honeyM: A Framework for Implementing Virtual Honeyclients for Mobile Devices

TJ OConnor
Information Technology and Operations Center
US Military Academy
West Point, NY 10996

Ben Sangster
Department of Computer Science
US Military Academy
West Point, NY 10996

ABSTRACT

This paper presents honeyM, a framework for deploying virtual mobile device honeyclients. Honeyclients provide the ability discover early warnings about novel attacks and exploitations and are typically deployed to protect wired infrastructure. In a wireless environment, honeyclients usually record attacks against the wireless access point. To identify attacks targeted specifically against mobile device users on wireless networks, we present honeyM. honeyM is a framework for virtual mobile honeyclients and contains a library of simulated mobile devices. honeyM emulates the wireless, Bluetooth, and GPS stacks of an actual mobile device in order to deceive mobile device fingerprinting tools. This paper discusses the design and framework for honeyM and demonstrates the necessity to provide a system to protect mobile users. In this paper, we implemented and evaluated the use of virtual honeyclients for mobile devices in two high risk environments, including a wireless security course and the DEFCON hacker conference.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: [Security and Protection]; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Security

Keywords

Mobile Device Security, Information Assurance, Bluetooth, WiFi, GPS, Wireless Security

1. INTRODUCTION

Protecting users against external attack from within the confines of a wired network is an enormous job. To succeed at network defense, an information assurance team must ensure intense scrutiny of traffic, develop fresh intrusion detection rules to prevent new attacks and exploits, and deploy immediate patches for the latest vulnerabilities. However, all of this is done within the physical perimeter of the network. Imagine placing an unsecured network connection in an organization's parking lot and still having to provide the same level of security. This is the difficulty for information assurance teams to protect mobile and wireless network clients.

Mobile devices are becoming ubiquitous as users require information from anywhere at anytime. All too often, in the tradeoff between functionality and security, mobile device users demand functionality. Examples of this lack of security to accommodate functionality include hard coded default Bluetooth passkeys and WiFi devices automatically joining available unencrypted wireless access points. The demand for functionality often places devices in a configuration that allows attackers to subvert the intended security mechanisms and protocols. As mobile device companies compete to release the latest versions of applications and firmware for devices, they often deploy devices with vulnerabilities. Attackers will continue to target mobile devices as the weakest link to our overall network defense. Anecdotally, we have heard stories of penetration testers mailing wireless devices to the corporate mailroom to attack a network via the wireless vector. Numerous reports of mobile devices vulnerabilities such as the recent Short Message Service (SMS) vulnerability reported by Kevin Mahaffey exist with known signatures are have been integrated into intrusion detection systems for mobile devices. However, administrators must be able to identify novel vulnerabilities without known signatures. We present the idea of using mobile honeyclients to collect these unknown vulnerabilities. By implementing a framework for mobile device honeyclients and evaluating it at two different hostile environments, we provide analysis and conclusions about our method to detect attacks against mobile devices.

The rest of this paper is organized as follows. Section 2 presents background information on wired and wireless honeyclients. In Section 3, we discuss the design of our framework for mobile device honeyclients. Section 4 presents the implementation in hardware and software of our mobile device honeyclients. In Section 5, we evaluate our honeyM framework by the ability to deceive mobile device attack tools and hackers at two hostile networks. In Section 6, we perform an analysis of our framework. Finally, in Section

7 we provide conclusions and promising direction for future work.

2. BACKGROUND

2.1 Overview of Honeypots

The idea of honeypots and deception is not a new technique in warfare. The United States Army's 23rd Special Troops Battalion arrived in Germany in the weeks following D-Day [5]. Concealed in the night darkness, they inflated rubber tanks, jeeps, and weapons. The soldiers of the 23rd STB tricked their enemy to attack the fake equipment by using several elaborate deception measures. As their enemy attacked the fake targets, the 23rd learned invaluable lessons about how their enemy was organized and how they executed attacks. Almost sixty years later, security professionals started applying this same method of deception to computer network security.

In early 2004, Provos from Google presented the idea of Honeyd, a virtual honeypot framework [10]. Honeyd simulates Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) services on a wired network. Using virtual honeypots, the system can convince adversaries that a honeypot is running a particular operating system or a particular vulnerable service. Network mapping and Operating System fingerprinting tools used by the attacker are fooled into believing the virtual device simulation is an actual machine on the network. Through the use of Honeyd, administrators can discover novel attacks while delaying attackers from attacking legitimate targets.

For wireless honeypots, Raytheon provided funding for The Hive, a wireless honeypot research project at the University of Florida in 2007 [3]. The university researchers built and tested honeypot simulated environment for wireless networked systems. They provided a standalone linux distribution that booted a honeypot simulator. The project focused on covering the wireless infrastructure such as access points from attack. The concept of wireless honeypots is not unique as a group known as WISE set up wireless honeypots as early as 2002 [1]. However, the university researchers provided the first virtual wireless honeypot that could simulate multiple wireless access points with a single piece of hardware.

Building upon this work, the Spanish HoneyNet Project introduced the idea of the Wireless HoneySpot [1]. Their project adds the novel concept of simulating both the wireless infrastructure and the individual wireless clients to detect unknown attacks on the 802.11 WiFi protocol. Focusing on detecting attacks against individual clients is a motivation for this paper. However, we address detecting attacks on the multiple communication vectors of mobile devices instead of just 802.11 WiFi.

The purpose of our project is to provide a honeyclient framework for mobile devices. Previous works have created honeyclients for individual protocols such as WiFi. However, our work is novel because it addresses creating honeyclients for mobile devices with multiple wireless communication protocols. We do not know of existing work that has created honeyclients for mobile devices with multiple wireless communication protocols. These protocols may include 802.11a/b/g/n WiFi, Bluetooth, infrared, GPS, 3G, WiMax or code division multiple access (CDMA). In this paper, we limited our honeyclient framework for mobile devices that have WiFi, Bluetooth, and GPS capabilities. Next, we present the design of our mobile device honeyclient framework that deceives an attacker into believing our honeyclient is an actual mobile device.

2.2 Overview of Mobile Device Attacks

In order to understand the types of honeyclient mobile device decoys we want to establish, we need to understand what attacks have succeeded in the past. Some examples of mobile device attacks include attacks against the Bluetooth, WiFi, infrared, or GPS protocols [9, 11, 14, 4]. More intelligent attacks even combine multiple weaknesses of different protocols to circumvent the overall security of the mobile device.

