The Pennsylvania State University

The Graduate School

Department of Computer Science and Engineering

CHARACTERIZATIONS OF VULNERABILITIES AND COUNTERMEASURES IN ADVANCED METERING INFRASTRUCTURE COLLECTORS

A Thesis in

Computer Science and Engineering

by

Adam Johannes Delozier

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

May 2011

The thesis of Adam Johannes Delozier has been reviewed and approved* by the following:

Patrick McDaniel
Associate Professor of Computer Science and Engineering
Thesis Adviser

Trent Jaeger
Associate Professor of Computer Science and Engineering

Raj Acharya
Professor of Computer Science and Engineering
Head of the Department of Computer Science and Engineering

*Signatures are on file in the Graduate School

Abstract

Electrical distribution and measurement techniques have been very cumbersome and inefficient for the utility companies for many years. Customers have also been affected by paying fixed rates whether they are using energy during grid downtime or peak hours. By integrating current technology, protocols, and interconnecting measurement devices to the Internet, a utility company can read meters on-demand and generate electricity more efficiently. With a high demand for this technology and a fast-paced production environment, many vendors are rapidly developing products using substandard security implementations with a single point of failure known as the gateway. If compromised, the gateway can disrupt power for regions or cause significant energy theft. In this thesis, we present a collection of vulnerabilities discovered, map each of them to an attack tree to reach our goals of compromising the gateway platform or denial of service, and provide countermeasures to prevent these types of attacks.

Table of Contents

List of Tables

List of Figures

Acknowledgments

I am most indebted to my advisor Dr. Patrick McDaniel. He has provided significant guidance and insight throughout my time at The Pennsylvania State University. With his optimism and patience during this thesis research, I was able to pursue success without hesitation.

I would like to thank Dmitry, Sergei, and Steve for their continuous support making this research possible. Also, I would also like to thank all members of the SIIS lab for their contributions from their vast knowledge in the field of security.

I have always been supported by my family and friends even though I was not always able to be with them. Their willingness to endure my absence is greatly appreciated.

Above all, I would like to extend my deepest gratitude to my beautiful wife. Without her infinite patience, I would never be where I am today. She has supported me through very difficult times and is always encouraging me to pursue all of my erratic endeavors.

Chapter 1

Introduction

With an era where computing is becoming pervasive, many products and services are being interconnected to provide convenience to both consumers and industry. One outdated service running an analog control system that has been recently receiving much needed updates is electrical distribution. With a major overhaul in its infrastructure, the electric grid is facing key component upgrades, including meters, which will provide different means of communication to its supplier. The new infrastructure, referred to as the smart grid, will include meters and aggregation devices, called the collector or gateway, which are connected together to record and communicate energy consumption information back to the utility company.

Prior to the smart grid development, meters were typically read monthly (or less frequently if estimated) by a utility worker that would visit each house. Now, an electrical supplier will be able to remotely read a meter periodically over a specified interval. With this capability, the supplier will be able to monitor a home's electric usage, producing a load profile at any given moment as well as charge the consumer based on the time-of-use (TOU) pricing scheme. With these advantages, the electric companies are now able to produce and distribute electricity more efficiently [13]. Many other capabilities exist with smart meters ranging from remotely disconnecting a consumers' meters, typically for back payments, to pushing firmware updates to the devices. This infrastructure provides

convenience to both the utility and the consumer, and as with any additional convenience introduces additional vulnerabilities and attack vectors. This massive deployment of the Advanced Metering Infrastructure (AMI) has significant financial support by the Department of Energy (DoE) [23].

Although we continually defend against threats in our computing infrastructure, we have not been able to provide a panacea to resolve all security concerns. All of the threats that have been affecting our computing infrastructure will now become vulnerabilities that can affect a new, vital resource, our energy supply. Prior to the smart grid, one of the few "attacks" that was typically performed by a consumer was energy fraud. Generally, a home owner would take a high risk in altering the meter on their house to read less energy than they were consuming. Any individual can find various articles on the Internet, such as Chilton's tutorial [6] on how to set up magnets to slow the meter down. The attack goal was to save money on their electric bill. Although this was successful for many individuals resulting in billions of dollars in loss [16], new technology will allow this attack and many others to be accomplished with a significant reduction in risk and likely from an area located far from the physical device by many more consumers.

A trivial attack would allow the malicious entity to manipulate their own electrical usage data without physical tampering of devices. This malicious entity could falsify meter data via the software that is executing on the device. If an attacker could construct an exploit for this vulnerability in a form that allows mass distribution, they could profit from selling this exploit to any consumer using a similar smart meter, thus broadening the attack coverage. The energy provider would not only lose money by means of falsified

data, but would have to incur the cost of updating all devices running software with the same vulnerability, referred to as the billion dollar bug [17]. Although these scenarios seem to only negatively affect a utility company a much more vivid scenario involves consumers and industry alike.

Allowing attackers to manipulate our electrical distribution can place any nation in danger given enough resources and a homogeneous AMI deployment [21]. A powerful attacker could construct an exploit to shut down the grid for many large regions for an extended period of time, which could result in loss of heat, food, and other essential resources and ultimately the death of many individuals. Any skilled malicious entity would attempt to attack a single point of failure for the AMI. This single point is represented as the gateway device in the network. The gateway can be a privileged and more powerful meter or an independent aggregation device. By attacking this central point, the malicious entity will be able to control sections of a neighborhood depending on how many meters are on the gateway's network. The two main objectives of a skilled attacker would be to compromise the integrity of the gateway platform or perform a Denial of Service (DoS) attack on the gateway affecting its corresponding, controlled meters and communication with the utility company. These objectives can be broken into subgoals and the subgoals can also be represented by multiple goals. This deep structure of goals can be accurately represented in attack trees discussed in Chapter 2.

In this thesis, we explore actual vulnerabilities discovered in a currently deployed system and its devices, assess the impact of these vulnerabilities against attack trees

where the goals align with a attacker's overall objective, and provide insight into recommended countermeasures. Penetration testing the smart grid exposes critical, undiscovered vulnerabilities and provides a concrete plan for implementing countermeasures.

1.1 Advanced Metering Infrastructure

The smart grid infrastructure is an emergent technology that promises to provide consumer and supplier-related benefits to electric grids around the world. By using existing back-haul network technologies such as GPRS, the telephony infrastructure, or the Internet, the smart grid provides a easy transition for electricity providers to integrate their grid to the new metering infrastructure. With the support of the DoE, massive deployment is being adopted by major cities in the United States. Other countries, such as Europe, have readily adopted this revolutionary movement in energy distribution as well. AMI is a network of sensors that provide energy usage data to the utility remotely and more frequently than the current analog deployment. This usage data is also available to the consumer and can be monitored with tools, such as Google PowerMeter, to track energy over time, predict the cost, and reduce the amount of energy consumed [10].

