CPRE/EE/SE 491
Senior Design

May1601 Project Plan

Submitted by

Jonathan Osborne	Team Leader
Nik Kinkel	Key Concept Holder
Korbin Stich	Key Concept Holder
Daniel Borgerding	Communication Leader
Jonathan Hope	Webmaster

Under the guidance of
Dr. Doug Jacobson

Prepared for
Alliant Energy

Fall 2015

Industrial Control System Honeypot and Traffic Monitor

Contents

1 Introduction **1**
- 1.1 Project Introduction . . . 1
- 1.2 Background . . . 1
- 1.3 Overview . . . 3

2 Requirements **4**
- 2.1 Overall Project Requirements . . . 4
- 2.2 Proposed Solution . . . 4
  - 2.2.1 Assessment of Proposed Solution . . . 5
- 2.3 Validation and Acceptance Testing . . . 6

3 Interface and System Description **8**
- 3.1 Technical Approach . . . 8
- 3.2 Design Specification . . . 9
  - 3.2.1 Layer 1: The External Interface . . . 10
  - 3.2.2 Layer 2: Low-Interaction Honeypot Micro-services . . . 11
- 3.3 Design and Implementation Stages . . . 12
  - 3.3.1 Stage 1 . . . 12
  - 3.3.2 Stage 2 . . . 13
  - 3.3.3 Stage 3 . . . 14
- 3.4 Functional Testing Plan . . . 15

4 Work Breakdown **16**
- 4.1 Project Schedule . . . 16
- 4.2 Risks and Feasability Assessment . . . 18
- 4.3 Cost Considerations . . . 18

5 Market Research **20**

6 Conclusion **22**

A Plugin Interface Details **23**
A.1 Honeypot Plugin Interface 23
A.2 Logger Plugin Interface 23

List of Figures

3.1 Simplified device internals 11

4.1 Tentative Fall 2015 Project Schedule Overview 16
4.2 Tentative Spring 2016 Project Schedule 17

A.1 Honeypot microservice plugin interface 23
A.2 Logging plugin interface 23

List of Tables

2.1 Validation and Acceptance Testing Plan 7

3.1 Functional Testing Plan 15

4.1 Fall 2015 Detailed Project Schedule 17
4.2 Spring 2016 Detailed Project Schedule 17
4.3 Summary of project components and implementation risk . . . 18
4.4 Summary of implementation costs 19

Section 1

Introduction

1.1 Project Introduction

Cybersecurity is critical in maintaining the integrity and reliability of the electrical infrastructure in our society. It is of utmost importance to ensure that our electric grid is resilient since it is one of the most complex and critical infrastructure which every sector depends upon. Recently, the roles of the electrical power sector have shifted and now companies have a huge responsibility in ensuring continued security and resilience of the power grid.

Attackers successfully compromised U.S. Department of Energy computer systems more than 150 times between 2010 and 2014. These numbers are expected to increase as computer systems begin to drive more of the power grid. According the Scott White,a professor of Homeland Security and Security management at Drexel University, "The potential for an adversary to disrupt, shut down (power systems), or worse ... is real here." As these threats become more persistent, so must be the defense. Our project focuses on on this defense of twenty-eight of Alliant Energy's power substations, to ensure any potential attacker is stopped as quickly as possible.

1.2 Background

With the dependency of electricity in the modern world, defending the functionality and integrity of electrical power plants is integral to the preservation and protection of our daily lives. Power plants handle extremely volatile resources on a continuous round-the-clock basis. In order to keep these systems up and running without fault they are monitored and controlled by numerous components on a SCADA(Supervisory Control and Data Acquisition) network. The integrity of this network is of vital importance. Normally these

networks are almost completely isolated from the outside world. However, should an intruder somehow gain access to this network, it is important that IT personal be notified immediately so the cause of the intrusion be identified and closed as soon as possible.

An ongoing cyber-espionage campaign against targets in the energy sector has given attackers the ability to sabotage operations against their victims. These attackers, known as Dragonfly, have managed to compromise a number of important organizations in the energy industry. Dragonfly follows in the wake of Stuxnet, targeting industrial control systems (ICS). Dragonfly has used trojanized software to deliver malware to nearly 2,800 known victims thus far. Their aim, as far as it is known, has only been for espionage purposes. However, should they have used the sabotage capabilities that were made open to them, they could have caused damage and or large-scale energy disruption in any of the many countries affected potentially costing companies millions of dollars in lost revenue. Because of such attacks, it is obvious that security for these kind of attempted breaches is decidedly necessary.