One early example of a Bluetooth attack on a mobile device is the BlueBug attack from the Trifinite organization (trifinite.org.) BlueBugging takes advantage of a mobile device with Bluetooth Radio frequency communication (RFComm) channels that do not require any authentication. Each mobile device offers 30 Bluetooth RFComm channels. In some implementations, these channels require neither authentication nor encryption. An attacker can simply connect to a vulnerable RFComm channel. Once connected, the attacker issues a series of commands that allow initiating calls, utilizing the SMS messaging system, updating the phonebook, call forwarding or forcing the device to utilize a certain cellular provider. After gaining access, the attacker can eavesdrop without any knowledge by the attacked device or user. Herfurt and Laurie discovered this vulnerability and demonstrated it successfully on 50 phones during CeBit 2004.

An example of a WiFi attack against mobile devices is KarmaMetasploit (metasploit.org). The KarmaMetasploit tool allows an attacker to advertise a rogue wireless network. The wireless network appears legitimate and can even serve up web-based content if properly configured. However, the toolkit will launch Metasploit client-side exploits against the user as soon as he or she connects. An attacker could use this toolkit to easily advertise a service such as "Free Public WiFi" at an airport and then attack unsuspecting victims. A recent audit by Airtight Security studied WiFi usage at fourteen different airports in the United States, Asia, and Canada before concluding that users at every airport were particularly vulnerable to this type of attack [8].

Airpwn is another example of a a WiFi attack (airpwn.sourceforge.net). Airpwn listens for wireless traffic on a poorly secured or unencrypted wireless network. An attacker can specify a set of regular expressions in Airpwn to automatically receive and then falsely reply to wireless traffic. An attacker could use Airpwn to redirect a user's web page request, reply maliciously to a Domain Name System (DNS) request, or even create an Address Resolution Protocol (ARP) storm by falsely replying the physical layer addresses.

Another medium for mobile device attacks is infrared. While infrared is simply a medium for communication, there are multiple means of implementing protocol stacks for infrared in mobile devices. These stacks exist in both stateful and stateless designs. Adam Laurie discovered a vulnerability in many hotel television stateless infrared systems that allowed him to obtain guests' names and their room numbers from the billing system [14]. His attack gained access to the email of guests who used web mail and allowed adding charges to a guest bill. Because some stacks are stateless, attackers will simply use fuzzing (or brute force randomization) tools that transmit malicious infrared codes to a target device. However, other mobile devices utilize complex infrared stacks that allow for such actions as Bluetooth-enabled device discovery or object-exchange (OBEX) over infrared.

Global Positioning System (GPS) is becoming an increasingly popular component of mobile devices. Where GPS once used to be used exclusively by the military and commercial sectors, most high-end mobile devices now are GPS enabled. Mobile applications can use GPS to direct a user to the nearest medical treatment facility or locate the nearest Starbucks. Regardless of the criticality of the use, GPS enabled devices are in frequent usage. While GPS contains an authentication protocol, it can be subverted by attackers. A research team led by Paul Kitner and Mark Psiaki at Cornell University demonstrated that suitcase-sized transmitters could be employed to fool devices [4]. They did this by repeatedly sending out false authentication signals, which mimic true GPS satellites and over time GPS receivers accept those false signals as genuine and report improper position data to the user.

In contrast to attacking a single mobile device communication vector such as Bluetooth, WiFi, or GPS, Kevin Mahaffey and John Hering of Flexillis Inc have had great success combining multiple different mobile communication vectors in order to attack mobile devices. In late 2008, the researchers discovered a vulnerability on the iPhone. They successfully managed to introduce an exploit via the Bluetooth Service Discovery Profile (SDP). Utilizing a specially crafted SDP message, the researchers loaded a framework of tools to attack the entire operating system of the phone. Further, their attack enabled access to a root shell on the iPhone device. This attack is very effective, however it relies upon the fact that they must first discover the address of the Bluetooth radio on the iPhone. Mahaffey and Hering discovered that they can simplify the discovery of a hidden Bluetooth Media Access Control (MAC) Address for the iPhone by passive capture of the WiFi traffic. Overall this is a very effective attack but is one that could easily be captured and discovered by mobile honeyclients. While we know about the iPhone SDP vulnerability because Flexillis discovered it, there are most likely unknown vulnerabilities that have been discovered and not reported. As phones and mobile devices continue to grow in functionality, the attackers motivation and opportunity to find novel exploits will also grow. Our research creates a platform to discover these novel attacks.

3. DESIGN

In this section we present honeyM, a framework for virtual honeyclients. Our framework allows the simulation of mobile devices with multiple protocol stacks. We introduce the design considerations in this section and then describe the actual implementation of our virtual honeyclient framework in the following section. We examine the design using a layered approach. Specifically, we examine the network, transport, application and user behavior that makes a mobile device unique. We then model those layers of behavior and log the necessary interaction with each of those layers in order to detect unknown attacks. Figure 1 depicts the design of honeyM. Further, we address some of the management aspects of our design such as logging, remote management, and virtualization that are necessary when creating mobile device honeyclients.

3.1 Network Layer Design Goals

We define the network layer as the generic protocols implemented in a mobile device, including Bluetooth, WiFi (802.11 a/b/g/n), or GPS. When considering how to design a system to mimic these protocols, we need to make some assumptions. First, we assume that the honeyclients framework will be used at the periphery of most networks. Such uses could include placing the system at the physical boundary of an organization or simply establishing the system at a private airport or clandestine meeting location. Next, we assume that their attacks may simultaneously use multiple protocols such as Bluetooth, WiFi, or GPS to compromise a device. Although an attacker could compromise a mobile device using attacks against cellular protocols such as CMDA or 3G, we will not simulate those protocols as the hardware required to attack and simulate those protocols is outside the budget of the average attacker.

Instead, our design focuses on the protocols that can be attacked with relatively inexpensive hardware. Our system includes support for the Bluetooth, WiFi (802.11 a/b/g/n), and GPS protocols. To succeed at imitating mobile devices, we must mimic the exact behavior of how the protocols act on a specific device. For example, the method used for pairing Bluetooth devices is implemented differently across different mobile devices. Creating fingerprints of application and mobile device behavior is ongoing work. However, Bratus et al at Dartmouth have shown that responses in particular wireless chipsets and drivers of mobile devices are differ enough that they can tell they can identify devices by intercepting 802.11 frames [2]. To properly mimic mobile devices, we must apply this same methodology of intensely scrutinizing frames across all network protocols. Next, we must consider how to correctly mimic transport layer of each protocol in mobile devices.

