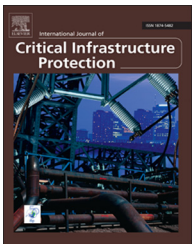
Available online at www.sciencedirect.com

ScienceDirect

www.elsevier.com/locate/ijcip



Constructing cost-effective and targetable industrial control system honeypots for production networks

Michael Winn[a], Mason Rice[a,*], Stephen Dunlap[a], Juan Lopez[b], Barry Mullins[a]

[a]Department of Electrical and Computer Engineering, Air Force Institute of Technology, Wright-Patterson Air Force Base, OH 45433, USA

[b]Applied Research Solutions, 51 Plum Street, Beavercreek, OH 45440, USA

ARTICLE INFO

Article history:
Received 11 January 2015
Received in revised form
1 April 2015
Accepted 15 April 2015
Available online 1 May 2015

Keywords:
Industrial control systems
Programmable logic controllers
Production networks
Honeypots

ABSTRACT

Critical infrastructure assets – and especially industrial control systems – are at risk. Malicious actors are constantly developing exploits that sneak past security controls. Honeypots offer an opportunity to acquire knowledge about the tactics, techniques and procedures used by malicious entities to compromise sensitive systems. However, the proprietary, and often expensive, hardware and software used by industrial control systems make it very challenging to build flexible, economical and scalable honeypots. This paper describes a technique that uses proxy technology to produce multiple high-interaction honeypots using a single programmable logic controller. The technique provides a cost-effective method for distributing multiple, authentic, targetable honeypots at slightly more than the cost of a single programmable logic controller.

Published by Elsevier B.V.

1. Introduction

On November 20, 2014, the Director of the National Security Agency, Admiral Michael Rogers, stated that several entities, including China, Russia and others, have the ability to disrupt electric utilities and other energy assets throughout the United States, potentially causing physical destruction, personal injury and even death [4]. Admiral Rogers expressed the desire to share threat information with the private sector, but he also implied that the private sector lacks the ability to collect data that could help prevent, detect and recover from cyber attacks. This is due, in part, to the reliance of IP-based protection devices such as intrusion detection systems and firewalls.

Further compounding the problem, malicious actors have a good understanding of current signature-based sensor technologies and can engineer malware that eludes such systems, making their activities almost undetectable. A honeypot is a proven method for learning about attacker tactics, techniques and procedures. However, the financial cost of implementing practical honeypots in industrial control networks is a major barrier.

This paper presents a technique for constructing low-cost industrial control system honeypots that are both authentic and targetable. The following section describes the background and establishes the context for the honeypot technique. Next, the technique involving the use of proxy technology is developed and the evaluation methods are described. Finally, the results of the evaluation are presented, along with the main conclusions and directions for further research.

2. Background

Successful honeypots balance authenticity, targetability, cost and risk. An authentic honeypot mimics the features of an operational system. More realistic features yield a more complex honeypot

*Corresponding author.
E-mail address: mason.rice@afit.edu (M. Rice).

http://dx.doi.org/10.1016/j.ijcip.2015.04.002
1874-5482/Published by Elsevier B.V.

that can collect more detailed data. A honeypot is useless if it is avoided by attackers. Therefore, a honeypot must be targetable: it should have a large enough presence to attract attackers. The financial cost of a honeypot is also a major consideration, especially with regard to its development, deployment, maintenance and operation, as well as the analysis of the collected data.

Several risks are associated with the use of a honeypot. These include attracting attackers and dealing with the consequences of an attacker discovering the honeypot and disabling or revealing the existence of the honeypot to others. The risks, consequences and mitigation measures must be considered very carefully before implementing a honeypot solution.

Some honeypots, called production honeypots, are created to function as decoy systems. The decoys obscure valuable assets while logging mechanisms such as `syslog` monitor anomalous activities. A targetable honeypot should incorporate multiple decoys to attract attacker attention away from hosts on a network. The decoys should also be positioned in relevant surroundings. For example, a lone programmable logic controller (PLC) on the Internet is of little interest to attackers because it is unlikely to be a part of a larger system. The decoy systems should also appear and function identically as their counterparts so that attackers cannot distinguish real systems from decoys. However, the cost of implementing such authenticity multiplies as the size (i.e., targetability) grows, quickly making this solution prohibitive. In other words, large honeypots should have footprints that are large enough to attract potential attackers, but the complexity of the equipment and the programming required generally results in less authenticity than what is required to capture and maintain the attention of serious malicious actors.

In order to balance cost with size and authenticity, a honeypot is designed to collect data about the different stages of a cyber attack: reconnaissance, enumeration, gaining initial access and maintaining access. Honeypots that are minimally authentic are suitable for detecting common reconnaissance probes while high authenticity honeypots are required to study complex attacks such as advanced persistent threats.

2.1. *Traditional honeypots*

Research on traditional (information technology) honeypots is plentiful and continues to evolve. Prior to 2006, traditional high-interaction honeypots were limited by costly physical hardware. Several research efforts, including the Honeynet Project [12], have examined the problem of creating and implementing realistic honeypots with suitable data collection mechanisms while minimizing their development, deployment and maintenance costs. Virtualization technologies became widespread in the commodity information technology market between 2006 and 2008, making it convenient and inexpensive to deploy multiple high-interaction virtual machines as honeypots to collect detailed data on cyber threats. However, traditional information technology honeypot techniques do not transfer or scale easily to industrial control systems.

2.2. *Industrial control system honeypots*

Industrial control system environments cannot leverage the advantages of virtualization like traditional information technology environments, so very few honeypot options are available that offer the same levels of functionality and flexibility as for traditional systems. Most current industrial control system honeypots incorporate low-interaction emulators [8]. Although low-interaction honeypots can be deployed on low-cost, flexible platforms such as Raspberry Pi or Gumstix devices, considerable effort is required to implement vendor-specific protocols and the platforms usually feature a limited set of capabilities (e.g., lack of authenticity and small size) [2,6].

High-interaction industrial control system honeypots can demonstrate more flexibility and the ability to collect detailed data on attack tactics, techniques and procedures. However, the cost of a full-scale test bed can be prohibitive [5]. Indeed, full-scale systems are extremely rare and their use is limited to controlled experimentation and research – they are rarely exposed to the Internet to collect live threat data. Furthermore, the test beds are often built to model specific environments (e.g., electric power grid, water treatment plant or refinery) and it is generally infeasible to adapt them to diverse industrial control environments.

A single programmable logic controller can cost several hundred dollars (e.g., approximately $900 for an Omron CP1L00 device) to several thousand dollars (e.g., approximately $5000 for an Allen-Bradley 1756-L61 system). The cost of procuring and deploying multiple devices limits their scalability (especially as somewhat expendable honeypots) [3,13]. For instance, the cost of deploying an authentic, targetable honeypot consisting of 30 L61 programmable logic controllers would amount to more than $165,000 for the initial hardware alone.

