CELLULAR BASEBAND SECURITY

A Thesis
Presented to
The Academic Faculty

by

Andrew Davis

In Partial Fulfillment
of the Requirements for the Degree
Bachelor Of Science in the
College Of Computing

Georgia Institute of Technology
May 2012

CELLULAR BASEBAND SECURITY

Approved by:

Assistant Professor Jonathon T. Giffin,
Committee Chair
College Of Computing
Georgia Institute of Technology

Assistant Professor Jonathon T. Giffin,
Advisor
College Of Computing
Georgia Institute of Technology

Assistant Professor Patrick Traynor
School of Computer Science
Georgia Institute of Technology

Date Approved: 27 April 2012

TABLE OF CONTENTS

CHAPTER I

INTRODUCTION

In recent years, the cellular phone has evolved into much more than a device for making calls. The combination of the PDA, digital media player, and cellular phone has become known as the smart phone. These smart phones are becoming more powerful with each generation and, with a big push from companies such as Apple and Google, are quickly becoming a necessary tool for many. The cellular network has seen numerous upgrades over the last few years to handle the increased load and demand required by newer, faster services. The mixed availability of services from region to region means that phones must support a wide array of protocols in order to ensure quality of service.

Due to the number and complexity of cellular protocols, baseband chipsets are used to handle all cellular network functions of these phones. A baseband chipset is often a physically separated system with its own CPU and operating system that communicates with the application CPU through either a bus or shared memory. These chips serve two purposes. They avoid having to redevelop cellular network functionality with each new phone and prevent the smart phone functionality from interfering with the cellular network.

The separation of the application and the baseband CPUs prevents user applications from directly manipulating cellular traffic. This separation is necessary because the cellular network was not designed with very strict security requirements. Cellular traffic can be encrypted, but there are few defenses between the phones and the towers. The phones trust that they receive well-formed and accurate information from the tower. When towers were very expensive, this posed little threat to the end users,

but today towers are cheap and can be carried in a backpack. Baseband systems offer little protection against memory corruption attacks lunched from malevolent tower as researcher Ralf-Philipp Weinmann has shown [27].

This thesis investigates the security testing of the baseband system. I developed a testing environment that is isolated to protect the existing cellular network. Using the testbed, I developed a fuzzer for SMS messages. I reversed the cellular baseband firmware of a popular smartphone using IDA Pro.

CHAPTER II

LITERATURE REVIEW

2.1 Software Memory Corruption

Memory corruption vulnerabilities plague the computing world. These vulnerabilities allow for an attacker to take control of the software running on a system. Although many defenses have been proposed and implemented, these vulnerabilities still exist in many systems today. Memory corruption vulnerabilities are the result of the programmer making a simple mistake with the low level management of the system. The more complex the system, the more likely the programmers are to make a mistake. Companies like Microsoft have developed processes such as SDL to help minimize the number of mistakes, but only a single mistake is necessary to compromise the system. The two categories of memory corruption vulnerabilities that lead to compromise will be discussed below.

Stack-based overflows are the result of improper handling of the stack resulting in compromised control flow. This type of vulnerability was introduced to the world in the paper "Smashing The Stack For Fun And Profit" by Aleph One [16]. In this tutorial, Aleph One walks through the dangers of the standard library functions in the C programming language. He demonstrates this on a common x86 processor and a Linux based operating system, but the basic principles have been applied to a variety of architectures and operating systems. The vulnerability is the result of the instruction pointer being pushed onto the stack when a function call is made. If this data is overwritten, then clearly the attacker has control of execution. Initially, these vulnerabilities were very dangerous as they were easy to discover and exploit. Today, this type of vulnerability is easy to find and repair, as well, and most traditional

computer systems have been patched. Cellular basebands, on the other hand, are large, complex systems that are expensive to maintain and are rarely updated, so they may still be vulnerable to this type of attack.