This is where Honeypots come in to play. A honeypot is a dummy network node. To an attacker, the honeypot looks like a poorly protected network; in reality, this honeypot is a fake system that is isolated from the rest of the organizations network and monitored closely by a security team. Honeypots disguise themselves as systems on the network, mimicking the behavior of other devices on the network while gathering information pertinent to identifying and nullifying the attack. Ideally, attackers on the network connect to the Honeypot and unknowingly provide information to the IT staff allowing them to interpret the source and content of the attack. The honeypot is able to catch specific types of network attacks directed specifically at certain SCADA network attacks. However it is unable to catch everything. For this reason honeypots are often combined with an intrusion detection system (IDS).

Typically, an Intrusion Detection System is used to detect and alarm on suspected intrusions. This is done using signature-based, statistical anomaly based, or protocol analysis detection. These detection techniques allow IDS systems to analyze network traffic and detect if there is any abnormal traffic in the network such as ping sweeps, denial of service attacks, and viruses/-worms etc. Any of these could potentially cause huge problems for a SCADA network because a honeypot alone would be unable to address these issues. Because of this it is advisable to combine the services of a honeypot and IDS to provide a variety of network protection.

1.3 Overview

The system we had in mind to ensure the best possible security for Alliant Energy would be to use a raspberry pi running with two separate network interface cards (NICs). One NIC would facilitate all the inbound and outbound traffic for all of the network and SCADA protocols we want to monitor. The second NIC facilitates all traffic for the passive IDS system. This port only listens for abnormal network traffic and reports it to the on-board logging service before sending the desired information to proper network security administrators. By using this design we fit the initial design criteria by implementing a fully functioning honeypot and a small IDS system to catch common types of attacks while also keeping the device cheap and easy to maintain.

This system will be comprised of layered services. The outermost layer of the device is external interface and the only point at which incoming network traffic is accepted. Access is controlled by an IPTables rule set to provide some resistance to attackers to help conceal the honeypot. What will be visible is an interface that any attacker will be able to interact with. Should an attacker attempt to login or exploit any of the services available through the aforementioned interface, the system will send alerts to a logger and alert the necessary IT staff.

The architecture in this system will be a flexible plugin architecture. Each micro-service(HTTP, HTTPS, SSH, etc.) will function as a plugin, and communicate with the logger plugin. This will create maximum flexibility and allow the system to be highly extensible. This is ideal because should an admin want to add a new micro-service or logging functionality, all they need to do is write a small plugin and the system will be able to handle it and begin running it.

A honeypot's greatest value lies in its simplicity, it's a device that is intended to be compromised. This means that there is no production traffic going to or from the device. Any time a connection is made to the honeypot, it is most likely to be a probe, scan, or even attack. Any time a connection is initiated from the honeypot, this most likely means the honeypot was compromised. We believe our honeypot will prove a viable security product. With increasing threats to not only the electrical sector, but the general population as a whole. It is entirely necessary to do everything possible to reduce or eliminate risks to an organization's critical assets; this is what our product aims to do.

Section 2

Requirements

2.1 Overall Project Requirements

The project we received from Alliant Energy was to create a low interaction SCADA honeypot system for 28 of their power substations. The system requirements for the device are as follows:

- Capable of interfacing with SSH, HTTP and HTTPS
- Have a fake login page with key logger to record login attempts
- Record logs of all connection attempts and send an alert to essential personnel upon irregularities such as increased ICMP traffic, port scans, and repeated attempts to connect to URLs.
- Rule set must be able to be fine-tuned to accommodate new/changed rules.
- Potential for a small intrusion detection system.
- Must be a low maintenance standalone device.
- Must be low power.
- Each of the 28 substations must be equipped with one.

