the potential to mitigate a large majority of mobile malware infections, we also discussed the potential counter attacks, scalability, and limitations of the Triton architecture.

Acknowledgements

We thank Tufan Demir, Gerry Eisenhaur and Mike Gagnon for their help with mobile malware analysis. We also thank Gustavo de los Reyes, Alex Bobotek and Dan Boneh for their insightful comments and feedback on the paper.

References

[1] Android and security. http://googlemobile.blogspot.com/2012/02/android-and-security.html.
[2] The bro network security monitor. http://bro-ids.org/.
[3] Cloudmark mobile platform. http://www.cloudmark.com/en/products/cloudmark-mobile-platform/how-it-works.
[4] Cyanogenmod wiki. http://wiki.cyanogenmod.com/index.php.
[5] Exercising our remote application removal feature. http://android-developers.blogspot.com/2010/06/exercising-our-remote-application.html.
[6] F-secure mobile threat report q2 2012.http://www.f-secure.com/weblog/archives/MobileThreatReport_Q2_2012.pdf.
[7] Infected apps in the android market. http://goo.gl/cFNiX.
[8] iphone security bug lets innocent-looking apps go bad. http://goo.gl/MMA2b.
[9] Lookout unveils the mobile threat network; verizon the first to adopt lookout api. http://goo.gl/bRxLt.
[10] New stealthy android spyware – plankton – found in official android market. http://www.csc.ncsu.edu/faculty/jiang/Plankton/.
[11] Okl4 microvisor.http://www.ok-labs.com/products/okl4-microvisor.
[12] Openggsn. http://sourceforge.net/projects/ggsn/.
[13] Remote kill and install on google android. http://jon.oberheide.org/blog/2010/06/25/remote-kill-and-install-on-google-android/.

[14] Security alert: Spamsoldier. https://blog.lookout.com/blog/2012/12/17/security-alert-spamsoldier/.
[15] Snort. http://www.snort.org/.
[16] Universal mobile telecommunications system. http://en.wikipedia.org/wiki/Universal_Mobile_Telecommunications_System.
[17] Vmware mobile virtualization platform. http://www.vmware.com/products/mobile/overview.html.
[18] A. Baliga, L. Iftode, and X. Chen. Automated Containment of Rootkit Attacks. In *Elsevier, Computers & Security*, volume 27, 2008.
[19] F. Beck and O. Festor. Syscall Interception in Xen Hypervisor. In *MADYNES Technical Report*, 2009.
[20] J. Bickford, H. Lagar-Cavilla, A. Varshavsky, V. Ganapthy, and L. Iftode. Security versus Energy Tradeoffs in Host-Based Mobile Malware Detection. In *Proc. 9th Conference on Mobile Systems, Applications and Services*, 2011.
[21] Think Broadband. Download test files. http://www.thinkbroadband.com/download.html.
[22] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A Distributed Storage System for Structured Data. In *Proc. 7th USENIX Symposium on Operating Systems Design and Implementation*, 2006.
[23] R. Chinchani and E. van den Berg. A Fast Static Analysis Approach to Detect Exploit Code Inside Network Flows. In *Proc. 8th Symposium on Recent Advances in Intrusion Detection*, 2005.
[24] M. Christodorescu. Behavior-based Malware Detection. In *University of Wisconsin-Madison*, August 2007.
[25] C. Cranor, Y. Gao, T. Johnson, V. Shkapenyuk, and O. Spatscheck. Gigascope: High Performance Network Monitoring with an SQL Interface. In *Proc. Conference on Management of Data*, 2002.
[26] C. Cranor, T. Johnson, O. Spataschek, and V. Shkapenyuk. Gigascope: A Stream Database for Network Applications. In *Proc. Conference on Management of Data*, 2003.
[27] B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, I. Pratt, A. Warfield, P. Barham, and R. Neugebauer. Xen and the art of virtualization. In *Proc. 19th ACM Symposium on Operating Systems Principles*, 2003.
[28] W. Enck, P. Traynor, P. McDaniel, and T. La Porta. Exploiting Open Functionality in SMS-capable Cellular Networks. In *Proc. 12th Conference on Computer and Communications Security*, 2005.

[29] T. Garfinkel and M. Rosenblum. A Virtual Machine Introspection Based Architecture for Intrusion Detection. In *Proc. 10th Network and Distributed Systems Security Symposium*, 2003.

[30] R. Greer. Daytona and the Fourth-Generation Language Cymbal. In *Proc. Conference on Management of Data*, 1999.

[31] G. Gu, R. Perdisci, J. Zhang, and W. Lee. BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-Independent Botnet Detection. In *Proc. 17th USENIX Security Symposium*, 2008.

[32] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. BotHunter: Detecting Malware Infection Through IDS-Driven Dialog Correlation. In *Proc. 16th USENIX Security Symposium*, 2007.

[33] G. Gu, J. Zhang, and W. Lee. BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic. In *Proc. 15th Network and Distributed System Security Symposium*, 2008.

[34] K. W. Hamlen, V. Mohan, M. M. Masud, L. Khan, and B. Thuraisingham. Exploiting an Antivirus Interface. *Computer Standards and Interfaces*, November 2009.

[35] J. Hwang, S. Suh, S. Heo, C. Park, J. Ryu, S. Park, and C. Kim. Xen on ARM: System Virtualization using Xen Hypervisor for ARM-based Secure Mobile Phones. In *Proc. 5th IEEE Consumer Communications and Networking Conference*, 2008.

