Cyber-Physical System Security with Deceptive Virtual Hosts for Industrial Control Networks

Todd Vollmer, Milos Manic, *Senior Members, IEEE*

***Abstract*—A challenge facing industrial control network administrators is protecting the typically large number of connected assets for which they are responsible. These cyber devices may be tightly coupled with the physical processes they control and human induced failures risk dire real world consequences. Dynamic virtual honeypots are effective tools for observing and attracting network intruder activity. This paper presents a design and implementation for self-configuring honeypots that passively examines control system network traffic and actively adapts to the observed environment. In contrast to prior work in the field, six tools were analyzed for suitability of network entity information gathering. Ettercap, an established network security tool not commonly used in this capacity, outperformed the other tools and was chosen for implementation. Utilizing Ettercap XML output, a novel four-step algorithm was developed for autonomous creation and update of a Honeyd configuration. This algorithm was tested on an existing small campus grid and sensor network by execution of a collaborative usage scenario. Automatically created virtual hosts were deployed in concert with an Anomaly Behavior (AB) system in an attack scenario. Virtual hosts were automatically configured with unique emulated network stack behaviors for 92% of the targeted devices. The AB system alerted on 100% of the monitored emulated devices.**

***Index Terms*—Industrial Control, Intrusion Detection, Network Security.**

I. Introduction

MANY modern complex control systems are interconnected via Ethernet networks. These networks, found deployed in areas such as chemical facilities or energy production, are utilized to deliver status and control information vital to the operation of physical systems. A compromised control system could have security, public safety, industrial or economical consequences [1],[2]. The need for resilient adaptive security systems, specifically developed for critical cyber-physical systems, is increasing with the elevated levels of cyber security threats in the modern world [3],[4].

Todd Vollmer is with the Idaho National Laboratory, Idaho Falls, ID 83415 USA (e-mail: denis.vollmer@inl.gov).

Milos Manic is with the University of Idaho, Idaho Falls, ID 83402 USA (e-mail: misko@uidaho.edu).

Furthermore, with the advent of the smart grid, the number of configurable devices to be deployed is relatively high. For example, in a typical Advanced Metering Infrastructure (AMI) system 1,500 wireless sensors report to one or multiple Wireless Access Points (WAP) nodes [5]. As of April 2010, almost 69 million of these meters were planned for deployment in the United States [6]. Assuming a uniform deployment of sensors, this plan calls for 46,000 WAP's. So, in addition to protecting existing networks, a large-scale deployment of new devices will soon be prevalent.

Network security monitoring systems are a significant part of a solution to protecting control systems. In most contexts, they are rarely capable of providing perfect intrusion detection [7],[8]. Deceptive systems, called honeypots, that emulate critical network entities have been deployed in tandem with monitoring solutions to improve detection accuracy and precision rates [9],[10].

It is difficult to list the definitive attributes of a network host necessary to attract an attackers attention. This requires analysis of attackers motivations, which may vary in depth and details depending on the situation. However, a reasonable assumption can be made that if any of the real devices on the network are a desirable target, than emulation of those systems would be a productive exercise. Given this premise and the issue of a large device deployment, a relevant concern is reducing the human effort involved while providing an improved security posture.

In addition to a honeypots faithful reconstruction of a host's network presence, automation is a key capability. According to John Ousterhout, there are four common steps for turning deployments from an enemy into a friend [11]. First, and most important, is automation. This is essentially a question of economy. It is usually cheaper to build better tools than manually manage the configurations of individual devices in a large system.

In this paper, the collaborative use of dynamic virtual honeypots in a control system network is introduced. Aspects of effective tools for identifying network host characteristics are examined. The presented algorithm focuses on automatically managing the complexity of self-configurable dynamic virtual hosts by adapting to an operational network environment. A self-updating model, based on passive monitoring of the network devices, is created and maintained. This model is used to configure deceptive network entities designed to draw the focus of malicious intent. Finally, a usage

scenario is examined to show how imitating a real network is useful when combined with an anomaly detection routine.

The remainder of this paper is organized as follows: Section II provides background information on honeypots and network scanning. Related prior work is found in Section III. Section IV defines the algorithm design and implementation details. A description of the test control system network and usage case is in Section V. Section VI provides an analysis of results and the presented algorithm. Section VII concludes the paper.

II. BACKGROUND

Lance Spitzner introduced the concept of a dynamic honeypot (DHP) in 2004. The idea was based on automatically configuring a honeypot by gathering information gleaned from network traffic. This type of system has the benefit of requiring little human input and can readily adapt to changes in network behavior. A DHP requires implementation in two key areas: 1. Network information gathering, and 2. Generating honeypot configuration for deployment.

Honeypots have uses other than presenting an emulated host. For instance, gathering malware by presenting a vulnerable service, acting as a mail host to collect SPAM email and acting as a 'tarpit' that consumes all attempts to break into a system. These uses are not explored in this paper but are provided for completeness.

Implementations of honeypots fall into two categories: high or low interaction. Low interaction virtual honeypots are used to gather information. They are simpler to deploy, less likely to be compromised and can work collaboratively with other agents. Additionally, they might distract an attacker from hitting real systems.

Honeyd, created by Niels Provos, is an open source project implemented as a small software daemon that creates virtual hosts on a network [12]. It is a low interaction honeypot that emulates an OS network stack and provides basic service functionality. Another advantage is the ability to deploy thousands of virtual hosts on a single host thereby reducing hardware costs.

Honeyd simulates the network stack and generally provides only superficial services. Because of this, an attacker is never able to gain access to the host by compromising a service but would quickly realize that something is amiss. The primary goal is not to entrap the attacker into spending all his effort on the deceptive system. It is to attract his attention, for at least a short time, and gather information that helps identify the attacker and a possibly compromised internal attack platform.

In this paper, Honeyd was evaluated and logic created to automatically configure it. The resulting configuration is designed to emulate, as close as possible, any user identified host on the network. This is in contrast to previous work that focused primarily on dynamically creating several honeypots, called a honeynet, that in aggregate are statistically similar to a network of hosts [13].

High interaction honeypot systems are typically hardware replicas of existing operational components that include the appropriate software. For purposes of this discussion, virtual machines are included in the High category. These systems do not mimic services but are deployed with working instances. This type of system provides a high fidelity solution that is less prone to discovery of its deceptive purpose by network intruders. However, they are at a higher risk for compromise by an attacker and require a more complicated deployment investment. Deploying a virtual machine is simpler than a hardware base system but still requires complex management scenarios for deploying a wide array of service software. This includes having copies of multiple OS distributions and server software.

Finally honeypots, high or low interaction, can only detect attacks directed at them. A competent attacker who discovers that a system is a honeypot will avoid any further contact with that system. The fidelity of the deception is in the presentation of the honeypot to the network. How the data is gathered to create this deception is important.