2.2 Proposed Solution

To house the necessary software to meet these expectations we have decided to create a standalone system using a Raspberry Pi micro-controller. The

Raspberry Pi is our preferred choice of hardware since it is a small Linux based micro-controller that is low in cost and power requirements. However, we would like to have multiple network interface cards in order to have one port for implementing SSH, HTTP/S and other services and one for our intrusion detection system to listen on the network.

To accommodate Alliants needs we have decided to implement a plugin architecture that allows for easy customization of the device to suit Alliant Energy's needs. By doing this any micro-service can be easily added and removed based on the needs of the company. The current default plugins are SSH and HTTP/S protocols. The SSH server plugin will serve as a false gateway to inside the SCADA system. The illegitimate user will be able to enter in a username and password at which time the SSH verification handshake will be terminated, and a log will be filed with the relevant information to be passed on to a network administrator for handling. The web server plugin will offer similar services for HTTP and HTTPS. We have implemented a fake login page visible to the unknown user. The login page will always return invalid credentials and will be protecting nothing. Later, if we wish to make the honeypot more interactive we could make the username and password vulnerable to cracking methods and have it protect false data to see how the user is tampering with it. This information will be gathered and reported to the network administrator via an outside Splunk server in Alliants central data center. The benefits of using a Splunk server is that it is a tested and proven utility that performs the exact function necessary to efficiently alert the necessary employees. The Splunk services are also configurable to capture logs of ICMP ping sweeps, port scans and unwanted URL attempts via an on-board intrusion detection system listening on an open network interface card.

2.2.1 Assessment of Proposed Solution

A Raspberry Pi is an ideal component for the Alliant Energy Honeypot because of it's small footprint and low power requirements. However the Raspberry Pi only comes with one interface card. In order to properly implement any kind of intrusion detection system it was necessary to add at least one network interface card. The remedy for this is simply adding another aftermarket USB NIC for the IDS to run on. Using this configuration any direct attacks will go through the integrated NIC on the Raspberry Pi while while indirect attacks such as ping sweeps will go through the the USB NIC.

Raspberry Pis also provide only 1GB of RAM which makes implementing a detailed intrusion detection system very difficult. However we are of the understanding that the IDS required by Alliant will only be used for simple

network attacks and should be able to run flawlessly on the Raspberry Pi hardware.

Originally the design to accommodate the emulated services on the honeypot were hard coded onto every device. This process works, but does have the fatal flaw of being very difficult to change if the needs of Alliant change. To account for any unforeseen changes to our design we decided to implement the services in the form of plugins. The decision to update the device to a plugin architecture is a choice we believe will make the honeypot much more valuable to Alliant Energy. Switching to this new architecture allows Alliant the ability to easily add and remove micro-services at will to fully customize the device to perfectly suit their needs.

This plugin architecture works seamlessly with our original Splunk design which will still receive logs generated on the device which constantly capture device information. When an suspicious log is seen by the logger the log is sent outbound to the Splunk server located in the Alliant's data center which will then be configured to alert essential IT personnel based on the requirements Alliant sees fit.

2.3 Validation and Acceptance Testing

The testing requirements for each project requirement are summarized in Table 2.1.

Component	Testing Plan
SSH, HTTP, HTTPS	Create scripts in GoLang using the test library to emulate several clients in addition to end to end testing.
Login	Observe successful end to end testing of Splunk logging with Base64 encoded username and password.
Logging	Attempt a series of port scans through NMAP. Attempt connections through SSH, HTTP and HTTPS. Validate the logging format. Unit test caching ability.
Rule Set	Try all permutations of potential rule sets and test them.
IDS	Run all services with IDS and ensure the device is still functional.
Device	Check system requirements with client specifications.
Power	Select a device that is suitable for the power constraints. Potentially provide power of Ethernet capability
Equipment	Monitor each device to ensure consistent uptime. Provide proper support for environmental factors

Table 2.1: Validation and Acceptance Testing Plan

Section 3

Interface and System Description

3.1 Technical Approach

Our design and implementation strategy is guided by a handful of core technical principles.

Pluggable Architecture

To accommodate evolving client needs and future maintainability and utility, our core deliverable will consist of a plug-in architecture for micro-services and logging modules, along with a default set of honeypot plug-ins and a Splunk logging module. By deploying a flexible plug-in architecture, additional logging back-ends and honeypot protocols can be integrated easily into the full system.

