May1601 Design Document

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

College of Engineering
Iowa State University of Science and Technology
Ames, Iowa

Fall Semester 2015

Contents

List of Figures

List of Tables

Section 1

Introduction

The purpose of this document is to outline the specifications and validations for the ICS/SCADA Traffic Baseline and Honeypot project. This document will provide detailed systems level analysis and layout of the various components included in the proposed product, as well as descriptive explanations of the testing and verification methods used to show that this design will meet the specs put forth by the client. Should the need arise, this document may be used by the client in the future to better understand the product's construction and how to implement/modify the product after the design team has finished.

1.1 Project Statement

The goal of this project is to create a standalone security device that can be placed in an industrial network to monitor traffic, looking for security-related deviations, and act as a low interaction Honeypot. The design team of May1601 will design this product according to the specifications put forth below.

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 and that the cause of the intrusion be identified and closed as soon as possible. This is where Honeypots come in to play. Honeypots disguise themselves as systems on the SCADA network, mimicking the behavior of other devices on the network while gathering information. Ideally, attackers on the network connect to the Honeypot unknowingly and provide information to the IT staff allowing them to interpret the source of the attack to determine the best course of action to mitigate the risk.

1.3 Deliverables

The product which will be produced for this project will be packaged within a Raspberry Pi microcontroller. We chose this device owing to its small physical footprint, small power consumption, its Linux based operating system, and relatively practical hardware capabilities. This device was also much more practical in terms of cost effective implementation, since up-scaling the device or attempting to design and manufacture our own on such a small scale would be costly and difficult to maintain long term. There will be (28) units delivered at the end of this project. Each unit will include:

- RaspberryPi Model 2 B
- 8GB MicroSD card
- 2.5A Micro USB power supply(5ft cable) with noise filter.
- USB 3.0 network adapter to RJ45 Ethernet connection.
- RaspberryPi plastic hardcase.
- A Docker container which will run Ansible and contain all necessary files for easy deployment.

1.4 Specifications

The device created by the May1601 design team shall be determined by the following specification rule set:

- The device shall be capable of interaction via SSH, HTTP, and HTTPS protocols.

- The HTTP and HTTPS protocols will be implemented via a fake login page with key logger to record attempts.
- The device will record and transmit logs of all connection attempts to Splunk server and alert essential personal upon any irregularities such as increased ICMP traffic, port scans, or repeated URL attempt requests.
- The device rule set will be malleable and easily updated in order to accommodate future modifications to the system.
- The device will be low maintenance, low power, and capable of standalone functionality.
- There will be (28) devices deployed in (28) electrical substations.
- The device will not interfere or interact with any other devices on the SCADA network.

Section 2

System Level Design

2.1 System Requirements

When requirements are gathered, it is important to break them up into "functional" requirements, and "non-functional" requirements. It is necessary to note the difference between the two, as they are key to understanding what to develop, and how to accomplish a project. Functional requirements are often thought of as a description of a system's behavior, and non-functional requirements elaborate performance characteristics of the system. In this section, a list of functional requirements, as well as non-functional requirements can be found.

2.1.1 Functional Requirements:

The Functional Requirements for the SCADA honeypot system are as follows.

- The system must interface with SSH, HTTP, and HTTPS protocols.
- The system will use a fake login page with key logger to record login attempts.
- The system will have the ability to record logs of all connection attempts.
- The system must contain a small intrusion detection system (IDS).
- The system must send alerts to administrative personnel upon detection irregularities that include, but are not limited to, increased ICMP traffic, port scans, and repeated attempts to connect to URLs.

- A rule set must be included that allows for fine-tuning to accommodate new or changed rules.

2.1.2 Non-Functional Requirements:

The Non-Functional Requirements for the SCADA honeypot system are as follows.

- The system must be low maintenance.
- Must be a standalone device.
- Must be low power.
- Allow the support of no fewer than 28 devices.
- Have an up-time of no less than 99 percent.