2.3. *Honeyd*

Honeyd is a low-interaction honeypot framework developed by Provos [10]. It is open-source software available free-of-charge under a GNU General Public License. Honeyd was designed to simulate network topologies and behavior for studying and defeating Internet worm propagation, including the well-known Slammer, Code Red and Blaster worms.

Honeyd provides an ideal framework for creating a high-interaction honeypot without having to deploy dozens of expensive programmable logic controllers. Specifically, Honeyd is freely available and can be used to create a large footprint. Honeyd also supports protocol independence in order to accommodate a wide range of industrial control equipment and can achieve the size required to implement effective production honeypots.

The central component of Honeyd is a plaintext configuration file that defines a host template. The Honeyd template specifies the characteristics that a honeypot host displays to attackers. A Honeyd host is created by associating an IP address with a template using the keyword “bind.”

The main characteristic of a Honeyd template is a “personality,” which is also the keyword used to identify the operating system that is to be mimicked by the template. Honeyd personalities are an implementation of the `nmap` fingerprint database `nmap-os-db` [7]. Honeyd can also generate media access control (MAC) addresses. With a specified name, Honeyd matches the vendor portion of the MAC address from the `nmap` implementation of the IEEE Organizationally Unique Identifiers (OUI) table [7] and generates the

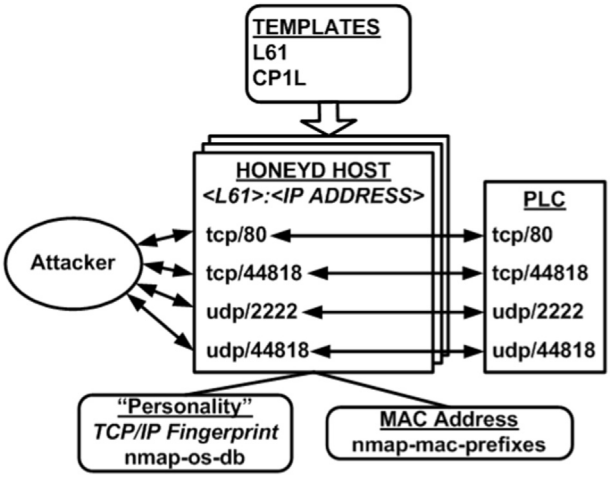


Fig. 1 – Honeyd host.

remaining host portion. The Ethernet address can also be explicitly specified, if desired.

Other characteristics of a template include ICMP type, TCP ports and UDP ports. The essence of the low-interaction capability provided by Honeyd is the response, which is carried out by associating scripts, executables or subsystems with ports. Honeyd features a built-in proxy capability that can forward incoming connections to a physical `host:port`, providing the potential for high-interaction functionality [9]. Fig. 1 presents the implementation of Honeyd and the interaction between an attacker and a programmable logic controller.

3. Honeypot methodology

This section describes the honeypot design considerations, scope and implementation.

3.1. *Design considerations*

The goal is to create a low-cost industrial control system honeypot that incorporates the key characteristics of a successful honeypot, especially targetability and high-authenticity. Honeyd is used as the underlying platform for the honeypot due to its inherent ability to generate a large footprint. The target should be convincing enough that an attacker, either human or automated, would be encouraged to progress through the reconnaissance, enumeration and initial access phases of an attack.

A common tactic during the reconnaissance and enumeration stages is to use a port scanning and fingerprinting tool such as `nmap`. An initial scan would quickly reveal the common TCP port 80 as open. Further exploration of the web page would reveal that the device is a programmable logic controller and lead to more specific reconnaissance and possibly exploitation of the decoy.

Fig. 2 shows `Zenmap` visualizations of an `nmap` scan of an empty subnet and a scan of the same subnet after launching Honeyd configured with a template of 75 Allen-Bradley 1756 ControlLogix programmable logic controllers. (`Zenmap` is an open source visualization tool that displays the `nmap` output

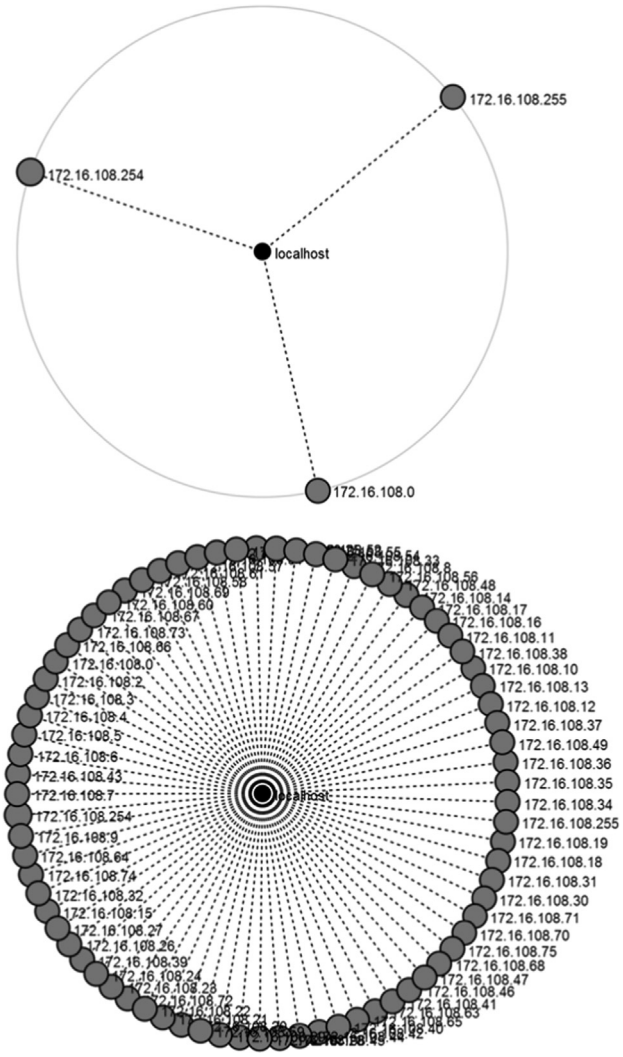


Fig. 2 – Attacker's perspective: empty subnet (top) with 75 Honeyd hosts (bottom).

in an interactive graphical user interface (GUI) [7].) Each dot in the figure is labeled with the IP address that responded to the `nmap` probes. The `Zenmap` GUI can display more detailed information collected from individual hosts.

3.2. *Pilot studies*

Pilot studies were required to determine the feasibility of using Honeyd as an industrial control system honeypot and to shape the specific metrics for evaluating its authenticity and targetability. The first pilot study involved the fingerprinting and exhaustive enumeration of the full range of ICMP/TCP/UDP responses for each programmable logic controller in the test environment using `nmap` and the command line options shown in Table 1.

The results were used to construct the Honeyd templates. To enhance efficiency, only the well-known ports (1–1024) and the industrial control system port (44818) were included in the subsequent experiments. This is a safe assumption because `nmap` requires at least one open and one closed port to conduct fingerprinting. Additional ports are unlikely to

Table 1 – nmap exhaustive enumeration options.

