



Interested in learning
more about security?

SANS Institute InfoSec Reading Room

This paper is from the SANS Institute Reading Room site. Reposting is not permitted without express written permission.

Designing and Implementing a Honeypot for a SCADA Network

Supervisory Control and Data Acquisition (SCADA) networks are increasingly under attack. In a world of wireless networks, removable media, and a desire to move to the Internet of Things, it is unrealistic to think that these networks can be isolated from every threat. Devices on SCADA networks are often not designed with security in mind, vendors are slow to create patches, and the systems usually are critical enough that scheduling downtime is a problem. A layered approach to securing SCADA networks that includes bo...

Copyright SANS Institute
Author Retains Full Rights



AD

Designing and Implementing a Honeypot for a SCADA Network

GIAC (GCIA) Gold Certification

Author: Charlie Scott, cscott@utexas.edu

Advisor: Richard Carbone

Accepted: June 7, 2014

Abstract

Supervisory Control and Data Acquisition (SCADA) networks are increasingly under attack. In a world of wireless networks, removable media, and a desire to move to the “Internet of Things,” it is unrealistic to think that these networks can be isolated from every threat. Devices on SCADA networks are often not designed with security in mind, vendors are slow to create patches, and the systems usually are critical enough that scheduling downtime is a problem. A layered approach to securing SCADA networks that includes both patching and monitoring is essential to the safety of those systems. This paper discusses a method of discovering attacks on a SCADA network by using a honeypot. An analysis of a facilities-based SCADA network was performed, a honeypot designed and built to mimic the common services of this network, and monitoring was put in place to detect when the honeypot was targeted. This resulted in more comprehensive monitoring of the SCADA network for attacks and added another layer of security to the existing patching routine.

1. Introduction

This paper is based on a facilities network filled with Supervisory Control and Data Acquisition (SCADA)-type devices, controlling and monitoring everything from elevators, to pumps, to generators, to smart meters, to building access control systems. The physical plant has a footprint that spans hundreds of acres and the digital footprint includes over 2,000 devices. No single group is responsible for managing these devices and, in many cases, vendors install these devices at the behest of a department and then they are not touched for years. While this network is isolated from the greater Internet, vendor contractors and technical staff often connect to it using outside laptops for maintenance tasks. This facilities network has long been a concern of the information security staff at the workplace because of the nature of the devices on the network, the lack of central responsibility for those devices, and the fact that vendors connect to it regularly. All of these concerns conspire to make it difficult to know if there is a bad actor lurking about or a compromised system on the network.

The goal of this paper is to improve the security monitoring of SCADA and facilities networks such as these, where information security operations groups may have limited access and visibility into the devices on those networks. A low-interaction honeypot was chosen because it required no modification of the existing network topology or devices. This honeypot was configured to send alerts to the information security operations staff via an intermediary Splunk server on a DMZ. From these alerts, information security staff can take action, such as tracking down the attacking host for incident handling purposes.

1.1. Challenges in Securing SCADA Networks

A common recommendation is to put SCADA devices on a network that is not physically connected to any other network (Stouffer, Falco, & Scarfone, 2011). An “air gapped” network, as it is often called, is not always an option, as there may be data from those devices that needs to be accessed (Weiss, 2010; Knapp, 2011). In addition, patching those devices may require a connection to either the Internet or a central patching server, such as a Windows Server Update Service (WSUS) server in the case of Windows

Charlie Scott, cscott@utexas.edu

systems, or a RedHat Satellite server in the case of RedHat Enterprise Linux. Stuxnet and Flame can use USB removable media as a vector, so physically separating a network does not guarantee that a system will remain unreachable to an attacker (Zetter, 2011; Zetter, 2012). Consultants and vendors are also an issue, as they may bring malware in on laptops and connect those systems to the SCADA network for maintenance purposes. Any network-aware malware on those systems will then also have access to the SCADA network. In other words, physical separation is not a guarantee that a SCADA network will be safe from attack and it should be treated as just another layer of security (Luallen, 2013).

Another challenge in securing SCADA networks is that performing software updates on SCADA devices can be difficult (Higgins, The SCADA Patch Problem, 2013). As mentioned, a physically separate or heavily firewalled network may prevent devices from being updated using centralized device management tools or the Internet-based software update services provided by Microsoft and Linux distributions such as RedHat. Difficult patch management often means that updates will not take place on a regular basis. Embedded devices, often running some stripped-down version of Linux, Windows CE, or a proprietary OS, may be even more difficult to update and may require a protocol such as TFTP or even a console connection. Updates for embedded OSes are usually rarer than those of the standard desktop and server OSes.

In addition to the difficulty involved in updating the software on SCADA devices, the vendors of these devices or the integrators used to bring them online may not allow them to be updated without the vendor or integrator first performing appropriate regression testing (Byres, 2012). Sometimes, updates on the part of the customer without the vendor first approving them may violate the warranty or support agreement. At this point, the customer is in a holding pattern while the vendor validates the patch, which may take days, weeks, or months.

In addition to the vendor concerns, there is the fear of disturbing equipment that has been operating fine for years, sometimes decades, for a security patch (Zubairi & Mahboob, 2013). There is always a chance that the update could break something else or

that the hardware might fail upon reboot. This is especially unnerving in a plant environment, where downtime may cost money or cause safety issues.

Finally, it may not be possible to implement additional security controls on SCADA networks or devices. For example, implementing anti-virus or a host-based intrusion detection system on SCADA-related workstations and servers might cause some unexpected behaviors, such as system slowdowns or active responses to perceived attacks (Wade, 2011). Intrusion prevention systems (IPS) placed inline on the SCADA network may block traffic it misidentifies as malicious.

1.2. Brief Overview of Honeypots

A honeypot can be thought of as a lure to attract attackers (Provos & Holz, 2008). Generally they take the form of physical or virtual systems that mimic the actual devices on the network, with heavy monitoring and logging so an attacker's actions can be studied. Although it may seem paradoxical for a security professional to *want* an attacker to be interested in them, there are a number of benefits that a honeypot provides.

Honeypots provide security researchers with a unique opportunity to study their enemy. By analyzing how real-world attacks are taking place, how often they take place, and what attackers are leaving behind (e.g. rootkits, Trojans, and exploits), a researcher can come up with better ways to defend against such attacks. For incident handlers, intrusion analysts, and others monitoring their network's security, a well-monitored honeypot can provide another indication of network attacks. Finally, a honeypot seeded with false but convincing information that might be desirable to an internal or external attacker (such as intellectual property or customer credit card numbers) can provide a diversion from more valuable, legitimate targets.

It is important to note that in some jurisdictions the use of a honeypot could be construed as entrapment (Provos & Holz, Virtual Honeypots: From Botnet Tracking to Intrusion Detection, 2008). It is best to read up on applicable laws and consult with a lawyer before embarking on this path.

1.2.1. Two Types of Honeypots

Researchers classify honeypots as either high-interaction or low-interaction (Provos, Developments of the Honeyd Virtual Honeypot, 2008). This is not, however, a measurement of the maintenance requirements for administrators of these honeypots (though sometimes it can be), but rather how much an attacker is able to do with them. A high-interaction honeypot is typically made up of the actual device, operating system, and applications one wants to be attacked. For instance, a researcher interested in attacks on Windows 2003 Server running the IIS 6 web server would actually build a physical or virtual machine running this OS and software. In the case of SCADA devices, a researcher might stand up an actual Schneider Electric Modicon M168 PLC. The primary advantage of a high-interaction honeypot is that the attacker can do whatever he wants to it, as he would against the actual device, because it *is* the actual device. This means that exploits actually work and can be analyzed. This is also the greatest disadvantage as well; because once the honeypot is compromised it will have to be rebuilt. Malware attacks against the honeypot system's BIOS, though rare, may make the system permanently untrustworthy (Hruska, 2009). Regardless, it also means that the honeypot can be used as a launching point for other attacks, which may cause legal complications.

A low-interaction honeypot takes an ordinary system, such as a Linux system, and runs software on it that mimics the operating system, network stack, and services that the researcher is interested in. In the abovementioned example of Windows 2003 with IIS 6, it may present services such as the common SMB ports and an IIS 6.0 banner on TCP ports 80 and 443. The honeypot will then log connections to these ports and any commands sent to them. Often, that is the extent of what they allow. A low-interaction honeypot simulating a Cisco router, for example, may present what looks like a standard Cisco IOS Telnet banner, but never allow an attacker to actually log in. Instead, the usernames and passwords they attempt (and other data sent) will be logged. Likewise, an FTP server may allow anonymous logins, but might not implement the full compliment of possible FTP commands. The primary advantage of a low-interaction honeypot is that it is relatively easy to maintain because, among other reasons, it is less likely to actually be compromised. Of course, this also means that a researcher may not get the full picture of what an attacker is trying to do. A common low-interaction honeypot is *honeyd*, which

Charlie Scott, cscott@utexas.edu

was created by Niels Provos for the HoneyNet Project (Provost, Developments of the Honeyd Virtual Honeypot , 2008).

1.2.2. Honeynets

Honeynets are simply collections of honeypots designed to look like common network services and servers (Provost & Holz, Virtual Honeypots: From Botnet Tracking to Intrusion Detection , 2008). For instance, you might have a honeypot that presents itself as a Windows domain controller, another as an intranet web server, another as a mail server, and so on, until it looks like a fairly complete corporate network. Honeynets can be made up of high-interaction honeypots, low-interaction honeypots, or a combination of both. Honeynets are often set up behind a system that serves as a *honeywall*, which provides a bridge to the honeynet and includes network monitoring, packet capture, and intrusion detection capabilities.

1.3. Benefits of Honeypots on SCADA Networks

There are several benefits to running a honeypot on a SCADA network as opposed to using other security measures. A honeypot does not modify existing SCADA network configurations including firewall and unified threat manager (UTM), or require the insertion of additional inline devices. Installing inline devices would cause downtime and, depending on the architecture, might be a single point of failure. The honeypot is simply plugged into the network as any other system would be and set up to run services that look like other devices on your SCADA network. Because the honeypot is not inline and not actually blocking malicious traffic, like an IPS would, this reduces the likelihood of network downtime caused by false positives.

Another advantage of a honeypot is that it can be configured to look like specific devices on a typical SCADA network. That is, it does not have to look like a generic IIS server on Windows, an OpenSSH service on Linux, or even a Telnet service on a Cisco router. You can configure the honeypot to look like a heating, ventilation and cooling (HVAC) system, building access control system (BACS) or industrial control system (ICS). This allows for the monitoring of attacks designed specifically to target the current infrastructure.