1.1.1 Devices and Infrastructure

The AMI infrastructure and smart grid deployment can be broken into three main components: smart meters, gateways, and a command and control server. Smart meters are devices that reside at a consumer's residence or a business facility. These smart meters are very similar to prior analog or current digital non-smart meters in

appearance but contain common or proprietary CPUs, micro-controllers, storage, and multiple communication interfaces. Meters in the United States reside in a socket-based enclosure inside the residence while in Europe they typically reside inside the residence in a non-socket enclosure. In addition to measuring and recording usage, smart meters can log power-related events, deliver log and usage data to the utility command and control server, and receive/send control-related messages such as remote disconnects, regulation requests for smart appliances, and much more [20].

1.1.2 Meter Networks

Millions of meters are being deployed rapidly [31] and require a sub-network architecture since it would be nearly impossible to interface each meter with a dedicated link to the network backbone. Currently, two major network topologies exist for meter deployments. In a deployed environment that uses the model represented in Figure 3.1 (B), one enhanced meter would have additional computation power and act as a gateway device aggregating the other meters' data. This deployment relies on a radio frequency (RF) mesh network. In a mesh network, each meter will act as a repeater in order to sustain greater coverage area from each meter that does not double as a gateway. If one meter does not double as a gateway, a dedicated device acts as a master or root node as illustrated in Figure 3.1 (A). This independent device has the same responsibilities as the enhanced smart meter but typically communicates via power-line communication (PLC) as opposed to its wireless mesh network alternative. By using PLC, the meters can reuse its power resource to signal commands and data over the power line. Devices in the PLC configuration use a star network topology. Note that networks within the

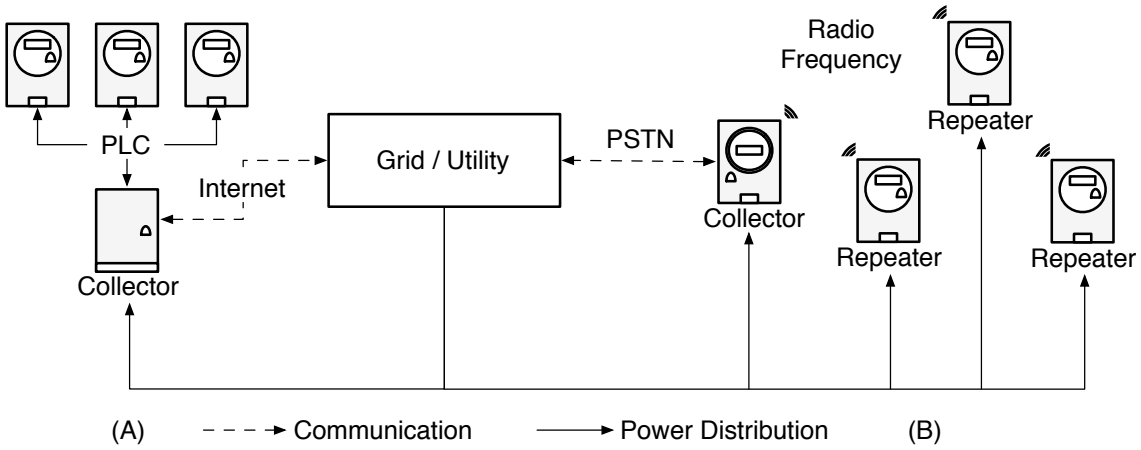

Fig. 1.1 Two example AMI network configurations. In (A) a power line communication (PLC) local network connects meters and a dedicated collector node. The collector communicates with the utility over the Internet. In (B) meters act as radio frequency repeaters to a collector node which itself functions as a meter. A backhaul link to the public switched telephone network connects the collector and utility [20].

same utility and region will be homogeneous and will be running the same software and firmware.

Either infrastructure setup in Figure 3.1 requires a connection to the back-end command and control server. This server typically resides in a utility office and is controlled by office personnel. This server would run on a typical Windows, Linux, or a similar commodity operating system and would contain its own vulnerabilities inherent to the OS and any additional software operating on it. This server will contain proprietary software used to control the gateway and meter devices currently deployed.

1.1.3 Device Maintenance

Maintenance for thousands of devices is a very complex scenario for small and large utility providers alike. Maintenance activities can be broken into two sections. Local (Site) maintenance is needed for installation of devices, executing commands that

are not available through the remote interface, and troubleshooting devices when network connectivity has been lost. Site maintenance is completed by a utility worker that will be running specialized software typically on a standard, consumer laptop. Each device is equipped with an optical probe that emits and transmits light probes. These optical ports operate at ANSI standard physical layer protocols [3] and can be access via a terminal client such as PuTTY [34]. Remote maintenance on devices including but not limited to remote disconnects, resetting meter usage, applying advanced payment, and firmware upgrades can be done at a utility office via the command and control server. This command and control server will deliver control messages to the devices requesting an action be performed.

1.2 Security

1.2.1 Systems Testing

Large systems such as the smart grid need to be evaluated for security during and after production to ensure the highest integrity during operations. During the software development life-cycle, many systems are Quality Assurance (QA) tested for features that are required by the system but often security is not considered a feature. Since designers, developers, and management need to plan for security in advance, security is generally left behind when the systems are given hard deadlines for deployment. If a company does not meet their deadlines, they will not be competitive in the market. Thus, security is generally not a requirement or even an afterthought. Hard deadlines are also driven by the motivation of implementing patches and bug fixes after vulnerabilities

are discovered and the system is already in production. Protecting and patching software vulnerabilities after the system has been deployed is considered Application Security [18]. Unfortunately, these vulnerabilities have already been discovered by third parties and exploited for their own benefit. Although this technique allows for the system to be deployed faster, it potentially allows a system riddled with vulnerabilities to be made public for malicious entities to take control away from the primary stakeholders. Many resources, such as [2, 19], exist for guiding security parallel to software development.

Ideally, the design plan for systems would allow time for security to be implemented and tested through the majority of the software life-cycle; however, this is not a common approach. In addition to allotting time, the designers and developers would have to be familiar with common security practices. Even when the system is designed to be secure, incorrect implementation of algorithms or protocols could negate any effort spent on securing the system. Another approach needs to be considered where security design, analysis, and implementation is supplemented with penetration testing.

Penetration (pen) testing includes but is not limited to static and dynamic analysis of source code, protocols, and configuration of a system. Pen testing is not unfamiliar to many security researchers and is typically used to audit systems after they have been deployed by a security consultant. Although, pen testing is most often completed after the system has already been implemented and deployed, some say that pen testing should be completed at component or unit level prior to integration as well. Although, this thesis addresses mainly penetration testing, security techniques such as external review, risk assessment, and static analysis should be completed during the software life-cycle [4].

1.2.2 AMI Security Implementation

Security generally goes unnoticed until vulnerabilities are exploited on a massive scale. From the Code Red worm [5] that infected thousands of machines to the more recent Stuxnet worm that targets industrial software, attacks are becoming more targeted and infecting more hosts. Stuxnet infected over 50,000 Windows systems and at least 14 Siemens control systems [30]. The AMI needs to be protected similar to any other critical infrastructure, possibly even more so than the others. The physical impact of exploits could have on the smart grid are astronomical and could have irreversible consequences. The government has recently stepped in and provided standards for analyzing risk, predicting and preparing for possible intentional failure, and an overall "analytical framework to develop effective cyber security strategies" for the smart grid. The *Guidelines for Smart Grid Cyber Security* is a three volume report released by the National Institute of Standards and Technology (NIST) in September 2010 [36]. This guide is a step in the right direction, but many major companies have already released and deployed systems throughout some of Europe and the United States. In order to utilize this guide, these manufacturers would have to step back in their development and maintenance cycles to use the NIST guidelines. Also, all the devices that are currently deployed will most likely not receive hardware upgrades or firmware upgrades any time soon. These upgrades will be made as needed due to the cost of upgrades for the utility company.