Option	Description
-sS	TCP SYN scan
-sU	UDP scan
-p*	Scan TCP and UDP ports 1–65535
-PE	ICMP echo probes
-PP	ICMP timestamp probes
-A	Enable operating system detection, version detection, script scanning and traceroute
-sC	Run all 101 nmap scripts in the "Default" category
-T4	Aggressive timing
-vv	Very verbose output

appear on the programmable logic controller or Honeyd platforms without intentional configuration.

The second pilot study involved a performance test to establish the range of data rates that would provide a quantifiable metric for evaluating targetability. A Python script was written to test the EtherNet/IP (ENIP) and HTTP protocols by opening a TCP connection to the programmable logic controller or Honeyd host, sending a request, asynchronously collecting and evaluating the response, and repeating the process for a fixed period of time. Connections were executed in separate threads in order to simulate simultaneous connections. At the end of the test duration, the TCP connections were closed gracefully and the Python script outputted the results to a text file.

The pilot study validated the accuracy of the performance testing script and established reasonable durations for testing. Testing conducted over 10, 30 and 60 s durations for each measurement revealed that 30 s was adequate.

The performance pilot study also revealed that, as the sending data rate increased, the programmable logic controllers responded by reducing the TCP window size (eventually to zero). Thus, basing the error rate on the traditional equation

$$Error\ Rate = \frac{send_{success} - receive_{success}}{send_{success}} \quad (1)$$

resulted in misleading measurements.

In the experiments described in this paper, the error rate was measured by considering the number of requests attempted against the number of valid responses

$$Error\ Rate = \frac{(send_{attempt} - send_{success}) + (send_{success} - receive_{success})}{send_{attempt}} \quad (2)$$

The TCP window phenomenon occurred consistently at error rates above 25%. Therefore, send rates for the performance test were established just below the 25% threshold (shown in Table 2).

Finally, the pilot study showed that the maximum number of simultaneous EtherNet/IP sessions was 64 for the L61 programmable logic controller and only four for the CP1L programmable logic controller. Testing the maximum number of sessions involved iteratively creating a TCP connection followed by registering an EtherNet/IP session. Both the programmable logic controllers stopped responding to the register_session request when they reached their maximums. The connection limitation is documented in the L61 specifications [1]; however, the limitation is not listed in the CP1L and CP1W EtherNet/IP module specifications. The limitations apply to the maximum number of simultaneous EtherNet/IP sessions that a programmable logic controller can handle at any given time, but they do not impact the ability of Honeyd to advertise hosts.

Table 2 – Data rates (requests per second).

Device	ENIP	HTTP
L61	300	50
CP1L	80	10

The TCP connections for the HTTP protocol timed out (e.g., RST) when no data was sent, and the programmable logic controller closed the connection gracefully (e.g., FIN, FIN-ACK, RST) at the conclusion of each reply. Note that the performance of the web server functionality was substantially lower than that of EtherNet/IP, especially for the CP1L programmable logic controller.

3.3. *Improving authenticity*

An authentic network device has a unique identity (e.g., IP address or MAC address) that distinguishes it from other network devices. Serial numbers and hostnames are examples of other types of identifiers. The unique identity must be consistent. When an attacker connects to a device using a certain address and queries the device for its identity, the response should consistently match what the attacker expects to see.

Extending the first pilot study, Honeyd was configured to advertise 75 programmable logic controllers. The results appeared successful at first (Fig. 2). However, authenticity flaws were discovered upon deeper examination of the Honeyd hosts. As seen in Fig. 3, all the Honeyd hosts were identical (i.e., no host was unique in any manner). A request for the web page from any of the Honeyd hosts returned the same static web page for the physical programmable logic controller, MAC address, hostname and serial number. Additionally, the IP address reported in the web page did not match the IP address used to connect to the Honeyd host. Native Honeyd and programmable logic controllers lack the ability to dynamically change these values; however, the Honeyd source code can be modified to accomplish this task.

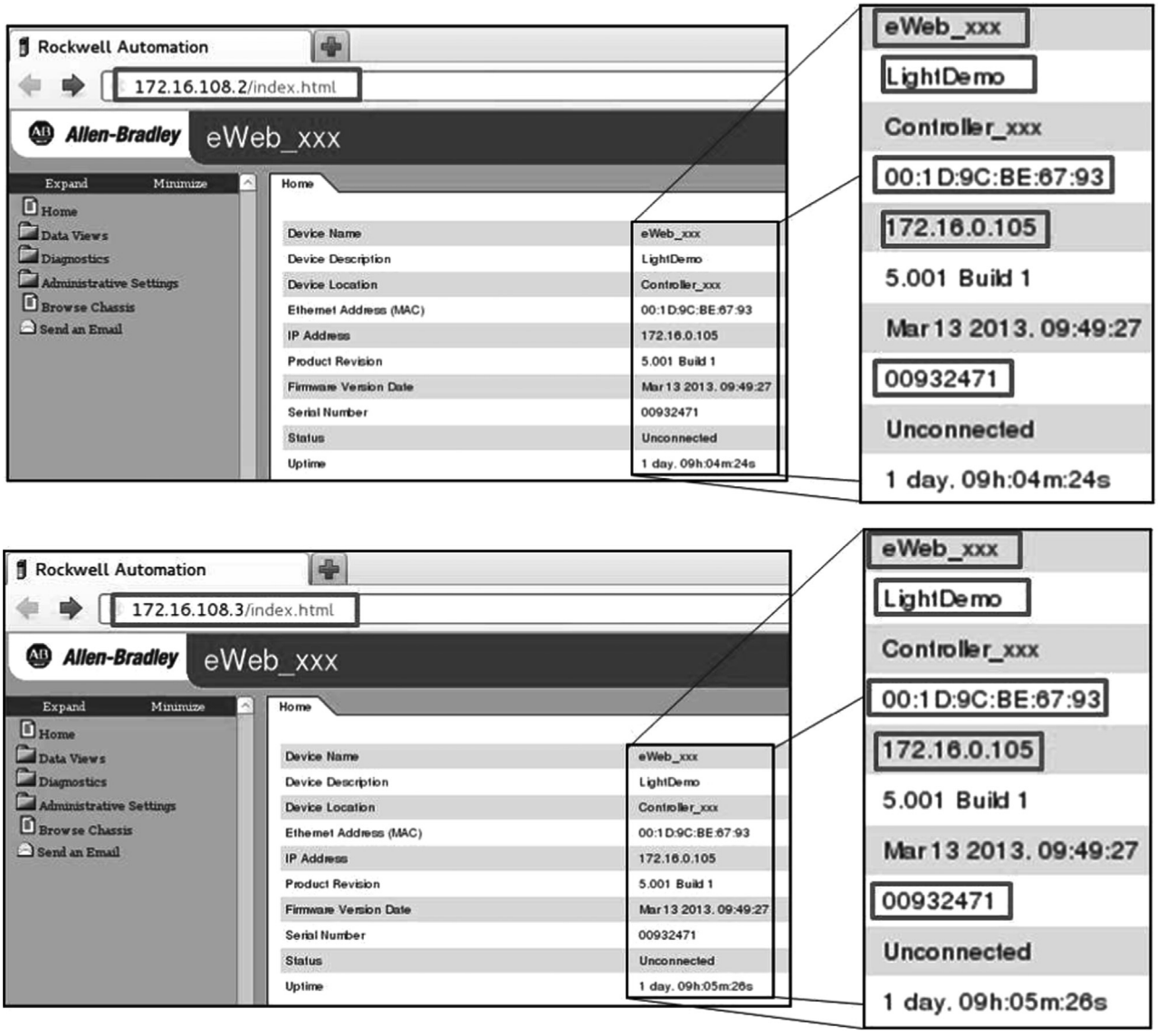


Fig. 3 – Authenticity flaws with Honeyd hosts.