Charlie Scott, cscott@utexas.edu

For all the upsides of running a honeypot on a SCADA network, there is one major downside: The system has to be monitored and alerts acted upon. Unlike an IPS, a honeypot will not automatically block attacks. Network security staff time may have to be allocated for monitoring, investigation, and remediation.

1.4. Existing Honeypots for SCADA Networks

Using a honeypot to help secure a SCADA network is not a completely new idea. Pothametsy and Franz from Cisco Systems proffered the idea as early as 2004 and created the SCADA HoneyNet Project (Pothametsy & Franz, 2005). Their goal was to create a module that worked with honeyd and would simulate a programmable logic controller (PLC), which is a specialized computer used for the control of machinery. The specific protocols that it simulated include FTP, HTTP, Modbus TCP, and Telnet. Modbus is a serial communications protocol used in PLCs for communication with other industrial electronic devices developed in 1979 by Modicon, now part of Schneider Electric (Knapp, 2011). This project is interesting and can still be used for simulation of PLCs, though the code has not been updated since 2005.

Digital Bond, an ICS security research and consulting company, also created a SCADA honeynet comprised of two virtual machines and has made them publically available (Digital Bond, Inc., 2014). One virtual machine acts as a PLC honeypot, while the other runs a honeywall that monitors network traffic. The honeywall can also be used to monitor a high-interaction honeypot in the form of a real PLC. On the honeywall, Digital Bond has included the Snort IDS and a number of signatures specific to the PLC. Services that the PLC honeypot simulates include FTP, Telnet, HTTP, SNMP, and Modbus TCP. The latest version of the honeynet, as of this writing, is 0.8 and is dated from 2011.

A more recent SCADA honeypot is Conpot (Rist, Vestergaard, & Haslinger, 2014), which simulates a Siemens SIMATIC S7-200 PLC, including the Modbus TCP, SNMP, and HTTP protocols. It was still being actively developed as of early 2014. Conpot is affiliated with the HoneyNet Project and can be optionally configured to report back attack data to that project, in order to provide researchers with an idea about how widespread attacks are.

Charlie Scott, cscott@utexas.edu

2. Designing and Implementing the Honeypot

Before deploying a honeypot, it is important for a network or security administrator to understand the network attack surface. If a network consists primarily of Windows systems and the administrator is mimicking a FreeBSD host using a honeypot, that honeypot will not likely provide the administrator with useful information concerning the network. Likewise, with a SCADA network, an administrator would not want to set up a honeypot with the personality of an Internet-facing DMZ. Mapping the SCADA network will give an administrator a better idea of what operating systems and services are running on the network. Once those services are identified, then he will have to choose which ones to include in the honeypot. Finally, he will need to implement the honeypot itself, architected so that results will still be accessible.

2.1. Mapping the Network Attack Surface

For this paper, the author chose to use Tenable's Nessus to map the attack surface of his facilities network. Any tool that can perform a discovery scan of a network should work. A good discovery scan will include not only hosts and ports that it finds, but also make an attempt to determine the operating system and the types of services that are running therein. However, Nessus was used because in addition to being able to perform a discovery scan, it also includes a number of SCADA-related vulnerability checks, including those for Modbus TCP service and Modicon PLCs (Tenable Network Security, 2014). One caveat: It can be very dangerous to run any kind of vulnerability or service scanner against a SCADA network, so it is best to get written permission, do the scan after-hours, and notify staff that support the devices on the network that this is being done.

2.2. Choosing the Honeypot and Services

The services included in the honeypot should mirror what is actually found in the facilities network in which it will reside. Having it mimic an existing device of some sort is ideal, especially if it concerns attackers who know what they are looking for. For this paper, the author chose to support, at a minimum, HTTP, SNMP, and Modbus. Modbus was chosen so that it would indicate to an attacker that this is some sort of SCADA device. SNMP was chosen because it can provide reconnaissance information to a would-

Charlie Scott, cscott@utexas.edu

be attacker. Finally, HTTP was chosen to provide some sort of interesting human-machine interface (HMI) to the attacker.

A device discovered on the facilities network that has all three of these protocols is the Schneider Electric PowerLogic ION6200 smart meter, pictured below.



Figure 1: A Schneider Electric PowerLogic ION6200 smart meter (Photo source: Schneider Electric, 2010.).

Because HTTP, SNMP, and Modbus were of primary concern, the author chose to use the Conpot honeypot. The fact that it is still community-supported, as opposed to the SCADA HoneyNet Project and the Digital Bond HoneyNet, was also a factor, as supportability is important for maintenance of the honeypot.

2.3. Implementing the Honeypot on the Network

The author installed the honeypot on a facilities network consisting of elevator controls, building access controls, smart meters for measuring power consumption, pumps, generators, and other devices used for the normal functioning and monitoring of facilities systems. There are three primary segments in this configuration, including the facilities network itself, a facilities DMZ network that contains tools and interfaces used to monitor or manage the devices on the facilities network, and the control network from which people manage or monitor the devices on the facilities network from the systems on the DMZ network. The control network and facilities DMZ are separated by a firewall, as are the facilities DMZ and facilities network. There is no direct

Charlie Scott, cscott@utexas.edu

communication between the control network and the facilities network; all traffic between the two must pass through the facilities DMZ. A high-level diagram of the network follows:

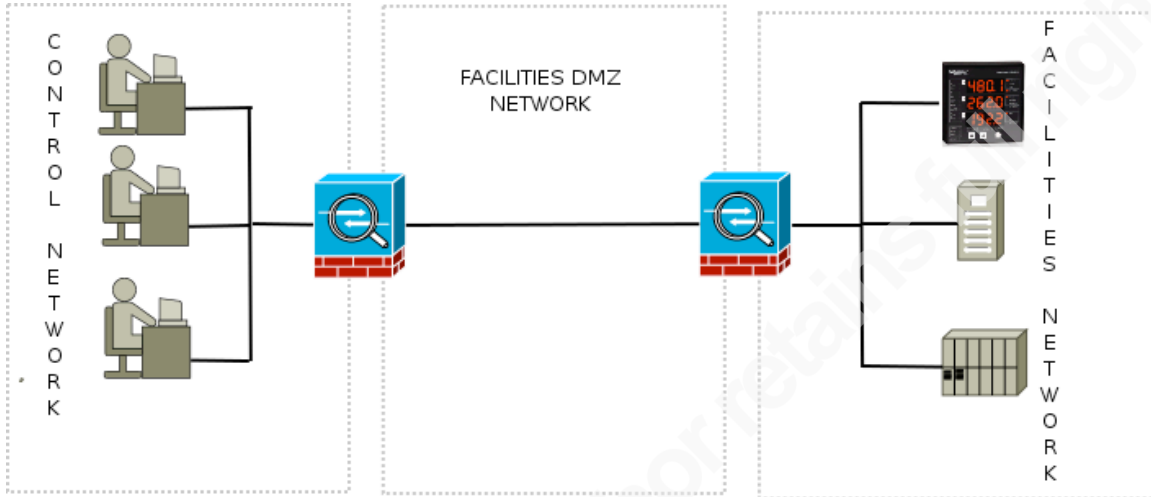


Figure 2: Diagram showing control network, facilities DMZ network, and facilities network prior to honeypot placement.

The honeypot itself is placed on the facilities network so that it looks like just another ION6200 smart meter. Rules are changed within the firewall to allow outbound Syslog (514/UDP) traffic from the honeypot to the Syslog and Splunk server on the facilities network DMZ. Additionally, Splunk's default port (8000/TCP) is allowed inbound from the control network so that Splunk can be accessed by those monitoring the honeypot. Finally, outbound SMTP (25/TCP) traffic is allowed from the Splunk server for email alerts. The following diagram illustrates this:

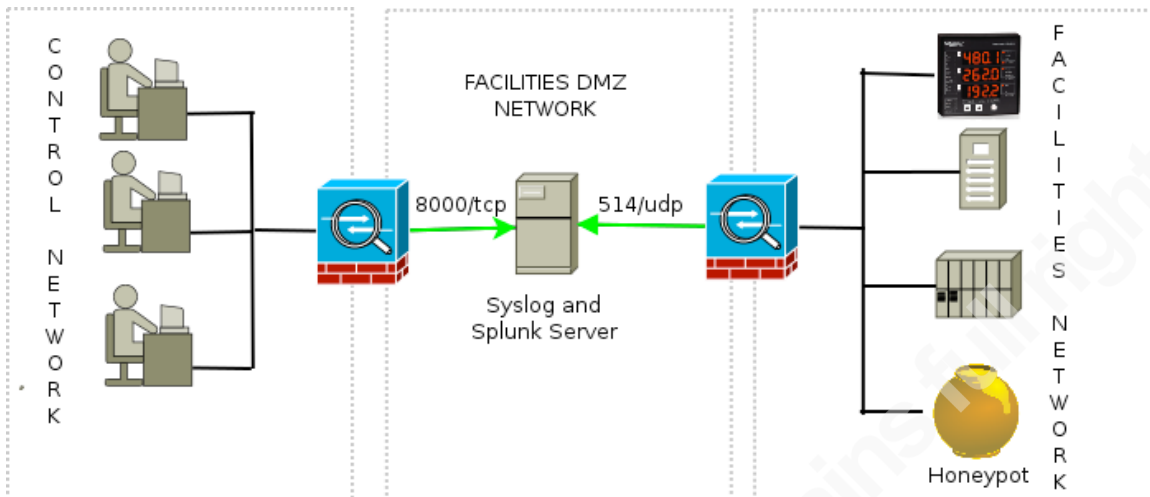


Figure 3: Diagram showing honeypot placement on facilities network (Honeypot graphic by Jonas Bock, Creative Commons Attribution-Share Alike 3.0 Unported).

2.4. Configuring the Honeypot

Originally, the author wanted to use RedHat as the Conpot host OS in order to be consistent with other supported devices in the environment, but the Conpot developers have not tested the honeypot on that platform and there were problems with dependencies. With some work, RedHat may have been possible, but it could present challenges with future updates and was abandoned in favor of a tested and functional distribution.

Ubuntu 12.04 LTS was the chosen distribution as it is tested by the Conpot team and has a relatively easy installation process (Glastopf Developers, 2013). The Ubuntu community, led by Canonical Ltd., will support Ubuntu 12.04 LTS until April 2017 (Ubuntu Community, 2014).