3.2 Transport Layer Design Goals

When we consider the transport layer for mobile devices, we think of the various transport layer protocols that will encapsulate our mobile device data communication. This includes the relatively common protocols such as TCP and UDP carrying our 802.11 communications to specific protocols such as the SCO protocol on Bluetooth intended for carrying hands free audio connections.

In order to mimic the exact behavior of each transport layer protocol, we must perform a thorough investigation of the specific protocol characteristics of each device since certain devices implement each protocol differently, including or excluding several of the optional fields in the protocol specifications. For TCP and UDP protocols this work has already been done. The p0f database contains a library of known devices and the specific behaviors they exhibit [13]. Factors such as the maximum segment size and particular employment of the TCP three-way handshake differ among devices. The p0f database uses all these subtle differences to distinguish between devices. Figure 2 shows how we can import the p0f database to create frames with specific device signatures. For Bluetooth, we must perform intense scrutiny of the characteristics of how each device implements transport layer protocol specifications.

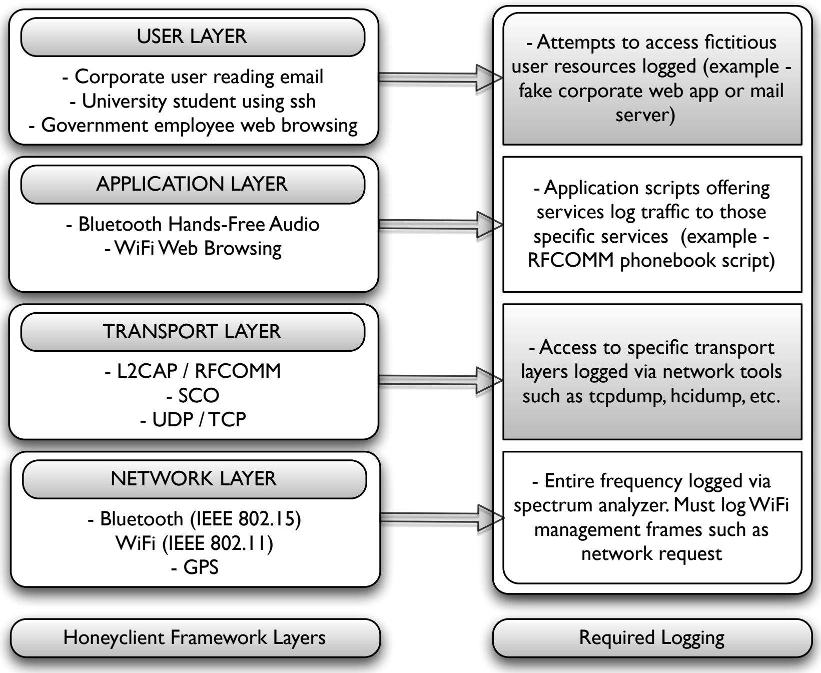

USER LAYER
- Corporate user reading email
- University student using ssh
- Government employee web browsing
APPLICATION LAYER
- Bluetooth Hands-Free Audio
- WiFi Web Browsing
TRANSPORT LAYER
- L2CAP / RFCOMM
- SCO
- UDP / TCP
NETWORK LAYER
- Bluetooth (IEEE 802.15)
WiFi (IEEE 802.11)
- GPS
- Attempts to access fictitious user resources logged (example - fake corporate web app or mail server)
- Application scripts offering services log traffic to those specific services (example - RFCOMM phonebook script)
- Access to specific transport layers logged via network tools such as tcpdump, hcidump, etc.
- Entire frequency logged via spectrum analyzer. Must log WiFi management frames such as network request
Honeyclient Framework Layers
Required Logging

Figure 1: Model for framework of virtualizing mobile device honeyclients and the required logging

```
pkt = p0f_impersonate(IP(dst='www.sigsac.org')/
TCP(sport=1025, dport=80, flags='S'),
osgenre='SymbianOS')
send(pkt)
```

Figure 2: An example of using the p0f database to impersonate a SymbianOS mobile device communication with sigsac.org.

To accomplish this level of scrutiny in Bluetooth, we utilized a Bluetooth Protocol Analyzer. This allowed recording entire Bluetooth conversations. To develop signatures of Bluetooth traffic at the transport layer, we used previously recorded benign traffic provided by the LeCroy Corporation. This traffic was generated by 33 different devices, including Bluetooth-enabled computers, headsets, phones, HID devices, and handheld computers. The benign traffic contained over 26,000 previously recorded packets. Next, we address how different applications on each mobile device create a unique signature of a device.

3.3 Application Layer Design Goals

The idea that applications act with different behaviors is not a new discovery. Provos created the idea of a personality engine for honeyd [10]. Essentially, the personality engine advertises the specific behavior, personality, and vulnerabilities for specific devices. In Provos's implementation, he correctly mimics the subtle differences between a full implementation Linux Mail Server and Microsoft Exchange. This deals not only with the banner information advertised on port 25 but also information like ensuring the optional fields in the TCP packets are constructed exactly how that specific brand of server would build them. In our work, we examine what this looks like for various mobile device applications that operate on WiFi and Bluetooth.

Several automated tools allow us to fingerprint the exact behavior of devices in order to build profiles of how the mobile devices applications offer different services. For Bluetooth, in particular, we used the sdpbrowse (Service Discovery Browse) tool and btaudit to investigate the services offered by each device. Figure 3 depicts using a tool to investigate the service discovery protocol of an iPhone using Bluetooth. The resulting information is then used to build a profile of the device for future usage by honeyclients.

Information released in the profile can tell us more about the vulnerabilities of each device. For example, Nokia phones with a MAC address beginning with 00:60:57 are vulnerable to BlueSnarf and Bluebugging attacks. We can advertise a phone with that specific vulnerability by using the behavior script represented in Figure 4. honeyM is extensible such that behavior scripts can be added and plugged into the project with simplicity. Other types of behavior that we can advertise include iPhones vulnerable to attacks on the service discovery protocol, Plantronic headsets vulnerable to implementations of the CarWhisperer attack on the Synchronous Connection-Oriented (SCO) protocol, or other applications vulnerable to attacks against TCP or UDP sockets.