The second type of vulnerability is caused when the programmer mismanages the heap memory resulting in a heap overflow. This type of vulnerability was popularized in the paper "w00w00 on Heap Overflows" by Matt Conover and is much harder to exploit than a stack-based overflow but is also harder to find and repair [18]. Conover shows that overwriting the heap metadata can result in the attacker gaining control. Like stack-based overflows, heap overflows occur when the programmer has made an error. Conover acknowledges that there are no complete solutions for protection aside from fixing the code. Unlike stack-based overflows, heap overflows are commonly found in systems today.

Fuzzing is the process of giving the program many bad inputs, monitoring behavior, and seeing if a vulnerability can be found [13]. The two classes of fuzzers are known as smart and dumb fuzzers. Smart fuzzers are intelligent about the bad inputs it produces and attempts to get better results while dumb fuzzers produces random input or randomly perturbs good input. Many security researchers have used fuzzers to find security vulnerabilities in popular desktop applications. Microsoft has recognized the importance of fuzz test and have integrated it into SDL (security development lifecycle) [13].

2.2 Embedded Systems Security Research

Embedded systems are hard to define, but they are generally headless systems that are embedded in a device that serves a few dedicated tasks. Due to decreasing computing costs and increasing device complexity these systems have become more like traditional computing environments and are exposed to networks. These systems include a wide range of devices including printers, routers, network cards, ATMs, and

cellular basebands. Researchers have shown that many of these systems are designed without security in mind. Arrigo Triulzi and Guillaume Delugré have shown that network cards have flaws that can lead to compromise [26, 19]. They demonstrated that exploits in a network card's firmware can lead to the attacker having control of the network card and thus all of the data the computer sends and receives over the network. Their work mostly focuses on the specifics of the network cards beginning tested, but the impact is the same as that of a baseband attack. These systems are harder to patch than traditional software and many people are not aware of their existence. Although not as apparent to the user, these systems play a critical role.

Embedded systems are harder to analyze for security vulnerabilities than more traditional applications. They are often complicated and interact very closely with their specific hardware. Debugging them often requires special software and hardware. These systems are protected as trade secrets and the source code is rarely available. They are also a lot harder to defend. Currently systems are designed to trust their embedded systems and have little to no defense in the event that one is compromised. Researchers Loïc Duflot, Yves-Alexis Perez, and Benjamin Morin have demonstrated that the integrity of a network card can be monitored to detect attacks [20]. They implemented a series of run time firmware integrity checks and were able to detect an attack. They concluded that while it is possible to monitor the integrity of a system's firmware, it is still a hard problem and is very system specific.

2.3 Baseband Security

The baseband system in a cellular phone is responsible for communicating with the cellular towers on behalf of a phone. It is most closely related to a network card in a computer. Weinmann demonstrated the first public attack on the baseband of a smart phone at the Chaos Communication Congress in Berlin [27]. His research revealed that most systems lacked even the most basic forms of exploit mitigations

and were very vulnerable to attacks. He also confirmed that an attack tower can be constructed on a modest budget and can be made fairly portable. Then he showed that an attack could take control of the cellular baseband and deliver a range of payloads from recording all traffic to turning the cell phone into a bug that records audio from the microphone and sends it to the attacker. Later, these attacks were shown to affect feature phones as well by researchers Nico Golde and Collin Mulliner [22]. They used specially crafted SMS messages to attack a wide variety of cellular phones with a similar setup to Weinmann.

2.4 GSM

GSM (Global System for Mobile Communication)[1] is a set of standards for cellular networks and for devices to communicate with these networks. These standards have had several additions over time and although many have their own names, most are colloquially called GSM.

The GSM has been the target of attacks in the past. Many of these attacks have been on different protocols specified by GSM itself. Most of these attacks, however, have resulted in a denial of service attack or compromising a message or call [25]. These vulnerabilities have not attacked a user's cellphone, but instead compromised the data it sends. Although, a baseband attack does involve the GSM network layer, the exploited target is the processor in the user's cell phone not the network.