2.4.1. OS Modifications and Hardening

This paper will not cover the installation of Ubuntu in detail, as the standard installation procedure was used. The server edition of Ubuntu 12.04 LTS was used because a graphical user interface and desktop applications (office applications, games, etc.) are not needed for this honeypot. Using a server edition keeps things simple.

Installation and configuration of the honeypot took place on a network that had access to the public Internet in order to update the operating system and install Conpot. This network was behind a firewall with network address translation (NAT) to protect it

during installation. Then, later, it was moved to the facilities network, which does not have direct access to the public Internet. Another option would be to use physical media, such as a USB thumb drive, to transfer Conpot to the system. Additionally, the system should be scanned for potential malware using a tool such as ClamAV or chkrootkit.

On an Ubuntu 12.04 LTS server installation, very little hardening is required as by default, no network-enabled services are running. If using a different distribution, it may be necessary to disable the firewall services that are running by default. Conpot has its own SNMP, HTTP, and Modbus services, so there is no need to run an Apache webserver or an SNMP daemon. In order to avoid conflicts or potential host-based vulnerable services, it is best to turn off all services.

Depending on the organization's security policy, it may be necessary to have a warning banner displayed at logon. The author modified `/etc/issue` and `/etc/issue.net` with the standard authorized warning banner for his organization:

Unauthorized use of UT Austin computer and networking resources is prohibited. If you log on to this computer system, you acknowledge your awareness of and concurrence with the UT Austin Acceptable Use Policy. The University will prosecute violators to the full extent of the law.

The SSH daemon was also installed and enabled during setup to assist with configuration using Ubuntu's APT package manager:

```
$ sudo apt-get install ssh
```

The following line in `/etc/ssh/sshd_config` was also uncommented so that the warning banner is displayed when someone attempts to connect via SSH:

```
Banner /etc/issue.net
```

SSHD was then restarted:

```
$ sudo service ssh restart
```

After the configuration is complete and SSH is no longer required, it was stopped and disabled with the following commands:

```
$ sudo service ssh stop
```

```
$ sudo update-rc.d ssh disable
```

Charlie Scott, cscott@utexas.edu

It is important that the honeypot not have any visible services that would not normally be on the type of device it is trying to mimic. SSHD, for instance, is not a service one would expect on an ION6200 smart meter, so it was disabled. If such a service must run then it is important to heavily firewall said service such that an attacker will not see those ports. Restrict access to only the machines from which they will be managed.

2.4.2. Installing Conpot

Installing Conpot on Ubuntu 12.04 LTS is very straightforward (Glastopf Developers, 2013; Haslinger, 2014), though on the server version of Ubuntu it will require a number of dependencies. The following dependencies were installed before installing Conpot:

```
$ sudo apt-get install git
$ sudo apt-get install cython
$ sudo apt-get install python-dev
$ sudo apt-get install python-pip
$ sudo apt-get install build-essential
$ sudo apt-get install libxml2-dev
$ sudo apt-get install libxslt1-dev
$ sudo apt-get install libevent-dev
$ sudo apt-get install snmp-mibs-downloader
```

After the dependencies were installed, GIT was used to clone *modbus-tk*, which is an implementation of the Modbus protocol in Python (Rist, *glastopf/modbus-tk*, 2013).

```
$ cd /opt
$ sudo git clone http://github.com/glastopf/modbus-tk.git
$ cd modbus-tk/
$ sudo python setup.py install
```

Finally, Conpot itself was installed and started to verify that it worked (note that the console log below has been truncated):

```
$ cd /opt
$ sudo git clone http://github.com/glastopf/conpot.git
$ cd conpot
$ sudo python setup.py install
$ sudo conpot
```



|_ |

```
Version 0.2.2
Glastopf Project
```

```
2014-05-25 12:19:03,094 Starting Conpot using template
found in: /usr/local/lib/python2.7/dist-packages/Conpot-
0.2.2-py2.7.egg/conpot/templates/default.xml
2014-05-25 12:19:03,095 Starting Conpot using configuration
found in: /usr/local/lib/python2.7/dist-packages/Conpot-
0.2.2-py2.7.egg/conpot/conpot.cfg
2014-05-25 12:19:03,095 Starting Conpot using www templates
found in: /usr/local/lib/python2.7/dist-packages/Conpot-
0.2.2-py2.7.egg/conpot/www/
...
2014-05-25 12:19:03,591 S7Comm server started on:
('0.0.0.0', 102)
2014-05-25 12:19:03,592 SNMP server started on: ('0.0.0.0',
161)
2014-05-25 12:19:03,719 HTTP server started on: ('0.0.0.0',
80)
2014-05-25 12:19:08,722 Privileges dropped, running as
nobody/nogroup.
```

As previously mentioned, Conpot's default template simulates a Siemens SIMATIC S7-200 PLC. While this might produce some interesting data, it is even more interesting to have the honeypot mimic something actually on the current facilities network; in this case a Schneider Electric PowerLogic ION6200 smart meter.

2.4.3. Modifying Conpot to Simulate an ION6200

Conpot is flexible enough that it can be modified to simulate other makes and models of SCADA-type devices. The majority of the configuration options are in a file called *default.xml* found in the *templates/* directory off of the directory where Conpot was installed (in this case, the *setup.py* program put it in */usr/local/lib/python2.7/dist-packages/Conpot-0.2.2-py2.7.egg/conpot/*). The first step in modifying the *default.xml* file is to make a backup of the old one:

```
$ cd /usr/local/lib/python2.7/dist-packages/Conpot-0.2.2-
py2.7.egg/conpot/templates
$ sudo cp default.xml default.xml.bak
```

Then copy *default.xml* to *ion6200.xml*, delete the original, and create a symbolic link between *ion6200.xml* and *default.xml*:

Charlie Scott, cscott@utexas.edu

```
$ sudo cp default.xml ion6200.xml
$ sudo rm default.xml
$ sudo ln -s ion6200.xml default.xml
```

In order to know what to modify in the *ion6200.xml* file, it is important to know how the ION6200 presents itself on the network. One way to do this is to go back to the Nessus scan used to get a map of the network attack surface and use what was discovered.

The following TCP ports and services were shown to be up during the scan of an ION6200: 80 (HTTP) and 502 (Modbus). The only UDP port open was 161 (SNMP). In addition, the SNMP walk that Nessus performed gathered the following information (with the revealed sysContact, sysName, and sysLocation redacted):

```
sysDescr   : ION 6200 to Ethernet
sysObjectID : 1.3.6.1.4.1.4346.11.1.2.1.1.1.15
sysUptime  : 0d 0h 0m 0s
sysContact : ██████████
sysName    : ██████████
sysLocation : ██████████
sysServices : 12
```

Rather than walk through the entire *ion6200.xml* configuration, the author has highlighted XML elements that need to be changed in the table below.

Table 1: Modified XML elements in the *ion6200.xml* configuration file.

Element(s)	Attribute/Sub-Element	Description
<conpot_template>	name	Template name
<conpot_template>	description	Template description
<core><databus>	<key_value_mappings>	Contains the key→value mappings for information referenced elsewhere in the configuration, such as the SystemDescription used by SNMP and other services.
<modbus>	<device_info>	Contains elements with the

Charlie Scott, cscott@utexas.edu

		Modbus VendorName, ProductCode, and MajorMinorRevision of the device.
<s7comm>	enabled	This should be set to “False” as it is a protocol specific to the S7-200 PLC.

In addition, the entire *ion6200.xml* configuration file is included in its entirety in Appendix A.

Conpot also supports a HMI via the HTTP server on TCP port 80 and can be configured to display a page of one’s choosing. An actual home page from an ION 6200 was recursively scraped using the wget tool:

```
$ wget -r -l 2 -t 1 --timeout=10 -nc -i ~/ions.txt
```

The wget tool created a directory from each IP address that included the HTML pages of the ION 6200’s web servers. Note that this is a dangerous process and could have repercussions on the equipment on a given network including the accidental modification of configurations. It should be approached with caution and it is better to obtain the pages from a test, rather than production, device.

In order to make these pages available to Conpot, a directory for the pages was created in Conpot’s root directory:

```
$ cd /usr/local/lib/python2.7/dist-packages/Conpot-0.2.2-py2.7.egg/conpot/  
$ sudo mkdir www  
$ sudo mkdir www/htdocs
```

The pages from an ION 6200 were then copied into the *www/htdocs* directory. Of course, on a real ION 6200 the pages are dynamic, but static pages are appropriate for a low-interaction honeypot such as this. The author is primarily looking for evidence of an attacker on the network (especially since nothing should be directly connected to it that

should not be there), rather than a deep knowledge of how they are attacking this particular device. The home page of the device is then visible on the honeypot when visited with a browser, as shown in the following screen capture.

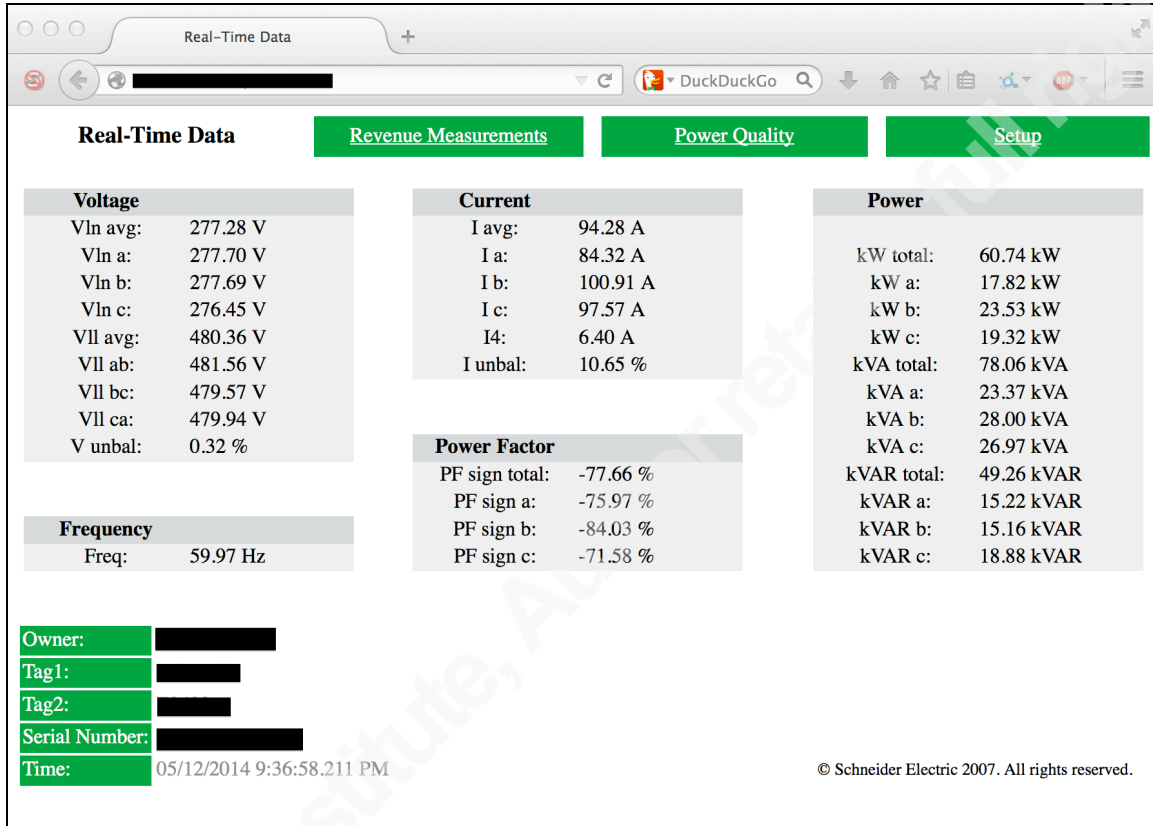


Figure 4: Screen capture of the ION6200 HMI.