For each device profile we created a honeyclient device profile Extensible Markup Language (XML) configuration document. Figure 5 depicts an XML configuration used in

```
sdptool browse --l2cap 00:23:6C:60:21:12

Attribute Identifier : 0x1 - ServiceClassIDList
  Data Sequence
    UUID16 : 0x111f - HandsfreeAudioGateway
    UUID16 : 0x1203 - GenericAudio
Attribute Identifier : 0x2 - ServiceRecordState
  Integer : 0x0
Attribute Identifier : 0x4 - ProtocolDescriptorList
  Data Sequence
    Data Sequence
      UUID16 : 0x0100 - L2CAP
    Data Sequence
      UUID16 : 0x0003 - RFCOMM
      Channel/Port (Integer) : 0x8
```

Figure 3: Results from a Bluetooth fingerprinting tool that shows an iPhone offering Generic Audio Connections over Bluetooth, indicating a OS Version 3.0 or greater of the iPhone OS.

honeyM. Each XML configuration contains the Organizationally Unique Identifier (OUI) address, the behavior required on each RFComm, SCO, L2CAP, TCP, and UDP port of the device. Additional information like broadcasted features, supported packet types, and advertised services are included into this XML configuration document as well. Prior to constructing such XML documents, we used tools such as btaudit and nmap to record the full advertised functionality on all ports of those specific devices. This ensures we have an accurate representation of each device. Now the only thing different between two iPhone honeyclients is the user. Next, we address how we configure and represent this user behavior in our model.

3.4 User Layer Design Goals

The purpose of our system is to collect unknown attacks. Thus, we want attackers to attack our simulated devices. Our system must be an attractive target for users and must leak information. This should be configurable by whoever employs the system. But at a minimum, this means that the service discovery and fingerprinting tools used by attackers should report the devices belong to specific (legitimate) users on the network. We assume an attacker would be excited to discover the simulated device belongs to the CEO of an organization or the head of research and development. This also means that we must now simulate traffic from the device.

The simulated traffic should observe the stateful design of the Bluetooth and WiFi protocols and not just fire random packets. The amount of information leakage in this traffic should convince an attacker of target legitimacy. This means it should include diverse and realistic traffic such as secure web browsing, FTP, SSH, VPN, and IPSEC. By creating a hybrid of traffic types, we can incentivize the attacker to continue attacking the device. If a simulated device just repeatedly visited a webpage, there would be no incentive for an attacker to risk herself by continuing the attack. Our tool includes tools realistic enough to spoof automated tools but it will fail under sustained human interaction. True traffic generation is an open problem and one we do not address in this work.

The amount of information leakage in this traffic should be configurable by an administrator of the system. For example, if an attacker compromises a phonebook of a device - we can't obviously give the CEO's call list. However, an administrator could choose some less sensitive numbers that would be representative of numbers that would belong in the phonebook of the CEO. Additionally, an administrator could place numbers in that phonebook with a trap should the actual attacker ever attempt to contact those numbers.

```
# Bind a Bluetooth RFCOMM Socket
server_sock=BluetoothSocket( RFCOMM )
server_sock.bind((dev,PORT_ANY))
server_sock.listen(1)

# Advertise a vulnerable Bluetooth Phoebook
advertise_service( server_sock, "Contacts",
   service_classes = [ SERIAL_PORT_CLASS ],
   profiles = [ SERIAL_PORT_PROFILE ] )
client_sock, client_info = server_sock.accept()

# Record time attacker connects
t = datetime.now()
timeStr=t.strftime("%Y-%m-%d-%H:%M:%S")
filename="rfcomm-"+dev+timeStr+".log"
f=open(filename,'w')
while True:

# Write received attack data to file
data = client_sock.recv(1024)
f.write(data)
if not data: break
client_sock.close()
server_sock.close()
```

Figure 4: An example of a Bluetooth RFComm advertised phonebook service that logs malicious activity.

In our implementation, we allow selecting a different user profile for each honeyclient. A *student profile* browses the local web university using WiFi and plays some of the latest music using Bluetooth SCO. In contrast, a *corporate profile* connects to a VPN, sends encrypted packets, and uses hands free dialing via Bluetooth L2CAP to call the fictitious corporate network. Having addressed how we mimic the different layers of mobile devices, we now address the design for managing our mobile honeyclient framework.

3.5 Management Design Goals

3.5.1 Logging

Logging the specific actions on the each layer of our honeyclients is critical to discovering novel attacks. Each layer has different logging requirements that enable us to learn more about the unique methods in which attackers hack mobile devices. For the network layer, it is important for us to log the entire frequency spectrums of each protocol. This means capturing WiFi management frames, Bluetooth Link Layer frames, and GPS signals. For Bluetooth and WiFi, this requires the use of specialized spectrum and protocol analyzers that can capture the entire 2.4 GHz spectrum and decode the protocol specifications.

For the transport layer, we must log activity on the particular transport layers using specific networks tools such as tcpdump and hcidump that log transport layer traffic on the TCP Protocol and Host Controller Interface respectively. This type of logging differs from logging activity on

```
<BTMAC>00:23:6C</BTMAC>

    <RFCOMMPort>
      <Port>13</Port>
      <Script>./behaivor/rfcomm/rfcommServerSDP.py</Script>
    </RFCOMMPort>

    <PSMPort>
      <Port>10</Port>
      <Script>./behaivor/l2cap/l2capServer.py</Script>
    </PSMPort>

    <BTClassVal>0x400200</BTClassVal>
    <HCIVersion>0x4</HCIVersion>
    <HCIRevision>0xFB</HCIRevision>
```

Figure 5: Partial XML configuration for a mobile honeyclient that has a Bluetooth radio with Logical Link Control and Adaptation Layer Protocol (L2CAP) and Protocol Service Multiplexing (PSM) Ports.

the application layer, where our application scripts offering services must perform individual logging. Thus, a honeyclient offering an unauthenticated connection must log when users attempt to connect to that service. Next, we will want to log access to fictitious user resources are accessed. Thus, if our simulated users connect to a fake corporate web app or mail server, we must log when a device in the vicinity of our honeyclient attempts to connect to that same fictitious resource.

Finally, we also want to maintain an situational awareness of what is happening in the vicinity of our simulated mobile devices. For example, our system may simulate a mobile device connected to a Wired Equivalent Privacy (WEP) network. We want to be able to identify when that access point in that network has been compromised to reveal the WEP key. In one method of compromising WEP, an attacker artificially sends thousands of IV packets in an attempt to break the key. Detecting this known attack and logging the key compromise allows our system to identify that all future traffic between the simulated mobile device and simulated access point is now able to be decrypted by an attacker. That is information we require as we research how attackers attempted to compromise our device.

