Global Information Assurance Certification Paper

This is a GIAC Gold Template

Monitoring network traffic for Android devices

GIAC (GCIA) Gold Certification

Author: Angel Alonso Parrizas, parrizas@gmail.com
Advisor: Dominicus Adriyanto

Accepted: 16 January 2013

Abstract

The same principles that organizations use to monitor network traffic go into their networks must be applied to the network traffic originating from mobile devices. This means that the techniques and tools, which would normally be used to collect and analyze network activity, can also be used to detect anomalous network traffic or network intrusions related to smartphones. This paper will therefore outline an architecture model, which can be used to analyze the network communications originating from Android devices and to detect any unusual traffic. As part of the exercise, a set of several tests involving real malware will be executed to gauge the effectiveness of said architecture. In addition to that, the aim of the exercise is to improve the detection mechanisms of the engine by creating new signatures to detect specific threats. Lastly, we will define incident-handling steps, which can be used to combat 0-day malware and known malware for which no signatures exist.

Introduction

In order to detect possible intrusions or any unusual patterns, several techniques have been used in the past. These techniques include packet capture tools and Intrusion Detection Systems (IDS). With the development of mobile platforms like Android, BlackBerry or iOS – to name but few, the concept of Local Area Network has changed immensely. Nowadays, any employee can use a smart phone to connect to his company's network when performing such trivial tasks as checking email or accessing shared resources. As a consequence of this, there are new attack vectors which can be used to compromise corporate infrastructures.

I already touched upon this subject by discussing a number of techniques which could be used to improve the security of Android devices in the paper "Securely Deployed Android Devices" (Alonso Parrizas, 2011). The aforementioned paper explains how to implement network filters for Android devices to enforce corporate security policies.

This paper wil go further and it will demonstrate how to implement a network architecture that allows detailed smartphone traffic monitoring detect an unauthorised connection or a compromised device. The traffic will be captured in real time, thus allowing forensic analysis in case there is an incident.

Preparation and the lab setup

The lab is composed of two main components:

1. The first component is smartphone and HTC desire Z (Vision) is chosen for the demo. This smartphone is connected to GSM network and WiFi.
2. The second component is a Virtual Private Server (VPS) running Ubuntu 12.04 hosted in Amazon. The VPS is a virtual machine serving as a VPN SSL Gateway.

The smartphone uses Android OS and runs a custom ROM, CyanogendMod 7.2.0 (Cyanogenmod, 2012) with a kernel version 2.3.7.

Additionally, there is another machine running Windows 7 OS and it has the Android SDK toolkit installed on it in order to interact with the smartphone.

A set of several additional security tools will be used in the project. OpenVPN (OpenVPN Technologies, 2002), and iptables (Netfilter, 1999) which will be used in the smartphone and the VPS and Snort (Sourcefire, 2001) which will be used in the VPS.

Routing the traffic to the VPS through a VPN tunnel

In order to be able to analyze the traffic flows, we will send all the traffic from the Android device – either via the 3G interface (rmnet0) or the wireless card (eth0), through the VPN tunnel established with OpenVPN, to the VPS. The source of the traffic could either be the 3G interface (rmnet0) or the wireless adapter (eth0). The smartphone will have to establish a VPN tunnel towards the OpenVPN end point which is under our sole control. At the same time, firewall policies are implemented on the smartphone (through iptables) and the policy only permit traffic towards the VPS. An example of the network architecture:

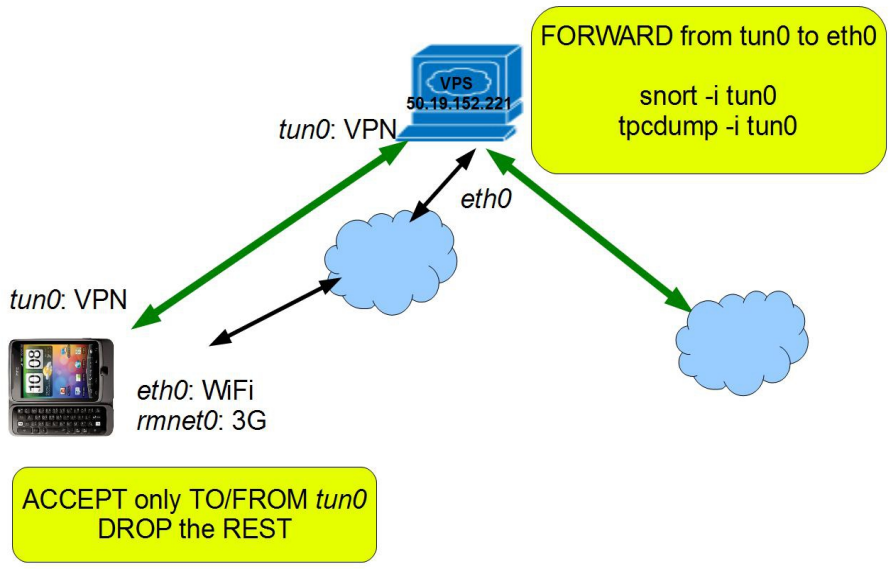

Figure 1: Network connectivity channel

The VPN tunnel uses digital certificates (public/private key pair) to authenticate the client and the server. Using digital certificates instead of a shared key gives higher flexibility, for instance we can revoke access in case if the smartphone is lost. In addition to that, PKI escalates better with a higher number of users.

Building the CA infrastructure and generating certificates is outlined in the OpenVPN manual but for reference, these are the main steps:

Define the variables that will contain the parameters of the certificate (ie. country, email, etc).

Initialize the PKI and build the CA.

Build the server keys and certificate.

Build the client certificate / keys.

Package the client key / cert and the server cert in a '.p12' file.

We will use the OpenVPN 'server.conf' example file as a template to create our own file. We will use AES with CBC and a key of 256 bit (AES-256-CBC) as it is the default cipher and mode used by OpenVPN. As we want all the traffic to be routed towards the VPS, it is necessary to propagate the default route '0.0.0.0/0.0.0.0' to the smartphone. This route will be pushed from the VPS to the smartphone after the VPN connection is established. Aside from that, a static IP is assigned to each individual user in order to enhance security. This approach helps greatly when creating specific Snort rules or firewalls rules.

In order to setup the VPN client in the smartphone, the first step is to import the certificates packaged in a '.p12' file, which must be password protected. This is done once the files are copied to the SD card. Afterwards, the VPN can be setup. Through the VPN connection configuration menu in the smartphone it is necessary to setup the IP of the VPS IP and the option 'redirect gateway'.