2.5. Monitoring and Alerting on the Honeytrap Results

The honeypot is on a facilities network that is not directly accessible from the control network. This means that if an attacker's activity generates logs on the honeypot, a human being will not know about it unless they are logged directly into the honeypot itself. Conpot can be configured to send logs to a remote Syslog server, which solves part of the problem as the logs can then be sent to a system on the facilities DMZ. However, there still needs to be a good way to view and search through the logs, as well as send alerts when there is activity. A useful tool for doing this is Splunk Enterprise, which will index logs for easier searching and alerting (Splunk, Inc., 2014).

2.5.1. Syslog Server Configuration

A logging server was set up in the facilities DMZ running rsyslog. Rsyslog is a high-performance, modular version of syslog (Rsyslog Community, 2014). In directory */etc/rsyslog.d* an additional configuration file called *60-conpot.conf* was added. The number sixty appended to the file name means that the Conpot-related configuration file will load last. What follows is the file's content:

```
module(load="imudp")
input(type="imudp" port="514")
local0.* /var/log/conpot.log
```

This file loads the imudp module, which sets up rsyslogd to use UDP, and configures it to listen on port 514. For any messages that come in to the local0 syslog facility it will log them to */var/log/conpot.log*.

On the Conpot system, the *conpot.cfg* file in */usr/local/lib/python2.7/dist-packages/Conpot-0.2.2-py2.7.egg/conpot/* was edited. Under the [syslog] section, the “enabled” parameter was set to True, the host set to the IP address of the Rsyslog server, and the “socket” set to “udp” so that it would send the messages to the remote host instead of the local device. The final portion of the configuration follows:

```
[syslog]
enabled = True
device = /dev/log
host = 10.20.14.130
port = 514
facility = local0
socket = udp ; udp (sends to host:port), dev (sends to
device)
```

As previously mentioned, the firewall between the facilities network and the facilities DMZ would also have to be configured to allow UDP port 514 from the Conpot server to the Rsyslog server. Once done, a restart of the Rsyslog service and of Conpot then sends the logs to the remote host, as show in this snippet:

```
May 26 15:38:20 Done scanning for mib files, recursive scan
was initiated from 1 directories and found 0 MIB files of 7
scanned files.
May 26 15:38:20 10.19.14.153 DataBus: Get value from key:
[SystemDescription]
May 26 15:38:20 10.19.14.153 Registered: OID (1, 3, 6, 1,
```

```

2, 1, 1, 1) Instance (0,) ASN.1 (sysDescr @ SNMPv2-MIB)
value ION 6200 E1176 dynrsp.
May 26 15:38:20 10.19.14.153 DataBus: Get value from key:
[Uptime]
May 26 15:38:20 10.19.14.153 Registered: OID (1, 3, 6, 1,
2, 1, 1, 3) Instance (0,) ASN.1 (sysUpTime @ SNMPv2-MIB)
value 0 dynrsp.
May 26 15:38:20 10.19.14.153 DataBus: Get value from key:
[sysContact]
May 26 15:38:20 10.19.14.153 Registered: OID (1, 3, 6, 1,
2, 1, 1, 4) Instance (0,) ASN.1 (sysContact @ SNMPv2-MIB)
value Charlie Scott dynrsp.
May 26 15:38:20 10.19.14.153 DataBus: Get value from key:
[sysName]
May 26 15:38:20 10.19.14.153 Registered: OID (1, 3, 6, 1,
2, 1, 1, 5) Instance (0,) ASN.1 (sysName @ SNMPv2-MIB)
value FAC eMeter dynrsp.

```

Now that Rsyslog is working, Splunk can be used as a means to view the logs and generate alerts.

2.5.2. Splunk Configuration

Splunk's configuration is based on indexes, which are repositories for Splunk data (Splunk, Inc., 2014). While all logs can be sent to the same index, having multiple indexes makes searching easier and more granular (as well as making it simpler to start over if something goes wrong). Therefore, the first thing to be done as a Splunk administrator is to create an index by going to the Settings menu and choosing Data→Indexes. Then add a new index called “conpot” and save it with the default values.

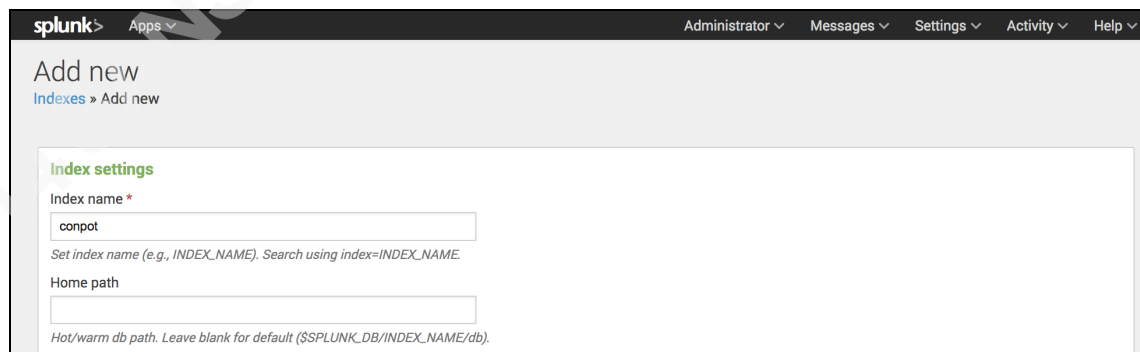


Figure 5: Adding a new index called “conpot” in Splunk.

After the index is created, it is time to add a data source. This is done from the Splunk home screen by clicking on the Add Data button. Choose the data type as “A file or directory of files” and then choose to consume any file on the Splunk server. Browse the server for the location of `/var/log/conpot.log`.

Charlie Scott, cscott@utexas.edu

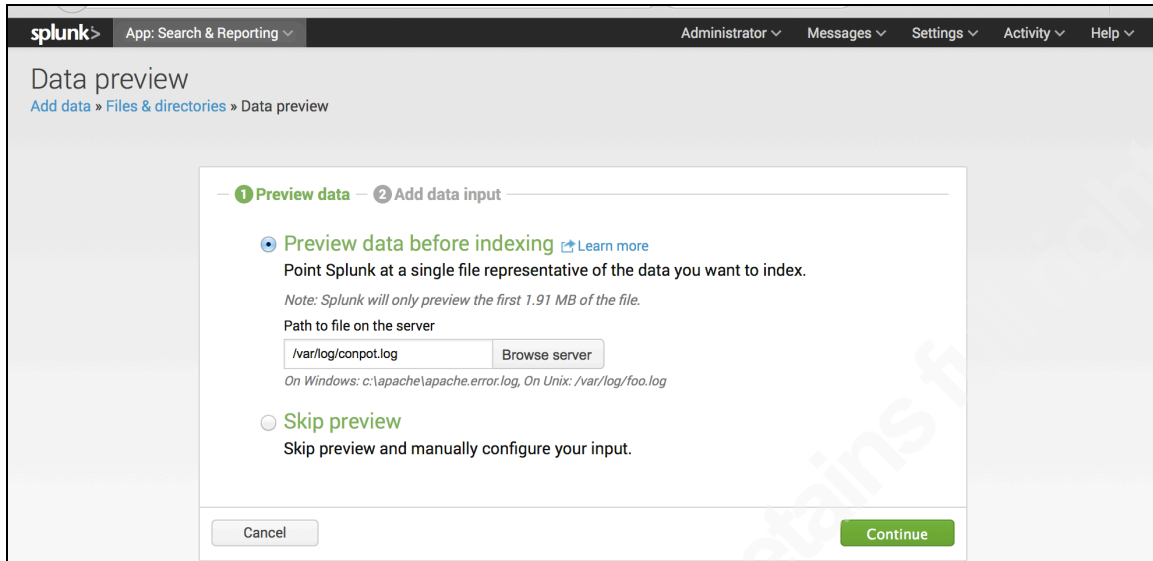


Figure 6: Adding the `/var/log/conpot.log` data source in Splunk.

Source types tell Splunk how to deal with certain types of data formats. For instance, there are source types for Syslogs and Apache logs. Splunk does not come with a source type that matches the format of Conpot logs, so it is best to select to “Start a new source type.” In Conpot, each line of a log is a discrete event, so choose “Every line is one event.”