2.2 Functional Decomposition

The SCADA honeypot to be built for twenty-eight of Alliant Energy's power-plant substation will have seven main components that allow it to mimic and log potential attacks. These components consist of a public firewall, an SSH server, a local webserver, an Ansible configuration manager, a log generator, a localized intrusion detection system, and a centralized Splunk server. To gain a more complete understanding of how data is processed through each honeypot the next few paragraphs will describe the purpose and functionality of each component.

2.2.1 Parts Breakdown:

Public Firewall

The public firewall also known as the public interface is one of two means of ingress to the system. The firewall setup uses Linux IP tables to control the flow of traffic to only information coming in on ports 22(SSH), 80(HTTP), 443(HTTPS) and port 2222. These ports are opened up for the reason that they are the bare minimum required to run the services of the on-board servers. This give the illusion that the honeypot is a secure service and helps to obscure its true intentions to a malicious user. Note: The public firewall does not interact with the second network interface card because it is monitored directly by the Intrusion Detection System.

SSH Server

The SSH server, setup after the public facing firewall, is a mock server that only simulates the authentication process of the SSH handshake. However the server setup on the honeypot is designed to terminate the handshake once the attempting user submits their credentials. The SSH server then sends the username, password and other important information to the logger to be processed into one cohesive log. The user is then denied access after a short delay to mimic the background process of checking a database.

Local Web-server

The local web-server implemented in the design of the honeypot system works in a similar fashion to the SSH server. Essentially if a malicious user attempts to connect via HTTP/HTTPS they will be directed to the webserver. This false web-server acts as an information gathering tool for Alliants admin. Once inside the firewall the user will be presented with a login page that asks for a username and password. Once entered the information is logged and sent to the logger just like the SSH server. Upon entering their credential the system will wait for a period of time before displaying that the credentials entered are not accepted. This provides the ability to log multiple attempts to access the system through HTTP/HTTPS.

Ansible Configuration Manager

The Ansible Configuration Manager is responsible for the setting up the initial configuration of each type of server and also pushing any patches that are necessary for the device to run properly. Because of this it will only be accessible by network users with administrative rights. By using Ansible we are able to setup a playbook that allows us to configure our device with whatever protocols and services we want and then push all changes to every device at once, remotely. This eliminates the need to physically go into every machine and make changes on a machine to machine bases.

Log Generator

The log generating component of the honeypot takes information from the all other components and formats them into an easy to read informative text file that can be sent to a central sever in Alliant Energy's data center. These logs will consistently record normal network activity as well as any attacks that occur on the device. The log format will be structured in accordance with Alliant Energy's choosing, but will contain information on the time the attack was initiated, what type of attack it was, the username and password, and other credentials. This data will then be relayed back to a Splunk server at Alliant's main data center for analysis.

Splunk Server

The splunk server is a component in Alliant's data center that receives the logs generated in the honeypot and analyzes them to determine if an attack has happened according to Alliant's definition of an attack. When an attack does occur the splunk server will contact an administrator via email to alert that an attack. From this information the administrators can make the appropriate response to keep the network secure.