3.3.1. *Adding search terms*

The first modification to the Honeyd source code involved adding a linked list `proxy_srchitem` to the template structure in `template.h`. Each `proxy_srchitem` contains `char` valued variables `find` and `replace` and their corresponding `int` valued variables `length`. Fig. 4 shows a conceptual representation of the modified template data structure. Storing the search terms in the data structure of each template upon initialization ensured the required consistency. To clarify, consider a scenario where an attacker uses a secure shell (i.e., `ssh`) to connect to a target computer with the IP address `10.1.10.12`. If the attacker were to then query the target with a command such as `ifconfig`, the expected output would be the IP address `10.1.10.12`. Repeated queries, or queries using alternate utilities such as `traceroute` would also consistently display `10.1.10.12`.

3.3.2. *Developing search terms*

The file `config.c` contains the code that implements and administers the templates, which are stored in a splay tree object defined in `tree.h`. The second modification adds a new function `proxy_build_srchlist()` that initializes the `proxy_srchitem` list added to the template object.

Honeyd supports plug-in modules for enhanced functionality. The keyword `option` triggers the configuration file parser to store the line of text in the linked list `honeyd_plugin_cfgitem` for use by plug-ins (see Fig. 5). The `option` items are further identified by a `plugin` name immediately following the `option` keyword. The name `icsproxy` is used to identify the search terms used in the modification.

The file `plugins_config.c` contains the new function `plugins_config_find_pkg_items()`, which extracts the options denoted as `icsproxy` from the `honeyd_plugin_cfgitem` linked list into a separate consolidated list, leaving the original `honeyd_plugin_cfgitem` list unmodified.

The last step is to store the search and replace terms in each template in the template tree. The new function `proxy_update_srchlist(tmpl)` in `config.c` iterates through each search term in the `icsproxy` consolidated list and adds it to the template search list. The associated replace terms for

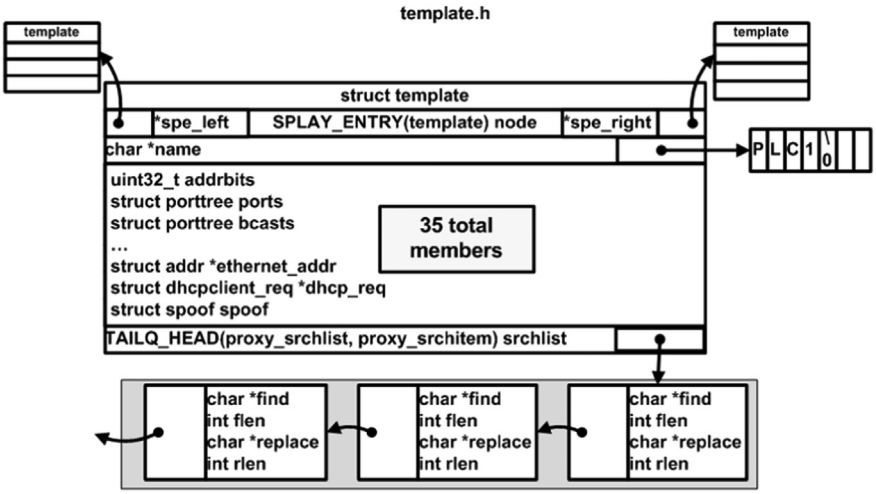


Fig. 4 – Adding the linked list of search terms to a template.

the IP address, MAC address, host name and serial number are calculated by utility functions added to `config.c`. The utility functions take advantage of the data available in the template structure to generate the appropriate replace term. Additional protocol-specific search terms can be added to the `switch()` statement in `proxy_update_srchlist` as required by specific implementations. Examples include data fields such as description, location and product code used by a particular protocol. The search options can also handle static search and replace pairs expressed using ASCII text or raw byte patterns. After all the templates are populated with their search and replace terms, Honeyd completes some remaining initialization tasks and begins to listen for connections to the IP addresses bound to the templates.

3.3.3. *Searching and replacing payloads*

The new file `icsproxy.c`, which contains the search and replace functions, was inspired by code originally developed by Shaw [11] for the C language equivalent of the PHP function `str_replace()`. Shaw's code leverages the standard C `strstr()` function to iteratively find all occurrences of the `find` string and replace them with copies of another string. However, the implementation relies on NULL terminated strings. In order to search and replace the full range of binary values, Shaw's code was adapted to use specified string lengths instead of NULL terminated ASCII strings.

To implement the inline search and replace functionality, a new function `processpayload()` was inserted in the `tcp_send()` and `udp_send()` functions in `honeyd.c`. Function `processpayload()` provides pointers to the appropriate template and data payload, along with the length of the payload being handled. Function `processpayload()` iterates through the template search list, calling the `replace_str()` function for each set of search and replace terms in the list. A pointer to the modified payload is returned and the sending function continues to wrap the TCP/IP packet for transmission to its destination. Fig. 6 presents the Honeyd architecture with the new find/replace list and payload modification features.

4. Evaluation

The overarching goal of the research was to construct a framework that enables a programmable logic controller to serve as the basis for a cost-effective, targetable and authentic honeypot. The evaluation of Honeyd with the `icsproxy` modifications (hereafter referred to as Honeyd+) involved (i) a functional test to evaluate authenticity; and (ii) a performance test to evaluate targetability (i.e., from the size perspective).

The test environment comprised a low-cost programmable logic controller (Omron CP1L with a CP1W-EIP61 EtherNet/IP adapter) and a high-cost programmable logic controller (Allen-Bradley 1756 ControlLogix with a 1756-EWEB module), both of which implement HTTP and the EtherNet/IP industrial protocol. Honeyd+ was hosted by a low-cost computing platform (Raspberry Pi running Raspbian priced at approximately $50) and a high-cost computing platform (HP EliteBook 8570w laptop running Ubuntu Server 14.04 LTS priced at approximately $1800).

Each Honeyd+ platform had two network interface cards, one for the public-facing IP space where the honeypots were deployed and the other configured on the programmable logic controller VLAN for communications between the programmable logic controller and the Honeyd+ platform. Both Honeyd+ platforms had two configuration files, one advertising 75 hosts corresponding to the L61 template and the other advertising 75 hosts corresponding to the CP1L template. The tests were launched from an external "attacker" personal computer running Kali Linux. The components were connected using a Cisco Catalyst 3550 switch configured with inter-VLAN routing (see Fig. 7).