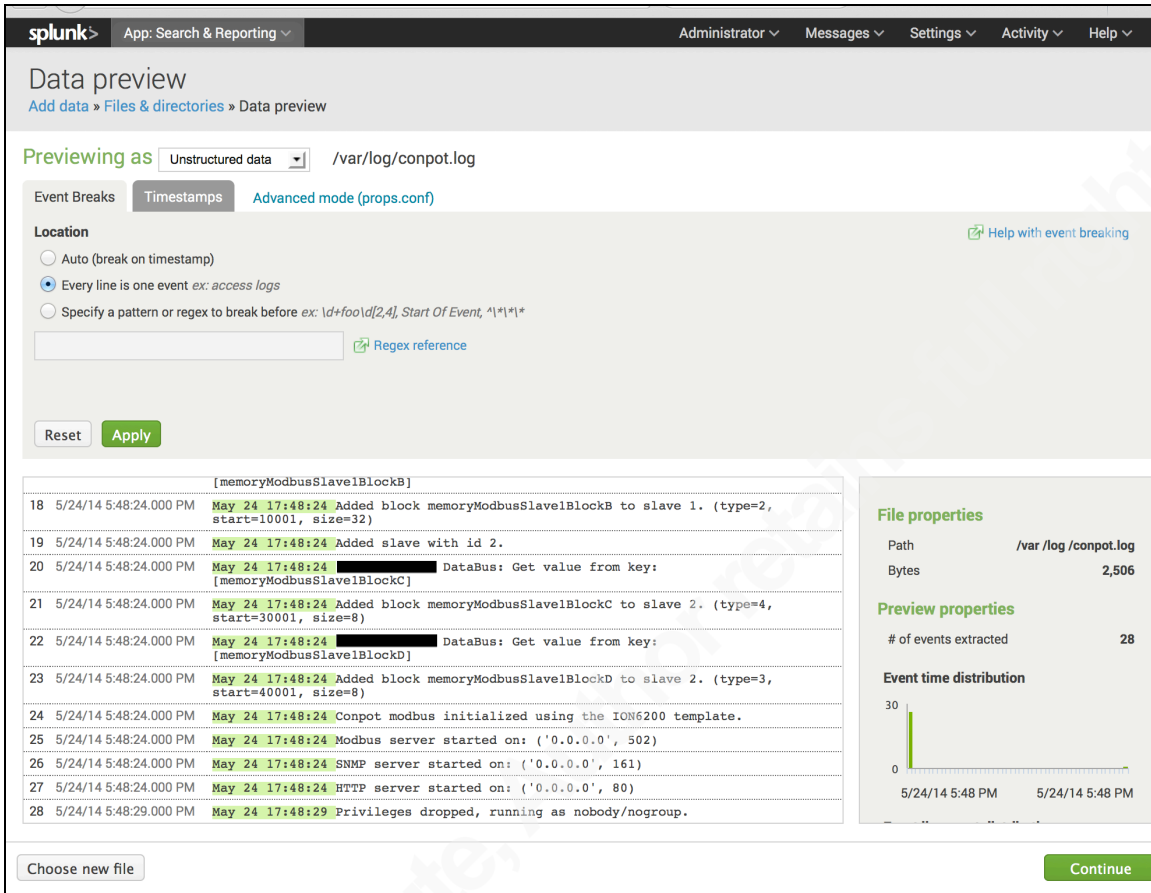


Figure 7: Telling Splunk how events are broken up in Conpot’s logs.

Then name the new source type as “conpot.”

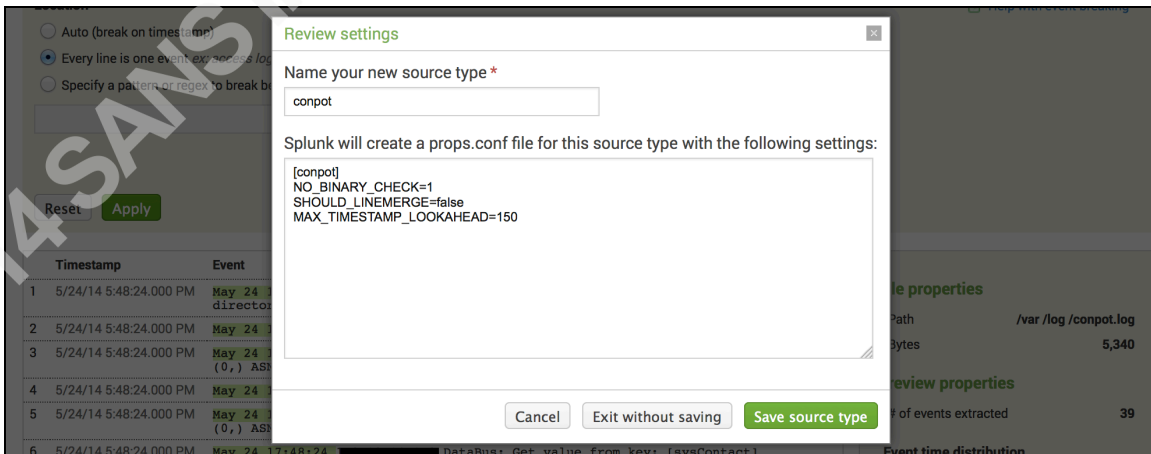


Figure 8: Naming the new source type as “conpot.”

Finally, set “conpot” as the destination index.

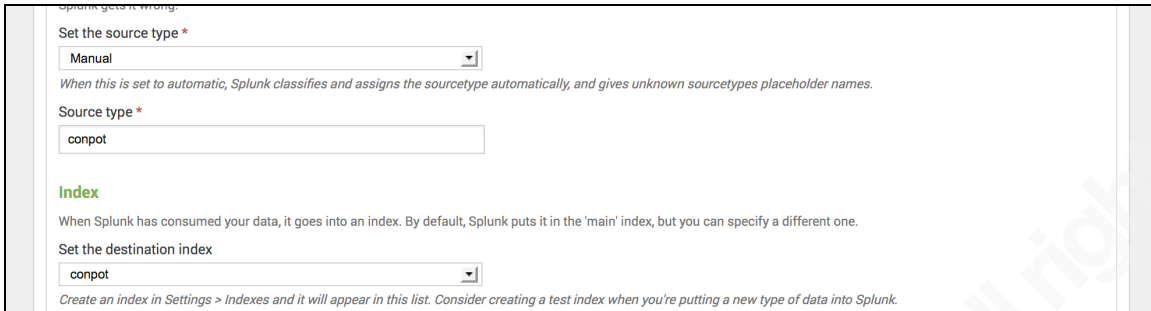


Figure 9: Setting the “conpot” index as the destination for the “conpot” source type.

When finished, a simple search on the “index=conpot” or “sourcetype=conpot” will show the Conpot log entries so far, though the search can be further refined. The following screen capture shows a search for all HTTP GET requests after a scan with a web application vulnerability scanner.

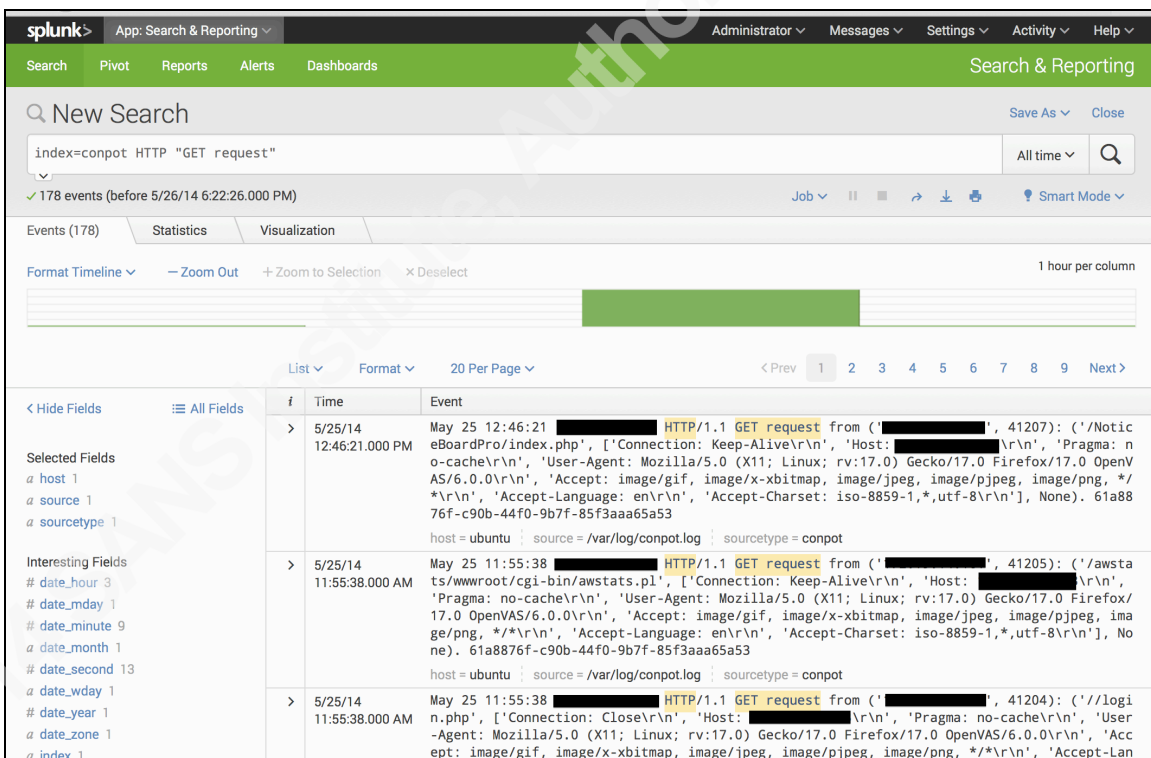


Figure 10: A simple search in Splunk for HTTP GET requests in the Conpot logs.

2.5.3. Alerting On Results

Splunk can generate alerts on results and either send an email or launch a script. For this project, the author chose to have Splunk send an email, which also requires

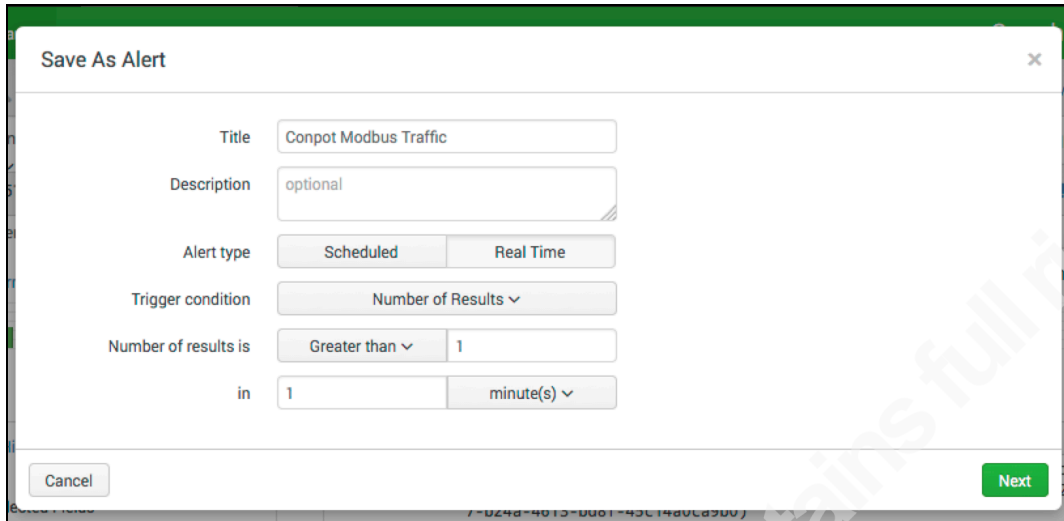
outbound SMTP traffic from the Splunk server to the mail server. The Splunk alerts were configured with the following information:

Table 2: Alerts added to Splunk for interesting Conpot events.