A. Passive vs. Active Scanning

The two primary means for gathering the necessary network host information to create a honeypot includes passive and active network scanning. Unfortunately, most research to this point provides minimal analysis on suitable tools for passive information gathering. This is a key enabling capability if the intent is to deceive an attacker into believing an emulated system is real. This paper corrects this deficiency by examining characteristics of six existing tools and consequently recommends a tool, previously not used in this context, called Ettercap [14].

In most of the literature reviewed, passive scanning has been implemented with P0f and occasionally Snort [13],[15]. P0f is a command line tool that utilizes an array of mechanisms to identify hosts in a network stream. It is a passive OS fingerprinting tool frequently cited in creation of dynamic virtual honeypots. Snort is inherently a rule based intrusion detection system.

The amount of information that may be gleaned from passive scanning is a limited subset of possible information [16]. A passive scanning based tool is restricted to only gathering data that is offered in the captured stream. If a service on a host is available, but not utilized, this data point will be missed. Active scanning may prove more successful at extracting this type of information.

Nmap is an active scanning tool that has proven useful for interrogating hosts on a network [17]. However, a downside to active scanning is the possible interruption of services on hosts. This problem is especially acute in control systems. For instance, ping sweeps on older systems have been known to disrupt normal operation and cause physical damage [18]. Active scanning also provides a beacon of network activity outside the norm and could be revealing for intruders listening in on the traffic. In either case of active or passive scanning, the resulting information may be used to configure a honeypot.

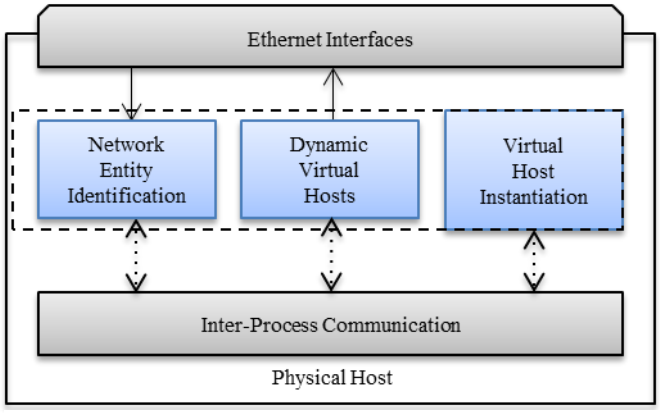

Fig. 1. Conceptual Design Diagram

III. Related Work

Dynamic honeypot solutions that gather network information, process that information into a configuration and deploy appropriately have been created as in [13],[15],[19],[20]. These papers propose monitoring methods that are active [19], passive [13], combined [15] or are ambiguous [20]. When passive monitoring is implemented, the chosen tool is typically P0f with no analysis of competing tools provided. Finally, the test implementations are all on non-control system networks.

There are two notable projects related to control system honeypots. The SCADA Honeynet project by Matthew Franz and Venkat Pothamsetty of the Cisco Critical Infrastructure Assurance Group (CIAG) was initially released in March of 2004 [21]. The project is not actively maintained, with a last release date of July 15, 2005, however it is still available from Sourceforge. The design utilizes Honeyd for simulating a set of services for a PLC. The major contributions of this project are service scripts, which include functionality for FTP, MODBUS, Telnet and a web server. However, the SCADA Honeynet does not consider automatic provisioning of the virtual hosts and is a manually configured project.

Digital Bond, Inc. is a control system security consulting and research group founded by Dale Peterson. Their SCADA Honeynet implementation is an evolution of the original project just described [22]. It utilizes two virtual machines instead of Honeyd. One virtual machine includes network monitoring tools such as Snort with Digital Bond's Quickdraw IDS signatures to detect activity. The other virtual machine simulates a PLC with several exposed services. There is no dynamic provisioning of hosts or services, although it is possible to replace the virtual machine PLC with an actual hardware component. This assures complete deception if the PLC is configured correctly with the added expense of an actual hardware device.

```
Create and update virtual hosts with following:
   Network Entity Identification.
   Write entities to XML.

   Read_data; from input files and Ettercap
   For each IP create a Dynamic Virtual Host
      Find_closest representative OS.
      Map_OS values to Honeyd names
      Create_MAC address for new hosts
      Create_Features for device specific behaviors
      Create_Config for virtual hosts
   End
```

Fig. 2. Pseudo code

TABLE I
Tool Capability Matrix

Tool	OS Identify	Port identify	MAC Vendor
ettercap	yes	yes	yes
ntop	yes	yes	yes
p0f	yes	no	no
snort	no	yes	no
tcpdump	no	yes	no
tshark	no	yes	yes

IV. Solution Design

This section describes the software tool evaluation and implementation logic of the solution. Fig. 1. shows the relationship of three key functional areas: Network Entity Identification (NEI), Dynamic Virtual Host configuration (DVH) and Virtual Host Instantiation (VHI). These act in a continuous cycle of processing and updating information represented by the dotted line box.

A pseudo code of the algorithm is shown in Fig. 2 with implementation details for procedures in italics found afterward in each section.

A. Network Entity Identification

The Network Entity Identification (NEI) component monitors network traffic from which it extracts the source, destination, and port activity. Information from the NEI is delivered to an implementation of the logic tasked with creating a dynamic honeypot configuration.

An evaluation was conducted on six passive network information gathering open source tools to determine their strengths and weaknesses relevant to support of automated configuration. The tools evaluated for providing network host identification are: p0F [23], Tshark [24], Ettercap, Snort [25], Tcpdump [26] and Ntop [27]. Of the six tools, Ettercap and Ntop provide well-formatted structured output as an option. Another tool, called SinFP [28], was removed from consideration because it did not execute correctly on the test sensor system.

In addition to identifying network entities, NEI needs to provide the information necessary to create a representative virtual network presence. The critically required capabilities examined were operating system identification, port or service identification per host and the capture of MAC addresses with a resolution to the appropriate vendor [15]. Considering the results in the critical capabilities matrix shown in Table I, it is apparent that Ntop and Ettercap fulfill all three criteria. Of the

TABLE II
TOOL PERFORMANCE RESULTS

Tool	# OS ID	# of Hosts	# of MACs	# of IPs
ettercap	16	45	35	44
ntop	0[a]	202	43	39
p0f	13	NA	NA	10
tshark	NA	NA	69	44
tcpdump	Not Tested			
snort	Not Tested			

[a]Ntop displays OS information only in the web output.

two candidates, Ettercap was chosen for its support of XML output, completeness of information provided from this output and available functionality for support of future work.

Table II presents the results when running the tools against the test network described in section V. The system, as configured during the test, had 46 physical connections to the network. The second column contains the number of operating systems identified by each tool. Ntop's identification of 202 hosts in column 3 contains duplicate entries for entities that have both IPv4 and IPv6 addresses. Additionally, records created for broadcast addresses inflate the host number. Ettercap outperformed or equaled the other tools in three of the four categories.