Bro Intrusion Detection System

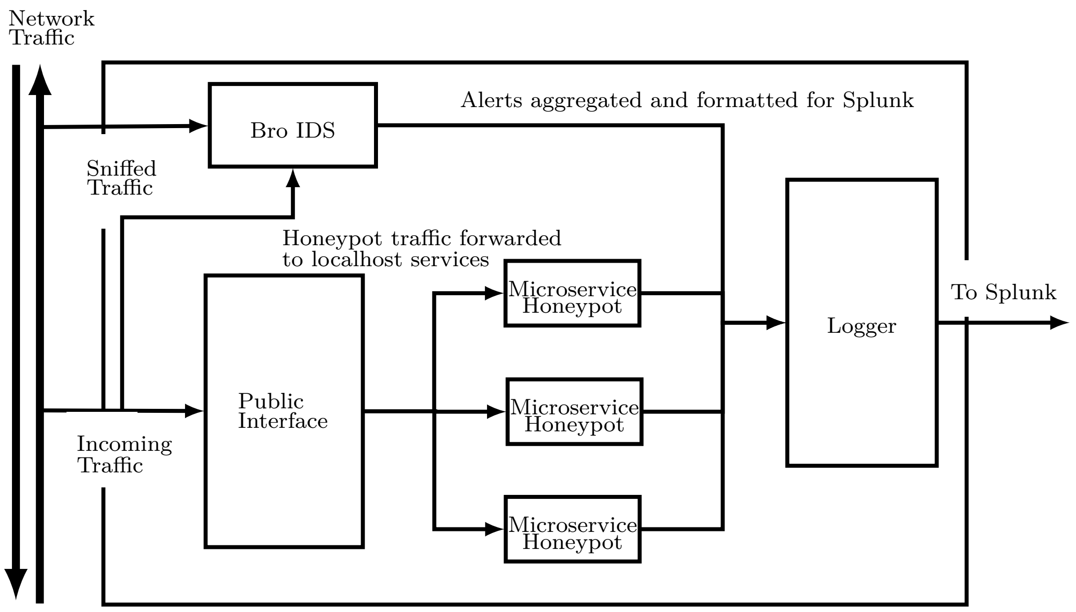
The Bro IDS software listens on a separate network interface card than the firewall and the other servers within the honeypot. This second NIC is unregulated and connects directly to the IDS, meaning that is allows all traffic in. By doing this the honeypot will be able to capture any other attack-/malicious traffic that occur within the SCADA system. However none of the other systems will be compromised as a result of this because each NIC is physically separate from each other and only connect at the logger which works the same way for either system.

2.3 System Analysis

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 scratch will minimize the resources required from the platform. This is important because the computer needs to be a small standalone device. A Raspberry PI only has 1GB of RAM for example and an IDS typically uses a lot of memory and process cycles. Most risk can be mitigated with proper contingency planning. Installing an IDS on a small platform such as a Raspberry PI poses a challenge. However we believe that we can compress the IDS to only include core functions Alliant wants to reduce the resources required to run it. The overall honeypot implementation carries a low risk of failure.

The overall plan to deal with unforeseen changes to the project is to run all software implementations in a plugin architecture on a standard raspberry pi with an additional network interface card added. The raspberry pi was chosen for its' adaptability and we are confident if any changes are needed the pi will be able to handle them. The flexible plugin architecture was chosen specifically because we cannot foresee all of the services Alliant might want to run in the future. By making a plugin architecture with a default set of plugins that were specifically asked for in the project we make it so Alliant gets the services they want today, and the ability to easily update the devices with new services in the future.

2.4 Block Diagrams



Section 3

Specifications

3.1 I/O Specification

The system I/O is broken down into three sections, each with a particular set of inputs and outputs:

1. Public Interface
2. Network Traffic
3. Administration

Note that all references to “Alert(s) to Splunk” are output of the form summarized in Figure 3.1, along with context-specific details (for instance, in the case of an attempted SSH login, the username and password supplied by the attacker) as detailed in Appendix A.

3.1.1 Public Interface

The public interface is visible to normal actors on the internal network. It is the only interface that attackers targetting the system will *directly* interact with (attackers may indirectly interact with the system through sniffed network traffic). Each honeypot service running on the system has a particular set of inputs defined by the underlying protocol specification for the particular service, as well as dual outputs: one output to the attacker (protocol-specific), and one output to the internal logging system (same format for all services).

A summary of the protocol-specific inputs and outputs follows.

Traffic Type	External Output	Internal Logged Output
SSH	Authentication Failed SSH Protocol Message	Alert to Splunk
Non-SSH	No Output	No Output

Table 3.1: SSH Honeypot Outputs

Traffic Type	Path	External Output	Internal Logged Output
HTTP	/login	Access Denied HTML Message	Alert to Splunk
HTTP	Any other path	HTML Login Page	Alert to Splunk
non-HTTP	N/A	No Output	No Output

Table 3.2: HTTP Honeypot Outputs

Traffic Type	Path	External Output	Internal Logged Output
HTTPS	/login	Access Denied HTML Message	Alert to Splunk
HTTPS	Any other path	HTML Login Page	Alert to Splunk
non-HTTPS	N/A	No Output	No Output

Table 3.3: HTTPS Honeypot Outputs

Traffic Type	External Output	Internal Logged Output
Valid SCADA Protocol	Access Denied Protocol Message	Alert to Splunk
Invalid SCADA Protocol	No Output	No Output

Table 3.4: SCADA Honeypot Outputs

SSH Honeypot I/O

Input All TCP traffic on port 22 sent to the device. The SSH Honeypot ignores non-SSH traffic on port 22.