4.1. *Experimental design*

This section describes the experimental design, including the functional and performance testing.

4.1.1. *Functional testing*

The first series of tests evaluated the search and replace functionality for the Omron CP1L and Allen-Bradley L61 programmable logic controllers. Using the data from the pilot

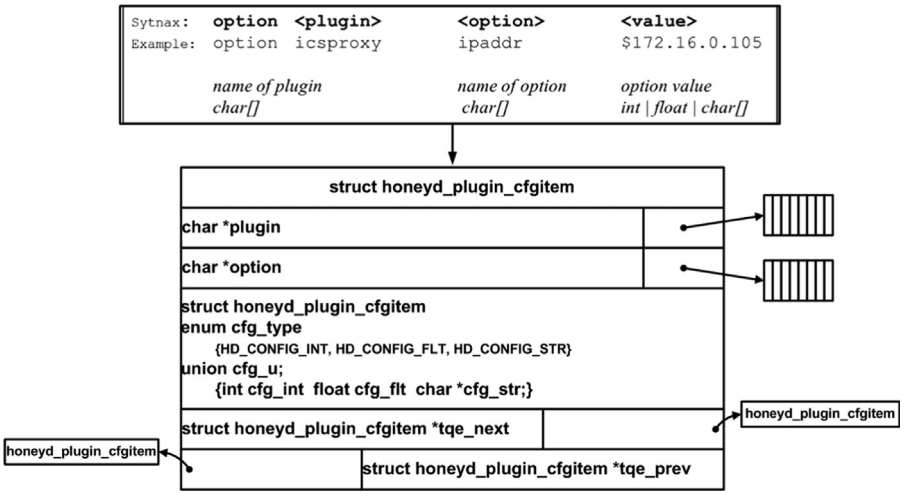


Fig. 5 – `icsproxy` options parsed from the configuration file (top) to a linked list (bottom).

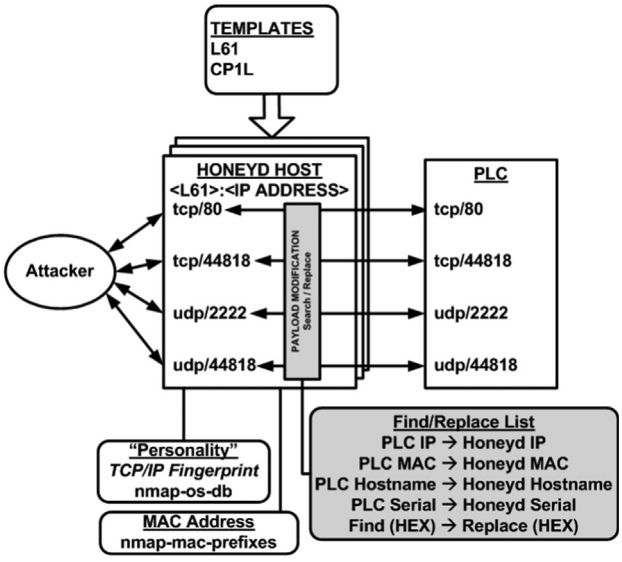


Fig. 6 – Honeyd host with `icsproxy` modifications.

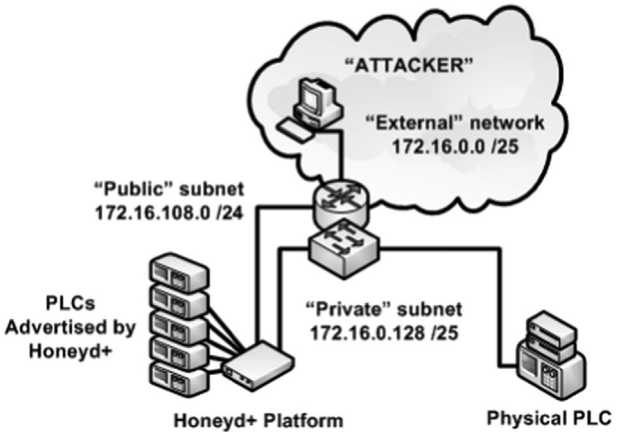


Fig. 7 – Evaluation environment.

tests, templates were created for the two programmable logic controllers that included terms for the IP address, MAC address, hostname, serial number and a static search and replace pair. The `honeydctl()` utility function included with Honeyd enables an operator to monitor the status of Honeyd. A modification to this function enabled it to list the search terms in each template (see Fig. 8).

The functional testing used `nmap` to conduct port scans and generate operating system fingerprints of the programmable logic controller and the Honeyd+ hosts running on the Raspberry Pi and HP laptop. The `wget` utility was used to download web pages via HTTP and the `nmap` scripting engine (NSE) `enip-info` script tested the proxy functionality for the EtherNet/IP protocol by sending the EtherNet/IP "List Identity" `0x0063` command. The tests were repeated three times and the results were compared using the `diff3` utility.

The functional testing was conducted using the command line options shown in Tables 3 and 4.

4.1.2. *Performance testing*

The second series of tests used the Python performance testing script and the parameters from the second pilot study to evaluate the performance of hosting Honeyd+ on the Raspberry Pi and laptop platforms, and the potential impact of multiple simultaneous connections. The performance testing investigated the error rates obtained by varying the platform, protocol and data rates. The data rate was computed as *threads* × *requests per second* (*per thread*).

The experiment used an incomplete factorial design with three replications ($n=3$). The response variable was the *Error Rate* as defined in the pilot study. Table 5 summarizes the experimental design.

Each thread represented a TCP connection from the attacker to the programmable logic controller (either a direct connection to the programmable logic controller or a connection through a Honeyd+ host). The experimental design is considered to be an incomplete block design because five levels were used for the L61 data rate, but only three levels for the CP1L data rate due to the constraints discovered in the pilot study (e.g., maximum number of EtherNet/IP sessions and the limited number of HTTP requests that could be handled).

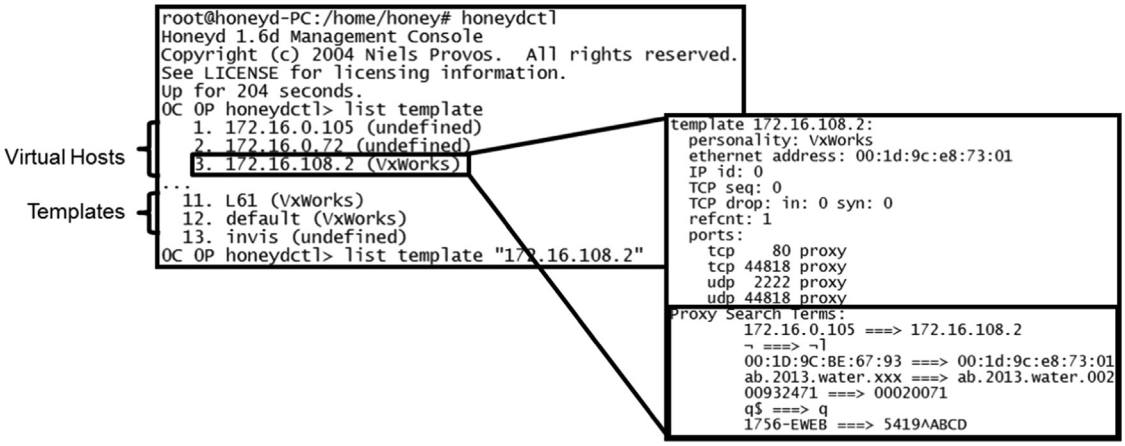


Fig. 8 – Validation of correct search terms to a template.