Ettercap is an extensible network manipulation and reconnaissance tool [14]. It is an established and popular tool in the hacking community. However, this paper is the first to establish its use as a source of information for dynamic honeypot creation. It was run as a daemon process with unified sniffing. In this mode it maintains internal network host records and updates them as new information is found. A binary log file is continuously updated as well. An Ettercap companion executable Etterlog is then run on the log file with a -x option to produce an XML file. This data file is the source for communication of the network entity information to the dynamic virtual host configuration process.

In conclusion of this section, the Ettercap tool was selected for identifying network entities. It provides information on host IP addresses, MAC values and port usage. When compared with the five tools listed in Table I, it performed as well or better than all of them. An additional key driving capability is Ettercap's formatted XML output that can easily be integrated into other systems. Communication within an automated system requires a defined consistent messaging system. Lastly, Ettercap is capable of performing more advanced operations that could be useful for future functional enhancements.

B. Dynamic Virtual Hosts

This section discusses the configuration creation of the Dynamic Virtual Hosts (DVH). These hosts emulate the network signature of actual systems on a physical network. Honeyd is a popular open source solution for virtual honeypots that provides a flexible and feature rich configuration capability. As autonomous configuration is a desired aspect for minimization of expensive manual configuration, Honeyd's configuration flexibility is an advantage. The overall goal is the automatic configuration and dynamic update of a variable length list of virtual hosts based on information gathered from actual hosts using Ettercap.

The following sections describe four functional areas in DVH: OS selection, OS name mapping, MAC creation and Service (port) emulation.

1) Operating System Selection

For any given host on a network, Ettercap may not be able to identify the operating system. If this occurs, for an emulation target, then an OS must be chosen. It is desirable to provide an exact match in network behavior. This does not necessarily require an exact match with the OS name in the database.

Read_data consists of extracting *n* host records *h* from the Ettercap entries and forming a record set *O* such that $O = \{h_1, h_2, \ldots, h_n\}$. *O* then becomes a source of information for creation of virtual hosts. The intention is to examine these records for similarities to an IP address *i* provided in a list of j target IP addresses where $IL = \{i_1, i_2, \ldots, i_j\}$. An assumption is being made that the hosts *h* on the network have an OS similar to a candidate *i* even if an exact match is not found.

Given that P_h is a set of port values for a host *h* and a network port set S_i for target *i*, *Find_closest* examines the intersections of $S_i \cap P_h$ for all *h* in *O*. The integer count of matching ports is stored for each intersection. In addition, the number of ports for the target is calculated. Given these values, a match percentage is calculated, e.g. two candidate ports and an intersection count of two constitute a 100% match. Candidates with a higher percentage were considered to be more similar. Some OS's utilize ports specific to services offered by that OS and they could be used in identification [16].

If a candidate OS is not identified by examining ports, then the MAC address is examined. *Find_closest* compares the vendor identification section of the candidate MAC address of *i* to the MAC addresses for each host *h* in *O*. If a match is found that has an identified operating system, this value is placed on a candidate list. After exhaustively examining *O*, the largest matching value, if one exists, from the candidate list is chosen as the OS. The assumption is that any hosts on the network that have the same NIC vendor may be performing similar functions and thereby have a similar operating system. As is described later, several control system vendors have an organizationally unique identifier for their network devices.

If no prior step has identified an OS, a random number *r* is generated in the range 0 to N where N is the cardinality (*O*). If the host record h_N OS field exists, this value is utilized. If not, a random value supported by Honeyd is chosen. In other words, a field is possibly selected for inclusion proportional to the relative frequency of its presence in *O*. Given that not all host records contain an OS, and possibly none of them; the completely random OS value is required.

Once an OS is identified by the selection algorithm or trivially identified by Ettercap further action, as described in the next section, is still required.

2) Operating System Name Mapping

The Honeyd configuration value for an operating system makes use of the Nmap version 1 database defined named values. Similarly, Ettercap utilizes its own defined name values that do not directly match Nmap. To make a functional configuration, a simple algorithm implemented in *Map_OS* was developed to associate Ettercap names with Nmap names. The algorithm's initial pass compares the word tokens of the OS names looking for case insensitive string matches. The number of word matches were summed and stored. After iterating through each possible OS combination, the one with the largest count total is presented as a candidate. Finally, each OS name combination is written to a file for reference during creation of the configuration.

3) MAC Creation

Honeyd provides two options for specifying the MAC address, either by vendor name or the six-octet string. Because Honeyd has hard coded vendor strings, the six-octet representation was chosen for use in the algorithm. Ettercap captures this MAC octet address for all hosts in *O*. The MAC protocol specifies that the first three octets are organizationally unique and should not overlap with any other vendor. Thus, in order to create a new MAC address that appears to come from a specific vendor, these first three octets were used. The vendor typically assigns the remaining three octets. In this algorithm these last three octets are created as described next.

In the *Create_MAC* function, the last three octets are randomly generated and appended to the end of the captured candidate vendor portion. This new MAC is then compared with all other MAC's noted in the Ettercap host list *O*. Any collision of addresses instigates a recreation of another random set of values. Given the 2^{24} possible values, the probability of a collision is low. Depending upon the security configuration of a deployed switch, these generated MAC values may require more refinement. For instance, if port security is enabled on the network switch the possible MAC's would have to be predefined.

4) Network Service Emulation

The host entries in *O* contain network ports, previously defined as P_h, that were active during the capture session. Along with the port number, a port service name is available. This service name is a human readable text value that is defined in an Ettercap configuration file called etter.services. Utilizing the service names contained in this file, a new configuration file called serv.conf was created. This file maps the service name to a service emulation script path.

The *Device_Features* function examines any service ports found in the Ettercap output and loads the serv.conf file. Any service name match to entries in the file results in the appropriate service script value placement in the Honeyd configuration. This enables the creation of service specific behaviors that furthers the goal of deception. Currently, the manual creation of scripts is necessary although some service scripts are already available from other projects. Automatic creation of these behavior scripts is another future area of exploration.

In addition to services found during passive scanning, a variable number of ports associated with the common services are randomly activated A common service mapping file for control system devices is utilized by the *Device_Features* function. It consists of a hierarchical MAC mapping structure. Generally, in the case of a control system device, the vendor portion of the MAC is directly tied to the device manufacturer enabling usage of the mapping file to find relevant services

Constructed utilizing XML, the file maps the vendor MAC to a list of common services that are possible to find activated on a device of this type. Each service in the file is described by the following attributes: port number, protocol, service description and action script. The action script specifies which script Honeyd should utilize, if any, when it sees traffic to this port. A value in this field will overwrite any previously defined default script found in serv.conf. This provides the capability to customize a response to this specific device type while still retaining generic service emulation functionality.

Each service description has an 'include' value. This is a floating-point value between 0.0 and 1.0. This value is compared to a randomly generated value in the appropriate range. If the random value is less than the include value, the port is added to the honeypot configuration. The intention is to vary port inclusion to represent the variability in device configurations.