The system tested in this thesis implements several security mechanisms such as encryption, hashing, one-time keys, and generally has an overall good security posture.

However, this system has several short-comings, as do most, even with these mechanisms in place. There was no general insight into the security model used to design and implement this system or the resource requirements but some simple late life-cycle pen testing would have solved most of these security holes discovered. Although pen testing can be a rigorous process, completing pen testing throughout the software development life-cycle can help our critical systems infrastructure. General unstructured pen testing may work well for attackers that have unlimited time and resources, but generally a goal-oriented, structured security plan needs to be developed where the threats are well-defined. To define a threat, one must put themselves in the mindset of an malicious entity. By enumerating the possible attack scenarios and what data or control the adversary may want, we can define a structured list of threats which can be represented in the form of a tree [37].

Chapter 2

Methodology

In this section, a structured approach to black-box testing is established. With a structured pen testing plan, vulnerabilities will be mapped to attacker goals and we will have a clear exploit of a specific resource. By narrowing the scope to specific goals, pen testers can focus on particular sections of the infrastructure and analyze that section in a depth-based approach rather than a broad, breadth-based approach which may have already been covered in the design or implementation of the system.

2.1 Vulnerability Testing

Vulnerability research can be a very complex process where standard pen testing tools, as complied in Linux live CD distributions such as BackTrack [27], are not always sufficient for finding and generating all possible exploits in a system. While these common pen testing tools will find standard known bugs and implementation mistakes, many different approaches need to be taken when doing an in-depth analysis of a system.

One common implementation error involves improper boundary checking. These exploits, known as buffer overflows, take advantage of the software by writing data over the expected range of memory and overwriting the return address. The return address is typically overwritten to a place where known malicious code is stored and later executed. Discovering buffer overflows is rather cumbersome when attempting to

find them manually, so fuzz testing, e.g. fuzzing, is used to generate and mutate data to test compiled software automatically. The advantage fuzzing has over robustness testing is that robustness testing usually does not apply randomness where necessary. Regression testing also is affected by the pesticide paradox where essentially the more you test software the more stable it becomes to the pre-established test cases [33].

If source code is provided, a static analyzer tool such as Klocwork Insight [14], provides a way to automatically detect common coding mistakes that lead to security flaws such as buffer overflows. However, since this tool uses pattern-based detection some unknown mistakes and custom protocol implementation flaws will go unnoticed. To look into implementation details of closed source software, it is often necessary to reverse engineer binary files. Reverse Engineering (reversing) is another common technique applied to find out additional, undocumented details about particular software or data files. Reverse Engineering generally involves taking an unknown binary file, typically executable formats, and disassembling them to understand the underlying software better [8]. Whether the code is a shared library or a proprietary executable, the information that can be obtained from reversing can provide insight into where the system is vulnerable.

In addition to fuzzing and reversing, an attacker might write custom tools to make the system behave in an unexpected manner. In order to perform any of these tasks, the attacker needs to understand the functionality of the system and be able to differentiate between expected and unexpected behavior. Now that some techniques for attacking a system are established, it is necessary to structure them in a goal-oriented way to define a path to success.

2.2 Attack Trees

Bruce Schneier presented the concept of attack trees in the Dr. Dobb's Journal in 1999 as an alternative to model threats against a system. Schneier defined attack trees to provide a "formal, methodical way of describing the security of systems, based on varying attacks" [29]. Attack trees have a historical backing where fault trees were used in analysis in numerous government systems. For example, the U.S. Nuclear Regulatory Commission published the *Fault Tree Handbook* shortly following the incident at Three Mile Island [38]. The root node of the tree is the attacker goal and the children on different levels of the trees are ways to achieve the goal. An attack would be traversing up the tree from a leaf node to the root. Nodes can be joined with *AND* or *OR* conjunctions. If an *AND* conjunction is denoted between two child nodes both actions are required to proceed to a higher level closer to the overall goal. The *OR* conjunction simply means that only one of the nodes is required to proceed to the root. Although there can be an arbitrary number of leaf nodes, we try to establish a reasonable granularity for subgoals. After sufficient research is completed on a particular node, that node can be denoted with a "P" for possible or "I" for impossible. This technique will provide weight to the tree and illustrate paths of possible attacks given the resources available.

In the following sections, we focus on two particular instances of attack trees that target the gateway device in a smart grid deployment. The trees were created to narrow the scope of the capabilities to denying service and compromising the integrity of the platform. Although these trees are concrete attack trees, a vendor independent attack tree can be created to abstract vendor-specific details away to focus on multi-vendor pen

testing [22]. Figure 2.1 and 2.2 represent the denial of service and platform integrity compromise goal respectively. The vulnerabilities described in Chapter 3 allow us to traverse at least one path through the tree, i.e. from a leaf node to the root node, but do not span all nodes in the tree.

2.2.1 Denial-of-Service

One very common, realistic attack is Denial-of-Service (DoS). Distributed DoS (DDoS) attacks where multiple machines are utilized against a target, e.g. a botnet disrupting a particular server's resources, are fairly common and very practical but for simplicity, throughout the rest of the paper we will be discussing DoS attacks that are implemented directly from one machine to the target. Any amplification in the amount of resources via machines or additional hardware will provide more significant results on larger deployed systems. The target lab used during these tests consisted of two meters, one gateway, one attacker laptop, and one web server where the laptop and web server were running Windows. Each machine was equipped with average resources from commodity hardware at the time of use. By denying service to the gateway node, commands sent from the command and control server, such as remote disconnect, can be blocked to prevent the meters attached to the gateway from receiving these messages. In addition to preventing commands from being delivered to the gateway, the gateway will not be able to respond back to the server with usage data, logs, or error messages.

The DoS attack tree presented in Figure 2.1 was derived from the archetypal tree for Denial of Service in [22] and grafted for the gateway node as opposed to a smart meter. The DoS tree is broken into two primary subgoals of suppressing command delivery or

execution from the server. An example of a path to attain command delivery suppression and inherently gateway DoS, would be to attain the identification information for the gateway and submit the "alive message" back to the server. Once these two subgoals are attained, we can spoof the gateway. Other methods for command delivery suppression would be to DoS the command and control server or drop packets to the gateway either in the backhaul network or by receiving them through a man-in-the-middle attack.

The second main subgoal to derive attacks from is suppressing command execution. Three basic subgoals of suppressing execution are exhausting gateway resources, configuring incorrect settings, and halting or locking the gateway. Any of these subgoals can be used to suppress execution and ultimately perform a gateway DoS. One traversal from a leaf node to the root node would be to allocate maximum sessions or leverage a software bug in the gateway source code, e.g. exploit a buffer overflow vulnerability, to halt or lock up the gateway software. By traversing a path from child to root nodes of the concrete attack tree presented in this section, it is clear how to achieve our goal of a gateway DoS.