Automation

Since our goal is to deploy *many* devices at different locations. We aim to automate as much of the build and configuration process as possible. Therefore, we will use Ansible [1] for easy remote setup for multiple devices. We will also use Docker [2] to configure the individual honeypot microservices. This setup will allow for simple and flexible device rollouts and updates.

[1] `http://www.ansible.com/`
[2] `https://www.docker.com/`

Safe Languages

The device will only deploy honeypot services written in a safe[3] language.

The core plugin architecture and the default plugins themselves will be written in Go [4].

Honeypots Don't Get Real Capabilities

Unlike medium- and high-interaction honeypots such as Kippo [5], our default honeypot microservices will not provide any real system capabilities. Since the device will be running in an internal, protected environment, *any* traffic should be considered an attack, and by only mimicking protocol initiation handshakes without providing actual program capabilities, we gain some protection against unknown exploits.

Heavily Restrict Traffic

The device will heavily restrict both ingress and egress traffic to a set of known, trusted servers. This will help mitigate any damage a compromised microservice on the device can cause.

Disguise Administration Interface

Since we aim to both trick an attacker into believing the services running on the device are legitimate, as well as restrict access to *functional* services on the device, the administration interface will only be accessible using Single Packet Authorization (SPA) [6].

3.2 Design Specification

The system has 2 primary layers of interfaces:

1. The External Interface, including components exposed to attackers and administrators as well as the alert logging interface

2. The Internal Component Interface, through which honeypot services communicate with each other

[3]Used informally here to mean languages that enforce type safety, memory safety, and provide built-in or standard library mechanisms to write safe concurrent code.

[4]`https://golang.org/`

[5]`https://github.com/desaster/kippo`

[6]`http://www.blackhat.com/presentations/bh-usa-05/bh-us-05-madhat.pdf`

Each layer has a particular interface for each component. See Figure 3.1 for a simplified diagram of the major device components.

3.2.1 Layer 1: The External Interface

The outermost layer of the device, the external interface, is the only point at which *incoming* network traffic is accepted. Access is controlled by an IPTables[7] ruleset.

The Public Interface

The public interface, visible to anyone on the device's network, is the interface through which attackers interact with the various honeypot micro-sevices. Since the micro-services are designed to run as unprivileged, isolated processes, the honeypots themselves run on high, non-standard ports (e.g. 8022 for SSH and 8080 for HTTP).

Firewall rules forward traffic from the open, standard port (e.g. 22 for SSH) to the high, non-standard port on local host where the actual honeypot is listening.

The Administration Interface

Devices must be updated and individually configurable, so there needs to be a way for administrators (or automated processes) to log in and perform routine maintenance. However, the device also should not expose a public SSH service with real login capabilities, and the honeypot SSH service can't be used for real system tasks.

Therefore, the device will use Single Packet Authorization to disguise the administration interface, only opening the port for a single connection to a particular internal IP address when presented with valid cryptographic credentials.

The Promiscuous Interface

The device will run an IDS in order to pick up anomalies and attacks on the greater internal network. The IDS will watch both incoming traffic to the device as well as sniff traffic on the internal network through a promiscuous interface.