Alert Name	Severity	Search
Conpot SNMP Session	Medium	index="conpot" sourcetype="conpot" "New snmp session from"
Conpot HTTP GET Request	High	index=conpot sourcetype=conpot HTTP "GET request"
Conpot Modbus Traffic	Critical	index="conpot" sourcetype="conpot" "Modbus traffic from"

Each alert was assigned a severity rating. The Modbus alert was given the highest severity rating of Critical, because it is very unlikely that a system would randomly be probing the Modbus service on TCP port 502. The HTTP GET request was given a High severity rating because it is the service someone might use to configure the mimicked ION6200. Finally, the Conpot SNMP session was given the severity of Medium, since not much except reconnaissance can be done from that service.

To set up an alert, after a search is successful click on the “Save As” menu in Splunk and select “Alert.” All alerts were configured from a real-time search, meaning that they would go off as soon as the trigger activity was detected. Splunk requires a trigger condition to determine when an alert should fire. For example, trigger conditions can be based on each result in the search, or a certain number of results in a search. The author used a condition that would generate an alert when the number of results from the search was greater than one within a one-minute window.



The screenshot shows a 'Save As Alert' dialog box. The fields are as follows:

Title	Conpot Modbus Traffic
Description	optional
Alert type	<input type="radio"/> Scheduled <input type="radio"/> Real Time
Trigger condition	Number of Results
Number of results is	Greater than 1
in	1 minute(s)

Buttons: Cancel, Next

Figure 11: Creating the trigger condition in Splunk for a Conpot event alert.

A very determined attacker, willing to go slowly, might not be detected, but most likely will set off an alert during a port or service scan. On the next page (as shown in the following screen capture), the alert is configured to show in the Triggered Alerts activity within Splunk, the severity is set, and the recipient email address configured. The author configured the alert to include only the links to the alert and search results so that sensitive information is not included in the email. Instead, someone will have to log into Splunk and view the generated alerts.

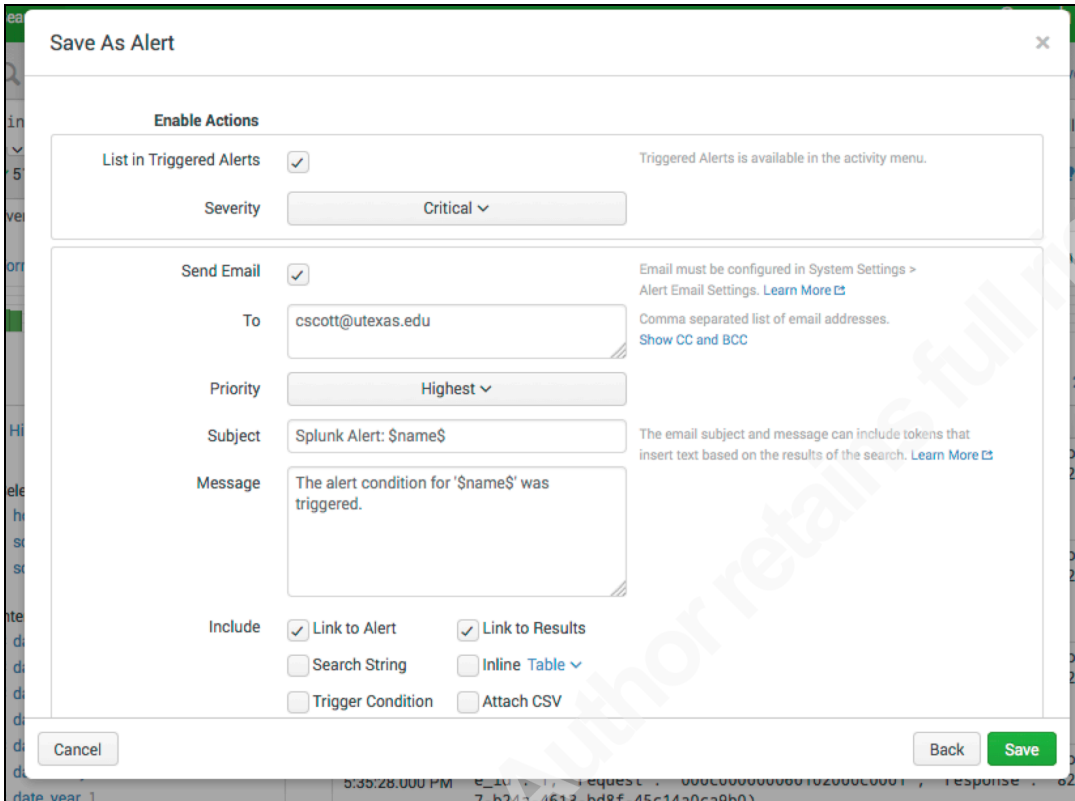


Figure 12: Setting the actions in Splunk for the Conpot event alert.

2.5.4. Generating and Reading Alerts

A Nessus scan of the Conpot honeypot using a default, Basic scan profile will trigger all three of these alerts: HTTP, SNMP, and Modbus. Indeed, doing so triggered an email alert sent to the author, as seen in the following screen capture.

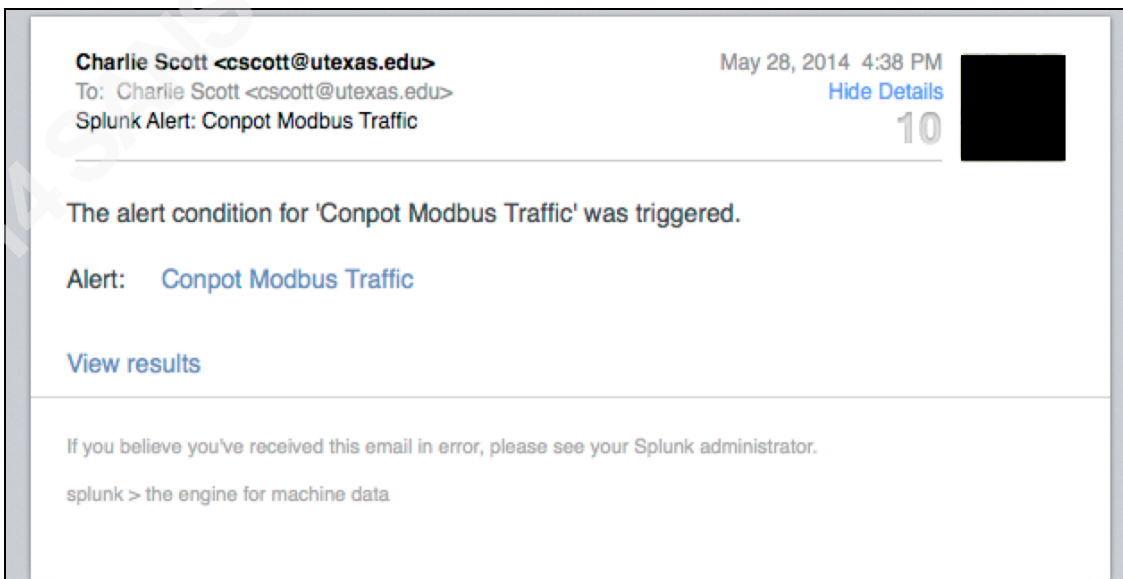


Figure 13: Sample email alert from Splunk generated by Modbus traffic against the honeypot.

Charlie Scott, cscott@utexas.edu

The “Conpot Modbus Traffic” hyperlink in the email links to the trigger history for that alert, while the “View results” hyperlink links to the specific search that caused the email alert. Also, under the Activity menu, there is a Triggered Alerts option that displays a report showing the fired alerts, including their severity, as shown in the following screen shot.

Time	Fired alerts	App	Type	Severity	Mode	Actions
2014-05-28 16:33:44 CDT	Conpot Modbus Traffic	search	Real-time	Critical	Digest	View results Edit search Delete
2014-05-28 16:33:44 CDT	Conpot HTTP GET Request	search	Real-time	High	Digest	View results Edit search Delete
2014-05-28 16:33:41 CDT	Conpot Modbus Traffic	search	Real-time	Critical	Digest	View results Edit search Delete
2014-05-28 16:33:41 CDT	Conpot HTTP GET Request	search	Real-time	High	Digest	View results Edit search Delete
2014-05-28 16:33:39 CDT	Conpot HTTP GET Request	search	Real-time	High	Digest	View results Edit search Delete
2014-05-28 16:33:38 CDT	Conpot HTTP GET Request	search	Real-time	High	Digest	View results Edit search Delete
2014-05-28 16:33:38 CDT	Conpot Modbus Traffic	search	Real-time	Critical	Digest	View results Edit search Delete
2014-05-28 16:33:35 CDT	Conpot Modbus Traffic	search	Real-time	Critical	Digest	View results Edit search Delete
2014-05-28 16:33:35 CDT	Conpot SNMP Sesssion	search	Real-time	Medium	Digest	View results Edit search Delete
2014-05-28 16:33:35 CDT	Conpot HTTP GET Request	search	Real-time	High	Digest	View results Edit search Delete
2014-05-28 16:33:34 CDT	Conpot HTTP GET Request	search	Real-time	High	Digest	View results Edit search Delete
2014-05-28 16:33:32 CDT	Conpot SNMP Sesssion	search	Real-time	Medium	Digest	View results Edit search Delete
2014-05-28 16:33:31 CDT	Conpot HTTP GET Request	search	Real-time	High	Digest	View results Edit search Delete
2014-05-28 16:33:29 CDT	Conpot SNMP Sesssion	search	Real-time	Medium	Digest	View results Edit search Delete
2014-05-28 16:33:28 CDT	Conpot HTTP GET Request	search	Real-time	High	Digest	View results Edit search Delete
2014-05-28 16:33:26 CDT	Conpot SNMP Sesssion	search	Real-time	Medium	Digest	View results Edit search Delete
2014-05-28 16:33:25 CDT	Conpot HTTP GET Request	search	Real-time	High	Digest	View results Edit search Delete

Figure 14: A “Triggered Alerts” report in Splunk.