3.5.2 Virtualization

The MITRE Honeyclient Project (www.honeyclient.org) provides an open source framework, designed to create and manage implementations of Honeyclient systems. The MITRE framework uses VMWare virtual machines to represent individual honeynet clients. By abstracting the honeyclients from the overall system, they reduce the risk of an attacker compromising the entire framework instead of just the unique honeyclient.

We assume we will receive both intelligent and brute force attacks against our simulated devices. Some attacks may simply deny service to the device. However, other attacks may result in a security compromise of the mobile device. Such a compromise should appear to give unlimited access to the devices, its services, and filesystem. The simulated device should respond exactly as an actual device would respond to the same attack. The attackers transactions on the filesystem is of particular interest to us. We want the ability study any code an attacker removes or plants on the newly compromised mobile device. To succeed at this, we must be able to safely virtualize a mobile device without compromising our overall system. We can mitigate the risks to our management system by virtualizing honeyclients in a system such as VMWare or XEN. This allows the honeyclient to be removed from the actual system managing our framework. Therefore, a root compromise in any device does will most likely not result in a root compromise of our overall system.

3.5.3 Remote Management

Remote management is a key component of our system. We want to be able to place the device in a hostile environment. Our physical presence may discourage attackers. This means we will not be able to safely monitor our device in this environment. Because our system serves no legitimate function, any attempt to contact mobile devices in the framework needs to be logged. That logging needs to be accessible to an off-site remote management location. At the remote location, we can investigate the logs to determine the attacker's intentions. For Bluetooth, WiFi, and GPS we want to log traffic to our specific mobile devices. Therefore, our remote management communications must not interfere with the communications we are logging. Thus, we use a 3G Data card in our system to provide remote callbacks for system management.

4. IMPLEMENTATION

4.1 Hardware

The hardware for testing honeyM included a small form factor computer with five WiFi radio cards, five Bluetooth radio cards, a frequency analyzer, and a USB GPS device. This allowed us using a single host platform to create four mobile honeyclients at any time. In order to change the OUI address of the WiFi cards and Bluetooth radios to mimic the OUI address, we required the use of specific cards. Further, the WiFi cards required the ability to be placed into monitor, managed, and master modes to support properly scripting traffic. Finally, we used the frequency analyzer to properly deconflict traffic on the 2.4 GHz wireless spectrum. Table 1 lists the hardware devices used in our experimentation.

Table 1: Hardware equipment used for mobile device honeyclients.

Use	Equipment
WiFi	5x USB AirPcap Tx WiFi Radios
Bluetooth	5x USB DBT120 Bluetooth Radios
GPS	1x generic USB GPS Device
Frequency	1x WiSpy 2.4 GHz Spectrum Radio
Management	3G Data Card
Logging	Artigo PC, 1 GHz, 1 GB Ram, 64 GB HD

4.2 Software

We installed Ubuntu 8.10 on our small form factor Artigo PC. Linux allowed us the most flexibility in logging traffic on the protocol stacks, generating artificial traffic, and masquerading as mobile devices. For Bluetooth traffic generation and logging we used the Linux BlueZ stack with HCITools (bluez.org). For WiFi traffic, we used Kismet (kismetwireless.net), Scapy (secdev.org) and LORCON (802.11ninja.net). These tools provided us the ability to log and generate 802.11 traffic at layer2.

4.2.1 Programming Languages

The entire source code for honeyM, including all behavior, traffic, and logging files is scripted using the Python programming language. We chose to use a scripting language that will allow future flexibility for others to modify our work. Further, Python allowed integration with the Scapy packet manipulation library. Scapy is a powerful packet manipulation library that can decode and forge a wide variety of protocols, including Bluetooth and WiFi. For example, figure 6 depicts partial Scapy code used to create an 802.11 frame containing a reply to a domain name request. We have included this specific functionality in our initial release of honeyM as well as several other scripts that utilize the Scapy library.

```
# Create an 802.11 Packet
dnsResp = Dot11(type = "Data", FCfield = "from-DS",
addr1 = addr1, addr2 = addr2)

# Append DNS Reply to 802.11 Packet
dnsResp /= DNS(id = id, qt = qt, qd = qd,
an = DNSRR(rrname = rrname, ttl = ttl, rdata=rdata))

# Send Packet
sendp(dnsResp)
```

Figure 6: An example of a Python Scapy script that creates an 802.11 packet containing a DNS reply.

4.2.2 Logging Software

We used multiple software packages to log traffic directed towards our mobile honeyclients. Table 2 lists all the software used in logging our honeyclients. For WiFi, we used Kismet to passively capture 802.11 layer2 traffic. Kismet supported exporting the captured data to a standard packet capture (PCAP) file that allowed us to import it into Wireshark for further inspection. While the Kismet tool continually swept all channels in the 2.4 GHz WiFi spectrum, we used Wireshark to log the specific traffic on each individual card. Additionally, we used the WiSpy commercial product to log wireless activity on the 2.4 GHz frequency. Figure 7 depicts the WiSpy tool measuring the activity of our honeyclients.

For Bluetooth, we used the HCIDump logging utility included with the BlueZ stack. HCIDump read only the Host Controller Interface (HCI) data coming from and going to our Bluetooth devices. While it is possible to log traffic on all layers of the Bluetooth Protocol stack, it requires specialized and costly equipment such as a protocol analyzer. In contrast, HCIDump provides us general information about the HCI events that occurred such as Remote Name Reads and Remote Inquiry Requests. While this limits the amount of information we can detect about Bluetooth attacks, recent work by Spill et. Bittau has shown promise for being able to simultaneously record all frames on the Bluetooth Spectrum [12].