As previously mentioned, we will implement firewall policies on the Smartphone and these policies will limit the allowed traffic to the one going through the tunnel. To begin with, we will implement a default DENY policy. Only the IN/OUT traffic going from/to the VPN's IP (50.19.152.221) and the encrypted traffic from/to the VPS going through tun0 interface (VPN interface) are ACCEPT. The rules implemented are as follows:

```
#!/system/bin/sh
/system/bin/iptables -F
# Traffic to localhost allowed
/system/bin/iptables -A INPUT -i lo -j ACCEPT
```

```
/system/bin/iptables -A OUTPUT -o lo -j ACCEPT


# Default deny policy for INPUT, OUTPUT, FORWARD
/system/bin/iptables --policy INPUT DROP
/system/bin/iptables --policy OUTPUT DROP
/system/bin/iptables --policy FORWARD DROP


# ACCEPT all the established connections
/system/bin/iptables -A INPUT -m state --state ESTABLISHED,RELATED -j
ACCEPT
# ACCEPT traffic to the SSH to manage the device
/system/bin/iptables -A INPUT -i tun0 -p 6 --dport 22 -j ACCEPT
#ACCEPT ALL the traffic coming from the tunnel
/system/bin/iptables -A INPUT -i tun0 -p 1 -j ACCEPT
# ACCEPT ALL the traffic coming to the VPS IP /system/bin/iptables -A
OUTPUT -d 50.19.152.221/32 -j ACCEPT
# ACCEPT ALL the traffic going to the tunnel interface tun0
/system/bin/iptables -A OUTPUT -o tun0 -j ACCEPT
```

The results of the rules applied to the smartphone can be seen below:

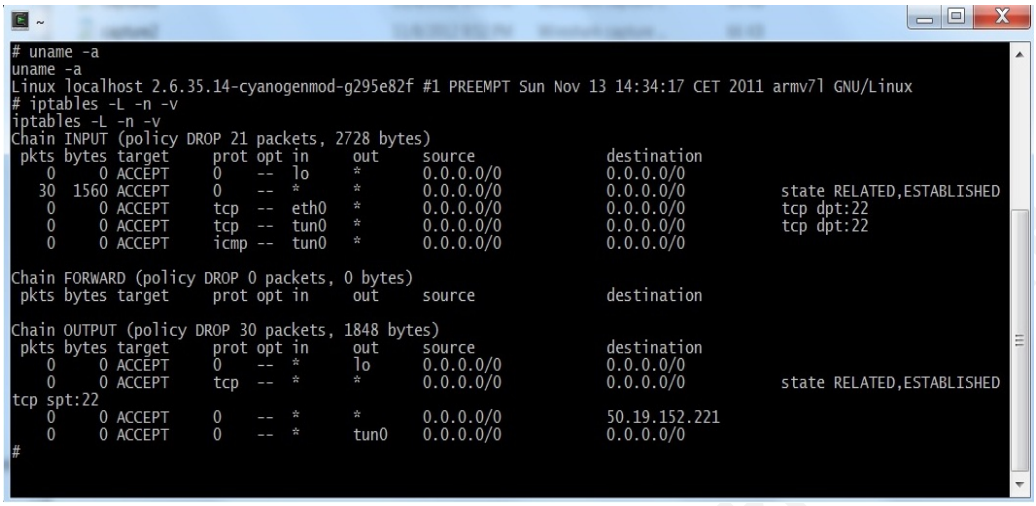

Figure 2: Firewall rules applied to the smartphone

In the Figure 2 ‘INPUT Chain’ is the set of rules applied to incoming traffic to the device. The default policy for this chain is ‘DROP’.

The first rule which affects the loopback interface (lo) is necessary as sometimes application needs to perform local sockets connections.

The second rule is in charge of accepting the traffic that already exists through the WiFi, VPN or 3G connection.

The third rule and fourth rules permit SSH traffic through the WiFi interface (eth0) and the VPN (tun0) in order to access the device remotely with SSH keys as it is explained in ‘Alonso Parrizas, A. (2011)’. The last one permits ICMP traffic through VPN tunnel for troubleshooting purpose .

The ‘FORWARD Chain’are the set of rules applied to the routed traffic and the default policy for these rules is DROP. It means that all traffic going from one interface to another interface (e.g. Wifi to 3G) is dropped by default.

The last chain is the ‘OUTPUT Chain’ which is the set of rules applied to outbound traffic from the smartphone. Similar to ‘INPUT Chain’ the default policy for these rules is ‘DROP’. The first rule is to permit the communication via sockets locally. The second rule allows existing traffic. The third rule permits to establish the VPN tunnel. The last rule allows the traffic going to the VPS through the VPN tunnel.

In order to reroute the traffic coming from the VPN tunnel to the Internet, iptables rules must be created in the VPS. The rules applied in the VPS are the following:

```
#!/bin/sh
/sbin/iptables -F
# Traffic to localhost and VPN interface tun0 allowed
/sbin/iptables -A INPUT -i lo -j ACCEPT
/sbin/iptables -A OUTPUT -o lo -j ACCEPT
/sbin/iptables -A INPUT -i tun0 -j ACCEPT
/sbin/iptables -A OUTPUT -o tun0 -j ACCEPT

# Default policy DROP Policy
/sbin/iptables --policy INPUT DROP
/sbin/iptables --policy OUTPUT ACCEPT
/sbin/iptables --policy FORWARD ACCEPT

# Incoming traffic allowed: HTTP, HTTPS, FTP and traffic established
/sbin/iptables -A INPUT  -i eth0 -p 6 --dport 80 -j ACCEPT
/sbin/iptables -A INPUT  -i eth0 -p 6 --dport 443 -j ACCEPT
/sbin/iptables -A INPUT -p tcp --dport ftp -j ACCEPT
/sbin/iptables -A INPUT -p tcp --dport ftp-data -j ACCEPT
/sbin/iptables -A INPUT -p ALL -i eth0 -m state –state
ESTABLISHED,RELATED –j ACCEPT

# NAT rule for the outgoing traffic coming the from the tunnel
 /sbin/iptables -A FORWARD -i tun0 -s 0.0.0.0/0.0.0.0 -d 0.0.0.0/0.0.0.0 -j
ACCEPT
# Forward the traffic between the VPN and the internet interface
/sbin/iptables -A FORWARD -i eth0 -o tun0 -m state --state
ESTABLISHED,RELATED -j ACCEPT
```

```
/sbin/iptables -A FORWARD -i tun0 -o eth0 -j ACCEPT
/sbin/iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE

Deny IPv6 traffic
/sbin/ip6tables --policy INPUT DROP
/sbin/ip6tables --policy OUTPUT DROP
/sbin/ip6tables --policy FORWARD DROP
```