3. Conclusion

SCADA networks typically contain business-critical and mission-critical devices. Consequently, anything that might cause support or downtime issues, such as an anti-virus, IDS, or firewall, is often avoided. A low-interaction honeypot can be an effective means of detecting hostile scanning and other activity on a SCADA network without modifying the existing network and system configurations. One caveat with honeypots is that they present potential legal complications, so it is best to check with a lawyer before embarking on this path.

Charlie Scott, cscott@utexas.edu

There are several SCADA honeypot projects that already exist, making implementation easier than starting from scratch with a generic honeypot configuration. Conpot is a good choice because it is under active development. This honeypot can be even more effective if it simulates a device that actually exists on the SCADA network, so understanding the attack surface of that network is important, and can be done with a scanner such as Nmap, Nessus, or other tools. To keep it simple, choose a device that has only a few services, but that might still be interesting to a potential attacker. Once a device is chosen, it is trivial to modify the Conpot configuration to match the HTTP, SNMP, and Modbus configurations on that host.

Because the SCADA network is not connected directly to a control network, intermediary logging and monitoring systems are required to collect and present information to the operators. Sending the honeypot logs to a Syslog server and indexing them with Splunk can allow the security operator to easily search honeypot activity, and be alerted when it appears that an attack is in progress. This allows a security operator to respond quickly to an event that might not have even been detectable before.

4. References

- Byres, E. (2012, October 12). *The Critical SCADA Security Patch that your Control System Isn't Getting*. Retrieved March 8, 2014, from Tofino Security: <https://www.tofinosecurity.com/blog/critical-scada-security-patch-your-control-system-isn%E2%80%99t-getting>
- Digital Bond, Inc. (2014, March 1). *SCADA Honeynet*. Retrieved March 1, 2014, from Digital Bond: <https://www.digitalbond.com/tools/scada-honeynet/>
- Forner, E., & Meixell, B. (2013, July 27). *Out of Control: Demonstrating SCADA Exploitation*. Retrieved January 12, 2014, from Black Hat Security Conference Media: <https://media.blackhat.com/us-13/US-13-Forner-Out-of-Control-Demonstrating-SCADA-WP.pdf>
- Glastopf Developers. (2013, January 1). *Ubuntu 12.04 LTS*. Retrieved May 21, 2014, from Conpot 0.2.2 Documentation: <https://glastopf.github.io/conpot/installation/ubuntu.html>
- Haslinger, D. (2014, April 18). *Conpot not initialising #125*. Retrieved May 24, 2014, from Github: <https://github.com/glastopf/conpot/issues/125>
- Higgins, K. J. (2014, January 15). *SCADA Researcher Drops Zero-Day, ICS-CERT Issues Advisory*. Retrieved February 1, 2014, from Dark Reading: <http://www.darkreading.com/applications/scada-researcher-drops-zero-day-ics-cert/240165420>
- Higgins, K. J. (2013, January 15). *The SCADA Patch Problem*. Retrieved January 12, 2014, from Dark Reading: <http://www.darkreading.com/vulnerability/the-scada-patch-problem/240146355>
- Hruska, J. (2009, March 25). *BIOS-level rootkit attack scary, but hard to pull off*. Retrieved May 28, 2014, from Ars Technica: <http://arstechnica.com/gadgets/2009/03/researchers-demonstrate-bios-level-rootkit-attack/>
- Knapp, E. D. (2011). *Industrial Network Security*. Waltham, MA, USA: Syngress.
- Luallen, M. E. (2013, February 1). *SANS SCADA and Process Control Security Survey*. Retrieved January 12, 2014, from SANS Institute: <https://www.sans.org/reading-room/analysts-program/sans-survey-scada-2013>
- Pothametsy, V., & Franz, M. (2005, July 15). *SCADA HoneyNet Project: Building Honeypots for Industrial Networks*. Retrieved February 1, 2014, from SCADA

Charlie Scott, cscott@utexas.edu

HoneyNet Project: Building Honeypots for Industrial Networks :

<http://scadahoneynet.sourceforge.net/>

Provos, N. (2008, July 15). *Developments of the Honeyd Virtual Honeypot* . Retrieved March 8, 2014, from Developments of the Honeyd Virtual Honeypot :

<http://www.honeyd.org/>

Provos, N., & Holz, T. (2008). *Virtual Honeypots: From Botnet Tracking to Intrusion Detection* . Boston, Massachusetts, US: Pearson Education, Inc.

Rist, L. (2013, November 11). *glstopf/modbus-tk*. Retrieved May 25, 2014, from Github: <https://github.com/glstopf/modbus-tk>

Rist, L., Vestergaard, J., & Haslinger, D. (2014, February 1). *CONPOT*. Retrieved February 1, 2014, from CONPOT: <http://conpot.org/>

Rsyslog Community. (2014, May 28). *rsyslog*. Retrieved May 28, 2014, from rsyslog: <http://www.rsyslog.com/>

Schnieder Electric. (2010, January 1). *Gain energy and insight with Power Logic*. Retrieved May 14, 2014, from Schneider Electric Corporate:

http://download.schneider-electric.com/files?p_File_Id=27479876&p_File_Name=PLSED106014EN.pdf

Splunk, Inc. (2014, May 29). *What is Splunk Enterprise?* Retrieved May 29, 2014, from Splunk: <http://www.splunk.com/view/splunk/SP-CAAAG57>

Stouffer, K., Falco, J., & Scarfone, K. (2011, June 1). *NIST Special Publication 800-82: Guide to Industrial Control Systems (ICS) Security* . Retrieved March 8, 2014, from NIST.gov - Computer Security Division - Computer Security Resource Center: <http://csrc.nist.gov/publications/nistpubs/800-82/SP800-82-final.pdf>

Tenable Network Security. (2014, 04 13). *Plugins: SCADA*. Retrieved 04 13, 2014, from Tenable Network Security:

<http://www.tenable.com/plugins/index.php?view=all&family=SCADA>

Ubuntu Community. (2014, May 20). *Releases*. Retrieved May 21, 2014, from Ubuntu Wiki: <https://wiki.ubuntu.com/Releases>

Wade, S. M. (2011, 1 1). *SCADA Honeynets: The attractiveness of honeypots as critical infrastructure security tools for the detection and analysis of advanced threats*.

Retrieved February 3, 2014, from Graduate Theses and Dissertations:

<http://lib.dr.iastate.edu/etd/12138/>

Weiss, J. (2010). *Protecting Industrial Control Systems from Electronic Threats*. New York, New York, United States of America: Momentum Press, LLC.

Zetter, K. (2011, July 11). *How Digital Detectives Deciphered Stuxnet, the Most Menacing Malware in History*. Retrieved January 12, 2014, from Wired: <http://www.wired.com/threatlevel/2011/07/how-digital-detectives-deciphered-stuxnet/>

Zetter, K. (2012, May 28). *Meet 'Flame,' The Massive Spy Malware Infiltrating Iranian Computers*. Retrieved March 8, 2013, from Wired: <http://www.wired.com/threatlevel/2012/05/flame/all/>

Zubairi, J. A., & Mahboob, A. (2013, December 11). *Securing SCADA Systems with Open Source Software*. Retrieved March 8, 2014, from DHA Suffa University: <http://www.dsu.edu.pk/index.php/en/downloads/category/7-dsu-workshop?download=60:protecting-scada-systems-using-open-source-software-honet-2013>

5. Appendix A – ion6200.xml Modified Template

The *ion6200.xml* file from */usr/local/lib/python2.7/dist-packages/Conpot-0.2.2-py2.7.egg/conpot/templates*, used to configure Conpot to simulate a Schneider Electric PowerLogic ION6200 smart meter.

```
<conpot_template name="ION6200" description="Simulation of a Schneider Electric ION6200 Smart Meter">
```

```
  <core>
```

```
    <databus>
```

```
      <!-- Core value that can be retrieved from the databus by key -->
```

```
      <key_value_mappings>
```

```
        <key name="FacilityName">
```

```
          <value type="value">"FAC"</value>
```

```
        </key>
```

```
        <key name="SystemName">
```

```
          <value type="value">"FAC eMeter"</value>
```

```
        </key>
```

```
        <key name="SystemDescription">
```

```
          <value type="value">"ION 6200 E1176"</value>
```

```
        </key>
```

```
          <key name="Uptime">
```

```
            <value type="function">conpot.emulators.misc.uptime.Uptime</value>
```

```
          </key>
```

```
        <key name="sysObjectID">
```

```
          <value type="value">"1.3.6.1.4.4346.11.1.2.1.1.15"</value>
```

```
        </key>
```

```
        <key name="sysContact">
```

```
          <value type="value">"Charlie Scott"</value>
```

```
        </key>
```

```
        <key name="sysName">
```

```
          <value type="value">"FAC eMeter"</value>
```

```
        </key>
```

```
        <key name="sysLocation">
```

```
          <value type="value">"FAC"</value>
```

```
        </key>
```

```
        <key name="sysServices">
```

```
          <value type="value">"12"</value>
```

```
        </key>
```

```
        <key name="memoryModbusSlave1BlockA">
```

```
          <value type="value">[random.randint(0,1) for b in  
range(0,128)]</value>
```

```
        </key>
```