An analysis of available vendor product specifications was used to create this file. For example, the test system contains a Rockwell Micrologix 1100 Programmable Logic Controller (PLC) and the possible services listed for this consist of Ethernet/IP, web services, SMTP email (outbound) and SNMP [29].

C. Virtual Host Instantiation and Update

The candidate emulation hosts are provided at startup as a list of IP addresses. It is assumed that if a host in the list disappears from passive sensing, the user still desires to have an emulated version of it. The overhead to maintain the missing hosts records is minimal. Of course, the actual system has to have appeared in the passive analysis during the monitoring period to create an initial virtual host configuration.

An initial configuration file is created by *Create_Host_Conf*. Changes to the configuration of the virtual hosts running under Honeyd are performed while the system is running. After a configurable time period, currently an arbitrarily chosen 60 seconds, etterlog is called on the ettercap daemon log file. The resulting XML output is saved and compared to an existing output file. Differences in network host activity are noted and stored on a list for possible action. Actions include adding network services, updating OS configuration and changing MAC addresses. A companion Honeyd executable file, called Honeydctl, provides this functionality.

A simple example Honeyd configuration file containing one virtual host configuration is shown in Fig. 3.

```
create vh1
set vh1 personality "Linux 2.4.xx"
set vh1 default tcp action reset
set vh1 default udp action reset
set vh1 default icmp action reset
add vh1 tcp port 23 "/script/router-telnet.pl"
set vh1 ethernet “00:00:BC:A1:00:23”
bind 192.168.1.125 vh1
```

Fig. 3. Honeyd Host Configuration

V. Usage Scenario and Results

In the following test scenario, scans and probes are directed at all devices on the network representing the reconnaissance phase of an intrusion. This assumes the attacker is an outsider and does not have a network map. The goal of the security system is to generate informational alerts about the anomalous presence. A secondary effect is the diversion of attention and effort of the attacker to a virtual honeypot system. Keys for success include: a faithful imitation of real devices on the network, a mechanism for monitoring activity directed at the honeypots, and appropriate communication of emulated IPs and alerts.

To improve the cyber security of network systems various approaches can be applied [30]-[32]. One of the most common approaches is anomaly detection. An anomaly detection system is trained on a set of normal network behaviors. The extracted behavior model is then used to detect anomalous behavior in any subsequently observed traffic.

One of the difficulties of this approach is building a comprehensive normal behavior model for a specific network communication system. Typically a user-defined period of activity is designated as ‘normal’. However, by definition, any network activity directed at a honeypot can be considered abnormal. This provides a definitive source of information for classifying traffic that does not require direct user interaction.

Anomaly Behavior (AB) implementation details are not covered in this paper but may be found in previous work of the authors [32],[33]. For this test scenario, an AB system was configured to monitor the virtual honeypot IP addresses and send alerts on any activity. The role of the automatically created honeypots is to attract and possibly delay an intruder on the network. This usage is similar to that proposed in [7] and [34].

The intended deployment is an operational control system network with a heterogeneous mix of hosts. There are two possibilities for timing when the honeypots are instantiated. The first approach, used in this test scenario, is to create the virtual hosts in advance of any anomalous situations. This would increase the probability of a network scan identifying the hosts. It removes the race condition between recognizing an anomaly and getting the hosts instantiated in time to get noticed.

The second approach, with the race condition, would be to instantiate the hosts after some indication of intrusion has occurred. This indication could come from a traditional intrusion detection system or some other security mechanism. Given the DVH use of virtual hosts with its reduced hardware requirements, a dedicated integrated host and low network impact; there is little benefit to delaying instantiation until after detection.

At the beginning of the scenario, all hosts are running and a sensor host with the virtual host logic is connected to the control network. As the NEI component becomes aware of changes in the host characteristics, the honeypots are automatically reconfigured to include the new behavior. The emulated hosts become more authentic appearing, in the service ports offered, over time. As already mentioned, this early instantiation reduces the risk of a stealthy intruder bypassing the honeypots, as they will most likely be present prior to the malicious activity.

A. Test Network

An existing small campus grid (SCG) and sensor network that physically exists in the Center for Advance Energy Studies in Idaho Falls, Idaho was used to test the algorithm. The network includes a suite of wireless sensors targeted at environmental conditions in the building, wind and solar renewable resources, and a variety of control system devices. The SCG is connected to a small wind turbine, a solar power station, and a wireless advanced metering infrastructure. Additionally, the network has several Windows based computers, web camera’s, a Rockwell Automation PLC and a National Instruments PLC.

The SCG network contains wireless systems from Emerson, Honeywell and Arch Rock. Each system connects wirelessly to the sensors via a wireless access point. These WAP gateways have a wired connection on one side of the network and wireless interfaces to remote environmental sensors. The network sensor device has visibility on the wired side of the connection. Each wired WAP connection has a variation in the method of Ethernet network protocols utilized that makes each one a unique challenge to emulate. For instance, the Emerson device transports data at the raw Ethernet level using a custom protocol.

The software for the implemented algorithm was deployed on a test host platform. This platform runs a 32 bit Ubuntu 12.04 operating system on a dual core Intel Atom 330 processor with 2 GB of DDR2 RAM, a 250 GB hard drive and three GigE network ports. One of the Ethernet ports was dedicated for use by the honeypot. Honeyd is capable of running multiple virtual hosts on one physical network interface. The second port was used to perform passive monitoring by NEI. The final port was connected to a second separate network used for management of the devices.

B. Test Steps and Results

A PERL implementation of the algorithm was run on the test sensor platform attached to the operational test network. In addition to OS emulation performance, seven network test probes were completed. Thirteen systems shown in the first

TABLE III
HOST IDENTIFICATION RESULTS

ID	Device	Mapped OS
1	Rockwell HMI	MS Windows ME, 2000 Pro or Advanced Server or Windows XP
2	Micrologix 1100 PLC	Novell NetWare 3.12 - 5.00
3	Arch Rock Server	Random
5	Honeywell HMI	MS Windows ME, 2000 Pro or Advanced Server or Windows XP
10	Arch Rock WAP	Random
25	D-Link WAP	Apple Airport Extreme Base Station (WAP)
99	D-Link Wireless camera	Apple Airport Extreme Base Station (WAP)
130	Arch Rock HMI	MS Windows ME, 2000 Pro or Advanced Server or Windows XP
150	Nat. Inst. PLC	MS Windows ME, 2000 Pro or Advanced Server or Windows XP
200	Emerson WiHart AP	Linux 2.4.16 - 2.4.18
215	HMI(Windows PC)	Windows for Workgroups 3.11 / TCP/IP-32 3.11b stack or Windows 98
253	Moxa 505A Switch	FreeBSD 4.4 for i386 (IA-32)
	Honeywell WAP	None[a]

[a]No information available for initial Ettercap OS determination.

column of Table III were evaluated; six control devices and seven more typical information technology devices. The ID column is used as a reference identifier and corresponds to the last octet used in the emulated IP address. For completeness, the Honeywell wireless access point is included. Because it does not utilize an IP address for communication, Honeyd cannot emulate this device.