2.2.2 Platform Integrity Compromise

Similar to the DoS tree constructed in Section 2.2.1, another concrete attack tree can be derived from the goal of compromising the platform integrity. This attack tree is represented in Figure 2.2 and has three subgoals of updating the firmware via web services, uploading a firmware image over the optical port, or physically tampering the gateway device. Being able to update the firmware via web services would require control of the diagnostic tool or submission of a raw HTTP request to the web service

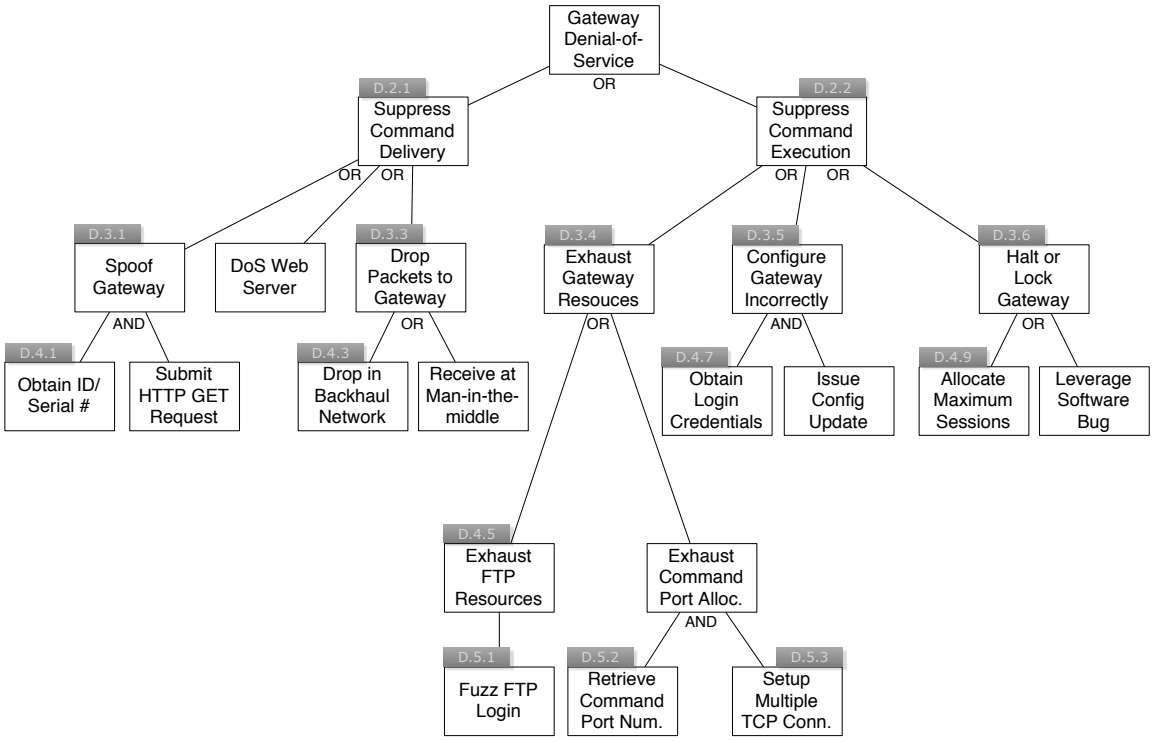

Fig. 2.1 Attack tree for goal of gateway Denial-of-Service (DoS)

in charge of updating the gateway's firmware. Controlling the diagnostic tool would require credentials and a valid image. Submitting a raw HTTP request would require obtaining an API Key via exploiting a code vulnerability or guessing the key, obtaining access to the utility providers LAN, and constructing a valid firmware image. Similar to controlling the diagnostic tool, to derive a path from uploading firmware via the optical port would require obtaining login credentials. However, since the optical port provides a different physical vector, a hardware sniffer could be installed or if a previous session was not closed correctly an attacker could use the that session. Physical tampering may pose a more difficult attack, but an attacker could replace the hardware with his own implementation that would provide additional functionality or remove functionality from the existing software. Although the trees presented are not an exhaustive list of possible

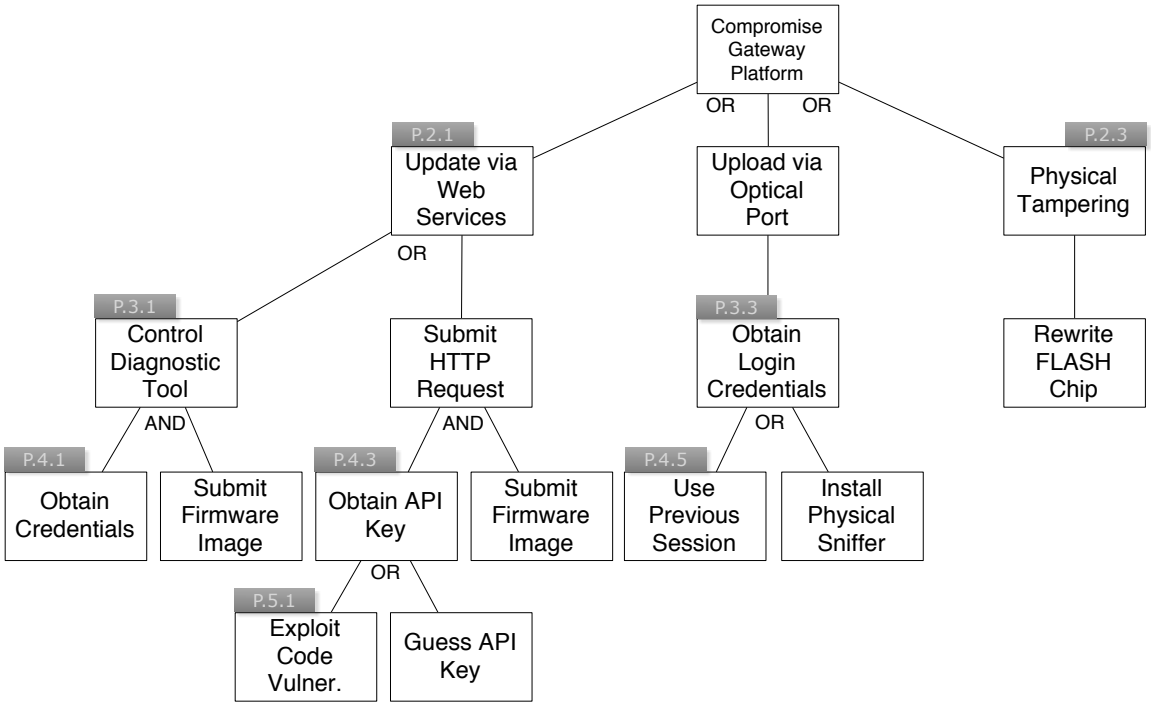

Fig. 2.2 Attack tree for goal of gateway platform integrity compromise

vulnerabilities, it presents a structured, goal-oriented approach to finding vulnerabilities via penetration testing.