2.5 OpenBTS

The OpenBTS project is an open source implementation of the GSM stack [9]. When coupled with a software radio, it provides a programmable base transceiver station which allows us to have full control of the cellular network. This control enables us to send malformed data without the fear of interfering with the public's cellular service. While, the OpenBTS system is complicated to setup and run, "OpenBTS for Dummies" [17] and the OpenBTS documentation [11, 4] provide documentation

for the project.

2.6 Reverse Engineering

Another method for finding memory corruption vulnerabilities is reverse engineering which involves reconstructing the program from the compiled binaries and attempting to find the programmer's mistakes. It is a time-intensive practice, but this is how Weinmann found baseband vulnerabilities in the past [27]. The standard tool used for reverse engineering is IDA Pro sold by Hex-Rays [6]. This tool contains little documentation, however Chris Eagle has written a comprehensive book on the tool [21].

2.7 Conclusion

Since their introduction, memory corruption vulnerabilities have plagued software. They have resulted in the loss of countless dollars and resulted in attackers compromising numerous systems. These vulnerabilities are well understood on traditional computing systems and defenses have been in development for years. Until very recently, however, little work had been done in the public sector with the security of embedded systems to prevent the same attacks. Given the rise of the smart phone and the fact that more and more people and businesses are relying on them every day, the security of these systems is critical. The baseband of the phone is its link to the rest of the network, and controlling it results in controlling the phone. This critical component has little protection today, but may be easy to harden.

CHAPTER III

METHODS

3.1 Reversing the Baseband

Since little information was available on the cellular baseband of the HTC Dream (commonly known in the United States, and hence referred to, as the G1) reversing the baseband's firmware was critical for understanding the inner workings of the system. The primary tool used for almost all reverse engineering of software today is IDA Pro, and it was used extensively in the reversing of the baseband system. The baseband on the G1 uses the ARM architecture which is supported by IDA Pro. Additional plugins were written and purchased to aid in the reversing. The goal was to use strings found in the source and disassembled functions to locate standard C library functions known to cause security vulnerabilities. Using IDA Pro's graphing ability, I determined which function has the greatest indegree on the call graph. I needed to write automated scripts to process portions of the code because the disassembled baseband is over 250MB. This was too large for a timely manual inspection.

3.2 Creating the Testing Environment

A testing environment is required to launch a baseband attack. The environment must be able to simulate the over-the-air connection between the tower and the cellular phone. This connection in GSM is handled by the Um interface which is responsible for the communication between the towers and the phones. In order to simulate this connection, I used OpenBTS with appropriate hardware. The OpenBTS project utilizes a software radio to implement the Um interface in an open source Unix application. This allows for complete control of the protocol through software so the

environment is not limited to baseband attacks. The OpenBTS system can be used for any application that requires the manipulation and control of the Um interface.

3.3 Using Information from Reversing

Any vulnerable functions discovered in the reversing above can be used to construct an exploit. This process involves determining what data sent by the tower is processed by the vulnerable functions and corrupting it. The goal is to cause the memory in the system to become corrupted and to gain flow control. If I can gain control of the instruction register or overwrite a function pointer, then I have control of the service.

3.4 Fuzzing

A fuzzing setup was contracted to aid in the discovery of vulnerabilities. A dumb fuzzer will send either random data or slightly perturbed good input to the phone in the hope that it will crash. A smart fuzzer is a program that can generate the data based on some set of rules. I reprogramed OpenBTS to function as a fuzzer and attack any phone on its network. I was hoping the baseband would crash while processing some corrupt data. A crash while parsing data is a good sign that memory corruption has occurred. Given the nature of fuzzing, it was necessary to automate this process. It is important to remember when fuzzing that the bug is the point of interest, not the crashes. Not all crashes lead to attacker control of the system. If control cannot be gained, a denial of service exploit may be possible depending on if/how the operating system attempts to recover.