1) Initiate Honeypots

An input text file for the DVH component contained two sets of space delimited IP addresses labeled R and E. List R contains the unordered IP addresses of real hosts. List E contains the list of IP addresses to be assigned to the emulated hosts. The lists represent a bijective function in that f: R->E is a one-to-one and onto mapping of set R to set E. A sample message with three hosts is shown in Fig. 4.

The same message was initially sent to DVH. Three virtual honeypots were created and verified by sending ICMP echo messages. After 60 seconds, a newly updated input text message was sent containing twelve test IP addresses for the hosts in Table III. The software automatically created configurations for all of the devices. Each emulated host was assigned its own unique IP and MAC address and was instantiated on the test sensor hardware.

These actions verified that the integrated communication mechanism works and virtual hosts are instantiated. Specifically, the NEI component created a network model stored as an XML file. A sample host entry is shown in Fig. 5.

```
R = 192.168.1.1 192.168.1.3 192.168.1.25
E = 192.168.2.1 192.168.2.3 192.168.2.25
```

Fig. 4. IP Emulation Message

The information from this file was retrieved by the DVH component to create configuration entries similar to those shown in Fig. 3.

```
<host ip="192.168.1.25">
  <mac>00:0C:29:77:61:78</mac>
  <os>D-Link DWL-900AP</os>
  <port proto="udp" addr="68" service="dhcpclient"/>
  <port proto="udp" addr="123" service="ntp"/>
  <port proto="udp" addr="137" service="netbios-ns"/>
</host>
```

Fig. 5. XML Host Record

A message file similar to Fig. 4., containing the emulated IP addresses, was sent to the Anomaly Behavior (AB) detection software. The message passing mechanism is a simple text file dropped into a specific directory. The application continuously monitors the appropriate directory for a new file. After receipt of the message the AB commenced passive monitoring of the twelve virtual hosts.

Of the thirteen devices initially chosen for emulation, ten specific operating systems were configured autonomously, two were 'random' and the Honeywell device was undetermined. The third column in Table III shows the Ettercap to Nmap mapped OS names selected by the algorithm. For table space purposes the 'MS Windows ME, 2000 Pro or Advanced Server or Windows XP' value has had its text reduced.

2) Network Scan Tests and Results

Nine tests, described next, were executed on the virtual hosts using Nmap, the Open Vulnerability System (OpenVAS) and the ping command line tool. Nmap version 5.21 was chosen to test the network presence of the emulated devices. This version utilizes the second generation Nmap OS database that is actively maintained. It uses a more robust guessing implementation for uncertain signatures. OpenVAS is a flexible comprehensive security scanning tool. It is capable of over 30,000 network vulnerability tests. A laptop, with Nmap, OpenVAS and ping installed, was assigned the IP address 192.168.1.15 and attached to the SCG network. The laptop filled the role of network intruder.

TEST 1: nmap -n -sP 192.168.2.0/24

This simple test performs a 'ping sweep' on all 256 addresses in the range that contains the 12 emulated devices. A combination of an ICMP echo request, TCP SYN to port 443, TCP ACK to port 80 and ICMP timestamp request are sent. Any system that responds to one of these requests is considered available on the network. All twelve of the emulated addresses were found in 2.2 seconds.

TEST 2: nmap -n -v -A -T4 -iL nmap.testhosts

This command line is the first example provided in the Nmap man page documentation. The –A option enables aggressive scan options including OS detection, version scanning, script scanning and traceroute. The –T4 option is a timing template that improves scan time on reasonably stable networks. Note, that by default, Nmap only scans 1000 of the most commonly used ports. It completed in 234 seconds.

OS detection in Nmap is based on a database of signatures.

Each fingerprint record in the database contains four fields: vendor, OS family, OS generation and device type. Output from detection includes lists of possible operating systems and device classes with an accuracy score. The score falls in a range of 0.0 to 1.0 with the later indicating a perfect match.

The OS detection produces large amounts information. For the 12 emulated devices, 223 device types and 40 OS matches were returned. In both cases, accuracy ranged from .85 to .97. As there were multiple results for most of the emulated devices, any of the entries that matched either the original device or its mapped OS were considered a success. Of the ten non-random devices, Nmap identified seven for a 70% success rate. Of the three that failed, no information was produced for device 2. Twenty-one incorrect entries were created for device 215. Device 5 was identified by one incorrect entry.

TEST 3: nmap -sU -sS -O --osscan-guess -n -p1-65535

The –sU option executes a UDP scan to each port specified. For some common ports a protocol specific payload is included but for most of them the packet is empty. The –sS option tells Nmap to send only a single SYN packet to each port. This is the initial packet sent in a TCP connect sequence. The –O option enables standard OS guessing while –osscan-guess makes Nmap guess more aggressively. Finally, the –p argument specifies to scan all possible ports instead of the default top 1000. The tool took 258 seconds to complete the configured actions.

The results from this Nmap execution were similar to those in TEST 2 with some exceptions. First, the correct OS guess for device 253 increased in accuracy by 2 points. Second, device 215 was correctly identified with an accuracy score of .86 were previously it had failed. This increased the overall identification rate to 80%. It should be noted that, because of the broad port scan range, port 44818 for device 2 was found. The port was missed in TEST 2. This is a common port used by the Rockwell Ethernet/IP protocol that is specific to that control system implementation.

TEST 4: nmap –sO –n

This scan sends IP packets and iterates through the eight-bit IP protocol field. The emulated hosts responded to only three of the 256 protocols: ICMP, TCP and UDP.

TESTS 5-7: ping -c2 –R, ping -c2 -T tsonly, ping -c2 -T tsandaddr

Utilizing the ping command line tool, ICMP echo requests were sent to the 12 emulated and 46 actual devices on the test network. ICMP packets are wrapped in an IP datagram and can contain IP option fields. Three rounds of requests were sent, one with the Record Route (-R) option, one with timestamp only (tsonly) and finally the option with both IP and timestamp (tsandaddr). All but one of the physical devices responded with varying levels of correctness to the pings. None of the 12 emulated devices responded correctly. The results for tests 4-7 are discussed later in section VI.

TEST 8 - OPENVAS:

The OpenVAS framework was leveraged to perform more intensive network probes than Nmap on the virtual hosts. A single large-scale discovery and vulnerability scan was executed against the 12 virtual hosts. Of the available 32,418 plugins 3,778 were enabled for the scan. Plugins are attributed to a wide variety of functional categories and enable specific scanning behaviors. Many of the plugins execute on the target host with the appropriate credentials. Host plugin types were disabled.