[7] `http://linux.die.net/man/8/iptables`

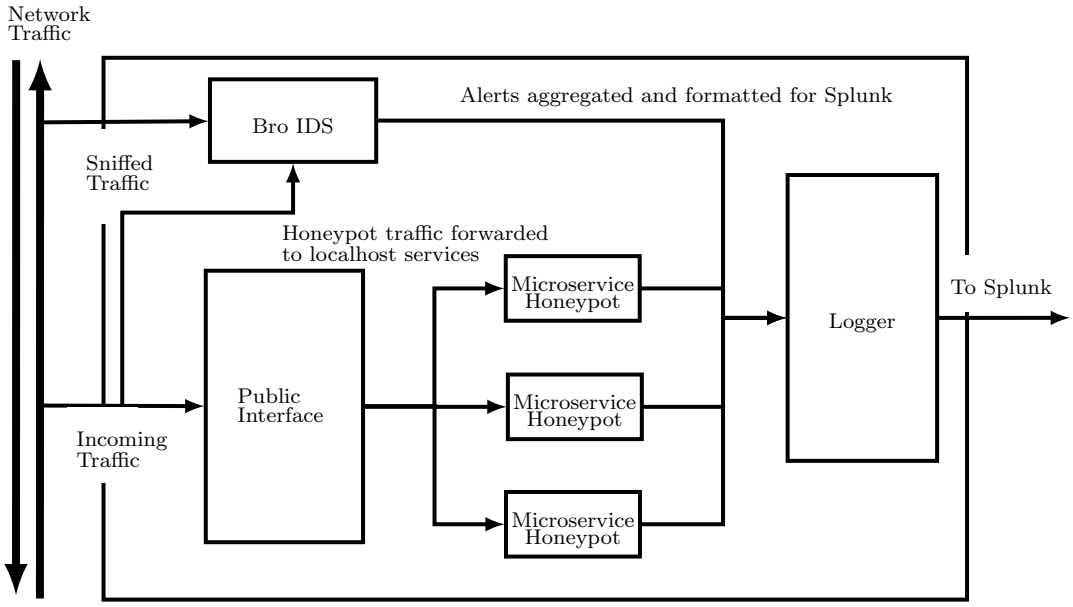

Figure 3.1: Simplified device internals

3.2.2 Layer 2: Low-Interaction Honeypot Micro-services

We will provide a structured plug-in architecture that can run clearly defined plug-ins for both arbitrary honeypot services as well as logging back-ends. The plug-ins then run as isolated and unprivileged processes, using inter-process communication over a well-defined protocol to communicate.

Each default honeypot micro-service plug-in set will be a low-interaction honeypot, without full protocol capabilities for its emulated service and with just enough functionality to discern:

- If a passive attacker is accessing the device
- If an active attacker is accessing the device
- What, if any, credentials an active attacker attempts to use
- Metadata about the attacker (IP address, MAC address, date/time, etc.)

Honeypot Plug-in Architecture

Since the Go programming language does not support dynamic linking, and thus compiling multiple plug-ins into a single executable becomes a messy

procedure, plug-ins themselves are implemented as independent programs that each implement our structured plug-in interface. We define two types of recognized plug-ins:

- Honeypot micro-service plug-ins
- Logging plug-ins

The plug-in architecture has two important interfaces.

A Structured Public Plugin Interface The plugin architecture defines a small, public interface that all plugins must implement. See Appendix A.1 for details of the honeypot plugin interface and Appendix A.2 for details of the logger plugin interface.

An IPC Mechanism Since plugins are isolated programs, they communicate using IPC (Inter-Process Communication). Specifically, plugins use Go's net/rpc library to communicate. Note however that this communication is handled transparently by the plugin architecture: plugin implementers do not perform RPC (Remote Procedure Call) with other components. The plugin controller itself performs RPC behind the scenes to coordinate plugin execution and communication, and by implementing the standard plugin interface, individual plugins get RPC behind the scenes for free.

3.3 Design and Implementation Stages

Design and implementation is broken up into three logical and somewhat self-contained stages. Each stage encompasses a design component, an implementation component, and an evaluation and testing component. Each stage implements a particular subset of the desired functionality.

3.3.1 Stage 1

Design and Implementation Stage 1 will consist of a number of initial tasks, design iterations, and prototyping. By completion of stage 1, the device will have the core plugin architecture implemented, run a few small honeypot microservices for common protocols, run a very minimal IDS rulset, and have a minimal build automation procedure.

Core Plugin Architecture

The core plugin architecture, including all public interfaces and the RPC and plugin registration mechanism, will be implemented in this early stage.

Minimal Build Automation

The device will be able to install all required dependencies automatically as part of the deployment procedure.

HTTP/HTTPS Honeypot

The device will be running an HTTP/HTTPS honeypot with a fake login page.

SSH Honeypot

The device will be running an SSH honeypot.

Pushing to Splunk

Both the HTTP/HTTPS honeypot and the SSH honeypot should be able to push alerts to an external Splunk server using a default Splunk logging plugin.

Open High Ports

At the client's request, the device will have 3 open high ports listening for incoming traffic.

Minimal IDS

An IDS will be used to push alerts to Splunk if any incoming traffic is seen on the 3 high ports.