In the following screenshot we can see the results of the applied rules:

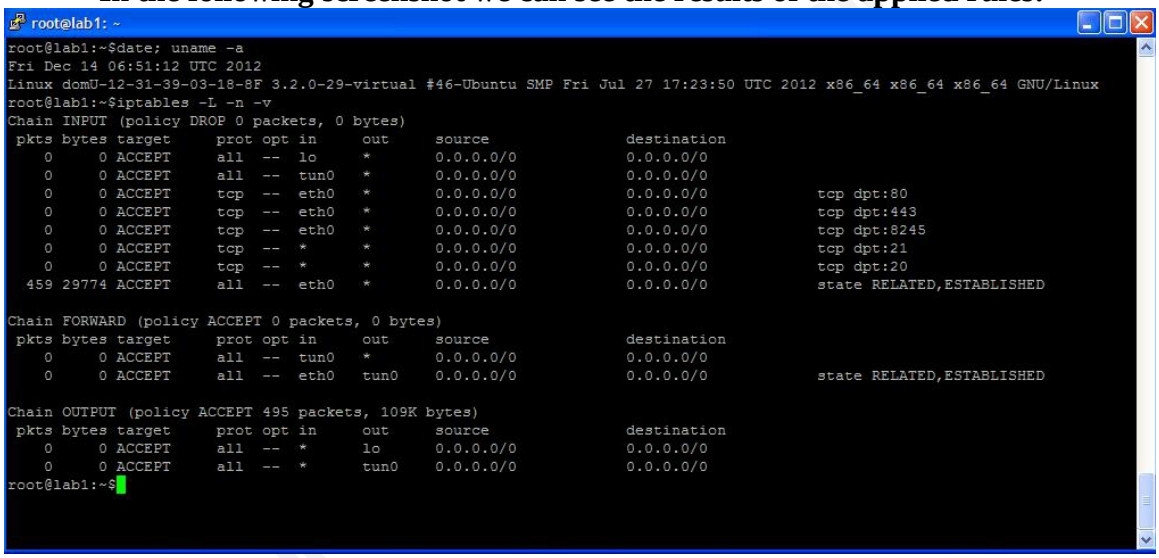

Figure 3: Firewall rules applied to the VPS

From the 'INPUT Chain' The second rule clearly shows that we are allowing all the traffic from the tun0 interface, (i.e. essentially all the VPN traffic). The third rule is in charge of the VPN traffic, but instead of using the standard OpenVPN port (1194/UDP) we will use port 80/TCP. This is in order to connect from networks, which only allows HTTP traffic as it was explained in detail in the paper 'Securely Deploying Android Devices' (Alonso Parrizas, 2011). The following set 'INPUT Chain' rules permit traffic to ports 443/tcp, 8245/tcp and 20/21 in order to manage the VPS remotely and apply patches. Finally, the last rule is for the traffic that already exists.

Regarding the FORWARD chain rules, the traffic is allowed by default as we want to forward and route the traffic from the physical interface (eth0) to the VPN interface (tun0) and vice versa.

The OUTPUT chain is configured to allow all the outgoing traffic.

Capturing the traffic and installing the IDS

Once the VPN tunnel is established and the traffic is being sent to the VPS, we can start monitoring the traffic. The traffic will be captured and inspected against Snort (Sourcefire, 2001).

Moreover, we will run tcpdump (tcpdump, 2012) to capture all traffic. There are several tools, which allow launching processes in the background and recovering them later. In our case, we used screen, which allows for spawning multiple shells as background processes and recovering them later. As we want to capture the full content of all packets and store all that information in a single file, we need to run tcpdump with the following switches:

```
screen tcpdump -i tun0 -s 0 -w /home/angel/capturex.cap
```

We also need to install and configure the Snort IDS. By default in Ubuntu 12.04 the Snort version is 2.9.2 which is quite current. To install it, we will use the 'apt repository' and the packages to install are: snort, snort-common and snort-common-libraries.

Besides those packages, it is necessary to install the Snort IDS signatures. This project will take advantage of two main signatures: snort (the registered version rules) and the Emerging Threats (Emerging Threats, 2012). All the rules will be updated daily with the oinkmaster script. Please note that the time stamp on the rules we ran our tests with is the 12th of November 2012. Once the signatures are up to date, it is necessary to select the appropriate signatures. Those signatures are the one which have been created for Android or are in some way related: emerging-mobile_malware.rules, emerging-web_client.rules, malware-cnc.rules. For these sets of rules, we will have all signatures enabled.

In addition, we will also choose some other signatures which could potentially be interesting: blacklist.rules, botnet-cnc.rules, community-bot.rules, community-virus.rules, malware-backdoor.rules, malware-cnc.rules , malware-tools.rules, emerging-trojan.rules, emerging-malware.rules, emerging-botcc.rules. As far as these go, we will simply leave the rules activated by default.

For our own signatures, we will use the file 'local.rules'.

The next step is to define the network and the interface we want to monitor. By default, in Ubuntu/Debian, this is done by modifying the /etc/snort/snort.debian.conf file, where the network and the interface are defined:

```
DEBIAN_SNORT_HOME_NET="172.16.1.1/24"
DEBIAN_SNORT_INTERFACE="tun0"
```

At this stage, the only remaining part is to run an instance of Snort with '/etc/init.d/snort start' command and to check if the process is running with the proper parameters:

```
/usr/sbin/snort -m 027 -D -d -l /var/log/snort -u snort -g snort -c
/etc/snort/snort.conf -S HOME_NET=[172.16.1.1/24] -i tun0
```

Analyzing the traffic

When analyzing network flows, it is necessary to understand what is going on and have great command of the tools to support the analysis. Tcpdump and Wireshark are among the best resources for this matter. Whilst tcpcump, is good enough to perform some basic analysis in a terminal shell (by creating filters for ports, protocols, IPs, etc.), Wireshark gives a much better view of the content and the qualities of each IP datagram or the TCP segments.

Once suspicious traffic is isolated, it is possible to use some of the interesting fields, like "content" or the "destination IP" to create additional rules. This information is key when creating IDS signatures, as will be explained in the following sections.

Creating our policy rules and testing them

We will need to define our own network policies in order to trigger alerts for all suspicious flows. In our case, we will define the following ports as allowed traffic: 53/tcp, 53/udp, 80/tcp, 123/udp, 443/tcp and 5228/tcp. They correspond to DNS, HTTP/S and Google Play traffic.

