Industrial Control System Honeypot

Group May1601

Team Aashwatth Agarwal, Dan Borgerding, Jon Hope, Nik Kinkel, Jon Osborne, Korbin Stich

Advisor Dr. Doug Jacobson

Client Alliant Energy

Introduction

Background

Supervisory Control and Data Acquisition (SCADA) systems play a vital role in maintaining and controlling critical infrastructure such as water treatment plants, oil pipelines, HVAC Systems, and the power grid. The societal importance of these services makes them a high-value target for well-funded, highly-skilled adversaries who increasingly turn to digital and electronic attack vectors, exploiting old and outdated protocols that were never designed to be exposed to the public internet. Thus, rapid intrusion detection and response is crucial for safeguarding SCADA networks.

A *honeypot* (a system designed to mimic a legitimate protocol, e.g. SSH, coerce an attacker into interacting with it, and then report attacker activity to an administrator) is an old and well-understood approach to detecting network intruders.

The Problem

Our client, Alliant Energy, requested many small, low-maintenance honeypot systems, each capable of speaking multiple network protocols (both SCADA and non-SCADA), able to log to many different backends, and able to sniff local network traffic for anomalies, to place inside each of their 28 electrical substations.

Unfortunately, many mainstream honeypot implementations have a number of problems that make them unsuitable for deployment in Alliant's substations, including:

- **Unsafe Languages:** Many existing honeypots rely on code written in languages without memory- or type-safety, unacceptable for a high-security environment.
- **Single-Protocol:** Most honeypots are designed to speak only a single protocol.
- **Complex Deployment:** Open source honepots are generally not designed to be deployed, updated, and configured en masse.
- **Expensive Hardware:** Commercial honeypots are too expensive to place in multiple locations across many subnets.

Solution

We designed a honeypot *plugin framework* (Figure 2) to enforce high-security process isolation, extreme extensibility, and easy and comprehensive testing. The system is tailored to run on Raspberry Pi consumer hardware and is cheap enough to buy in quantity, while still being powerful enough to run an Intrusion Detection System (IDS) capable of watching local network traffic for anomalies. Finally, we delivered a set of configuration management utilities that allow our client to deploy to many different locations and update all devices in a *single-step*.

Project Requirements

Functional	Non-Functional
Low-interaction SSH, HTTP/HTTPS and DNP3 honeypots	Low maintenance
Minimal IDS	Low power consumption
One-step mass deployment and remote management	Cheap
Highly extensible	Withstand harsh substation environment

Concept

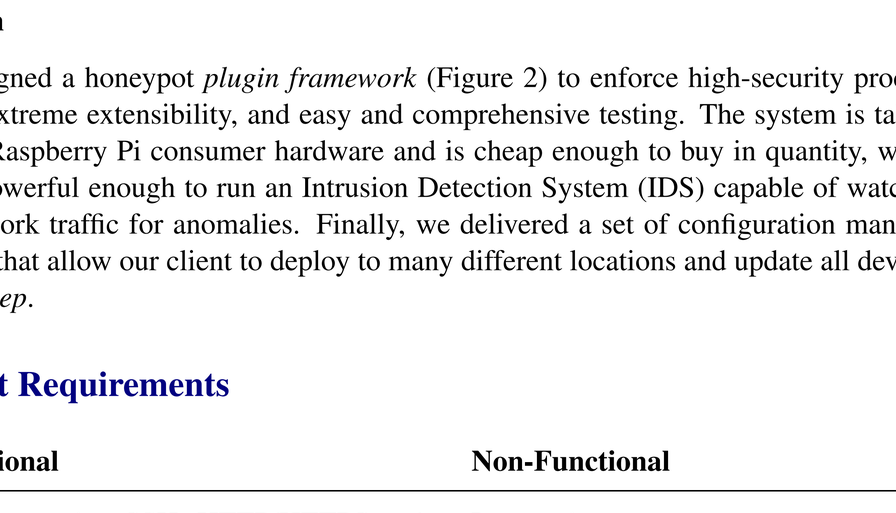

Figure 1: High Level Design

Each device has two network interfaces: one in promiscuous mode forwarding local network traffic to an internal IDS, and another forwarding incoming traffic to the honeypot plugin system. Local device firewalls, administrator keys, package dependencies, and other site-specific artifacts are setup globally through the Ansible configuration management system.