3.5 Understand Capabilities and Design of the Baseband System

In order to understand the impact of a baseband attack, it is first necessary to understand the capabilities of the baseband system. This varies from system to system so I limited my focus to the baseband of the G1. Important capabilities to examine

were its communication to the application CPU, memory access (preliminary research indicates that it may have DMA to all of the application CPUs memory), and cellular capabilities. Understanding of the capabilities and the operating system on the baseband came from a combination of reverse engineering and research. It was necessary to learn as much as possible about the workings of the baseband. Using the code recovered in the reverse engineering described above, any source code available online, the work already done by the jailbreak communities, and any documentation offered by the component suppliers were all necessary to get as complete of a picture as possible.

3.6 Application Operating System Defenses

The baseband system is installed by the manufacturer of the phone and is often inaccessible to the user. The application CPU is therefore the easiest place to install defenses. On the G1 the application CPU will be running Android which is open source and based on the Linux kernel, so the information about its design is extensive. The research done to determine the impact helped understand how the baseband interacts with the rest of the system. Safe-guarding these links will be the key to defense. There is most certainly little the application can do to prevent a baseband attack, but it could possibly be able to detect and recover from an attack to minimize the damage. This may require reworking the operating systems trust model to be able to treat the baseband system as hostile. This is time consuming to implement and would most likely require extending the Android kernel and interfacing with a proprietary system (the baseband). However, similar designs could provide guidance and it should be possible for a design to be established.

CHAPTER IV

RESULTS

4.1 Reverse Engineering

The baseband firmware was downloaded from an HTC support site. The firmware was not immediately recognizable as a standard executable format. It was suspected that the binary maybe an ELF file with a special header given other researchers had encountered this format. I confirmed this finding with an online post [12] and that the baseband was running the OKL4 microkernel developed by OK Labs [8]. This is a common kernel for embedded devices and in itself provides nothing but basic operating system functionality. I suspect the baseband functionality is provided by services running on top of the OKL4 kernel. Given that all of the baseband code was packed together in a single binary file, the amount of disassembled code was enormous. I developed some Python scripts that attempted to graph out the code to find functions of interest, but the results were inconclusive. The older version of IDA Pro I was using seems to have trouble dealing with the ARM code and the volume of code being processed. A newer version of IDA Pro was ordered, but I have yet to receive it. The Hex-Rays decompiler [7] was also ordered and may have proven useful, but I have have yet to receive it. Another idea was to compile the Open Source OKL4 Kernel and use Zynamics BinDiff [14] to locate interesting functions such as strcpy and memcpy, however, BinDiff requires the newest version of IDA Pro. Zynamics also makes a tool, BinNavi [15], that is designed to help with finding vulnerabilities in binaries, but again it requires the newest version of IDA Pro.

4.2 Lab Setup

A cellular testbed was developed as I carried out my research. A USRP 1 software programmable radio that was modified to operate on GSM frequencies was combined with the OpenBTS software stack. This gave us access to everything needed to emulate the traffic that occurs between a cellular radio and the BTS.

One problem with running such a lab is the legality of transmitting on the frequencies used by GSM. Without a license from the FCC it is illegal to transmit on those frequencies at the power levels necessary. Another issue is 911 service. Calls to 911 emergency services are referred to as SOS calls. When making an SOS call, a cellular phone does not care which tower it is transmitting to and all towers must accept any SOS call. This is why if you pick up any cell phone (even one that has not been activated) and dial 911 the call will go through as long as you are in range of a tower. This presents a problem with running our testbed. If our tower was given an SOS call, it could not route the call because we are not connected to the phone network. To prevent any incident, we place the testbed in a shielded box that prevents any signal from leaking out or getting in. This ensures that only test phones inside the box can connect to the network.