Output The SSH Honeypot output is summarized in Table 3.1.

HTTP Honeypot I/O

Input All TCP traffic on port 80 sent to the device. The HTTP Honeypot ignores non-HTTP traffic on port 80.

Output The HTTP Honeypot output is summarized in Table 3.2.

HTTPS Honeypot I/O

Input All TCP traffic on port 443 sent to the device. The HTTPS Honeypot ignores non-HTTPS traffic on port 443.

Output The HTTPS Honeypot output is summarized in Table 3.3.

SCADA Honeypot I/O

This section summarizes the inputs and outputs for *all* SCADA honeypot services. Note that particular SCADA protocols, while existing on the device as different services, will have logically equivalent I/O, simply formatted to conform to the particular SCADA protocol in use. SCADA honeypot I/O is thus summarized as a group.

Input Any SCADA traffic sent to the device on an open port.

Output The SCADA Honeypot service output is summarized in Table 3.4.

3.1.2 Traffic

Traffic inputs to the system originate from two sources:

1. Sniffed network traffic
2. High port traffic to device

High port traffic to the device is traffic sent directly to an open port on the device on which there is honeypot service running.

Sniffed Network Traffic

Input Sniffed network traffic is traffic observed by a network interface listening in promiscuous mode. This input is any ethernet traffic on the same network as the device.

Output In all cases, output is either none or an alert logged to Splunk. The decision of whether or not to send an alert is made by the system IDS.

3.1.3 Administration

Device administration is done through SSH. In all cases, administration traffic consists of SSH protocol messages.

3.2 Interface Specifications

The system deals with four primary interfaces:

1. Administration Interface
2. Public Interface
3. Splunk Interface
4. Promiscuous Network Interface

A brief description of each primary interface follows.

3.2.1 Administration Interface

All administration is done over SSH. The administration interface is simply an SSH server, listening on a *different* network than the public interface (an internal wireless network, for instance). Administrators confirm access with each device through Single Packet Authentication to validate they have administrative rights to the device.

3.2.2 Public Interface

The public interface is the interface other users/hackers on the same network as the device are presented with. This interface is a set of open ports, one for each honeypot micro-service, and a handful of open high ports with no services listening.

```
{"event":{
    "source address": string
    "source port": int
    "service type": string
    "service type":{
        "service specific field A": type,
        "service specific field B": type,
        "service specific field N": type,
    }
}
```

Figure 3.1: General Alert JSON Format

Since the device will be deployment to numerous (28) different locations, and every device's configuration may vary slightly (i.e. a different set of high ports open, or a different SCADA protocol running), an exhaustive listing of open ports is not practical or possible at this design stage (the list of open ports is configurable by the client).

However, every device follows the same pattern: one open port per honeypot microservice, and a handful of open high ports.

3.2.3 Splunk Interface

The Splunk interface is the interface the device uses to push alerts to administrators. The Splunk interface is interacted with through a REST API, the details of which are specified by Splunk itself[1].

The alerts themselves are formatted JSON[2] using format specified in Figure 3.1, displayed in pseudo-JSON for convenience (format subject to change based on continuing feedback from client).

Each "service type" entry is contains service-specific details (for instance, an SSH alert may contain a username field, a password field, and a key field). Additional data such as time of alert and originating device are handled through metadata inherent in a Splunk REST API call.

See Appendix A for a full example of an alert from the HTTPS microservice honeypot.

[1] `http://docs.splunk.com/Documentation/Splunk/latest/RESTREF/RESTprolog`
[2] `www.json.org`

3.2.4 Promiscuous Network Interface

The final interface is the promiscuous network interface. This is simply a network interface listening in promiscous mode, and the system's IDS will interpret the traffic and, if appropriate, log an appropriate alert to Splunk using the REST API and an appropriate JSON format 3.1.