Unit Testing and Bug Fixes

Each component of Stage 1 will have full unit tests (where applicable).

3.3.2 Stage 2

Design and Implementation Stage 2 will extend the work done in Stage 1 and provide the first SCADA protocol honeypot microservice plugin.

Better Deployment Automation

In Stage 2, general system hardening, custom firewall setup, and IDS configuration will be automated as part of the deployment step.

Promiscuous Network Interface

A second network interface will be added to listen in promiscuous mode, used by an IDS to pick up alerts from the greater internal network.

Custom IDS Configuration

The IDS will be configured to alert on domain-specific and network-specific indicators, such as known energy-sector malware and internal ping sweeps.

OPC Honeypot

A honeypot microservice plugin will be implemented to emulate the OPC protocol, the first SCADA protocol used on the device.

Unit Testing and Bug Fixes

Each component of Stage 2 will have full unit tests (where applicable).

3.3.3 Stage 3

Design and Implementation Stage 3 will finalize build automation, provide a full integration testing suite, and allow time to add additional, custom honeypot plugins to fit the client's partcular needs at different deployment sites.

Finalize Build and Deployment Automation

Build and deployment automation procedures and configuration needs will be finalized with the client.

Single Packet Authorization

Stage 3 will integrate a small Single Packet Authorization module into the device for remote administration.

Custom SCADA Protocols

By working closely with the client, any additional SCADA protocols needed will be implemented using the plugin microservice pattern established in design stages 1 and 2.

Unit Testing and Bug Fixes

Each component of Stage 3 will have full unit tests (where applicable).

Integration Testing

A full integration test suite will be written to test data flow through the system and, when possible, work to verify system correctness.

3.4 Functional Testing Plan

Each primary device componenet will have both unit tests and, when applicable, integration tests. The device testing plan is specified in Table 3.1. Note that all microservice testing procedures will follow the same strategy, so they have been collapsed to one table entry.

Component	Unit Tests	Integration Tests
Plugin Architecture	✗	Tested with plugins
Build Automation	✗	Test deployment on virtual machines
IDS	✗	Script attacks and verify alerting
Microservices	✓	Attempt to login to active service
Logger	✓	Verify logging correct on Splunk instance
Single Packet Authorization	✓	Verify login succeeds/fails
Promiscuous Interface	✗	Verify IDS receives traffic

Table 3.1: Functional Testing Plan

Section 4

Work Breakdown

4.1 Project Schedule

This section will outline the assumed tentative schedule for this Senior Design project. Gantt charts will be used to show the progress of this project over several revisions of this Project Plan and serve as a visual time line of the projects accomplishments upon completion in May. The schedules below are subject to change as the need arises.

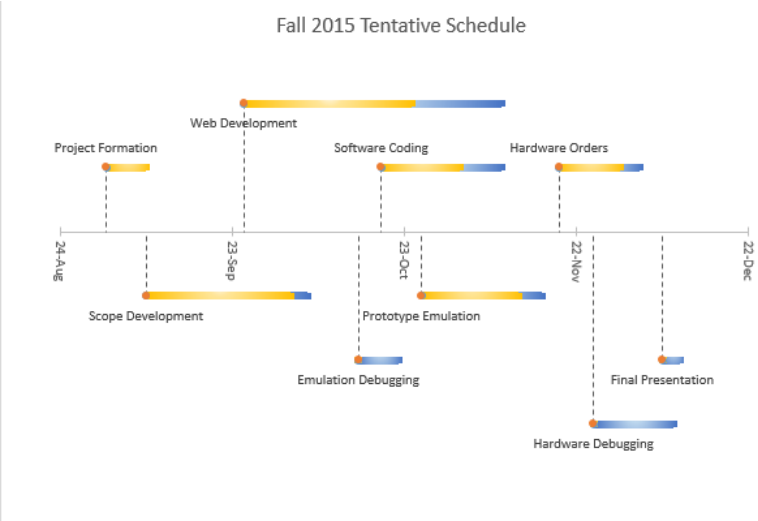

Figure 4.1: Tentative Fall 2015 Project Schedule Overview

Task	Start	Duration	End
Project Formation	2015-09-01	7	2015-09-08
Scope Development	2015-09-08	28	2015-10-06
Web Development	2015-09-25	45	2015-11-09
Emulation Debugging	2015-10-15	7	2015-10-22
Software Coding	2015-10-16	21	2015-11-9
Project Formation	2015-10-26	21	2015-11-16
Hardware Orders	2015-11-19	14	2015-12-3
Hardware Debugging	2015-11-25	14	2015-12-9
Final Presentation	2015-12-7	3	2015-12-10