Now, we will have to create the snort rules which theses rules trigger alerts whenever traffic on ports other than those defined above is detected:

```
alert tcp $HOME_NET any -> $EXTERNAL_NET !$HTTP_PORTS,!5228,!53
```

```
(msg:"NOT STANDARD TCP PORTS";sid:9999999; rev:0;)

alert udp $HOME_NET any -> $EXTERNAL_NET !53,!123(msg:"NOT STANDARD UDP PORTS";sid:9999998; rev:0;)
```

To test our signatures, we will establish a connection from the Android device, using the netcat tool. We will specify a random destination IP and set the destination port to 9999 and then verify that the alert is generated in /var/log/snort/alert:

```
[**] [1:9999999:0] NOT STANDARD TCP PORTS [**]

[Priority: 0]

11/12-19:29:56.422536 172.16.1.99:37946 -> 147.156.1.1:9999

TCP TTL:64 TOS:0x0 ID:25761 IpLen:20 DgmLen:60 DF

******S* Seq: 0xA82B1A31  Ack: 0x0  Win: 0xFAF0  TcpLen: 40

TCP Options (5) => MSS: 1350 SackOK TS: 492084 0 NOP WS: 1
```

In addition to that, we need to check, if the traffic is stored by tcpdump.

```
root@lab1:/var/log/snort# tcpdump -nr /home/angel/capturex.cap host 147.156.1.1

reading from file /home/angel/capturex.cap, link-type RAW (Raw IP)

19:29:47.402027 IP 172.16.1.99.37946 > 147.156.1.1.9999: Flags [S], seq 2821397041, win 64240, options [mss 1350,sackOK,TS val 491182 ecr 0,nop,wscale 1], length 0
```

```
root@lab1:/var/log/snort# tcpdump -nr tcpdump.log.1352747222 host 147.156.1.1

reading from file tcpdump.log.1352747222, link-type RAW (Raw IP)

19:29:47.402021 IP 172.16.1.99.37946 > 147.156.1.1.9999: Flags [S], seq 2821397041, win 64240, options [mss 1350,sackOK,TS val 491182 ecr 0,nop,wscale 1], length 0
```

At this stage, we have confirmed that the traffic is being stored correctly and that

alerts are generated by Snort whenever the flows are not in line with the network policy.

Testing the platform with real malware

This will be the ultimate test as this time. We are going to install real malware on the Android device and analyze the traffic flows against the IDS. The malware samples will be obtained from the following website: http://contagiominidump.blogspot.com.es/ which is a good resource for Android malware. At the time when we carried out our tests, the latest malware was published on Friday October 19, 2012.

All malware is password protected and once downloaded onto our workstation, we need to deploy it on the smartphone with the Android SDK toolkit. The command used to install any Android application which comes in the APK format is 'adb install'.

1.1.1. Analysis of 'Android - Fake Installer /Fake Lookout (TrojanFakeLookout.A.)'

The detailed analysis of this malware can be found in 'TrustGo (2012)'. In short, this particular malware steals information from the device, and sends it to a specific server. For our analysis, it is most important to know is that the communications with the external server are carried over HTTP. In fact, this includes some commands which are sent to the infected device in order to steal specific information (eg: SMS; data from the SD, etc)

Once the malware has been pushed onto the device, we have to execute the application. At the same time – and in real time, we are checking the alerts coming from snort with the 'tail –f /var/log/snort/alert' command. This means that if any signature is matched, we will see an alert. Unfortunately, no single alert has been raised which means that the signatures from emerging-threat and Snort don't catch this specific malware.

As we are capturing all the traffic going through the VPN tunnel though, we can see which flows have been opened between the smarphone and the Internet. This permits us to dig into the traffic and with the help of tcpdump and Wireshark (Wireshark, 2012), we can analyze it.

The first strange flow we detect is an HTTP connection to the host 68.178.232.100. Looking into the packets further into the packet capture, we detect the following HTTP request:

```
GET /controls.php HTTP/1.1
User-Agent: Dalvik/1.4.0 (Linux; U; Android 2.3.7; HTC Vision Build/GRI40)
Host: thelongislandpress.com
Connection: Keep-Alive
Accept-Encoding: gzip
```

The URL requested is http://thelongislandpress.com/controls.php, which matches with the string hxxp://[hidden]press.com/controls.php reported in TrustGo (2012, October 17). This is the server from which the commands are sent to the device though the HTTP response.

Based on this data, we can take the following step and create a signature which matches the string http://thelongislandpress.com/controls.php . As soon as it's done, we can run the test again while monitoring the traffic. A possible signature is:

```
alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS
(msg:"MALWARE Android TrojanFakeLookout.A"; flow:established,to_server;
content:"/controls.php";nocase; http_uri; content:"Host: thelongislandpress.com";
http_header; metadata:service http; reference:url,blog.trustgo.com/fakelookout/;
sid:88888801; rev:1;)
```

This Snort signature matches any TCP traffic going to the HTTP ports in addition to some specific strings once the connections has been established. This is the detailed explanation:

Alert: this means that an alert will be triggered whenever the rule is matched

Tcp: protocol type

$HOME_NET: defined as 172.16.1.1/24

$EXTERNAL_NET: defined as anything different to $HOME_NET

$HTTP_PORTS: Any HTTP port (80)

msg:"MALWARE Android TrojanFakeLookout.A": The message which will be generated when the alert is triggered

flow:established,to_server: That the TCP connection is already established and the traffic is going from the client to the server.

content:"/controls.php";nocase; http_uri: The string "/controls.php" is in the URL

content:"Host: thelongislandpress.com"; http_header: The string thelongislandpress.com in the 'host' field under the HTTP header

In order to test the traffic, we need to run the snort command pointing to the capture file:

```
/usr/sbin/snort -m 027  -d -l /var/log/snort2 -u snort -g snort -c
/etc/snort/snort.conf -S HOME_NET=[172.16.1.1/24] -r /home/angel/capturex.cap
```

In the meantime, we check the alerts in real time:

```
[**] [1:88888801:1] MALWARE Android TrojanFakeLookout.A [**]
[Priority: 0]
11/08-21:00:55.125712 172.16.1.99:59058 -> 68.178.232.100:80
TCP TTL:113 TOS:0x0 ID:24542 IpLen:20 DgmLen:223 DF
***A**** Seq: 0x393EB8F8  Ack: 0x9A02E635  Win: 0xFF48  TcpLen: 20
[Xref => http://blog.trustgo.com/fakelookout/]
```

1.1.2. Analysis of 'Android Fakelash - Android SMS trojan'

The second piece of malware we will analyze is a fake version of the popular flash plugin. The full analysis of the malware can be found in Fortiguard (2012, September 18). This malware steals SMS messages and sends them via HTTP. Most likely this is used to steal SMS authentication tokens used for online banking. Our analysis will be focused on the HTTP traffic, as it is how the information is transported.