Chapter 3

Vulnerabilities

Assessing the vulnerabilities in a system is a very challenging task. Although a security researcher is able to make an arbitrary endpoint to the assessment which could be a checklist of known vulnerabilities in design and common code or protocol implementation errors, general vulnerability assessment could contain no endpoint to a deployed system. For example, vulnerability researchers continue to find errors in Windows OS implementations everyday. Microsoft continues to release patches, however these patches could contain errors or make other sections of software behave incorrectly and pose a risk.

Since control systems, such as AMI, are very complex and involve multiple operating systems, different software suites, and standardized or proprietary protocols, the approach for vulnerability assessment should be done for multiple entry points. This assessment will focus on gateway vulnerabilities. Since the gateway is the control point, frequently deployed as even a single-point of failure in some deployments, it is a highly valued asset in the AMI. This device, as opposed to the smart meters, runs more expensive hardware and more complex software to aggregate meter data and administer a neighborhood of meters. As stated in the Section 1.1.2, some gateway devices can be one of the meters in the deployment or it can be an independent device. Although, the

primary focus is on the gateway, it is critical to understand the other components of the AMI and the interfaces that allow these components to communicate with each other.

3.1 Fingerprinting

When black box testing, the vulnerability researcher will need to find out preliminary information about the system. This information includes but is not limited to the type of Operating System (OS), commodity software versions, open ports, and any technical documentation that would allow further investigation on how the system operates. A common tool used to fingerprint the OS type and scan for all open TCP ports is nmap [1]. By using nmap on the SUT gateway, we were able to identify a control channel port where the web server sends requests. In addition to the control channel, SMTP and FTP were also open.

Since the command and control server is running as a web server, information can be obtained from the headers or 401 error pages. This information provides the type of web server and version information for further testing, unless the header information is forged which is rare. Skipfish, a web application security reconnaissance tool [39] was run against the web server to find any additional information from the web server. By fingerprinting the web server, we were able to obtain the OS type and web server version information.

Another technique for obtaining information is sniffing. Tools like WireShark [9] provide packet-level information on a variety of links. The link that WireShark was used on was the LAN where utility office workers would reside in a real system. The information that was recovered from the link was in plaintext. After reading further

documentation, the vendor is under the assumption that the utility will provide a VPN solution for external interfacing. However, if an attacker was able to probe inside the network, they would have full access to the commands being sent between the client and server. The traffic between the gateway and the web server was encrypted and the communication protocol was unknown. The authentication being used between the gateway and the web server is MSCHAPv2, which has not been broken yet.

3.2 Randomization

While analyzing security systems, one important feature to evaluate is non-standardized methods for random number generation. When software is used to create a pseudo-random number, entropy should be evaluated unless the system and protocol is generally accepted by the scientific community or standardized. If a standard protocol is unable to be used, the process of evaluating the entropy of the pseudo-random number generator (PRNG) should be rigorous and done by many individuals from the scientific community. Vendors should only implement their own cryptographic systems when absolutely necessary. When features or attributes, e.g. MAC address or timestamp, of the system are used in generation of the random number, these features should be masked or aggregated in a hash chain to avoid exposure of information or reduce the search space of the number generation. Failure to make an acceptable random number will give the adversary the advantage of lowering the search space until they are able to guess the number within a reasonable amount of time.

The Universally Unique Identifier (UUID) version 1 algorithm exposed information about the user's network interface card, but version 4, commonly used today as a

source of uniqueness, uses a random or pseudo-random device source for 15 out of 16 bytes. The remaining byte is used to denote the version number of the algorithm, which would be "4", and another predictable, reserved 4-bit value [15]. Microsoft has implemented a UUID version 4 algorithm for the .NET Framework that generates a UUID structure, referred to by Microsoft as GUID, given any random bit stream. By the Microsoft Communication Protocol Program (MCPP), the entropy of the byte stream can be very low and the GUID generation algorithm will not detect or correct low entropy generated GUIDs [24].

Figure 3.1 shows the plot of GUIDs over a 24 hour period. A script was created to request a new session key from the AMI web service, generated via the .NET Framework GUID class, every minute during the day. Although only one 24 hour period is shown, this test was performed several days and appears to have no uniform correlation or clustering of session keys. We can assume that the PRNG seeding the GUID algorithm is producing enough entropy. No specific randomization flaw was discovered, however the use of GUIDs is prevalent throughout the system and other issues presented in this chapter require general knowledge of their purpose.

Table 3.1 A list of vulnerabilities discovered from SUT.

Reference	**Description**	**Nodes affected**
V1	Web Services Key Distribution and Authentication	P.4.3
V2	Gateway Spoof	D.3.1
V3	Gateway Control Port DoS	D.5.3
V4	Optical Sniffing	D.4.1, D.5.2
V5	Optical Session Hijacking	P.4.5
F1	HTTP Fingerprinting	P.2.1
F2	nmap Fingerprinting	P.2.1, D.2.1, D.2.2

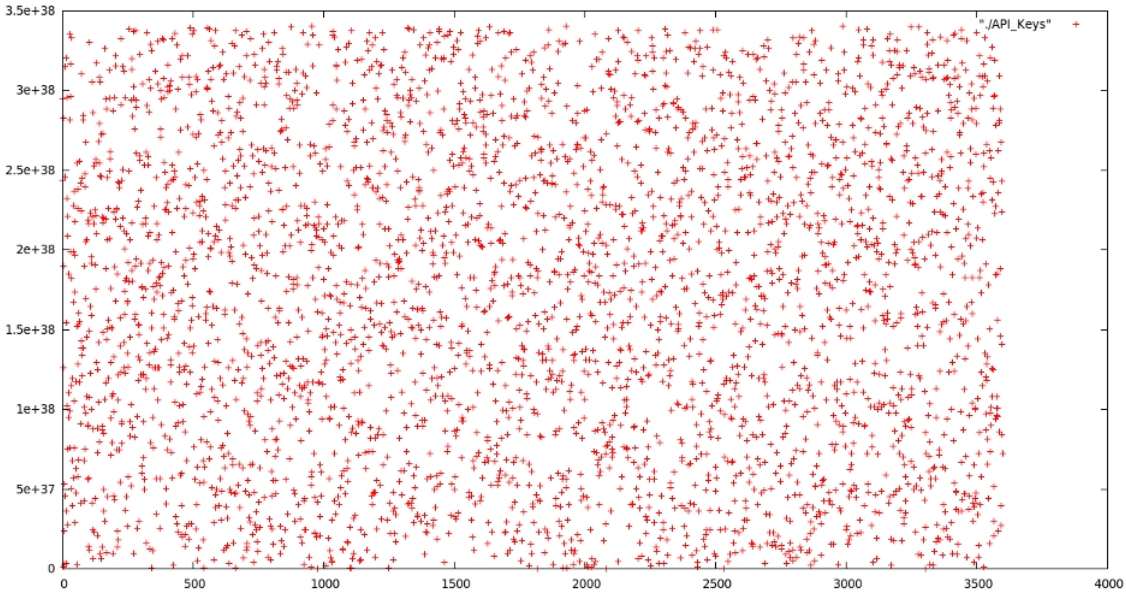

Fig. 3.1 A sample of session keys generated over a 24 hour period.