Table 4.1: Fall 2015 Detailed Project Schedule

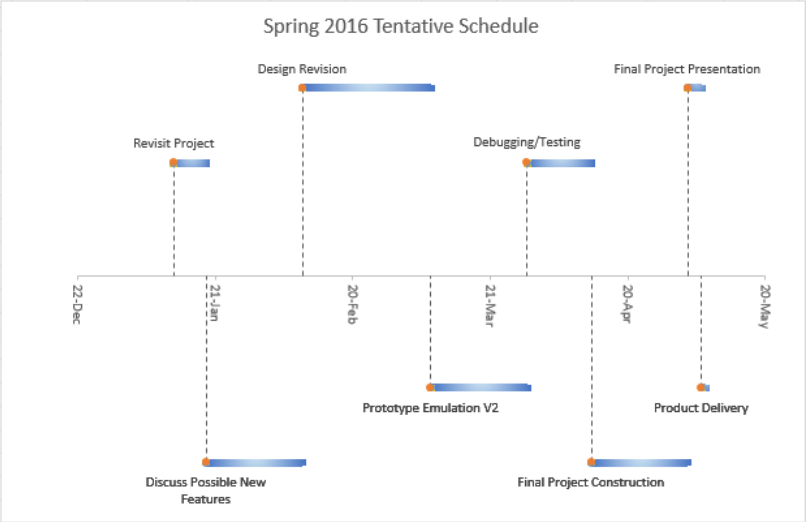

Figure 4.2: Tentative Spring 2016 Project Schedule

Task	Start	Duration	End
Revisit Project	2016-01-12	7	2016-01-19
Discuss Possible New Features	2016-01-19	21	2016-01-19
Design Revision	2016-02-09	28	2016-03-08
Prototype Emulation V2	2016-03-08	21	2016-03-29
Debugging/Testing	2016-03-29	14	2016-04-12
Final Project Construction	2016-04-12	21	2016-05-03
Final Project Presentation	2016-05-03	3	2016-05-06
Product Delivery	2016-05-06	1	2016-05-07

Table 4.2: Spring 2016 Detailed Project Schedule

4.2 Risks and Feasability Assessment

This project is routinely implemented in industry with high success rates. Several software implementations of SCADA honeypots already exist in the open source community. However, our implementation will be custom because the client has a secondary goal of including an IDS on the system. Creating the honeypot software from conception will minimize the resources required from the platform. This is important because the computer needs to be a small standalone device. A standard Raspberry PI only has 1GB of RAM for example. An IDS typically uses a lot of memory and process cycles. Most risk can be mitigated with proper contingency planning, as summarized in Table 4.3.

Component	Risk	Contingency Plan
Web Server Honeypot	Low	Utilize open source nginx[7]
SSH Server Honeypot	Low	Use open source Kippo SSH[8]
Event Alerts	Medium	Switch from REST API to SMTP
SCADA Honeypot	Medium	Use open source Conpot[9]
Intrusion Detection System	High	Use IPTables Logging

Table 4.3: Summary of project components and implementation risk

Installing an IDS on a small platform such as a Raspberry PI poses a significant challenge. Typically and IDS requires a lot of processing power and memory consumption. However after confronting our client at Alliant we have decided to go with an minimal IDS system to reduce resource consumption.

4.3 Cost Considerations

Alliant Energy requires service in twenty eight different locations. Each location will house a standalone minimal computer. The CanaKit Raspberry PI was chosen because it is customizable and holds more memory than standard versions. An additional network interface will also be provided from an external network adapter to accommodate the need for an IDS system. 4.4.