As the first step, we look at the existing rules in Snort to see if anything has already been created for this particular malware. This time, we see that there already is a rule:

```
/etc/snort/rules/malware-cnc.rules:alert tcp $HOME_NET any ->
```

```
$EXTERNAL_NET $HTTP_PORTS (msg:"MALWARE-CNC Android/Fakelash.A!
tr.spy trojan command and control channel traffic"; flow:to_server,established;
content:"/data.php?action="; nocase; http_uri; content:"&m="; distance:0; nocase;
http_uri; content:"&p="; distance:0; nocase; http_uri; content:"&n="; distance:0; nocase;
http_uri; metadata:policy security-ips drop, service http;
reference:url,blog.fortiguard.com/android-malware-distributed-by-malicious-sms-in-
france/; classtype:trojan-activity; sid:24251; rev:1;)
```

This signature matches the same protocol, ports, source/destination IP, and direction of the flow as the previously mentioned signature. However, on this occasion, it matches different strings inside the content of the packet:

content:"/data.php?action="; nocase; http_uri : the string “/data.php?action=” in the URI

content:"&m="; distance:0; nocase; http_uri: the string “&m=” in the URL following the previous one.

content:"&p="; distance:0; nocase; http_uri: the string “&p=” in the URL following the previous one.content:"&n="; distance:0; nocase; http_uri: the string “&n=” in the URL following the previous one.

Which is the following URL:

data.php?action=XXX+&m=YYY+&p=ZZZ+&n=WWW

Unfortunately, this tests reveals that there is no alert triggered by the malware generated traffic. Looking at the network flows with tcpdump, there is an HTTP connection with the host androidoutdate.co.cc, which can be resolved to the following IP: 89.248.172.19. Analyzing the HTTP request with Wireshark, we can see the following HTTP request:

```
GET /data.php?action=cmd&online=ffffffff-c32c-1f6d-2bbc-
fab90033c587&m=null&ver=Flash HTTP/1.1
User-Agent: Dalvik/1.4.0 (Linux; U; Android 2.3.7; HTC Vision Build/GRI40)
Host: androidoutdate.co.cc
Connection: Keep-Alive
```

```
Accept-Encoding: gzip
```

If we look at the report published in Fortiguard (2012, September 18) we can see that fields in the URL requested match the one in the report:

/data.php?action=cmd&online=UUID&m=PHONE NUMBER&ver=flashpayer11

Therefore, the orignal Snort signature is partially matching some of the fields. In order to make it work though, we will need to modify the signature to match all the fields:

content:"/data.php?action="; nocase; http_uri : the string "/data.php?action=" in the URI

content:"&online="; distance:0; nocase; http_uri: the string "&online=" in the URL following the previous one.

content:"&m="; distance:0; nocase; http_uri: the string "&m=" in the URL following the previous one.content:"

&ver="; distance:0; nocase; http_uri: the string "ver=" in the URL following the previous one.

So, the final signature is as follows:

```
alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS
(msg:"MALWARE-CNC Android/Fakelash.A!tr.spy trojan command and control
channel traffic - VERSION GCIA"; flow:to_server,established; content:"/data.php?
action="; nocase; http_uri; content:"&online="; distance:0; nocase; http_uri;
content:"&m="; distance:0; nocase; http_uri; content:"&ver="; distance:0; nocase;
http_uri; metadata:policy security-ips drop, service http;
reference:url,blog.fortiguard.com/android-malware-distributed-by-malicious-sms-in-
france/; classtype:trojan-activity; sid:88888802; rev:1;)
```

We can test the signature and we immediately get the following results:

```
[**] [1:88888802:1] MALWARE-CNC Android/Fakelash.A!tr.spy trojan
command and control channel traffic - VERSION GCIA [**]

[Classification: A Network Trojan was detected] [Priority: 1]

11/11-16:22:52.835667 172.16.1.99:54971 -> 89.248.172.19:80

TCP TTL:46 TOS:0x0 ID:63821 IpLen:20 DgmLen:301 DF
```

```
***A**** Seq: 0x89151AE5  Ack: 0xF7698A01  Win: 0x1B00  TcpLen: 32

[Xref => http://blog.fortiguard.com/android-malware-distributed-by-malicious-sms-in-france/]
```

1.1.3. Analysis of 'Android SimpleTemai'

This malware downloads some files from the Internet through an HTTP request, and installs a backdoor on the device. A full analysis of the network flows can be found in Mobile Sandox (2012, July 31). The report shows that the connections are made to two different URLs – namely: http://wap.juliu.net or http://ads.wapx.cn.

As before, we executed the malware on the mobile device but no alerts are generated by Snort. We, therefore, search for the domain name string in all the signatures (eg: grep "wap" /etc/snort/rules/*) in order to detect any other possible existing signature. This yields no results.

While executing the malware we noticed an HTTP connection to the domain wap.juliu.net:

```
GET /control.html?imei=352212045402919&sim=null&imsi=null&model=HTC%20Vision&release=2.3.7&qd=01300602323&gamename=com.polarbit.rthunderliteok&script=001 HTTP/1.1

User-Agent: Dalvik/1.4.0 (Linux; U; Android 2.3.7; HTC Vision Build/GRI40)

Host: wap.juliu.net
```

The interesting part is that the 'IMEI' of the infected device is sent along in the request. This is quite suspicious. If we search for an existing signature with the 'IMEI' string, we find out that there is something of relevance:

```
/etc/snort/rules/emerging-mobile_malware.rules:alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (msg:"ET MOBILE_MALWARE Possible Mobile Malware POST of IMEI International Mobile Equipment Identity in URI"; flow:established,to_server; content:"POST"; http_method; content:"imei="; nocase; http_uri; reference:url,www.met.police.uk/mobilephone/imei.htm; classtype:trojan-activity; sid:2012848; rev:1;)
```

This signature matches any HTTP POST request which contains the IMEI string.

We can adapt this signature to fire an alert whenever the same string is detected within an HTTP GET request. We just need to substitute the string “POST” with “GET”.

Once we have created the new rule, we can run snort again against the capture file and see if an alert is generated:

```
[**] [1:88888803:1] ET MOBILE_MALWARE Possible Mobile Malware GET of IMEI International Mobile Equipment Identity in URI [**]
[Classification: A Network Trojan was detected] [Priority: 1]
11/11-17:03:41.452832 172.16.1.99:51225 -> 119.191.58.17:80
TCP TTL:43 TOS:0x0 ID:30374 IpLen:20 DgmLen:350 DF
***A**** Seq: 0x784DE1D6  Ack: 0xAEC32DB7  Win: 0xFEC9  TcpLen: 20
[Xref => http://www.met.police.uk/mobilephone/imei.htm]
```

Truth is that we could have tweaked the signature to detect only the traffic going to the domain wap.juliu.net and with the string ‘imei’, however the disclosure of the imei through any HTTP request is enough to generate an alert so that we can investigate it.