3.3 Hardware/Software Specifications

Hardware Specification

The following components make up the hardware specification:

- Raspberry Pi 2 Model B
- Quad-Core 900 MHz Processor
- 1 GB RAM
- 8 GB Micro SD Card
- 150 Mbps WiFi Adapter
- 2.5A USB Power Supply with Micro USB Cable
- CanaKit Raspberry Pi 2 Case

Software Specification

All system components implemented by this development team will be written in the Go Programming Language[3].

Table 3.5 summarizes the free software tools the system will use.

Tool Name	Purpose	Brief Description
Raspbian[4]	Operating System	Debian GNU/Linux ARM port
Ansible[5]	Configuration and Setup	Build and deployment automation
Docker[6]	Isolated microservice environment	Linux container wrapper

Table 3.5: Free Software Tools Used

[3]https://golang.org/

[4]https://www.raspbian.org/

[5]http://www.ansible.com/

[6]https://www.docker.com/

Section 4

Testing and Modeling

Testing for this project can be done through a combination of unit testing and simulations. Four components are required to be tested. Ansible, SSH, WebAuth, and the Splunk Logger. Ansible will require simulations to determine if it can successfully install and configure multiple devices. SSH, WebAuth, and Splunk will too require simulations to evaluate their ability to handle several clients. In addition, these three components will require unit testing to check for edge cases and invalid inputs.

4.1 Simulations and Modeling

An open source program named Vagrant will be used for simulations. Vagrant allows for the quick creation of a virtual machine that can be used in this instance to create a simulated device. Ansible will take a newly created machine, run its' installations, and start all services. Multiple machines can be created by using a Vagrant file. Additional machines can be created in a similar method with the ability to make SSH, HTTP, and HTTPS connections. This will simulate several clients attempting to communicate with our Honeypot.

4.2 Implementation Issues and Challenges

The biggest challenge to testing is proper configuration of Vagrant and unit testing for inputs and edge cases. Writing appropriate test code for that will emulate client SSH, HTTP, and HTTPS calls will also be a challenge.

4.3 Testing Procedures and Specifications

The following list can be used as a procedural guideline for testing. Specifications can be described as all services performing their function correctly.

1. Install and configure Vagrant on the host machine.
2. Run Vagrant file to start Vagrant on several virtual machines.
3. Start Ansible to install services on VM. Ansible configures and installs IP-Tables firewall, WebAuth, SSH, and Splunk logger
4. Start Vagrant on client VM's.
5. Simulate HTTP, HTTPS, and SSH calls. SSH and Http calls are logged to external Splunk server using JSON format. If there is an error, the Splunk logger will cache the event.
6. Evaluate Splunk Logs for JSON formatting
7. Use GoLang Test package to build test cases for each unit in WebAuth, Splunk, and SSH.
8. Run test cases
9. Evaluate results

Section 5

Conclusion

With the culmination of the concepts outlined in the above document, the design team of May1601 believes that specifications outlined and agreed upon between we the team and the client will be satisfied. With the inclusion of a public interface, SSH and WEB servers, minimal IDS support, and event logging we believe that our product will perform as advertised while maintaining a minimized footprint in both cost and power consumption. By keeping our product smaller and simple we believe our honeypot will prove a viable and cost efficient alternative to commonly used open source honeypots. With a planned prototype implementation by the end of the Fall 2015 semester, there will be ample time in the Spring to modify the current design of the product or implement further functionality which the client may desire.

Appendix A

Full HTTPS Alert

```
{"event":{
    "source address": "127.0.0.1",
    "source port": "443",
    "service type": "HTTPS",
    "https":{
        "method": "POST",
        "path": "/login",
        "parameters":{
            "username": "root",
            "password": "toor",
        }
    }
}
```

Figure A.1: A sample Splunk alert

Figure A.1 shows an example HTTPS alert. The alert is logged to Splunk's REST API in the listed JSON format using a POST request.