Plugin Framework Design

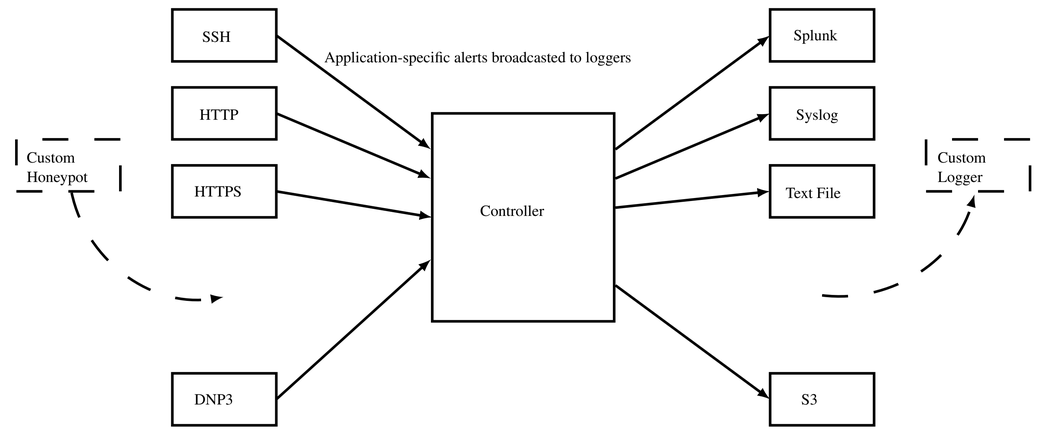

Figure 2: Pluggable Honeypot and Logging Framework

Figure 2 shows the basic architecture of the honeypot plugin framework. Both honeypot and logging plugins run as separate processes and are completely isolated from one another. Communication is done via a message passing protocol, where honeypots submit events to the controller, and the controller then broadcasts the event to all logger plugins.

New plugins need only implement a simple, two-function interface to be automatically integrated into the broader plugin system. This approach means that plugins for new protocols and logging backends can be developed rapidly, tested in isolation, and integrated into the whole system easily.

Functional Modules

Component	Technical Details
Plugin Framework Controller	Go programming language. Manages child honeypot and logger subprocesses, coordinates communication over messaging interface (localhost TCP or UNIX sockets).
Honeypot Plugins	Go programming language. Mimics legitimate network protocol handshakes (e.g. SSH, DNP3), collects network activity metadata, and sends to controller.
Logger Plugins	Go programming language. Receives alerts from controller as a byte string and submits to particular backend (e.g. local database, remote logging endpoint).
Intrusion Detection System	Industry-standard Snort IDS. Configured with custom ruleset designed for small substation SCADA network. Pushes alerts to remote endpoint.
Configuration Management	Ansible Provisioning Software. Variable arguments/paremeters allow administrators to configure and deploy the honeypot system to multiple locations.

Testing

Strategy

Unit Testing

- Each honeypot plugin tested in isolation
- Failures, crashes, and malicious input simulated for controller
- Isolated design components leads to effective functional-style testing

Integration Testing

- Each honeypot protocol connected to, used, log output verified
- Real-world attacks simulated
- Stress-tested for high traffic volume
- Automatic streamlined VM Provisioning

Environment

Simulated Substation Network

- Vagrant used to create a minimal test network of virtual machines
- Separate machines for virtual device, logging endpoints, and simulated attacker
- IDS rules tested with live traffic
- Inter-system traffic and behavior monitored in real time