[36] X. Jiang. An evaluation of the application verification service in android 4.2. http://www.csc.ncsu.edu/faculty/jiang/appverify/.

[37] X. Jiang, X. Wang, and D. Xu. Stealthy Malware Detection Through VMM-based "Out- of-the-Box" Semantic View Reconstruction. In *Proc. 14th Conference on Computers and Communications Security*, 2007.

[38] T. Johnson, S. M. Muthukrishnan, V. Shkapenyuk, and O. Spatscheck. Query-aware Partitioning for Monitoring Massive Network Data Streams. In *Proc. Conference on Management of Data*, 2008.

[39] S. T. Jones, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Antfarm: Tracking Processes in a Virtual Machine Environment. In *In Proc. USENIX Annual Technical Conference*, 2006.

[40] A. Karasaridis, B. Rexroad, and D. Hoeflin. Wide-scale Botnet Detection and Characterization. In *Proc. 1st Workshop on Hot Topics in Understanding Botnets*, 2007.

[41] H. Kim and B. Karp. Autograph: Toward Automated, Distributed Worm Signature Detection. In *Proc. 13th USENIX Security Symposium*, 2004.

[42] P. P. C. Lee, T. Bu, and T. Woo. On the Detection of Signaling DoS Attacks on 3G Wireless Networks. In *Proc. 26th IEEE Conference on Computer Communications*, 2007.

[43] L. Litty, H. A. Lagar-Cavilla, and D. Lie. Hypervisor Support for Identifying Covertly Executing Binaries. In *Proc. 17th USENIX Security Symposium*, 2008.

[44] L. W. McVoy and C. Staelin. lmbench: Portable Tools for Performance Analysis. In *Proc. USENIX Annual Technical Conference*, 1996.

[45] Y. Nadji, J. Giffin, and P. Traynor. Automated Remote Repair for Mobile Malware. In *Proc. 27th Annual Computer Security Applications Conference*, 2011.

[46] J. Oberheide and F. Jahanian. When Mobile is Harder Than Fixed (and Vice Versa): Demystifying Security Challenges in Mobile Environments. In *Proc. 11th Workshop on Mobile Computing Systems and Applications*, 2010.

[47] J. Oberheide and C. Miller. Dissecting Android's Bouncer. In *Summer-Con*, 2012.

[48] B. D. Payne and W. Lee. Secure and Flexible Monitoring of Virtual Machines. In *Proc. 23rd Annual Computer Security Applications Conference*, 2007.

[49] P. Porras, H. Sadi, and V. Yegneswaran. An Analysis of the iKee.B iPhone Botnet. In *Security and Privacy in Mobile Information and Communication Systems*. 2010.

[50] A. Srivastava and J. T. Giffin. Automatic Discovery of Parasitic Malware. In *Proc. 13th Conference on Recent Advances in Intrusion Detection*, 2010.

[51] P. Traynor, M. Lin, M. Ongtang, V. Rao, T. Jaeger, P. McDaniel, and T. La Porta. On Cellular Botnets: Measuring the Impact of Malicious Devices on a Cellular Network Core. In *Proc. 16th Conference on Computer and Communications Security*, 2009.

[52] K. Wang, G. F. Cretu, and S. J. Stolfo. Anomalous Payload-Based Worm Detection and Signature Generation. In *Proc. 8th Symposium on Recent Advances in Intrusion Detection*, 2005.

[53] D. Whyte, E. Kranakis, and P. C. van Oorschot. DNS-based Detection of Scanning Worms in an Enterprise Network. In *Proc. 12th Network and Distributed System Security Symposium*, 2005.

[54] Y. Zeng, X. Hu, and K. G. Shin. Detection of Botnets Using Combined Host-and Network-level Information. In *Proc. 40th Conference on Dependable Systems and Networks*, 2010.

Biographies

Arati Baliga is an Independent Security Consultant and Adjunct Professor at the Department of Computer Science and Engineering at NYU Polytechnic School of Engineering. Her expertise and interests are in mobile, systems and network security. Previously, she was a Security Researcher at the AT&T Security Research Center where she worked on building carrier based defenses against mobile malware.

Arati obtained her Ph.D from the Department of Computer Science at Rutgers University. Her dissertation work on stealth malware detection won the outstanding paper award at the Annual Computer Security and Applications Conference (ACSAC 2008).

Jeffrey Bickford is a member of the AT&T Security Research Center. He is interested in mobile device security with a focus on using virtualization techniques to create a secure and robust mobile platform. His recent work has focused on designing a secure enterprise architecture for protection against APT attacks. Jeff also enjoys investigating real world attacks and analyzing mobile malware samples. He completed his M.S. at the Rutgers University Department of Computer Science.

Neil Daswani is a member of the information security group at Twitter. He was formerly the CTO of Dasient, Inc. prior to its acquisition by Twitter. Before co-founding Dasient, Neil had served in a variety of research, development, teaching, and managerial roles at Google, Stanford University, DoCoMo USA Labs, Yodlee, and Bellcore (now Telcordia Technologies). While at Stanford, Neil co-founded the Stanford Center for Professional Development Software Security Certification Program.

His areas of expertise include security, wireless data technology, and peer-to-peer systems. He has published extensively in these areas, frequently gives talks at industry and academic conferences, and has been granted several U.S. patents. He received a Ph.D. and a master's in computer science from Stanford University, and earned a bachelor's in computer science with honors with distinction from Columbia University.