1.1.4. Analysis of ‘Android KungFu variant’

This is the most sophisticated malware as it can modify some critical components of the operating system. In order to accomplish that, the malware first needs to download some components from the Internet through HTTP but not using standard ports. The full analysis and report can be found in ‘Aicuxiao, 2012’.

When executing this malware on the smartphone we notice several alerts matching the signature we created to detect any non-standard flow:

```
[**] [1:9999999:0] NOT STANDARD TCP PORTS [**]
[Priority: 0]
11/24-04:10:50.237622 172.16.1.99:54122 -> 114.112.190.30:7500
TCP TTL:64 TOS:0x0 ID:61659 IpLen:20 DgmLen:60 DF
******S* Seq: 0xD6B7A4BF  Ack: 0x0  Win: 0xFAF0  TcpLen: 40
TCP Options (5) => MSS: 1350 SackOK TS: 21604 0 N
```

Other than this match, no other Snort alerts have been triggered, although there are some specific signatures for this particular malware:

sid-msg.map:2013020 || ET MOBILE_MALWARE DroidKungFu Checkin || url,www.fortiguard.com/encyclopedia/virus/android_droidkungfu.a!tr.html || url,www.redmondpie.com/droidkungfu-new-hard-to-detect-android-malware-threat-on-the-loose-steals-user-data-and-more/ || url,extraexploit.blogspot.com/2011/06/droidkungfu-just-some-piece-of-code.html

sid-msg.map:2013022 || ET MOBILE_MALWARE DroidKungFu Checkin 2 || url,www.fortiguard.com/encyclopedia/virus/android_droidkungfu.a!tr.html || url,www.redmondpie.com/droidkungfu-new-hard-to-detect-android-malware-threat-on-the-loose-steals-user-data-and-more/ || url,extraexploit.blogspot.com/2011/06/droidkungfu-just-some-piece-of-code.html

sid-msg.map:2013023 || ET MOBILE_MALWARE DNS Query for gongfu-android.com DroidKungFu CnC Server || url,www.fortiguard.com/encyclopedia/virus/android_droidkungfu.a!tr.html || url,www.redmondpie.com/droidkungfu-new-hard-to-detect-android-malware-threat-on-the-loose-steals-user-data-and-more/ || url,extraexploit.blogspot.com/2011/06/droidkungfu-just-some-piece-of-code.html

sid-msg.map:2013063 || ET MOBILE_MALWARE DroidKungFu Checkin 3 || url,blog.fortinet.com/androiddroidkungfu-attacking-from-a-mobile-device/ || url,www.fortiguard.com/encyclopedia/virus/android_droidkungfu.a!tr.html || url,www.redmondpie.com/droidkungfu-new-hard-to-detect-android-malware-threat-on-the-loose-steals-user-data-and-more/ || url,extraexploit.blogspot.com/2011/06/droidkungfu-just-some-piece-of-code.html

sid-msg.map:2013967 || ET USER_AGENTS Suspicious User-Agent (adlib) || url,blog.trendmicro.com/connections-between-droiddreamlight-and-droidkungfu/

sid-msg.map:2013968 || ET MOBILE_MALWARE Android/KungFu Package Delete Command || url,blog.trendmicro.com/connections-between-droiddreamlight-and-

```
droidkungf
```

If we analyze all these connections, we can see that the following IPs and ports are involved: 114.112.190.30:7500, 58.221.44.102:7500, 180.210.34.207:8511

As far as the first IP goes, there are two HTTP sessions requesting the following resources:

```
GET http://dd.phonego8.com:7500/ad/nadp.php?v=1.5&id=all

GET http://dd.phonego8.com:7500/ad/nadp.php?
v=1.5&id=CHNF&u=352212045402919
```

As for the second request, the IMEI of the device is sent through the field "u=" which isn't very nice. In this case, the IMEI signature we created for the 'SimpleTemai' was not triggered. This happened because the signature was searching for the 'imei=' string instead of 'u=' – and because the HTTP port used here is not the standard one (80), but 7500.

Looking at the analysis published in Mobile Sandbox (2012, September 8) we can see that the malware connects to several domains. One of those domains, happens to be dd.phonego8.com so this could be a good lead. Let's then create a signature which matches that domain on port 7500 – but first, let's finish analyzing the rest of the connections.

This TCP connection to 58.221.44.102:7500 is a POST to:

```
ad.imadpush.com:7500/AppManager/index.php/AppPoster/mgPush/getPush
```

And the information sent in the POST is:

```
imei=352212045402919&packagename=com.tebs3.cuttherope&versionname=1.1
.5&versioncode=6&IMEI=352212045402919&login_way=1&user_detal_info=1&rq_po
ster=1&user_detal_info=1&dId=10000
```

This matches the analysis performed in Mobile Sandbox (2012, September 8) perfectly, however, this traffic has not matched the signature 'ET MOBILE_MALWARE Possible Mobile Malware POST of IMEI International Mobile Equipment Identity in URL' mentioned in section 'Android SimpleTemai' because port 7500 is not defined as a

standard HTTP port in the variable $HTTP_PORTS.

Finally, the last flow towards the host 180.210.34.207:8511 features a GET HTTP request:

```
GET ad.pandanew.com:8511/search/s2.php?
i=352212045402919&c=NCuttherope&v=17&b=htc_wwe&m=HTC+Vision&sv=10
```

The first parameter in the request is clearly the IMEI. This has not been detected by the IMEI signatures as in the other cases - the destination TCP port is not a standard one and the string matched is not 'imei' but 'i'.