Table 3 – `nmap` options used in functional testing.

Option	Description
`-sS`	TCP SYN scan
`-sU`	UDP scan
`-pT:1-1024,44818, U:1-1024,2222,44818`	Scan specified TCP/UDP ports
`-PE`	ICMP echo probes
`-PP`	ICMP timestamp probes
`-A`	Enable OS detection, version detection, script scanning and `traceroute`
`-script ''enip-info"`	Use the `nmap` script `enip-info` (sends the List Identity command)
`-T4`	Aggressive timing
`-vv`	Very verbose output

Table 4 – `wget` options used in functional testing.

Option	Description
`-e robots=off`	Ignore the "disallow" directives in `robots.txt`
`-user-agent=''Mozilla/5.0 (Windows NT 6.3; rv:36.0)Gecko/20100101Firefox/36.0"`	The L61 requires iframes, determined by checking the user agent; it redirects the default "wget/1.13.4" to an error page, so the –user-agent switch fools the L61 into serving the web page to a Firefox browser
`-max-redirect=1`	Limits the maximum number of redirections to one to keep the test within scope

Each protocol, platform and data rate combination was measured 50 times for a period of 30 s and the mean *Error Rate* for each factor-level combination was used in the analysis. The programmable logic controller was rebooted prior to starting each platform/rate measurement. The order of the measurements was randomized.

4.2. *Limitations*

Certain limitations had to be considered prior to implementing Honeyd+. First, the search and replace functionality was tested on protocols such as EtherNet/IP that use length-based error checking. Protocols that use encrypted or compressed communications cannot be supported. Protocol-specific algorithms, such as error checking, encryption and compression, could potentially be incorporated in the Honeyd+ source code, but this was outside the scope of the research.

Provos [10] has demonstrated that a traditional implementation of Honeyd can support 2000 TCP requests per second with 65,536 hosts. However, the pilot study conducted in this research showed that programmable logic controllers support significantly fewer connections, depending on the manufacturer, protocol and model. Additionally, since multiple Honeyd+ hosts communicate with a single programmable logic controller, any modification or degradation of the programmable logic controller is reflected in subsequent connections to all the Honeyd+ hosts.

5. Results

The functional and performance testing were performed successfully without complications or discernible errors. The functional testing demonstrated the authenticity of Honeyd+ in all the test cases. The performance testing focused on the targetability characteristic of Honeyd+ and was successful for the lower data rates with the EtherNet/IP protocol; however, error rates at the higher data rates were not optimal.

Table 5 – Experimental factors and levels.

Factor	Experiment 1	
PLC (categorical) Platform (categorical)	CP1L Baseline-CP1L PC-CP1L Pi-CP1L	
Protocol (categorical)	EtherNet/IP	HTTP
Data rate[a] (interval)	1 × 80 2 × 40 4 × 20	1 × 20 2 × 10 4 × 5
Aggregate data rate	80	20
Factor	**Experiment 2**	
PLC (Categorical) Platform (categorical)	L61 Baseline-L61 PC-L61 Pi-L61	
Protocol (categorical)	EtherNet/IP	HTTP
Data rate[a] (interval)	4 × 75 5 × 60 12 × 25 10 × 30 20 × 15	1 × 50 2 × 25 5 × 10 10 × 5 25 × 2
Aggregate data rate	300	50

[a] Fixed data rate in requests per second denoted as *Threads* × *Rate*.

```
## CP1L Search Terms
option icsproxy ipaddr $172.16.0.106

option icsproxy ethaddr $00-1D-4B-F0-22-66

option icsproxy hostname $CP1W-EIP61

option icsproxy serial $4bf02266

option icsproxy $(hex)435031572d4549503631 $(hex)524a3558214e51413932
#                     C P 1 W - E I P 6 1        R J 5 X ! N Q A 9 2

## L61 Search Terms
option icsproxy ipaddr $172.16.0.105

option icsproxy ethaddr $00-1D-9C-BE-67-93

option icsproxy hostname $eWeb_xxx

option icsproxy serial $00932471

option icsproxy $(hex)313735362d45574542 $(hex)353431399324714344
#                     1 7 5 6 - E W E B        5 4 1 9 ^ A B C D
```

Fig. 9 – Search terms used in the authenticity test.

5.1. *Functional testing*

Advertising a subnet of 75 Honeyd+ hosts with the five search terms in Fig. 9 yielded similar results as shown in Fig. 2. Deeper examination of the `nmap` port enumeration and operating system fingerprinting results, and inspection of the web pages and the response to the List Identity command demonstrated the successful application of the search terms in all cases. Figs. 10–12 highlight the changes to one of the Honeyd+ hosts compared with the physical programmable logic controller for each of the tools tested.