Charlie Scott, cscott@utexas.edu


```
<key name="memoryModbusSlave1BlockB">
  <value type="value">[random.randint(0,1) for b in range(0,32)]</value>
</key>
<key name="memoryModbusSlave1BlockC">
  <value type="value">[random.randint(0,1) for b in range(0,8)]</value>
</key>
<key name="memoryModbusSlave1BlockD">
  <value type="value">[0 for b in range(0,32)]</value>
</key>
  <key name="Copyright">
    <value type="value">""</value>
  </key>
<key name="s7_id">
  <value type="value">""</value>
</key>
<key name="s7_module_type">
  <value type="value">""</value>
</key>
<key name="empty">
  <value type="value">""</value>
</key>
</key_value_mappings>
</databus>
</core>
<protocols>
  <snmp enabled="True" host="0.0.0.0" port="161">
    <config>
      <!-- Configure individual delays for SNMP commands -->
      <entity name="tarpit" command="get">0.1;0.2</entity>
      <entity name="tarpit" command="set">0.1;0.2</entity>
      <entity name="tarpit" command="next">0.0;0.1</entity>
      <entity name="tarpit" command="bulk">0.2;0.4</entity>

      <!-- Configure DoS evasion thresholds
      (req_per_ip/minute;req_overall/minute) -->
      <entity name="evasion" command="get">120;240</entity>
      <entity name="evasion" command="set">120;240</entity>
      <entity name="evasion" command="next">240;600</entity>
      <entity name="evasion" command="bulk">120;240</entity>
    </config>
    <mibs>
      <mib name="SNMPv2-MIB">
        <symbol name="sysDescr">
          <!-- Value is key in databus -->
          <value>SystemDescription</value>
        </symbol>
      </mib>
    </mibs>
  </snmp>
</protocols>
</core>
```

```

</symbol>
<symbol name="sysUpTime">
  <value>Uptime</value>
</symbol>
<symbol name="sysContact">
  <value>sysContact</value>
</symbol>
<symbol name="sysName">
  <value>sysName</value>
</symbol>
<symbol name="sysLocation">
  <value>sysLocation</value>
</symbol>
<symbol name="sysServices">
  <value>sysServices</value>
</symbol>
</mib>
</mibs>
</snmp>
<modbus enabled="True" host="0.0.0.0" port="502">
  <device_info>
    <VendorName>Schneider Electric</VendorName>
    <ProductCode>PowerLogic</ProductCode>
    <MajorMinorRevision>ION6200</MajorMinorRevision>
  </device_info>
  <slaves>
    <slave id="1">
      <blocks>
        <block name="memoryModbusSlave1BlockA">
          <!-- COILS/DISCRETE_OUTPUTS aka. binary output, power on/power
off
          Here we map modbus addresses 1 to 127 to S7-200 PLC Addresses
Q0.0 to Q15.7 -->
          <type>COILS</type>
          <starting_address>1</starting_address>
          <size>128</size>
          <content>memoryModbusSlave1BlockA</content>
        </block>
        <block name="memoryModbusSlave1BlockB">
          <!-- CONTACTS/DISCRETE_INPUTS aka. binary input.
          Map modbus addresses 10001-10032 to S7-200 PLC inputs
starting from I0.0 -->
          <type>DISCRETE_INPUTS</type>
          <starting_address>10001</starting_address>
          <size>32</size>

```

```

        <content>memoryModbusSlave1BlockB</content>
    </block>
</blocks>
</slave>
<slave id="2">
    <!-- This slave does some measuring. (analog inputs).
    Map modbus addresses 30001-30009 to S7 PLC analog input bits
AIW0-AIW8 -->
    <blocks>
        <block name="memoryModbusSlave1BlockC">
            <!-- Will be parsed with eval() -->
            <type>ANALOG_INPUTS</type>
            <starting_address>30001</starting_address>
            <size>8</size>
            <content>memoryModbusSlave1BlockC</content>
        </block>
        <block name="memoryModbusSlave1BlockD">
            <!-- Maps to S7-200 PLC addresses HoldStart+8 -->
            <type>HOLDING_REGISTERS</type>
            <starting_address>40001</starting_address>
            <size>8</size>
            <content>memoryModbusSlave1BlockD</content>
        </block>
    </blocks>
</slave>
</slaves>
</modbus>
<s7comm enabled="False" host="0.0.00" port="102">
    <system_status_lists>
        <ssl id="W#16#xy1C" name="Component Identification">
            <system_name id="W#16#0001">SystemName</system_name>
            <module_name id="W#16#0002">SystemDescription</module_name>
            <plant_ident id="W#16#0003">FacilityName</plant_ident>
            <copyright id="W#16#0004">Copyright</copyright>
            <serial id="W#16#0005">s7_id</serial>
            <module_type_name
id="W#16#0007">s7_module_type</module_type_name>
            <oem_id id="W#16#000A">empty</oem_id>
            <location id="W#16#000B">empty</location>
        </ssl>
        <ssl id="W#16#xy11" name="Module Identification">
            <!-- Not really sure what these are supposed to contain -->
            <module_identification
id="W#16#0001">empty</module_identification>

```

```
<hardware_identification
id="W#16#0006">empty</hardware_identification>
  <firmware_identification
id="W#16#0006">empty</firmware_identification>
  </ssl>
</system_status_lists>
</s7comm>
<http enabled="True" host="0.0.0.0" port="80">
  <global>
    <config>
      <!-- what protocol shall we use by default? -->
      <entity name="protocol_version">HTTP/1.1</entity>
      <!-- if we find any date header to be delivered, should we update it to a
real value? -->
      <entity name="update_header_date">>true</entity>
      <!-- should we disable the HTTP HEAD method? -->
      <entity name="disable_method_head">>false</entity>
      <!-- should we disable the HTTP TRACE method? -->
      <entity name="disable_method_trace">>false</entity>
      <!-- should we disable the HTTP OPTIONS method? -->
      <entity name="disable_method_options">>false</entity>
      <!-- TARPIT: how much latency should we introduce to any response by
default? -->
      <entity name="tarpit">0</entity>
    </config>

    <!-- these headers will be sent with each response -->
    <headers>
      <!-- this date header will be updated, if enabled above -->
      <entity name="Date">Sat, 28 Apr 1984 07:30:00 GMT</entity>
    </headers>
  </global>

  <!-- how should the different URI requests be handled -->
  <htdocs>
    <node name="/">
      <!-- force response status code to 302 -->
      <status>302</status>
      <headers>
        <!-- these headers will be sent along with this response -->
        <entity name="Content-Type">text/html</entity>
        <entity name="Location">/index.html</entity>
      </headers>
    </node>
    <node name="/index.html">
```

```
<!-- this tarpit will override the globally set tarpit for this node -->
<tarpit>0.0;0.3</tarpit>
<headers>
  <entity name="Last-Modified">Tue, 19 May 1993 09:00:00
GMT</entity>
  <entity name="Content-Type">text/html</entity>
  <entity name="Set-cookie">path=/
```

```
<headers>
  <entity name="Last-Modified">Tue, 19 May 1993 09:00:00
GMT</entity>
  <entity name="Content-Type">text/html</entity>
</headers>
</node>
</htdocs>

<!-- how should the different status codes be handled -->
<statuscodes>
  <status name="400">
    <!-- 400 (BAD REQUEST) errors should be super fast and responsive -->
    <tarpit>0</tarpit>
    <entity name="Content-Type">text/html</entity>
  </status>
  <status name="404">
    <!-- 404 (NOT FOUND) errors should be super fast and responsive -->
    <tarpit>0</tarpit>
    <entity name="Content-Type">text/html</entity>
  </status>
  <status name="501">
    <!-- 501 (NOT IMPLEMENTED) errors should be super fast and
responsive -->
    <tarpit>0</tarpit>
    <entity name="Content-Type">text/html</entity>
  </status>
</statuscodes>
</http>
</protocols>
</conpot_template>
```



Upcoming SANS Training

[Click Here for a full list of all Upcoming SANS Events by Location](#)

Rocky Mountain Fall 2017	Denver, COUS	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS Baltimore Fall 2017	Baltimore, MDUS	Sep 25, 2017 - Sep 30, 2017	Live Event
Data Breach Summit & Training	Chicago, ILUS	Sep 25, 2017 - Oct 02, 2017	Live Event
SANS Copenhagen 2017	Copenhagen, DK	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS London September 2017	London, GB	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS Oslo Autumn 2017	Oslo, NO	Oct 02, 2017 - Oct 07, 2017	Live Event
SANS DFIR Prague 2017	Prague, CZ	Oct 02, 2017 - Oct 08, 2017	Live Event
SANS Phoenix-Mesa 2017	Mesa, AZUS	Oct 09, 2017 - Oct 14, 2017	Live Event
SANS October Singapore 2017	Singapore, SG	Oct 09, 2017 - Oct 28, 2017	Live Event
Secure DevOps Summit & Training	Denver, COUS	Oct 10, 2017 - Oct 17, 2017	Live Event
SANS Tysons Corner Fall 2017	McLean, VAUS	Oct 14, 2017 - Oct 21, 2017	Live Event
SANS Brussels Autumn 2017	Brussels, BE	Oct 16, 2017 - Oct 21, 2017	Live Event
SANS Tokyo Autumn 2017	Tokyo, JP	Oct 16, 2017 - Oct 28, 2017	Live Event
SANS Berlin 2017	Berlin, DE	Oct 23, 2017 - Oct 28, 2017	Live Event
SANS Seattle 2017	Seattle, WAUS	Oct 30, 2017 - Nov 04, 2017	Live Event
SANS San Diego 2017	San Diego, CAUS	Oct 30, 2017 - Nov 04, 2017	Live Event
SANS Gulf Region 2017	Dubai, AE	Nov 04, 2017 - Nov 16, 2017	Live Event
SANS Miami 2017	Miami, FLUS	Nov 06, 2017 - Nov 11, 2017	Live Event
SANS Milan November 2017	Milan, IT	Nov 06, 2017 - Nov 11, 2017	Live Event
SANS Amsterdam 2017	Amsterdam, NL	Nov 06, 2017 - Nov 11, 2017	Live Event
SANS Paris November 2017	Paris, FR	Nov 13, 2017 - Nov 18, 2017	Live Event
Pen Test Hackfest Summit & Training 2017	Bethesda, MDUS	Nov 13, 2017 - Nov 20, 2017	Live Event
SANS Sydney 2017	Sydney, AU	Nov 13, 2017 - Nov 25, 2017	Live Event
SANS London November 2017	London, GB	Nov 27, 2017 - Dec 02, 2017	Live Event
SANS San Francisco Winter 2017	San Francisco, CAUS	Nov 27, 2017 - Dec 02, 2017	Live Event
SIEM & Tactical Analytics Summit & Training	Scottsdale, AZUS	Nov 28, 2017 - Dec 05, 2017	Live Event
SANS Khobar 2017	Khobar, SA	Dec 02, 2017 - Dec 07, 2017	Live Event
SANS Munich December 2017	Munich, DE	Dec 04, 2017 - Dec 09, 2017	Live Event
European Security Awareness Summit 2017	London, GB	Dec 04, 2017 - Dec 07, 2017	Live Event
SANS Austin Winter 2017	Austin, TXUS	Dec 04, 2017 - Dec 09, 2017	Live Event
SANS Frankfurt 2017	Frankfurt, DE	Dec 11, 2017 - Dec 16, 2017	Live Event
SANS Bangalore 2017	Bangalore, IN	Dec 11, 2017 - Dec 16, 2017	Live Event
SANS SEC504 at Cyber Security Week 2017	OnlineNL	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS OnDemand	Books & MP3s OnlyUS	Anytime	Self Paced