All twelve devices and their open ports were discovered during the scan. The activity took 21 minutes and 44 seconds to complete and a scan report was produced. At the initial level of detail, the finished scan report looked similar to those reports from scans against the actual hardware. However, several differences were found when looking at the details. All of the devices had a common warning about a multicast address response flaw that could lead to a denial of service attack. This kind of similarity could possibly be leveraged to facilitate identification of virtual hosts. In this particular instance, a configuration change to the virtual hosts would remove the commonality.

Another similarity arose from what was missing in the details. Service port information, on actual hosts, usually provides a variety of information. For instance, a service note on an open SSH port might detail what SSH protocol versions are supported and any vulnerabilities discovered. For each virtual host few of the services had additional information, which is unusual. As was noted in the Network Services Emulation portion of the Design section, it is possible to add service emulation scripts to the virtual host configuration. Only the default scripts that came with Honeyd were included in the algorithm implementation. Another possible solution is to configure the service ports to interact with a *subsystem*. A subsystem is a complete application that is capable of executing on its own. They run as a separate process. Honeyd acts as a proxy and passes the connection to the subsystem. This provides a greater depth of deception but could also introduce more configuration complexity and security issues.

TEST 9 - ANOMALY TEST:

As was mentioned earlier in this section, a message with the 12 emulated IP addresses was sent to the AB component. The function of the AB component is to passively monitor host traffic and send alert messages. If the AB component receives an input IP, for which it has not been trained, then it will consider all traffic to it as abnormal. This is a convenient feature for the intended use of honeypots in this system. The AB posted abnormal behavior messages for %100 of the monitored emulated hosts during tests 1-7. A sample message is shown in Fig. 6. It contains the source IP address, destination IP address and the IP layer protocol number involved. In the example protocol 1 indicates ICMP.

```
<meta:event>
  <dtime>2012-09-14T01:00:00</dtime>
  <confidence>.9</confidence>
  <type>behavior</type>
  <window-size>20</window-size>
  <ip_list>
    <ip>192.168.2.25</ip>
    <ip>192.168.1.15</ip>
  </ip_list>
  <proto_list>
    <proto>1</proto>
  </proto_list>
</meta:event>
```

Fig. 6. Anomaly Behavior Message

VI. Analysis

Although honeypots, physical or virtual, emulate real operations systems at some level, there is no guarantee that attackers would perform a scan of a network. However, if one is conducted having emulated devices similar to actual devices can provide a benefit to the security of the system. Minimally it makes the attackers analysis of the devices difficult by increasing the amount of data to analyze. Additionally, the attacker will waste time and effort if an emulated device is chosen for further probing. This provides defenders with extended opportunities to identify intruders on the network.

Based on information from the tests, Industrial control network protocols are a viable candidate for emulation by the presented algorithm. Application ports are fixed, unusual ports that readily identify the use of a particular protocol. Given the passive nature of information capture, active network sessions are needed to discover the ports and nature of the service. For instance, the test system contains a Rockwell Micrologix 1100 processor that uses EtherNet/IP for communication. The network traffic from the operator HMI to this device occurs on port 44818 using TCP. The TCP connection is maintained for the duration of the session. The traffic between the HMI and control device is regular in size and timing. The packet lengths were as follows: 19.15% between 40-79 bytes, 80.82% between 80-159 bytes and 0.04% between 160-319 bytes. The average packet size is 95.861 bytes. This regularity benefits the anomaly detection algorithm as well.

In the background section is a discussion on the choice of passive scanning for host information. One side effect of passive scanning is the inability to directly identify network ports not in use. While the control system network traffic is typically regular and a constant connection, it or other services may not be enabled. However having these inactive services as part of the virtual hosts is beneficial to the presented deception. The mechanism to support this capability is found in the Device_Features function described in the solution design section. Originally created to add optional services for control devices, it can also be used to ensure service ports for hosts are added to the virtual host configurations. For example, host ID 130 in Table III runs a Microsoft Windows 2000 Server OS. An Nmap scan of the device revealed 6 open TCP ports. Passive scanning identified 4 of the ports. One of the missing ports was for a terminal service that was not accessed during the test time frame. This terminal service port was subsequently added to a service file with a probability of addition set to 1.0. All subsequent reruns of the test scenario then included this port in the configuration for that virtual host. Prior to this configuration change the passive discovery tool discovered 30 of the 33 of the ports found in an active Nmap scan on the twelve test devices.

Tests one through three in the previous section were designed to evaluate the network presence of the virtual hosts. Test one verified that as a base case 100% of the virtual hosts were discoverable on the network. At a superficial level they appeared to be legitimate devices. Test two provided a more in depth network probe designed to verify the OS representations. The scan correctly identified 70% of the devices. A more intensive OS scan in test three correctly identified 80% of the emulated OS's. So given both a superficial and more intense scan the virtual hosts appear to resemble actual hosts, at a 70% or 80% accuracy rate. This shows the end result of an effective integration of the information gathering, communication and host creation logic.

A. Scalability and Security Issues

Scalability of the presented solution relies primarily on the capability of the hardware host. Honeyd is technically capable of emulating 65,535 hosts. Testing by the Honeyd authors shows that on a modest system thousands of different honeypots are possible [12]. To validate this claim, a test with 986 virtual hosts was run on the test platform. The Honeyd OS signature database contains 986 entries. Each host configuration was created similar to Fig. 3. with a unique OS entry from the database and an IP address.

The Nmap command in TEST 1 was then executed targeting the 986 IP's. The top command was run on a 1 second interval to capture CPU and memory usage of the Honeyd daemon. At rest, prior to the Nmap tests, 8,748 KB's of memory was consumed. 8,860 KB's were used at the conclusion of the test. The average CPU utilization was 0.3% with a standard deviation of 1.23% and a maximum of 14.9%. This testing is not comprehensive but does validate that, at a superficial level, a large number of virtual hosts can be created. Honeyd is single threaded and with more intensive probing it is possible to maximize utilization of a single CPU. The test system has two CPU's and can continue to function even if this occurs.

The tested hardware host uses a Long Term Support (LTS) version of Ubuntu 12.04. This OS has a five-year support cycle that includes security upgrades. As part of the hardware design, three physical Ethernet ports were specified. The ports are all assigned to a specific communication task to avoid a complete denial of service situation. For instance, if a large number of honeypots are active and consuming the entire bandwidth of a single port the system can still communicate on another port assigned to the management network.

Updates to the host OS, communication of alerts and IP

monitoring/emulation lists are delivered on a separate management network. The second interface is configured as a passive read only interface on the operational network. This means it is not directly addressable from another host on the network. One security concern is a possible flaw in the monitoring software attached to the interface. The third interface is for use by the honeypot software to present its emulated hosts on the operational network. The most likely threat to the host is from this interface. This is a logical outcome considering the honeypots are designed to attract the attention of those with nefarious intent.

This threat is partially mitigated by the design of Honeyd. The software runs as a restricted user and, by default, does not provide any real services to compromise. For instance, on a high-interaction honeypot there are real shell services that might be compromised. Note that this does not rule out a denial of service or exploitation of a possible flaw in Honeyd itself. In addition to the Honeyd features, a host monitoring system such as OSSEC [35] can be utilized to provide self-monitoring.