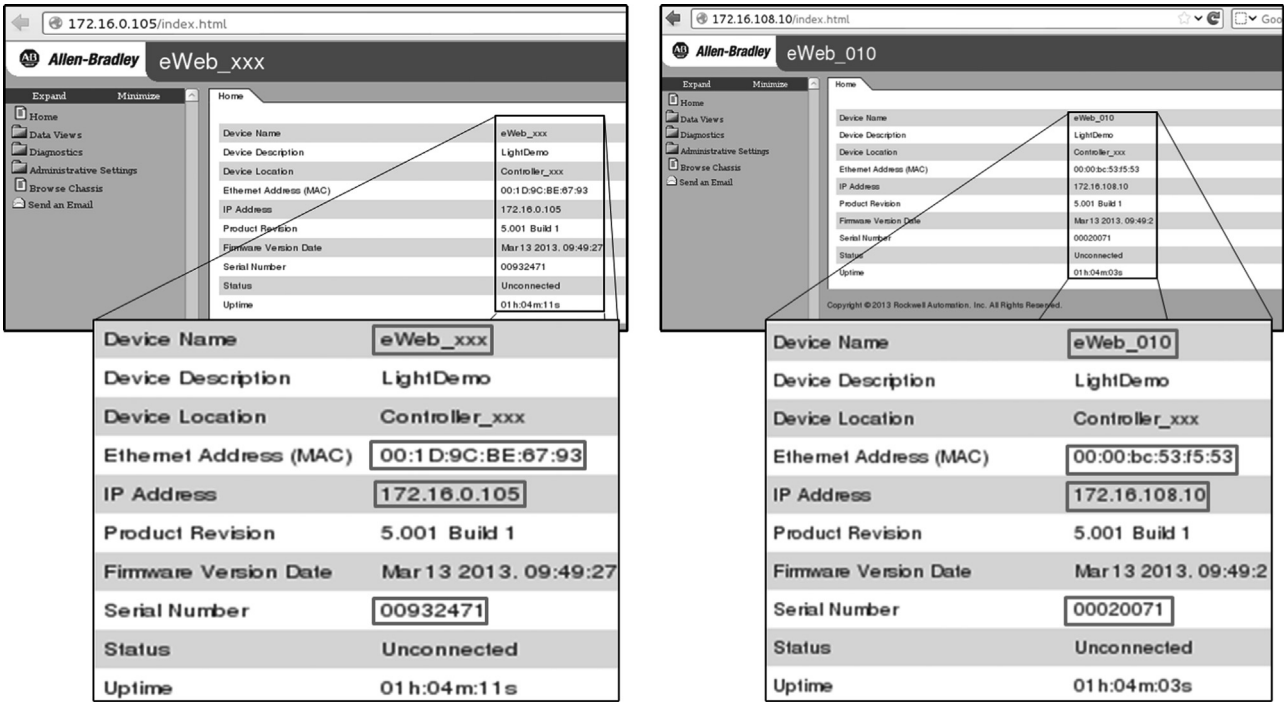


Fig. 10 – Web page response: actual web page (left) and Honeyd+ host web page (right).

5.2. *Performance testing*

Performance testing with the EtherNet/IP protocol revealed an increased mean error rate at each level across all the platforms as the number of connections increased, despite the constant aggregate data rate. In the case of HTTP, the average mean error rate on the L61 was higher, but was more consistent for the Raspberry Pi and laptop computer than the programmable logic controller. The mean error rate for HTTP with the CP1L was approximately 88% across all platforms, which is consistent with the observations in the pilot test (Tables 6 and 7).

The total mean error rates suggest that the Raspberry Pi and laptop computer both increased the error rates. However, for the EtherNet/IP industrial protocol, small differences exist in the mean error rates for the Raspberry Pi, laptop computer and programmable logic controller. In the case of HTTP, the Raspberry Pi and PC Honeyd+ platforms have higher error rates than the programmable logic controller.

6. Conclusions

Honeyd+ is a feasible and reduced-cost technique for deploying multiple physical honeypots of the same size. The experimental results demonstrate that Honeyd+ could represent 75 authentic programmable logic controllers on the Raspberry Pi and laptop platforms, each of them containing five search terms. Functional testing reveals that the programmable logic controller performance falters above five simultaneous connections, depending on the protocol and manufacturer. Furthermore, analysis reveals that a low-cost Raspberry Pi device has about the same performance as an expensive laptop.

Honeyd+ clearly has applications as a research honeypot. Honeyd+ inherits the additional capabilities of the native Honeyd, such as simulating the routing of multiple network segments. Other industrial control system components, such as human–machine interfaces, sensors and actuators, could be added to create a more comprehensive system. However, additional modifications are required to compensate for the limitations imposed by programmable logic controllers. Substituting a script or separate web server is a possible solution; incorporating caching functionality and/or stateful memory are areas for further research.

Honeyd+ can also be used as a decoy in a production network. Honeyd+ templates on Raspberry Pi devices installed at multiple remote sites could cloak real assets while using just one centrally-located programmable logic controller. Honeyd+ incorporates the simple logging capability from native Honeyd, which catalogs attempted connections by source IP, source port and operating system fingerprint. More verbose monitoring of a centrally-located programmable logic controller with open source tools such as Honeywall or Wireshark could enhance data collection. In such an application, attackers would have an increased chance of interacting with a honeypot during the reconnaissance and enumeration stages, giving critical infrastructure asset owners and operators an opportunity to respond to the intrusions.

The views expressed in this paper are those of the authors and do not reflect the official policy or position of the United States Air Force, United States Army, Department of Defense or United States Government.

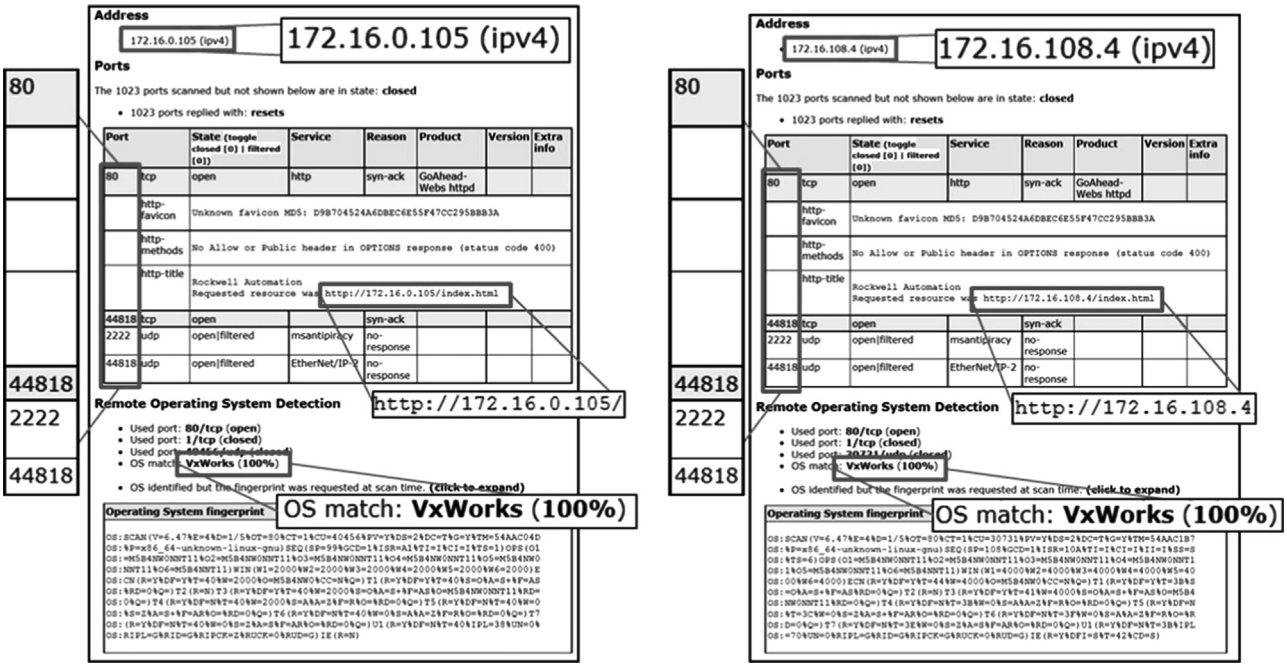


Fig. 11 – nmap port enumeration and fingerprint: actual response (left) and Honeyd+ host response (right).