[7] `https://www.nginx.com`
[8] `https://github.com/desaster/kippo`
[9] `http://conpot.org/`

Model	Unit Cost	Vendor	Total Cost
Raspberry PI B+	$69.99 Plus Tax	CanaKit	$1960 Plus Tax[10]
USB 3.0 Gigabit Ethernet Adapter	$16.99 Plus Tax	Anker	$476 Plus Tax[11]

Table 4.4: Summary of implementation costs

[10] http://www.amazon.com/CanaKit-Raspberry-Complete-Original-Preloaded/dp/B008XVAVAW/ref=sr_1_1?s=pc&ie=UTF8&qid=1444258362&sr=1-1&keywords=raspberry+pi

[11] http://www.amazon.com/Anker-Gigabit-Ethernet-Adapter-Supporting/dp/B00NOP70EC/ref=sr_1_4?ie=UTF8&qid=1444258449&sr=8-4&keywords=usb+to+nici

Section 5

Market Research

Currently, Industrial Controls Systems across the US are being probed by potential threats every day. These SCADA networks can be extremely important to the US infrastructure and cannot be allowed to be tampered with by foreign threats. However the current state of the industry is one where equipment is often run for stringent, long term schedules with very little down time allotted. This can make keeping a SCADA network up to date difficult without opening access to the net. Concurrently, allowing any kind of outside access puts the systems at greater risk. In order to find and prevent these infiltrations, Honeypots are placed inside SCADA networks across the US to gather information about potential attackers. these Honeypots are quickly becoming an important part of any respectable SCADA network infrastructure.

There exist many standalone open source Honeypots which are available for easy implementation. One such system is Conpot, a low interaction server side SCADA designed to be easily implemented and modified. By default, this software runs a basic emulation of Siemens s7-200 CPU which is a PLC(Programmable Logic Controller) which could likely be found in a SCADA environment. The interactive protocols in this default install include MODBUS, HTTP, SNMP and s7comm[12]. Another powerful Honeypot tool available to industry is the HoneyD Daemon. This system created has the capability to simulate thousands of customizable virtual hosts on a network at the same time. It allows for the configuration of numerous services and includes advanced logging and adjustable network settings to prevent attackers from discovering it's true identity[13].

Our group chose not to use open source Honeypot systems like CONPOT or HoneyD on the basis that we desired to create our own system from

[12] `http://mushorg.github.io/conpot/`

[13] `http://www.honeyd.org/general.php`

scratch. These products, though readily available to the public and our client, would not have satisfied the minnimized hardware and software footprint that our client desires. By keeping our system simple, well keep maintenance time to a minimum while still providing useful information about potential threats to IT personal. Our small standalone device will also allow for quick streamlined installation and be kept out of the way of working personal.

Section 6

Conclusion

For this project our senior design group was assigned to design and implement a SCADA honeypot system for 28 of Alliant Energy's power plants. Each individual system has the requirement of being low power, low maintenance, able to track and log any access attempts thorugh SSH, HTTP, and HTTPS as well as alert the proper administrators if any of these attempts were to happen. In addition to the required specifications we decided to include the possibility of implementing a simple intrusion detection system as requested by Alliant's security team.

To meet these features we have decided to implement our honeypot as a Raspberry PI running a Debian OS. The Raspberry PI offers an ideal platform for hosting the set of services needed to properly secure the honeypot since it is highly customizable and provides the necessary storage and memory requirements. On the Raspberrry PI we will implement two seperate servers, one for SSH and one for HTTP/S. Both of these will be stored on a Docker image that will reinforce the standard that we have set for these devices to be easy to update and control. These criteria are consistent with how Alliant views illegitimate users attempting to access the device as well as how they want their systems to communicate with it. By implementing these services we will be able to secure each SCADA network in accordance with Alliant Energy's request and produce a quality product for our client.

Appendix A

Plugin Interface Details

The public plugin interface description follows for both recognized types of plugins. Note that code is in Go.

A.1 Honeypot Plugin Interface

```
type Honeypot interface {
        Start()         error
        Stop()          error
        Restart()       error
}
```

Figure A.1: Honeypot microservice plugin interface

A.2 Logger Plugin Interface

```
type Logger interface {
        LogEvent(event *Event)          error
        LogEvents(events []Event)       error
}
```

Figure A.2: Logging plugin interface