3.3 Web Services Authentication

One attractive option for businesses is to open a web services interface to the public. By using web services and some type of authentication token, the business could share information such as its inventory to allow the public to write their own software to query non-critical parts of their database. Even internal systems are now moving towards the ease of use that web services allow. The web server of the SUT has several web services available for utility office workers to control the system. The utility would simply need to write an interface that submits Simple Object Access Protocol (SOAP) messages to the web services running on the utilities server. Once a SOAP message is sent, it is queued to perform the task detailed in the message (e.g. remote disconnect, measurement, firmware update).

To prevent unauthorized individuals from sending messages to the control server, a user must provide an API key with any service request. The only exception to this rule is use of a user login request. A user login request contains the following three fields: username, password, and authentication type. The SUT uses GUIDs, see Section 3.2, to encode log message, control messages, and other various strings. It was discovered that there is only one authentication type GUID provided by the SUT and that is "Default Authentication". By testing this GUID in the third field, an API key was returned along with a success message. Reversing the web service code revealed that when the default authentication type GUID is defined the username and password field are ignored. The default authentication type is defined in a shared constants file that would be the same for every deployment and can not be changed since the checks are hard-coded into the services. This vulnerability with the web services key distribution and authentication will be known throughout as V1 (see Table 3.1).

3.4 Gateway Spoof

When the SUT gateway boots, it will send a ping message to the command and control server. Upon receiving a ping message, the server updates the corresponding database table with an IP address of the sender, current connection status, and last communication timestamp. Following the ping communication, the server attempts to make a connection to a default, but configurable, port on the gateway.

An adversary can send a ping message impersonating the gateway simply by spoofing easily attainable credentials, such as identifiers and serial numbers, passed via the HTTP GET method. Since this request is sent over plain-text and not under an SSL

connection, the information is accessible provided the attacker has access to monitor the network traffic with a tool such as WireShark [9]. The gateway database table on the server will update and the status remains "Connected" even if no further connections are established on default connection port.

The adversary needs to have access to the network where the web server resides. Depending on how the utility provider sets up their network, i.e. using VPNs, MAC Address filtering, and other filters, the adversary could use an ethernet cable connected to the gateway to obtain network connectivity if no filtering is set up. The adversary also needs the ID and serial number of the gateway they plan on spoofing. This information can be obtained via the optical port without being logged in. This information is transferred unencrypted over the optical port upon reboot of the gateway.

By spoofing the gateway, the adversary can prevent control messages of high or low priority from being delivered to a meter. An example scenario is as follows:

1. An adversary crafts a HTTP GET packet with a correct serial number and ID of a gateway.

2. The utility provider attempts to send a Remote Disconnect command to a meter via the gateway.

3. The message fails to get delivered to the gateway since it assumes the rogue, spoofed gateway is the correct gateway and determines no problems exists since the gateway status is marked as "SUCCESS".

Verification of this issue was made via a laptop residing on the LAN with a gateway, meter, and the server. The laptop sent multiple HTTP GET request, once

a second, using the simple TCP/IP network utility netcat [11] with the gateway's ID and serial number. The database residing on the server was then inspected for changes. Additionally, a remote disconnect request was submitted from the command and control server after the spoofed message was received and the remote disconnect, a high priority message, was not received by the gateway.

3.5 Control Port DoS

The control port discovered in Section 3.1 is one of the target vectors for stress testing. The gateway accepts command connections on a default port. This connection is established as part of the handshake protocol between the web server and the gateway. When an adversary attempts to create a connection to this port on the gateway without any subsequent exchange in communication, the DC forks this process off and in several minutes expires a stale connection and regains that resource. Once several connections are made by an adversary (this can be from the same IP), the web server's attempt to connect to the gateway are either dropped or queued. With the web server unable to send messages to the gateway, the adversary could prevent execution of remote connects/disconnects, remote firmware updates, load profile requests, and any other high or low priority commands.

By using netcat in a bash script, we demonstrated that by requesting continuous connections to the gateway on the default command port, we could stop any priority messages. It wasn't until the processes on the malicious machine were stopped and the web server sent an additional gateway connect command that the remote disconnect occurred. After using a very large amount of connections, we concluded that a minimum

of six connections is required to induce this state. The netcat connections would not expire on the DC and prevented commands from the web server until they were killed. This vulnerability is referenced in Table 3.1 as V3.

3.6 Optical Communication

One vector of attack that may go unnoticed in the initial security design plan is the local optical communication interface. The optical interface used on smart meters and gateway devices is a local interface for a service technician to communicate with the device. This port is embedded in the device and is easily accessed from the exterior shell. With a standard optical-to-USB cable [35], which can easily be purchased from an online distributor, an adversary can communicate with the meter through a terminal client, such as PuTTY. The optical port is used for local administration of the meters including reading data usage (if the utility is unable to read the device remotely), firmware upgrades, troubleshooting hardware/software, or many other built in capabilities. When adding additional communication vectors to a system, the vendor needs to evaluate this vector the same degree as any other. Since this vector implies that the attacker must interact with the device within close proximity, the utility company might require less stringent security requirements for authentication, integrity, or confidentiality.

Once a user logs into the console of the gateway, the default timeout is 60 minutes which is longer than most standard session timeouts. Unless the user explicitly types a command to terminate the communication, the session will stay alive for the default timeout period. This default timeout can be set to any arbitrary time other than the default of 60 minutes. Most utility workers usually do not participate in the design

of the software and may not realize the consequences of terminating the session and may bypass this by disconnecting the cable after their troubleshooting or maintenance is complete. If an adversary can persuade a utility company to send a utility worker out to complete maintenance on a device by physically disrupting the communication or making false claims about its stability, the adversary has a substantial window of opportunity to hijack the session the utility worker opens. Hijacking the session only requires the aforementioned cable and physical access to the device within the timeout period of a non-terminated session. By issue a command found in the help interface, the password for the device is displayed in plaintext to enable the attacker to login in subsequent times. The ability to hijack an optical session and obtain the password will be referenced as V5.

Since the communication between the optical port and the device attached via USB are not encrypting the traffic, which is typical over a serial connection, an adversary could construct some type of sniffing device and record the information including the login username and password. This sniffing mechanism could reside in the device with its own storage and the security flags that protect physical tampering could be ordered and replaced without the utility knowing [22]. This sniffing device that would be attached to the box could also tamper with a firmware upload that is taking place via the optical port. An adversary could detect transmission and corrupt the transfer giving them the ability to upload custom, malicious firmware. This vulnerability is referenced as V4.

Chapter 4

Countermeasures

Creating countermeasures for a system is a tedious process when some attacks may be unknown during the design and implementation phases. A vendor who is incorporating security throughout the software life cycle could set up what seems to be a flawless design, but there are many possibilities why pen testing still reveals flaws with systems. It is critical that a concrete plan be established for implementing countermeasures. By mapping vulnerabilities to attack trees, i.e. marking the node with a "P", we are not only are able to visualize a path to the target attack goal, but we can also establish what countermeasures are needed and where they exist in relation to the vectors.