Examining the available signatures for this malware a little further in order to understand why none of them have been triggered:

```
emerging-mobile_malware.rules:alert tcp $HOME_NET any ->
$EXTERNAL_NET 8511 (msg:"ET MOBILE_MALWARE DroidKungFu Checkin";
flow:established,to_server; content:"POST "; depth:5; nocase;
content:"/search/sayhi.php"; distance:0; nocase; sid:2013020; rev:1;)

emerging-mobile_malware.rules:alert tcp $HOME_NET any ->
$EXTERNAL_NET 8511 (msg:"ET MOBILE_MALWARE DroidKungFu Checkin 2";
flow:established,to_server; content:"POST "; depth:5; nocase; content:"search/rpty.php";
distance:0; nocase; sid:2013022; rev:1;)

emerging-mobile_malware.rules:alert tcp $HOME_NET any ->
$EXTERNAL_NET 8511 (msg:"ET MOBILE_MALWARE DroidKungFu Checkin 3";
flow:established,to_server; content:"POST "; depth:5; nocase;
content:"/search/getty.php"; distance:0; nocase; sid:2013063; rev:1;)

emerging-mobile_malware.rules:alert tcp $HOME_NET any ->
$EXTERNAL_NET $HTTP_PORTS (msg:"ET MOBILE_MALWARE
Android/KungFu Package Delete Command"; flow:established,to_server;
content:"/search/isavailable"; http_uri; content:".php?imei="; http_uri; content:"&ch=";
http_uri; content:"&ver="; http_uri; content:"User-Agent|3A 20|adlib/"; http_header;
classtype:trojan-activity; sid:2013968; rev:1;)
```

Port 8511 is defined in signatures 1, 2, 3 which matches the flow to 180.210.34.207:8511, but the HTTP method doesn't match (POST vs GET). Furthermore, the strings of the PHP script request do not match as the signature checks for

“search/sayhi.php”, “search/rpty.php” and "/search/getty.php” whereas the traffic analyzed contains “/search/s2.php”. The fourth signature is not so restrictive regarding the matching of the port, but the string that is supposed to match the content search is for a specific User-Agent which is not the case here.

Having this information, we can now create a signature that matches the following:

TCP port 8511, GET request, and the string /search/s2.php

The signature created is:

```
alert tcp $HOME_NET any -> $EXTERNAL_NET 8511 (msg:"ET MOBILE_MALWARE DroidKungFu Variant - GCIA"; flow:established,to_server; content:"GET"; content:"/search/s2.php";  sid:88888804; rev:1;)
```

And once we test the traffic against the new signature we get the following alert:

```
[**] [1:88888804:1] ET MOBILE_MALWARE DroidKungFu Variant - GCIA [**]
[Priority: 0]
11/24-05:24:58.737643 172.16.1.99:45206 -> 180.210.34.207:8511
TCP TTL:64 TOS:0x0 ID:10645 IpLen:20 DgmLen:170 DF
***AP*** Seq: 0x13F93447  Ack: 0xF5313366  Win: 0x7D78  TcpLen: 32
TCP Options (3) => NOP NOP TS: 466437 295592359
```

Having all the traffic fully stored on the VPS file system, we can perform any network analysis in case anything happens. This captured traffic is useful whether a device is compromised by a 0-day malware that triggers no alert. Aside from that, the initial rules created to detect any non-standard connection can be useful to detect 0-day malware which is not using HTTP ports.

In such situations, the analyst could use tpcdump or wireshark against the capture file (/home/angel/capturex.cap) searching for any unusual connections or in fact, anything simply suspicious.

Incident response to malware

Although the purpose of this paper is to cover the analysis of the network flows, we are going to highligt some incident response steps required for handling Android malware.

1.1.5. Detection of known malware with non-existing signature

In the first case, we will cover malware that has been already reported but no snort signatures are yet available.

Several tools and techniques were proposed in the past to improve the security of the Android mobile operating system (Alonso Parrizas , 2011) (link). For example, removing the possibility to install any non-authorized software or install an antivirus/antimalware. This could be used as part of the preparation phase of the incident handling process, but in this case we will focus our efforts on the network flows.

1. Preparation: During this phase, a security analyst with intrusion analysis skills must be designated. The analyst will most likely follow Android security forums, malware news, etc, in order to be aware of new malware, which could pose a significant threat for the business. One good resource for Android malware related information is: http://contagiominidump.blogspot.com/ where samples can be downloaded. Also, it is absolutely necessary to prepare and build a lab where the analyst can perform his tests. This lab needs to consist of:

   An Android device: where the malware can be executed

   A computer running Linux with USB ports to connect to the Android device and push the malware. Also, this machine needs to have network interfaces to analyze the traffic going through it (Snort, tcpdump, Wireshark). This system will receive the traffic from a mirrored port in a Switch.

   A WiFi Access point/router with an isolated network (disconnected from the production/development environments) but with Internet access. The traffic from the lab network must be mirrored to the network card of the computer (e.g.: there are several Linksys models which support this functionality).

In addition to that, a backup and restore policy must be created in order to recover the Android device after each malware installation. In other words, a clean backup of the base image and the base setup (baseline image) has to be created. This image will then be reinstalled onto the device after each test with any given malware is completed.

2. Identification: During this phase, the security analyst will identify any possible malware, which could potentially be an issue for the business. As the malware in question has been already been reported, most likely, its analysis will be (publicly) available. If such report exists, it is important to understand what network communications is the malware involved with – and simply, what the malware ultimately does (e.g: stealing sensitive information directly from the device and sending it through a TCP connection). Once a sample of such malware exists, the analyst needs to test it himself by pushing it to the Android device and executing. The traffic needs to be captured and stored on the lab computer with snort running. As there probably will not be any signatures available yet, no alert will be fired – unless of course, other signatures matched (e.g: the one regarding the non-standard HTTP ports). With the information gathered from the network flows and butilizing the analyst's skills, it should be possible to create a custom signature for use with snort (as we have done and outlined in sections 3.2.1 to 3.2.4). This is where the experience of the particular analyst and his knowledge of snort are crucial. Note that should the malware use SSL, the analyst has to use additional tools and techniques (Fahl, Harbach, Muders, Smith, Baumgärtner and Freisleben, 2012). Once the signatures have been created, the malware can be executed again in order to test if Snort now triggers an alert.
3. Containment: As the malware is executed in a controlled environment there is no need of applying any additional countermeasures.
4. Eradication: same as for containment.
5. Recovery: during this phase and after the detection phase is over, the Android device needs to be restored to its initial state. This means that it needs to be flashed with the baseline image. Even when done, we still need to monitor the network flows in order to detect any anomaly in the outgoing traffic.

6. Lessons learned: Although theses tests are carried out in a contained environment and under full control, we still need to notify Android users about the malware its potential impact for the end user (e.g.: information leakage or identity theft).

1.1.6. Detection of 0-day malware

This scenario is quite different as there is no definition, no signature and no information of our malware. As a result, the analysis is by definition more difficult and in many cases, the malware will not be detected in the early stages. The incident handling process will therefore be a bit different from the previous one.