Additionally, the specific application layer scripts allowed logging of traffic. The RFComm service trap originally listed

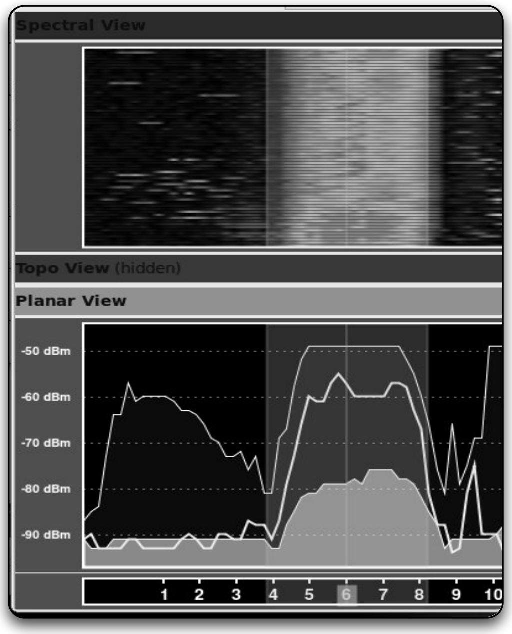


Figure 7: WiSpy 2.4 GHz frequency logging tool depicting an attack on Channel 6.

in Figure 4 logs all traffic intended for the advertised RFComm service. Similarly, any TCP, UDP, SCO, or L2CAP ports offering service are logged by the specific application layer script. To log GPS traffic, we used the gpsd server that reads NMEA sentences from our GPS unit. Python provides a nice interface to this called, gps.py that allowed pulling gps information in real time. Thus we could tell the altitude, date time, mode position, status, and number of satellites at any given point during our testing.

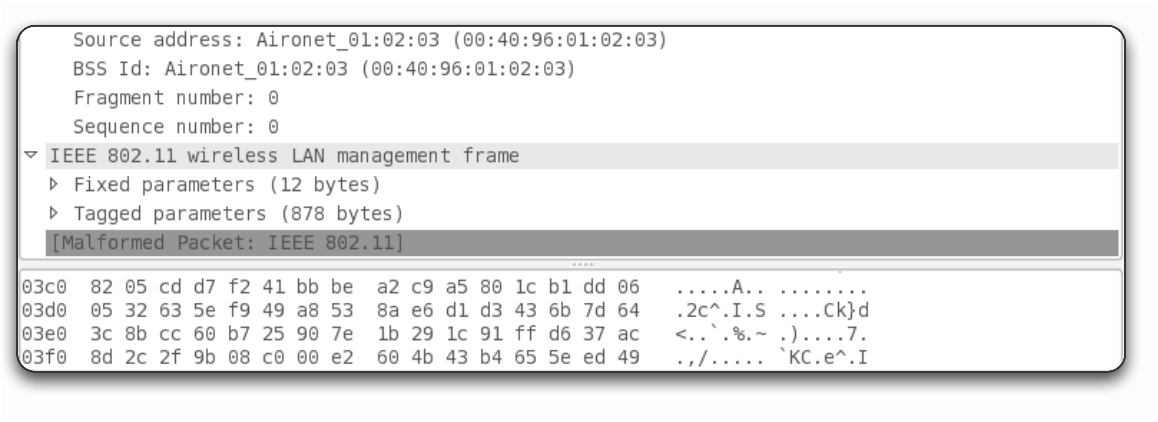
Table 2: Software used for honeyclients.

Tool	Description
Kismet	Logs layer 2 traffic on 802.11 protocol
HCIDump	Logs Bluetooth HCI layer interaction
SpecTools	Logs 2.4 GHz Wireless Spectrum
Wireshark	Logs TCP and UDP traffic on WiFi cards
gpsd	Logs GPS traffic.

5. EVALUATION

We evaluated honeyM by utilizing several different mobile device fingerprinting tools. This included the BTScanner, hcinfo, and btaudit software under Linux, the Bluetooth Explorer console under Mac OS X, and the BlueScanner, tool under Windows. All tools reported seeing the discovered devices as the actual intended mobile devices. Figure 8 depicts a Bluetooth scanning utility identifying our honeyclient as an Apple iPhone device. Since our device profiles were built upon extensive investigation using nmap and btaudit, we ensure we are correctly deceiving adversaries.

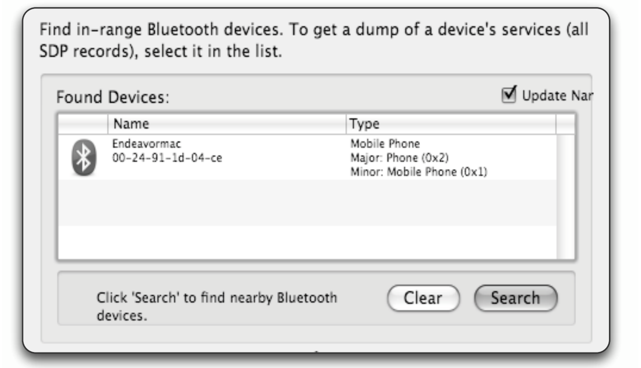
Next, we tested our honeyclients by employing them in two different hostile environments, the DEFCON Hacker



```
    Source address: Aironet_01:02:03 (00:40:96:01:02:03)
    BSS Id: Aironet_01:02:03 (00:40:96:01:02:03)
    Fragment number: 0
    Sequence number: 0
▽ IEEE 802.11 wireless LAN management frame
  ▷ Fixed parameters (12 bytes)
  ▷ Tagged parameters (878 bytes)
  [Malformed Packet: IEEE 802.11]

03c0  82 05 cd d7 f2 41 bb be  a2 c9 a5 80 1c b1 dd 06   .....A.. ........
03d0  05 32 63 5e f9 49 a8 53  8a e6 d1 d3 43 6b 7d 64   .2c^.I.S ....Ck}d
03e0  3c 8b cc 60 b7 25 90 7e  1b 29 1c 91 ff d6 37 ac   <..`.%.~ .)....7.
03f0  8d 2c 2f 9b 08 c0 00 e2  60 4b 43 b4 65 5e ed 49   .,/..... `KC.e^.I
```

Figure 9: A Malicious WiFi Probe Response sent to a honeyclient during during the SANS wireless security course. During this phase students attempted to compromise devices by fuzzing various devices with malformed 802.11 Probe Responses.



Find in-range Bluetooth devices. To get a dump of a device's services (all SDP records), select it in the list.

Found Devices: ☑ Update Nar

Name	Type
Endeavormac 00-24-91-1d-04-ce	Mobile Phone Major: Phone (0x2) Minor: Mobile Phone (0x1)

Click 'Search' to find nearby Bluetooth devices.

Figure 8: Deception of a Bluetooth service discovery tool, showing a honeyclient as a mobile phone.