The approach that should be taken when designing countermeasures from attack trees should be to correct the child node marked with a "P" closest to the root node, e.g. attacker goal. By providing a countermeasure that will be closer to the root node, the attack surface for the overall goal is reduced more significantly than if a countermeasure were provided lower in the tree. Once a node is covered by a countermeasure all children derived from that node will no longer be vulnerable. Ideally, a countermeasure that would protect the root node would be considered the best countermeasure for that tree; however, it would be extraordinarily difficult to provide a solution that would give coverage over the entire attack surface. Since our pen testing was completed after implementation

and was not involved during the design of the system, most countermeasures will be provided in the lower levels of the tree. To reiterate, the best approach to implementing countermeasures is continually considering security during the entire software life-cycle phase and targeting nodes that are higher in the tree.

Providing a countermeasure at higher nodes will require an abstraction of the general solution to solving security issues with the SUT and is possibly not feasible. The feasibility of the countermeasure could rely on the financial resources of the vendor or customer, or it could be that no countermeasure is currently available that would provide broad enough coverage. An alternative to providing a countermeasure at a higher node is to cover each leaf node in the tree until you exhaust all possible paths to the subgoal. Once a countermeasure is in place for an intermediate node, this will disrupt the path to the goal. This approach would not leave any room for additional goals that may fall under the same subgoal but were not anticipated when the attack tree was designed. Also, making many small patches to solve a higher problem may alleviate the immediate issues; however, this may lead to more vulnerabilities being introduced into the system. These additional vulnerabilities appear when adding "quick-fix" patches that when separate are tested to work and are secure, but when combined cause unexpected results. In Section 4.1, we examine some countermeasures for the attack trees described in Chapter 2 and reason why some of the attack surface would not be feasible to cover as a vendor.

4.1 SUT Countermeasures

4.1.1 Platform Integrity Compromise

The first attack tree that is explored for countermeasures is the Gateway Platform Compromise tree 2.2. One subgoal of this tree is updating the gateway firmware or configuration via the web services interface. The focus of the countermeasure will be to secure the web services interface and meet the requirements of confidentiality, integrity, and authenticity. Specifically, we want to ensure that the firmware image that was created by the vendor has not been corrupted. Making the web services interface security may require a compounded security solutions, i.e. layers of security. For authentication, if the system has a Windows domain, the user's account credentials can be passed to the web service or basic authentication, e.g. using username and password [26]. Although this provides a standard authentication mechanism for the user to access the utility web service, the data on the wire should meet the confidentiality and integrity requirements as well. The communication to the web server should be encrypted or any attacker with access to the internal network will be able to view and potentially modify any of the traffic between the client and server. The communication with the web server should be forced to use TLS/SSL [7].

Code signing is a technique to apply public key cryptography to ensure the integrity and authenticity of the code [25]. For the web services interface that is running on a server using public key cryptography to validate the new firmware image for a gateway is not nearly as expensive as performing this operation on the gateway's limited resources. To sign the code, the developer would sign the firmware image to be uploaded

with the single private key. The server would then validate that any firmware image it receives must be able to be read with the general public key that is available. Considering the size of firmware images are generally less than 1.5 MB in size, this process would not exhaust the resources of the server.

The second subgoal of the Gateway Platform Compromise tree is the ability to update the gateway platform via the optical interface. Similar to the generalized confidentiality issue of the web services, the optical port needs to encrypt the traffic to and from the gateway. The optical port on the SUT follows the ANSI physical layer standard however the application layer is proprietary. The IEEE currently has a proposal out for optical port communication on all 7 OSI layers; however, a preliminary glance at this standard reveals that encryption is not being pushed for this medium [32]. One solution to encrypt the traffic and prevent eavesdropping on the link would be to use Secure Shell (SSH). By using SSH, the exchange of passwords will be encrypted along with any data transfer preventing a sniffer from being used and would also prevent a session from being resumed by another device. Unfortunately, the code signing approach will not work well on the gateway, as it contains limited resources to perform the expensive algorithms involved in public key cryptography.

The final subgoal of the Gateway Platform Compromise tree is the ability of an adversary to physically tamper with the device. Depending on the deployment these devices could be locked away in storage boxes or mounted on top of poles. These feats alone will deter a typically adversary, but a sophisticated adversary with particular motivations will defeat these seemingly trivial current fixes for physical access. Unfortunately, we can not defend against every attacker goal, so vendors focus on a cost trade-off of

security features. Certainly, no utility company is going to pay someone to guard each gateway, so they are willing to risk physical access. This subgoal was added to the attack tree for thoroughness. Currently, the money to invest in physical tamper resistance technologies simply does not pay off for vendors.

4.1.2 Denial-of-Service

Being able to design countermeasures for the subgoals of suppressing command delivery and command execution is difficult at any stage of the systems life cycle. Due to the granularity, we will be focusing on providing countermeasures for nodes below these first subgoals. The first node we observe is the "Spoof Gateway" node. This node is a result of V2 and F1 in Table 3.1. To counter these vulnerabilities, we will focus on the software's implementation failure. The cause of V2 is a result of updating the database prematurely during the ping phase of communication. Aside from sending the request in the clear over HTTP, this is a bug that would require the web server only to update the database with the IP and information of the pinging gateway after it has confirmed the communication link and authentication has occurred successfully. The rest of the communication outside of the ping request is encrypted with the key of the gateway. To resolve the leakage of information during the HTTP request this initiation should be encrypted as well.

A solution that would protect many access vulnerabilities found in this system would be to implement IPSec. By using a layer 3 solution, application layer protocols that currently exist and are used since they are standardized would be protected if an attacker would hop on the link at a non-authenticated endpoint. With the current

push to deploy an IPv6 solution, IPSec will be more common since it was developed in conjunction with the IPv6 standard [12].

The next two subgoals of focus are the web server DoS and dropping packets destined for the gateway. The gateway does not take a direct countermeasure for these goals since they are dealing with strictly the interface for the gateway device. Being able to resolve a DoS attack directly on a server is a very difficult challenge. Making sure you have the correct infrastructure, including a DMZ, is good practice. Another way to deter DoS attacks is to use load balancing on the server and the machines on the edge of the network. If the utility workers are capable of accessing the command and control server from outside the facility, a VPN solution should be used and the link between the internal address and the web server should be encrypted using TLS/SSL or a similar solution.

The next set of countermeasures are generated for goals under the suppressing command execution tree. The first subgoal evaluated from this subtree is exhausting the gateway's resources. The vulnerabilities found under this node are specific to a implementation bug and information leaks. The stress testing on the FTP port via fuzzing the FTP login was unreliable to generate exhaustion; however, the exhaustion occurred several times. This exhaustion could be alleviated by providing the layer 3 solution, so that attackers would need to be authenticated before accessing this common application layer protocol. The information displayed over the optical port without logging in should be suppressed or left as a configuration that is turned off for deployment. It is fairly common that debug is accidentally left on in deployed environments.

The subgoal of configuring the gateway incorrectly requires an attacker to be logged into the device. This countermeasure was covered under the optical countermeasure in Section 4.1.1. In addition to authenticating securely, the system should be tested with a variety of configurations and provide a filter mechanism that only allows specific types and sizes of data for a particular configuration parameter.