Finally, it is not required that Honeyd and the anomaly behavior routines reside on the same machine. However, by condensing the software installs to one platform, it simplifies configuration. It creates a more secure mechanism for passing messages, as the information never leaves the machine. This also provides an opportunity to explore the recently expanding computational capabilities of low power multi-CPU devices.

B. CPU and Memory Performance Measurements

The DVH configuration logic, when implemented in Perl and run on the test machine previously described, took .7s clock time to run and utilized 21 MB's RAM. The input Ettercap XML file contained 46 host entries and the resulting Honeyd configuration file included 12 devices. When running this configuration file, Honeyd consumed 5.7 MB's of RAM. During active scanning with Nmap, this would increase to 7.2 MB's. Ettercap was run continuously in daemon and logging mode on the test machine. It utilized 6 MB's of RAM and would utilize up to 60% CPU time when the Ethernet port it was monitoring was utilized to transfer data files.

C. Tool Limitations Discussion

This section describes issues found with the tools used to implement the automatic configuration algorithm. They are provided as findings relevant to the specific tools and are not detractors directly related to performance of the algorithm presented in Section IV. The deficiencies found are special cases of characteristics that are not commonly examined. A review of literature has found similar types of weaknesses in other honeypot implementations so this is not necessarily unique to Honeyd [36].

Examination of the emulated test systems, using the Nmap protocol scan, revealed a Honeyd limitation. As was noted in TEST 4, the emulated hosts only responded to three protocols. When run against real devices in the test network, a variety of responses are noted. This includes a varying number of protocols acknowledged. A review of the Honeyd source code reveals that basic support for other protocols could be added.

An issue with the handling of the IP options field was discovered with Honeyd in tests 5-7. The IP datagram format consists of a header, option and data sections [37]. The option section is a variable length list of options, up to 40 bytes, that is not typically used. Of the five currently defined options, two are relevant to this project: Record Route and Timestamp. Record Route requests that the target, and each hop on the path to it, add their IP to a list in the option field. Timestamp has three request variations: timestamp only, timestamp with IP and a preloaded IP list. Honeyd does not support any of the options.

As this field in the IP header is optional, support by vendors vary from none to dropping packets that contain options. A study done in 2005 found a 45% success rate for Record Route and a 36% success rate for the Timestamp option when implemented in a SYN packet sent to web servers located on the Internet [38]. Another study, performed in 2010, on 267,736 Internet addresses found a 47.7 % response rate to Timestamp requests delivered in an ICMP Echo Request [39]. These studies show, that despite being optional, a significant number of devices provide some level of support and therefore makes it a concern for emulation. Honeyd is built with a library named libdnet. This library has the requisite functionality to correct the issues noted.

Honeyd uses an older version of the Nmap database scheme. There are two primary drawbacks to this situation. First, the fingerprints are not updated and are missing more modern signatures. Second, there is very little control of the emulation behavior outside of the signature definitions. Effectively this means emulation is dependent almost entirely on the definitions. A better solution might be a melding of a historical signature with observed characteristics found in the live network traffic and upgrading to the latest Nmap version 2 formatted database.

Finally, as was noted earlier, a host on the network used a custom protocol that did not utilize IP addresses and consequently was not recognized by Ettercap. The host communicates using raw Ethernet frames and is characteristic of a Honeywell proprietary protocol. However Ntop did notice the host communication and tracked the host in its node list.

The choices are to choose Ettercap, Ntop or merge data from both sources of information. A weakness with Ntop is in the data export routine. The interface does not contain all the information needed to create a configuration. Ettercap was chosen based on its XML output functionality and general recognition performance. It is possible to correct the Ntop programmatic interface to provide all necessary information as it is an open source project and it internally tracks the data. Likewise, Ettercap could be modified to track host information on IP's or MAC addresses.

VII. Conclusion

An algorithm was proposed and demonstrated to automatically deploy deceptive virtual network entities in a control system network. Six open source passive network-monitoring tools were evaluated and Ettercap was chosen for host identification. This differs from prior work in the field in which p0f is typically used. The algorithm created unique network stack signatures for twelve of the thirteen test devices. Eight of the twelve emulated devices were correctly identified by an aggressive Nmap scan. OpenVAS found all twelve devices but some abnormal details were found. Several deficiencies with both the monitoring tools and virtual honeypot implementation Honeyd were discovered and discussed. These problems are: non-IP based traffic, OS identification database support, missing information and well formatted program output.

In order to show the necessary depth of the proposed automatic deployment and configuration, a usage scenario was executed. In this scenario an anomaly detection system monitored the network activity of the honeypots. The role of the automatically deployed honeypots was to attract and possibly delay an intruder on the network. The primary enabling technologies included continual host monitoring, reconfigurable deceptive virtual hosts and a network anomaly behavior monitor. The benefits of the presented system include: 1) reduced operator interaction, 2) low operational network impact, 3) increased awareness of the security situation, and 4) an independent view of hosts and services that are active on the network. The behavior system alerted on %100 of the packets targeted at the virtual hosts.

This work has identified several areas of possible future research. The use of virtualized networks and devices derived from the automated system presented could subsequently be used as a standard test bed for a variety of IDS systems. An obvious category of work would be correcting the deficiencies found in the support software. Finally, service emulation scripts are manually created. Autonomously developing service behaviors that emulate observed network communications would further the goals of deception and IDS testing.

References

[1] D. A. Shea, "Critical infrastructure: control systems and the terrorist threat," Library of Congress, Report for Congress RL31534, Jan. 2004.

[2] Y. Huang, et al., "Understanding the physical and economic consequences of attacks on control systems," *Int. J. Critical Infrastructure Protection*, pp. 73-83, 2009.

[3] C. Rieger, D. Gertman, and M. McQueen, "Resilient control systems: next generation design research," in *Proc. 2nd IEEE Conf. Human System Interactions*, May 2009, pp. 632-636.

[4] G. Rueff, B. Wheeler, T. Vollmer, T. McJunkin, "INL Control System Situational Awareness Technology Final Report", INL, Idaho Falls, ID, Rep. EXT-11-23408, Jan. 2013.

[5] T. Iwao, K. Yamada, M. Yura, Y. Nakaya, A. Cardenas, S. Lee, and R. Masuoka, "Dynamic Data Forwarding in Wireless Mesh Networks," in *Proc. IEEE SmartGridComm*, 2010.

[6] The Edison Foundation (2010, Apr.). Utility-Scale Smart Meter Deployments, Plans & Proposals. [Online]. Available: http://www.edisonfoundation.net.

[7] A. Carcano, et al., "A multidimenisonal critical state analysis for detecting intrusions in SCADA systems," *IEEE Trans. Ind. Informat.*, vol. 7, no. 2, May 2011.

[8] R. Sommer and V. Paxson, "Outside the Closed World: On Using Machine Learning for Network Intrusion Detection," in *Proc IEEE Symp. on Security and Privacy*, May 2010.