Conference and the SANS Wireless Security Course. In both environments, it was clearly advertised that all wireless and mobile technologies were susceptible to attack and there was no expectation of privacy among users. We first tested our framework at the SANS 617 Wireless Ethical Hacking, Penetration Testing, and Defenses course. During SANS 617, the students used tools to exploit WiFi networks, Bluetooth devices, WiMAX and proprietary wireless systems. This provided us an outstanding opportunity to refine advertising our mobile honeyclients and ensure we were logging the appropriate level of traffic. Figure 9 depicts an attack on the WiFi radio of a honeyclient during the SANS Wireless Security Course. In this particular attack, a student attempted to perform device fuzzing by sending malformed probe responses to the wireless radio of the device. Fuzzing a wireless radio is a frequent method used by hackers to find new exploits that cause a device to fail or act in an unpredictable manner.

Following SANS, we employed our mobile honeyclients at the DEFCON 17 hacker conference. We advertised our clients as the phones and mobile devices of high visibility personnel in the Capture the Flag room, where hackers were competing in a contest to attack different computer systems. As show in Figure 10, several attackers attempted to query and read information from our devices. We saw several attacks on the 802.11 WiFi protocol as well. These included attacks such as KarmaMetaSploit, where an adversary would advertise a false access point and then inject attacks when a benign user connected to that access point. One particular fake access point was found next to the Wall of Sheep, including a visual depiction of users connecting to unsecured access points.

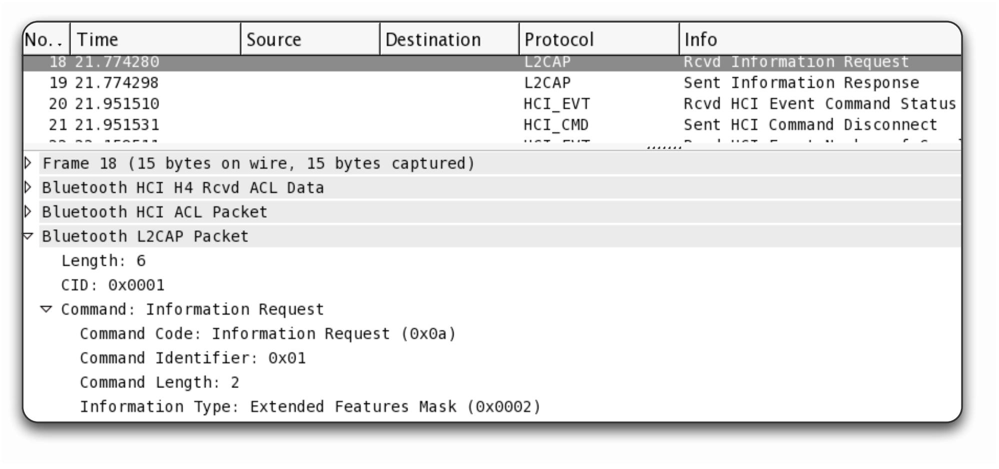
6. ANALYSIS

In this section, we address some of the limitations learning during our testing, the potential legal constraints concerning a mobile honeyclient framework, and the possibility to integrate this work into existing intrusion detection systems.

6.1 Limitations

Through the test environments at DEFCON and SANS, we learned methods for attracting attackers. For example, Bluetooth devices are easily advertised as it is inherent in the protocol specifications. When the device is set to discoverable, it broadcasts connection information. Further, the Bluetooth specification allows for service discovery and remote feature reading information, that allows a hacker to query the specific capabilities of a device. Through the use of these discoverable advertisements, we can easily attract a hacker. We learned in both environments that the more attractive the name of the device, the more likely a hacker is going to be to connect to it and try to penetrate it.

However, attracting hackers to attack our WiFi radios is much different. There are no means to set a WiFi radio to the equivalent of discoverable mode. Instead, we must transmit enough information so that an attacker listening to the entire BSSID or broadcast hears our information and considers it attractive enough to attack. For example, after repeated failure of attracting attackers at DEFCON, we started repeatedly broadcasting WLAN Probe Requests for WiFi Networks that do not exist. Knowing these networks did not previously exist until we broadcasted a request to join them, any subsequent Probe Responses we heard we then considered suspect. Another means was to create and then join networks with interesting Service Set Identi-



```
No.. Time          Source    Destination    Protocol   Info
  18 21.774280                              L2CAP      Rcvd Information Request
  19 21.774298                              L2CAP      Sent Information Response
  20 21.951510                              HCI_EVT    Rcvd HCI Event Command Status
  21 21.951531                              HCI_CMD    Sent HCI Command Disconnect
▷ Frame 18 (15 bytes on wire, 15 bytes captured)
▷ Bluetooth HCI H4 Rcvd ACL Data
▷ Bluetooth HCI ACL Packet
▽ Bluetooth L2CAP Packet
    Length: 6
    CID: 0x0001
  ▽ Command: Information Request
      Command Code: Information Request (0x0a)
      Command Identifier: 0x01
      Command Length: 2
      Information Type: Extended Features Mask (0x0002)
```

Figure 10: Evidence of a hacker remotely reading the features of a Bluetooth radio on a mobile honeyclient captured at the DEFCON Hacker Conference.

fiers (SSID). Thus, we created networks called *TopSecret, BestBuy-Corp, and AirForce-UseOnly.* These networks advertised connected clients by MAC address. Subsequently, attackers had knowledge of the existence of our WiFi radios. There are design considerations in both methods for advertising WiFi radios, most notably whether or not we want to appear connected to a network.

Advertising that a device is GPS capable is obviously much different since this protocol does not have inherent advertisement or broadcast mechanisms. Thus, we must advertise GPS devices either via the Bluetooth or WiFi mechanisms in order to attract a hacker. For example, the Apple iPhone has a GPS capability. We might want to broadcast a WiFi frame with the hostname of our device as *TJs iPhone* in order disclose to an attacker that we have a GPS capability.

6.2 Legal Constraints

There are two major legal considerations when employing honeyclients. In this section, we examine both and consider what is legal to record and what is legal to broadcast.

In an ideal honeyclient framework, we would record the entire wireless spectrum to detect attacks. This is obviously illegal and could be construed an illegal federal wiretap because we would gain access to information from multiple sources that did not intend for us to receive their communication. Thus, we must settle for attacks targeted only towards our specific mobile honeyclients. In states with ”one-party consent” statutes, we can do within the framework of the law. Nevada, where DEFCON was held, is a *one-party* consent state and thus we were allowed to record all traffic as long as we were party to the conversation. Twelve other states require *two-party* consent for wiretapping. Maryland, the location of SANS Wireless Security, is one of the twelve *two-party* states. However, the nature of what we want to record is a mitigating factor at least. We are recording malicious activity, where we are the recipient. Would a prosecutor hold a victim responsible for taping harassing phone calls to her residence?