The final subgoal off the command execution suppression node is halting or locking the gateway. The two issues at hand for this goal are as simple as a specific implementation bug to the most difficult of leveraging a software bug such as a buffer overflow. V3 can be fixed by correctly terminating invalid connections instead of leaving them in the buffer. Also, IPSec would prevent unauthorized users from connecting to the control port of the gateway. Leveraging a software bug, such as a buffer overflow, would take extensive static analysis and fuzz testing. The ability to protect against attacks, such as buffer overflow exploits, still plagues software from companies such as Adobe, Windows, and Apple. The ability to cover this node, as mentioned above, is not yet feasible. Though many techniques have been deployed to avoid overwriting the return address [28, 21], offensive techniques have also been used to defeat most of these countermeasures.

4.2 Summary

With market competition and desire from entities like the Department of Energy, many vendors are concerned with deploying their infrastructures quickly. The tight deadlines and non-standard security practices have created a plethora of issues that were exposed through penetration testing. Pen testing should be a process that is completed throughout the software life cycle in addition to other security checkpoints. Standards

groups, such as NIST and IEEE, are trying to guide the stability and uniformity of implementation for smart grid vendors; however, the standards are not being created as quickly as the products that are being released. By performing post deployment black box pen testing, we were able to identify vulnerabilities in a deployed system and generate real attack scenarios. The use of attack trees has guided the focus of pen testing and provided an general structure for where and how countermeasures should be applied. The countermeasures presented in this paper are targeted to specific patches for the SUT that need to be made and general security posture issues. Protocols such as IPSec and TLS are very promising, but have yet to become adopted by the majority of vendors. Effectively applying countermeasures in the AMI is essential to the movement of securing our infrastructure systems as we proceed into a pervasive, highly connected society.

References

[1] Nmap Reference Guide. http://nmap.org/book/man.html.

[2] Julia Allen, Sean Barnum, Robert Ellison, Gary McGraw, and Nancy Mead. *Software Security Engineering: A Guide for Project Managers*. Pearson Education, Inc., 2008.

[3] American National Standards Institute. C12.18 Protocol Specification for ANSI Type 2 Optical Port, 2006.

[4] B. Arkin, S. Stender, and G. McGraw. Software penetration testing. *Security Privacy, IEEE*, 3(1):84 –87, 2005.

[5] CERT/CC. CERT Advisory CA-2001-19 "Code Red" Worm Exploiting Buffer Overflow In IIS Indexing Service DLL. http://www.cert.org/advisories/CA-2001-19.html, 2001.

[6] Dan Chilton. Cut your electric bill in half without conserving energy. *DIY Life*, September 2007.

[7] Tim Dierks and Eric Rescorla. The transport layer security (tls) protocol. RFC 4346, 2008.

[8] Eldad Eilam. *Reversing: Secrets of Reverse Engineering*. Wiley, 2005.

[9] WireShark Foundation. Wireshark. http://www.wireshark.org/.

[10] Google. Google powermeter. http://www.google.com/powermeter/about/index.html.

[11] *Hobbit*. netcat. http://nc110.sourceforge.net/.

[12] S. Kent. Security architecture for the internet protocol. RFC 2401, 1998.

[13] Chris S. King. The Economics of Real-Time and Time-of-Use Pricing For Residential Consumers. Technical report, American Energy Institute, 2001.

[14] Klocwork. Klocwork insight. http://www.klocwork.com/.

[15] P. Leach, M. Mealling, and R. Salz. A universally unique identifier (uuid) urn namespace. RFC 4122, 2005.

[16] Electric Light and Power Magazine. Reducing revenue leakage. http://uaelp.pennnet.com/, 2009.

[17] Patrick McDaniel and Stephen McLaughlin. Security and Privacy Challenges in the Smart Grid. *IEEE Security & Privacy Magazine*, May/June 2009.

[18] G. McGraw. Software security. *Security Privacy, IEEE*, 2(2):80 – 83, 2004.

[19] Gary McGraw. *Software Security: Building Security In*. Pearson Education, Inc., 2006.

[20] Stephen McLaughlin, Dmitry Podkuiko, and Patrick McDaniel. Energy Theft in the Advanced Metering Infrastructure. In *Proceedings of the 4th International Workshop on Critical Information Infrastructure Security*, 2009.

[21] Stephen McLaughlin, Dmitry Podkuiko, Sergei Miadzvezhanka, Adam Delozier, and Patrick McDaniel. Embedded firmware diversity for smart electric meters. In *Proceedings of the 5th USENIX Workshop on Hot Topics in Security*, HOTSEC '10, 2010.

[22] Stephen McLaughlin, Dmitry Podkuiko, Sergei Miadzvezhanka, Adam Delozier, and Patrick McDaniel. Multi-vendor penetration testing in the advanced metering infrastructure. In *Proceedings of the 26th Annual Computer Security Applications Conference*, ACSAC '10, pages 107–116, New York, NY, USA, 2010. ACM.

[23] Rick Meritt. Stimulus: DoE readies $4.3 billion for smart grid. *EE Times*, February 2009.

[24] Microsoft Developer Network. Globally unique identifiers (guid): Quality of random bits. `http://msdn.microsoft.com/en-us/library/cc246028.aspx`.

[25] Microsoft Developer Network. Introduction to code signing. `http://msdn.microsoft.com/en-us/library/ms537361.aspx`.

[26] Microsoft Developer Network. Web service authentication. `http://msdn.microsoft.com/en-us/library/ms154673.aspx`.

[27] BackTrack Linux Organization. Backtrack linux. `http://www.backtrack-linux.org`.

[28] Gerardo Richarte. Four different tricks to bypass stackshield and stackguard protection. *World Wide Web*, 1, 2002.

[29] Bruce Schneier. Attack Trees. *Dr. Dobb's Journal*, December 1999.

[30] Bruce Schneier. Stuxnet. `http://www.schneier.com/blog/archives/2010/10/stuxnet.html`, 2010.

[31] SmartMeters. Echelon ships two million smart meters, March 2010. `http://www.smartmeters.com/the-news/853-echelon-ships-two-million-smart-meters.html`.

[32] EndDevic End Device/Telemetry Interface Unit Subcommittee. P1701 - standard for optical port communication protocol to complement the utility industry end device data tables. `http://standards.ieee.org/develop/project/1701.html`.

[33] Ari Takanen, Jared DeMott, and Charlie Miller. *Fuzzing for Software Security Testing and Quality Assurance* . Artech House Publishers, 2008.

[34] Simon Tatham. PuTTY. http://www.chiark.greenend.org.uk/~sgtatham/putty/.

[35] TesPro. Tp-usb optical probe for meter communication. http://www.tespro.com.cn/en/TP-USB.php.

[36] The Smart Grid Interoperability Panel – Cyber Security Working Group. Smart grid cyber security strategy and requirements draft nistir 7628, February 2010.

[37] H.H. Thompson. Application penetration testing. *Security Privacy, IEEE*, 3(1):66 –69, 2005.

[38] W.E. Vesely, F.F. Goldberg, N.H. Roberts, and D.F. Haasl. *Fault Tree Handbook*. U.S. Nuclear Regulator Commission, 1981.

[39] Michal Zalewski. skipfish. http://code.google.com/p/skipfish/.