The network itself has had some issues. The OpenBTS software is very unstable and has many issues that had to be fixed or worked around. Once the software was stable enough to begin testing, we ran into an interesting issue with our setup. Every phone tested can see the network, but none can connect. We can confirm that it is indeed our network that the phones are seeing, because if we change the MCC and MNC (numbers that identify the carrier to the phone) the network changes names. For example, if we can set the MCC to 310 and the MNC to 410 and the phone see the network as AT&T and if we change the MNC to 260 it shows the network as T-Mobile. We have tried many combinations of phones and SIM cards and none seem to work. We are in ongoing communication with the company that sold us the

USRP 1 and are attempting to trouble shoot the issue. Our current best guess is there is something wrong with our hardware.

4.3 Fuzzing

Although the cellular testbed is not yet in a state were the fuzzer can be run, I did research and developed a fuzzer while waiting for the components of the testbed to arrive. The fuzzer was built by modifying the OpenBTS software to send malformed packets to any phone connecting to the network. Due to the nature of the setup, some significant changes had to be made compared to a traditional fuzzer.

Traditional fuzzers are attacking a piece of software and are running on the same systems as the target. They can use a debugger to determine if the software has crashed so they know when they have found something interesting. In our case, the fuzzer is completely separated from the target. To solve this problem, two changes were made from a traditional fuzzing setup.

The first change that was made was that all randomness was removed from the fuzzer. Fuzzers typically mutate data randomly, although they sometimes follow certain rules. The fuzzer I developed ran through its test deterministically and, most importantly, repeatably. To do this, I use a list of bytes that are commonly significant for program control such as a null byte and the using a loop: take a valid SMS message, send the message with a byte replaced with the control byte, and repeat for every byte in the message. This limits to number of test cases run, but this was necessary given the constraints on the environment.

The second change involves the recording of the results. A traditional fuzzer such as Peach [10] uses a debugger to determine if a program has crashed. Then it may use a program such as !exploitable [3] or crashwrangler [2] to determine the likelihood of the crash being exploitable. In the case of our cellular testing, attaching a debugger was not possible. Instead, a normal (not fuzzed) SMS message was sent between each

test. The message contained an incrementing counter. This would allow us to trace back to determine which iteration caused the crash.

The GSM protocol is a very large attack surface for fuzzing. It is a very complex protocol with lots of extensions. For the purposes of our tests, I focused on the SMS protocol. The standard SMS protocol in the United States sends messages using the 7 bit GSM character set. I focused on fuzzing the message field with a mutation engine derived from a one discussed in "A Bug Hunter Diary" [23].

CHAPTER V

DISCUSSION

5.1 Impact of Successful Attack

If an attacker can gain control of a target's baseband then they have gained a significant amount of power. The baseband system controls all network access to the phone. With the rise in popularity of smartphones, more and more businesses have become reliant on them. There are many possible impacts of this type of attack.

5.1.1 Espionage

Espionage is the most obvious impact from a baseband attack. The attacker would be able to examine any and all network traffic from the device. Today, smartphones pull email, calendars, and files, often automatically. These would be intercepted by the attacker who could also make requests as if he were the user. End-to-end encryption in a high layer would help mitigate this attack. However, it is unclear what other accesses the baseband has to the rest of the system. It maybe possible for it to access the data on the smartphone in which case the attacker can simply steal the encryption keys. The amount of access the baseband has to the rest of the system is dependent on the design of the particular phone. The phone functionality of the baseband may also be used to spy on the user. The cellular phone can be activated without ringing the phone[27]. This effectively turns the phone into a bug.

5.1.2 Denial of service

Another attack that is a more likely threat, because it requires less work, is the launching of a denial of service attack the could disrupt cellular service. This attack could target specific users or attack all users in range. It would cripple the communication

of the users cellphone until it could be repaired or the user could find a replacement. This could be used to prevent a user from contacting emergency services in the event of an attack. The country of Georgia claims that Russia launched a denial of service attack in coordination with a physical one [24]. It is reasonable to believe that these types of coordinated attacks will become more popular as countries become more dependent on their networks.