The second legal consideration to our framework is changing the MAC addresses of our radios in an attempt to deceive attackers. Each MAC address contains an OUI address that is registered to a specific organization. In fact, this is how we attract attackers. We pretend to have a MAC address from Apple or Nokia in an attempt to lure attackers. However, borrowing these addresses and broadcasting them on a large scale would certainly violate Federal Communications Commission (FCC) rules governing the use of those addresses. This is certainly an open legal problem and one that must be considered when emplacing any type of honeyclient. Wired honeyclients face the same obstacles as the devices must borrow MAC addresses in order to emulate thousands of clients on the network.

6.3 Towards Integration into IDS

The intent for our work is to create a tool that collects novel attacks against mobile devices so we can add those signatures to our existing intrusion detection systems and firewalls. Thus, we must address the feasibility integrating signatures into intrusion detection systems.

In 2004, a tool dubbed HoneyComb takes the data captured on honeypots and applies pattern-matching techniques and protocol conformance checks on multiple levels in the protocol hierarchy to network traffic [6]. The system then creates precise traffic signatures that can be autonomously imported into intrusion detection systems and host based firewalls to prevent future attacks. Combined with multiple honeypot sensors, this can be a very effective means to hardening the defense of a wired network. A similar system could be employed on a wireless network.

The major shortcoming is that few intrusion detection systems exist for mobile devices. WiFi intrusion detection systems exist and have proven successful in defending attacks against the 802.11 protocol [7]. However, individual protocol intrusion detections for Bluetooth, and GPS are considered open academic problems. OConnor and Reeves created a signature-based Bluetooth Intrusion Detection system but no production level systems exist [9]. Further, we are unaware of any GPS IDSs. As we move towards a framework for creating mobile device honeyclients, we must examine the open problem of a mobile device intrusion detection system as well that treats the device holistically, collating all the communication vectors for that device.

7. CONCLUSIONS AND FUTURE WORK

In this paper we presented honeyM, a framework for creating virtual mobile honeyclients. honeyM mimics the behavior of mobile devices in order to deceive attackers and gain insight on unknown attacks. We demonstrated the increasing necessity to learn more about mobile device attacks by providing anecdotal evidence of recent attacks against mobile devices. We provided an overview of the design of the honeyM, including our layered approach for imitating mobile devices, the logging, remote management, and virtualization aspects of our design.

In our experiments we demonstrated that honeyM could simulate several different vulnerable mobile devices and deceive multiple mobile device scanning and detection tools. Specifically, we evaluated honeyM by placing it into two different hostile environments (SANS and DEFCON), where hackers attempted to attack the mobile honeyclients.

In conclusion, this paper demonstrated how honeyM could be used to detect novel attacks against mobile devices located at the periphery of our networks. As the processing power and storage capabilities of these devices increase, so will the need to protect the security of these devices. We presented honeyM as an attempt to combat this threat by gaining insight to new attacks.

The initial release of honeyM presented in this paper successfully mimics devices using WiFi, Bluetooth, and GPS as communications vectors. However, recent work has shown security vulnerabilities in WiMax, ZigBee, DECT, and other mobile device protocols. Future work will include integrating each of these protocols into the honeyM framework. Further, as individual device vulnerabilities are identified, they can easily be integrated into honeyM.

Acknowledgements

We would like to thank the following for their support, helpful ideas, and feedback: National Reconnaissance Office, West Point Information Technology and Operations Center, Greg Conti, Tom Cook, Erik Dean, Brad Haines, and Mike Kershaw.

8. REFERENCES

[1] N. Al-Gharabally, N. El-Sayed, S. Al-Mulla, and I. Ahmad. Wireless honeypots: survey and assessment. In *ISTA '09: Proceedings of the 2009 conference on Information Science, Technology and Applications*, pages 45–52, New York, NY, USA, 2009. ACM.

[2] S. Bratus, C. Cornelius, D. Kotz, and D. Peebles. Active behavioral fingerprinting of wireless devices. In *WiSec '08: Proceedings of the first ACM conference on Wireless network security*, pages 56–61, New York, NY, USA, 2008. ACM.

[3] R. Brooks. Wireless honeypots: Innovative software technologies to identify wireless attacks and perform risk management. *Raytheon Technology Today*, 2:14–15, 2007.

[4] H. M. El-Bakry and N. Mastorakis. Design of anti-gps for reasons of security. In *CIS'09: Proceedings of the international conference on Computational and information science 2009*, pages 480–500, Stevens Point, Wisconsin, USA, 2009. World Scientific and Engineering Academy and Society (WSEAS).

[5] P. C. Jussel. Corps tactical deception: Who's fooling whom? In *U.S. Army Command and General Staff College. School of Advanced Military Studies. Monograph*, page 44, Fort Leavenworth, KS, Sep 1990.

[6] C. Kreibich and J. Crowcroft. Honeycomb - creating intrusion detection signatures using honeypots. In *In Proceedings of the Second Workshop on Hot Topics in Networks (HotNets-II*, 2003.

[7] P. LaRoche and A. N. Zincir-Heywood. 802.11 network intrusion detection using genetic programming. In *GECCO '05: Proceedings of the 2005 workshops on Genetic and evolutionary computation*, pages 170–171, New York, NY, USA, 2005. ACM.

[8] D. Lowe. Airtight study at worldwide airports reveals wireless security risks for travelers and airport operation. Technical report, AirTight Networks, Chicago, IL, 2008.

[9] T. OConnor and D. Reeves. Bluetooth network-based misuse detection. In *ACSAC '08: Proceedings of the 2008 Annual Computer Security Applications Conference*, pages 377–391, Washington, DC, USA, 2008. IEEE Computer Society.

[10] N. Provos. A virtual honeypot framework. In *SSYM'04: Proceedings of the 13th conference on USENIX Security Symposium*, pages 1–1, Berkeley, CA, USA, 2004. USENIX Association.

[11] J. Ryoo, Y. B. Choi, T. H. Oh, and G. Corbin. A multidimensional classification framework for developing context specific wireless local area network attack taxonomies. *Int. J. Mob. Commun.*, 7(2):253–267, 2009.

[12] D. Spill, M. Ossmann, and M. Steward. Bluetooth, smells like chicken. *DEFCON 17*, 2009.

[13] M. Zalewski. Strange attractors and tcp/ip sequence number analysis. http://razor.bindview.com/publish/papers/tcpseq.html.

[14] K. Zetter. A hacker games the hotel. Wired News, Jul 2005.