[9] L. Chao, M. Sumiko, K. Hirotsugu, "Dynamic Hybrid System of Honeypot and IDS for Network Security Analysis", IPSJ SIG Notes, vol. 2013, no. 26, pp. 1-5. December 2013.

[10] M. A. McQueen and W. F. Boyer, "Deception used for Cyber Defense of Control Systems," in *Proc. 2nd IEEE Conf. on Human System Interaction*, May 2009.

[11] J. Ousterhout," Is Scale Your Enemy, Or is Scale Your Friend?," in *Communications of the ACM*, vol. 54 No.7, July 2011, pp .110-111.

[12] N. Provos and T. Holz, *Virtual Honeypots*, Boston, MA: Addison-Wesley, 2007.

[13] J. Hieb and J. H. Graham, "Anomaly-Based Intrusion Detection for Network Monitoring Using a Dynamic Honeypot," Intelligent Systems Res. Lab., Univ. of Louisville, TR-ISRL-04-03, Dec 2004.

[14] Ettercap network sniffer. [Online]. Available: http://ettercap.sourforge.net/.

[15] C. Hecker and B. Hay, "Securing E-Government Assets Through Automating Deployment of Honeynets for IDS Support," in *Proc. 43rd Hawaii International Conf. on System Sciences*, HI, 2010, pp. 1-10.

[16] F. Gagnon and B. Esfandiari, "A Hybrid Approach to Operating System Discovery Based on Diagnosis," *Int. J. Network Mgmt*, vol. 21, pp. 106-119, Mar 2011.

[17] G. Lyon, *Nmap Network Scanning*, Sunnyvale, CA: Insecure.Com, 2008.

[18] D. P. Duggan, "Penetration Testing of Industrial Control Systems," Sandia National Laboratories, Tech. Rep. SAND2005-2846P, Mar. 2005.

[19] C. Hecker, K. L. Nance, and B. Hay, "Dynamic Honeypot Construction," in *Proc. 10th Coll. For Information Systems Security Education*, Adelphi, MD., 2006.

[20] X. Jiang and D. Xu. (2004). BAIT-TRAP: A Catering Honeypot Framework. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.84.3.

[21] V. Pothamsetty and M. Franz. SCADA Honeynet Project. [Online]. Available: http://scadahoneynet.sourceforge.net/.

[22] Digital Bond Incorporated. SCADA Honeynet. [Online]. Available: http://www.digitalbond.com/tools/scada-honeynet/.

[23] P0f, available: http://lcamtuf.coredump.cx/p0f.shtml.

[24] Tshark network analyzer. [Online]. available: http://www.wireshark.org/.

[25] M. Roesch, "Snort: Lightweight intrusion detection for networks, " In *Proc. of the 13th Conf. on Systems Administration*, Berkeley, CA, Nov. 7–12 1999, pp. 229–238.

[26] Tcpdump packet analyzer. [Online]. Available: http://www.tcpdump.org/.

[27] Ntop network traffic probe. [Online]. Available: http://www.ntop.org/.

[28] P. Auffret, "SinFP, Unification of Active and Passive Operating System Fingerprinting", *Jour. in Comp. Virology*, Vol. 6, No. 3., pp. 197-205, Aug. 2010.

[29] *Micrologix Ethernet Interface User Manual*, Rockwell Automation, Publication 1761-UM006E-EN-P, Aug. 2005.

[30] O. Linda, T. Vollmer, and M. Manic, "Neural Network Based Intrusion Detection System for Critical Infrastructures*,"* in *Proc. Int. Joint Conf. on Neural Networks*, June 14-19, 2009, pp. 1827-1834.

[31] S. Zhong, T. Khoshgoftaar, and N. Seliya, "Clustering-based network intrusion detection," in *Int. J. Reliability, Quality and Safety*, Vol. 14, No. 2, pp. 169-187, 2007.

[32] O. Linda, T. Vollmer, and M. Manic, "Improving cyber-security of smart grid systems via anomaly detection and linguistic domain knowledge," in *Proc. IEEE Symp. on Resilience Control Systems*, Salt Lake City, UT., Aug. 2012.

[33] T. Vollmer, M. Manic, O. Linda, "Autonomic intelligent cyber sensor to support industrial control network awareness", *IEEE Trans. Ind. Informat.*, DOI: 10.1109/TII.2013.22703, to be published.

[34] Y. Jain, S. Singh, "Honeypot based secure network system," *Int. Jour. On Comp. Sci. and Eng*., Vol. 3, No. 2, Feb 2011.

[35] A. Hay, D. Cid, and R. Bray, *OSSEC HIDS Host-based intrusion detection guide*. Burlington, MA: Elsevier, 2008.
[36] S. Mukkamal, K. Yendrapalli, R. Basnet, M. K. Shankarapani, and A. H. Sung, "Detection of Virtual Environments and Low Interaction Honeypots," In *Proc. Information Assurance and Security Workshop*, June 2007, pp. 92-98.
[37] *Internet Protocol*, DARPA RFC 791, Sept. 1981, [Online]. Available: http://www.ietf.org/rfc/rfc791.txt.
[38] R. Fonseca, G. Porter, R. H. Katz, S. Shenker, and I. Stoica, "IP Options are not an option," UC Berkeley, EECS-2005-24, Dec. 9, 2005.
[39] J. Sherry, "Applications of the IP Timestamp Option to Internet Measurement," B.S. thesis, Dept. Comp. Sci. and Eng., Univ. of Washington, 2010.

Todd Vollmer (M'06–SM'13) Todd Vollmer, IEEE Senior Member, received his B.S. and M.S. degrees in computer science from South Dakota School of Mines and Technology in 1996 and 1998 respectively. He is currently pursuing a Ph.D. in computer science from the University of Idaho. Todd has worked as a Research Scientist at the Idaho National Laboratory since 2006. Prior work involved satellite system software development for Lockheed-Martin and DigitalGlobe, Inc. He has several recent published papers at IEEE conferences on computational intelligence and computer security.

Milos Manic (S'95-M'05-SM'06) received the Dipl. Ing. and M.S. degrees in electrical engineering and computer science from the University of Nis, Serbia in 1991 and 1997 respectively, and a Ph.D. degree in computer science from the University of Idaho in 2003. Dr. Manic is an Associate Professor at the University of Idaho. He has over 20 years of academic and industrial experience, including an appointment at the ECE Dept. and Neuroscience program at University of Idaho. As university collaborator or principal investigator he lead number of research grants with the National Science Foundation, Idaho National Laboratory, EPSCoR, Dept. of Air Force, and Hewlett-Packard. Dr. Manic is a Secretary of IEEE Industrial Electronics Society and is a member of several technical committees and boards of this Society such as IES Committees (previously the Boards) for Conferences and for Publications. Dr. Manic has published over hundred refereed articles in international journals, books, and conferences.