5.1.3 Identity Theft

As smartphones become more prevalent, more functionality is being shifted onto them. Google has introduced google wallet which allows an Android smartphone to replace your credit card [5]. Credit card swipers have been used to steal card information for quite some time, however, if all of the information is on the phone, then the attackers can execute the attack wirelessly. This would allow an attacker to setup in a busy confined space such as an airport or shopping mall and steal the identities of any user in range.

5.2 Possible Attackers

There are two categories of attackers in most cyber attacks. Governments are investing heavily in cyber warfare and criminals continue to exploit new technologies for commercial gain.

5.2.1 Governments

It is well known that governments utilize the cellular networks to fight crime and spy on their citizens. It has never been confirmed but it is reasonable to assume that several governments have back doors into users cellphones and are able to use them for espionage purposes. In the case of monitoring domestic phones, the government would not need a baseband exploit since that control they network and to some extent the phones already. A baseband attack could be an effective surveillance system on

a foreign network.

5.2.2 Criminals

Criminals are continuously looking for new vectors to carry out attacks. They have proven to be quick to adapt to new technologies and have already started to carry out attacks on smart phones. Although, baseband attacks are an excellent vector for attack, they do pose some threats to the criminals executing them. These threats will be addressed in the defenses section.

5.3 Defenses

Currently, no defenses exists to protect against a baseband attack. Defending against these attacks is difficult because there has been little research done on them.

5.3.1 In Phone Defenses

There is little a phone can do to defend itself against a baseband attack. This is especially true if the cellular protocols themselves cannot be modified to provide strong authentication. One approach would be to add up to date exploit mitigations to baseband systems. These mitigations are present in desktop systems and have yet to prevent exploitation, but have increased the time and effort needed to develop an exploit. Another solution would to implement some monitoring functionality so that the integrity of the baseband could be monitored from the application CPU. This would require major changes and it is unclear how a monitor could be trusted if the baseband is compromised. All of these techniques have a overhead to them that would increase the complexity of the systems, requiring more powerful chips and more power, thus lowering the phones battery life.

5.3.2 Network Defenses

Since performing a baseband attack requires broadcasting, defense can be placed in the cell towers themselves. Towers can be monitoring for any rogue towers and can

alert the authorities when one is detected. There is little a tower can do for the user. They cannot reliably contact the user since the attacker now controls his phone. The carriers could blacklist the target and prevent them from getting network access, but if the purpose of attack was to deny service then carriers are only assisting the attackers. The network itself could be upgraded so that phones could cryptographically verify that they were connecting to a carrier tower, but this would require a major overhaul of both the cellular networks and the phones.

5.3.3 Proximity

The primary defense against this attack is proximity. An attacking tower must be in range of its target. The attacker is also transmitting a powerful radio signal that can be used to locate the tower. The farther the attacker must be from the target the more powerful the signal must be broadcasting. An attacker could mitigate this risk by using a directional antenna or using a very low power transmitter and getting very close to his target. The attackers can also remain mobile given that our entire testbed could easily fit in a backpack. Coupled with some batteries, an attacker maybe able to move before being caught.

CHAPTER VI

CONCLUSION

Everyday society relies more on cellular networks to survive. The next generation of devices are being built on these networks. For years, desktop systems have been plagued by security vulnerabilities that have been studied and mitigated. These vulnerabilities have begun to surface in smartphones and the mitigations for desktops have been applied. However, smart phones have additional systems that desktops do not. The security of these systems is largely unstudied, but they have already been deployed in the field. The baseband is an example of one such system where compromise could be disastrous.

REFERENCES

[1] "3GPP." http://3gpp.org/.

[2] "Crashwrangler." https://connect.apple.com/cgi-bin/WebObjects/MemberSite.woa/wa/getSoftware?bundleID=20390.

[3] "!exploitable crash analyzer - msec debugger extensions." http://msecdbg.codeplex.com/.