1. Preparation: During this phase, we are running a production environment, all Android devices are deployed in accordance with the architecture proposed in sections 2.1 and 2.2 – as well as the steps described in the paper 'Securely Deploying Android Devices' (Alonso Parrizas, 2011). This means that all the traffic is captured in real time and verified by an instance of Snort. During this phase, an Incident handler has to be designated in order to perform the analysis once a device becomes compromised. In order to recover such a device later, it is important to have a backup policy, in which daily or weekly backups of each Android device are done and kept in safe repository (there are several tools available on the market). The incident handler will have SSH access through keys to each Android device in order to perform investigations.
2. Identification: This is surely the most difficult part, as there are no signatures, which could possibly detect the malware and trigger an alert. We have already deployed signatures, which can detect any unusual traffic (e.g.: non standard HTTP ports), and if we are lucky enough, we will already receive some alerts. However, if this is not the case, we will have to take a look at some other options:

   Detect if any application has been installed without our knowledge. In order to do that, we need to run the command 'pm list packages' which will list all the packages installed. It is also possible to acquire this information by looking at the file system – namely, the directory where installed applications reside: 'ls –lahtr /data/app'

Check for any listening port, which could be a backdoor. Some of the Android malwares create backdoors. It is then necessary to check the listening sockets with a tool like 'netstat'. This information can be correlated with the captured network traffic.

Detect files, which have been modified / added in the last hours (the time stamp will be important when correlating this information with the network flows). The command 'find' will do just that when supplied with proper parameters. (e.g.: find / -mmin 2). Once we have a list of all modified files, we need to investigate their purpose. For instance, the 'Kungfu Variant' malware will modify the /data/data/data/com.noshufou.android.su/databases/su.db file in order to grant 'su' access to the application itself. This should already be something suspicious for the investigating analyst.

Check permissions of all applications installed through the Android GUI interface. Most of Android malware requests special permissions (like access to the SD card, or to the SMS message DB, etc). The incident handler therefore needs to check which permissions are granted, in order to detect any anomaly.

Once we have timestamps of various key files (or applications) we can correlate the data with the network flows, which are all the time being captured and stored as explained in section 2.2.

Looking at the stored network flows – and the timestamps of the files and / or applications, which caught our interest, we can check what has changed in the network before and after installation or modification the said components. For instance, we can detect which websites were visited before the system was compromised and check if any application was in fact downloaded to the device. Moreover, we can check the traffic after the system has been compromised – and find out, what kind of information has been forwarded. Having this information, will allow us to create customized signatures as it has been done in sections 3.2 and 3.3.1.

3. Containment: As we have full control over the network through the VPN tunnel, we can easily create firewall rules on the VPS itself, in order to filter specific IPs and / or ports. This will be the easiest way to contain the

incident. Once we have identified which ports, protocols, and IPs are involved; we can add an iptables rule to the incoming or outgoing traffic – depending on how the traffic flows (e.g.: Backdoor). A simple rule to filter the HTTP traffic towards IP 1.1.1.1 is:

```
/sbin/iptables iptables -A INPUT  -i tun0 -p 6 --dport 80 –d
1.1.1.1/32  -j DROP
```

4. Eradication: During this phase, the malware has to be removed from the infected device. This can be done through the shell with the commands:

```
pm clear package_name
pm uninstall package_name
```

5. Recovery: Sometimes removing and cleaning the data from the malware package is not enough and it is necessary to recover from a clean backup. That is why the backup policy which was mentioned in step 1, is really critical. If this is the case, we can recover the device from the clean backup quite easily. It is necessary to monitor the flows (once the signatures have been created in step 2) in order to detect if any other device (or the same one again) is compromised by the same malware. An important point is to keep an eye on the iptables rules we created in order to see if there is any match. This means that there would be additional traffic towards the blacklisted IP, which is not being caught by the Snort rule. This therefore needs to be investigated and the network flows analyzed as it has been done in other sections.
6. Lessons learned: Once we understand how the device has been compromised, we need to inform the user what has gone wrong (e.g: browse to dangerous websites) and how this should be avoided in the future. Lastly, we need to communicate the incident to the rest of the company in order to make them aware of the new threat (security awareness program).

Conclusions

Throughout this project, we have been able to deploy a network architecture which permits to closely monitor the network flows originating from the smartphone. This model permits an Intrusion Analyst to monitor the network connections in real time and review the flows in order to perform forensic analysis.

With this setup, we have been able to analyze several pieces of malware, test the effectiveness of our IDS and learn how the alerts are generated. This analysis has permitted to improve the existing Snort signatures for Android. Moreover, we have been able to define a standard network policy, which can be used as a baseline when detecting deviations. Lastly, we have defined the incident handling steps to analyze malware and how to respond to 0-day malware.

References

Aicuxiao (2012, January 17). *Android.KungFu series variants depth analysis and complete clean-up methods* [Web blog]. Retrieved from http://www.aicuxiao.org/2012/01/17/20403.html

Alonso Parrizas, A. (2011, September 22). *Securely deploying Android devices*. Retrieved from http://www.sans.org/reading_room/whitepapers/sysadmin/securely-deploying-android-devices_33799

CyanogenMod (May 2011). *CyanogenMod ROM* [software]. Retrieved from http://www.cyanogenmod.com/

Emerging Threats (2012). *Emerging Threats* [Web blog]. Retrieved from http://www.emergingthreats.net/

Fahl, Harbach, Muders, Smith, Baumgärtner, Freisleben (October 2012) *Why Eve and Mallory Love Android: An Analysis of Android SSL (In)Security* http://www2.dcsec.uni-hannover.de/files/android/p50-fahl.pdf

Fortiguard (2012, September 18). *Analysis of Android Fakelash* [Web blog]. Retrieved from http://www.fortiguard.com/av/VID4163531

Mobile Sandbox (2012, July 31). *Analysis of Android SimpleTemai* [Web blog]. Retrieved from http://mobilesandbox.org/xml_report_static/?q=10453

Mobile Sandbox (2012, September 8). *Analysis of Android KungFu variant* [Web blog]. Retrieved from http://131.188.31.187/xml_report_static/?q=44659

NetFilter (1999). *Iptables*. [software]. Retrieved from http://www.netfilter.org/

OpenVPN Technologies. (2002). *OpenVPN* [software]. Retrieved from http://openvpn.net/

Screen (2012) *Manual screen* [software]. Retrieved from. http://linux.die.net/man/1/screen

Sourcefire (2001). *Snort* [software]. Retrieved from http://www.snort.org/

Tcpdump (2012). *Tcpdump* [software]. Retrieved from http://www.tcpdump.org/

TrustGo (2012, October 17). *Analysis of TrojanFakeLookout.A* [Web blog]. http://blog.trustgo.com/fakelookout/

Wireshark (2012). *Wirehark* [software]. Retrieved from http://www.wireshark.org/