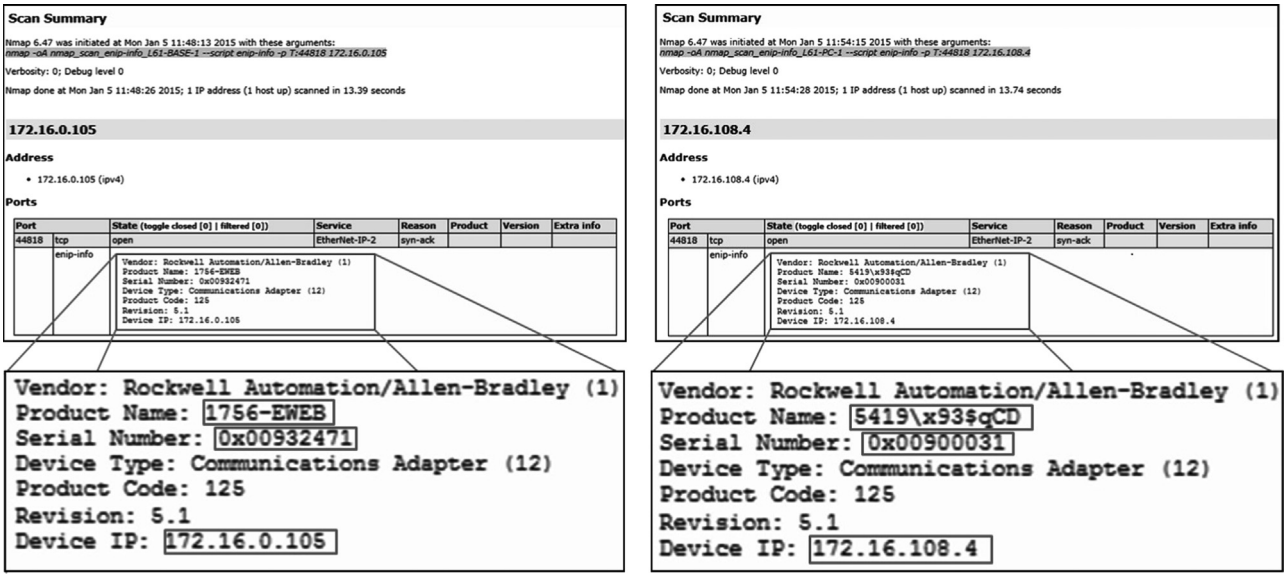


Fig. 12 – List Identity (0x0063) command via nmap script: actual response (left) and Honeyd+ host response (right).

Table 6 – Total mean error rates for the L61 controller.

ENIP (300)	Physical PLC no Honeyd (%)	Honeyd on Laptop (%)	Honeyd on Raspberry Pi (%)	HTTP (50)	Physical PLC no Honeyd (%)	Honeyd on Laptop (%)	Honeyd on Raspberry Pi (%)
4 × 75	9.3	11.4	7.9	1 × 50	0.7	32.4	32.1
5 × 60	14.4	16.4	12.7	2 × 25	0.1	32.5	32.2
12 × 25	29.5	30.1	27.4	5 × 10	14.1	32.3	31.9
10 × 30	35.1	36.2	33.4	10 × 5	18.0	32.3	31.9
20 × 15	58.7	63.9	60.1	25 × 2	54.1	58.5	57.4
Total	**29.4**	**31.6**	**28.3**	Total	**17.4**	**37.6**	**37.1**

Table 7 – Total mean error rates for the CP1L controller.

ENIP (80)	Physical PLC no Honeyd (%)	Honeyd on Laptop (%)	Honeyd on Raspberry Pi (%)	HTPP (10)	Physical PLC no Honeyd (%)	Honeyd on Laptop (%)	Honeyd on Raspberry Pi (%)
1 × 80	0.00	0.00	0.00	1 × 10	87.66	88.66	88.73
2 × 40	3.45	0.18	2.40	2 × 5	85.77	88.87	89.14
4 × 20	11.04	28.36	29.51	5 × 2	86.77	88.68	88.93
Total	**4.83**	**9.51**	**10.64**	**Total**	**86.74**	**88.74**	**88.93**

REFERENCES

[1] Allen-Bradley, 1756 ControlLogix Controllers, Technical Data, Publication 1756-TD001H-EN-P, Rockwell Automation, Milwaukee, Wisconsin (literature.rockwellautomation.com/idc/groups/literature/documents/td/1756-td001_-en-p.pdf), 2013.

[2] D. Berman, Emulating Industrial Control System Devices Using Gumstix Technology, M.S. Thesis, AFIT/GCO/ENG/12-13, Department of Electrical and Computer Engineering, Air Force Institute of Technology, Wright-Patterson Air Force Base, Ohio (dtic.mil/cgi-bin/GetTRDoc?AD=ADA563104), 2012.

[3] R. Bodenheim, J. Butts, S. Dunlap and B. Mullins, Evaluation of the ability of the Shodan search engine to identify Internet-facing industrial control devices, *International Journal of Critical Infrastructure Protection*, vol. 7 (2), pp. 114–123, 2014.

[4] C-SPAN, Cybersecurity threats (www.c-span.org/video/?322853-1/hearing-cybersecurity-threats), November 20, 2014.

[5] Idaho National Laboratory, INL's SCADA Test Bed, Idaho Falls, Idaho (www.4vip.inl.gov/research/national-supervisory-control-and-data-acquisition-test-bed), 2014.

[6] R. Jaromin, Emulation of Industrial Control Field Device Protocols, M.S. Thesis, AFIT-ENG-13-M-27, Department of Electrical and Computer Engineering, Wright-Patterson Air Force Base, Ohio (www.dtic.mil/cgi-bin/GetTRDoc?Location=U2&doc=GetTRDoc.pdf&AD=ADA582482), 2013.

[7] G. Lyon, Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning, Insecure.Com, Sunnyvale, California, 2008.

[8] V. Pothamsetty and M. Franz, SCADA HoneyNet Project: Building Honeypots for Industrial Networks (scadahoneynet.sourceforge.net), 2005.

[9] N. Provos, HONEYD(8), *OpenBSD System Manager's Manual* (www.citi.umich.edu/u/provos/honeyd/honeyd-man.pdf), 2002.

[10] N. Provos, A virtual honeypot framework, *Proceedings of the Thirteenth USENIX Security Symposium*, 2004.

[11] L. Shaw, A portable C string replacement function: `repl_str()`, creativeandcritical.net (creativeandcritical.net/str-replace-c), 2009.

[12] L. Spitzner, The Honeynet Project: Trapping the hackers, *IEEE Security and Privacy*, vol. 1 (2), pp. 15–23, 2003.

[13] S. Wade, SCADA Honeynets: The Attractiveness of Honeypots as Critical Infrastructure Security Tools for the Detection and Analysis of Advanced Threats, M.S. Thesis, Department of Electrical and Computer Engineering, Iowa State University, Ames, Iowa (lib.dr.iastate.edu/cgi/viewcontent.cgi?article=3130&context=etd), 2011.