[4] "Gnu radio wiki." http://gnuradio.org/redmine/projects/gnuradio/wiki/OpenBTS.

[5] "Google wallet." http://www.google.com/wallet/.

[6] "Hex-rays." http://www.hex-rays.com/.

[7] "Hex-rays decompiler." http://www.hex-rays.com/products/decompiler/index.shtml.

[8] "Open kernel labs." http://www.ok-labs.com/.

[9] "Openbts." http://openbts.sourceforge.net/.

[10] "Peach fuzzer." http://peachfuzzer.com/.

[11] "Range networks openbts wiki." https://wush.net/trac/rangepublic.

[12] "Reversing radio baseband." http://www.reddit.com/r/ReverseEngineering/comments/grpzq/reversing%5Fradio%5Fbaseband/.

[13] "Sdl process: Verification." http://www.microsoft.com/security/sdl/discover/verification.aspx.

[14] "Zynamics' bindiff." http://www.zynamics.com/bindiff.html.

[15] "Zynamics' binnavi." http://www.zynamics.com/binnavi.html.

[16] Aleph One, "Smashing the stack for fun and profit," *Phrack*, November 1996.

[17] Apvrille, A., "Openbts for dummies." http://gnuradio.org/redmine/attachments/219/fordummies.pdf.

[18] Conover, M., "w00w00 on heap overflows." http://www.cgsecurity.org/exploit/heaptut.txt, 1999.

[19] Delugré, G., "Closer to metal: Reverse engineering the broadcom netextreme's firmware." http://esec-lab.sogeti.com/dotclear/public/publications/10-hack.lu-nicreverse_slides.pdf, 2010.

[20] Duflot, L., Perez, Y.-A., and Morin, B., "Run-time firmware integrity verification: what if you can't trust your network card?." http://www.ssi.gouv.fr/IMG/pdf/Duflot-Perez_runtime-firmware-integrity-verification.pdf, 2011.

[21] Eagle, C., *IDA Pro Book*. no starch press, 2nd ed., July 2011.

[22] Golde, N. and Mulliner, C., "Sms-o-death: from analyzing to attacking mobile phones on a large scale." http://mulliner.org/security/sms/feed/smsodeath_mulliner_golde_cansecwest2011.pdf, 2011.

[23] Klein, T., *A Bug Hunter's Diary: A Guided Tour Through the Wilds of Software Security*. No Starch Press, 2011.

[24] Markoff, J., "Before the gunfire, cyberattacks." http://www.nytimes.com/2008/08/13/technology/13cyber.html, August 2008.

[25] Traynor, P., McDaniel, P., and La Porta, T., *Security for Telecommunications Networks*. Advances in Information Security, Springer, August 2008.

[26] Triulzi, A., "The jedi packet trick takes over the deathstar: taking nic backdoors to the next level." http://www.alchemistowl.org/arrigo/Papers/Arrigo-Triulzi-CANSEC10-Project-Maux-III.pdf, 2010.

[27] Weinmann, R.-P., "All your baseband are belong to us." https://cryptolux.org/media/hack.lu-aybbabtu.pdf, 2010.

Cellular Baseband Security

Andrew Davis

21 Pages

Directed by Assistant Professor Jonathon T. Giffin

Memory corruption vulnerabilities have been exploited by attackers to compromise systems for years. Traditional computing systems have had years of defenses developed for them such as ASLR and DEP. However, these technologies have not been implemented in embedded systems. Embedded systems have great power in a computing system and control many critical functions. If an attacker can compromise one of these systems, such as a network interface, then it is a serious threat. In a cellular network the network interface is known as the baseband. Cellular protocols are far more complex than traditional network protocols, so a few companies offer systems that can be integrated into a phone. The cellular network is becoming the primary communication network for many and disrupting cellular communication can have a devastating impact. I hope to find a memory corruption vulnerability in the baseband of a cellular phone and develop defenses to protect the phone against it.