SEVENTH FRAMEWORK PROGRAMME
Theme SEC-2011.2.5-1 (Cyber attacks against critical infrastructures)

D4.4 Device fingerprinting

Contract No. FP7-SEC-285477-CRISALIS

Workpackage	WP 4 - System Discovery
Author	Marco Caselli, Frank Kargl, Valentin Tudor
Version	0.2
Date of delivery	M24
Actual Date of Delivery	M24
Dissemination level	Public
Responsible	UT

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n°285477.

SEVENTH FRAMEWORK PROGRAMME
Theme SEC-2011.2.5-1 (Cyber attacks against critical infrastructures)

The CRISALIS Consortium consists of:

Symantec Ltd.	Project coordinator	Ireland
Alliander		Netherlands
Chalmers University		Sweden
ENEL Ingegneria e Innovazione		Italy
EURECOM		France
Security Matters BV		Netherlands
Siemens AG		Germany
Universiteit Twente		Netherlands

Contact information:
Dr. Matthew Elder
Symantec Limited
Ballycoolin Business Park (GA11-35)
Blanchardstown
Dublin 15
Ireland

e-mail: matthew_elder@symantec.com

Contents

Abstract

Device fingerprinting forms an important building block of automatic system discovery. The ability to learn about the types of devices in a critical infrastructure network, their operating system, and possibly also the software and services they are running provides valuable information, e.g., for efficient automated vulnerability discovery, and network-driven detection as analyzed in WP6. We will look at both active and passive fingerprinting approaches and put a special focus on automation.

This document broadens and deepens the work described in D4.1 "Device fingerprinting preliminary results". We provide a solid state of the art mainly focusing on device fingerprinting. Also, we outline a reference architecture for a fingerprinting tool and describe its modules. We define important requirements and challenges when trying to transfer or extend fingerprinting in the industrial control systems (ICS) domain. With respect to D4.1, we added some tests on ICS components using both state-of-the-art fingerprinting tools and a new tool developed within the CRISALIS project.

1 Introduction

In the ICT field, fingerprinting is a set of activities that exploit different kinds of information in order to outline or describe devices and software running a computer network. Fingerprinting is an important step in many activities related to information security and is often used as a basic approach for working in unknown ICT environments.

Device fingerprinting is the most known and well-studied kind of fingerprinting. It is used to remotely recognize devices' hardware, operating systems, application software (and their respective versions or configurations) running a network. The aim of this activity is to better understand potential vulnerabilities in the network, to analyze attacks, to help assessing risks of chain attacks and finally to derive proper countermeasures.

Both malicious and legitimate users can take advantage from fingerprinting. The former might fingerprint a network or a host to choose exploits needed to penetrate the systems. The latter can, instead, use fingerprinting as the first step towards a vulnerability assessment. Moreover, fingerprinting can be a useful instrument to detect network anomalies. A system that changes over time, due to an update or for the effect of a cyber-attack, might modify its fingerprint too. For this reason, monitoring variations of the fingerprints can be a useful activity also for intrusion detection.

Fingerprinting has an important advantage with respect to manually collect information on a network. Manually collected information is often outdated or otherwise incomplete. On the other hand, fingerprinting approaches can exhibit mis-categorizations. For this reason, the two approaches should always be considered complementary and security mechanisms should rely on both information sources.

Fingerprinting can be divided into two main categories: active and passive. Active fingerprinting requires tools that actively query the system to obtain information needed to outline fingerprints. Passive fingerprinting tries to acquire such information in a less intrusive way by listening to network communication. Usually, active fingerprinting has more chances to succeed in recognizing the system. This is because active fingerprinting implies collecting all the information needed to form a fingerprint while passive fingerprinting collects only the information that is available on the communication channel at that moment. However, performing active scans is not always possible. Active operation are noisier and can be detected. As an example, in SCADA systems (Supervisor Control And Data Acquisition), active fingerprinting is not desirable due to potential computational overload of system's components [1]. Active probing a device on the network

increases the number of frame a device has to process. Programmable Logic Controllers (PLCs) and Remote Terminal Units (RTUs) are usually tuned on the processes they are controlling and cannot support such increase in the amount of traffic. In this situation, any further communication flow caused by the fingerprinting activity, could lead to an undesirable state where a system component cannot satisfy expected requests [2].

Any kind of fingerprinting implies three basic activities:

- Gathering
- Storing
- Dataset updating

The Gathering phase refers to the type of collected data. The most common way to fingerprint ICT systems implies the inspection of specific values in the TCP and IP headers. When aggregated, these elements form a 67-bits signature for each device working in a standard network. Monitoring fields and values of different network protocols is also possible. In fact, it is possible to use all the layers of the ISO/OSI stack [3], from the Application layer (Protocol-based fingerprinting) to the Physical layer (Signal-based fingerprinting). Besides the ISO/OSI stack it is also possible to record completely different information. For example specific situations allows to use data related to timing and communications patterns [4], [5].

Storing procedures allows to use such information in different ways and with numerous software of analysis. Beyond this, specific data structures can be very efficient but fit just few tools. This property can be exploited in situations in which the speed of the fingerprinting procedure is essential. For example, some fingerprinting tools rely just on specific fields of TCP and IP headers. Such information is sometimes stored as a message digest to save space and speed up the comparison process. However it is impossible to use the digest to search also for fingerprinting similarities and not just for exact matches.

Finally, keeping fingerprints information updated is a fundamental issue for a successful analysis. This is due to properties of ICT hardware and software quickly changing over time. This means that new information must be added to databases each time unknown fingerprints are shown. Such operation can be difficult because: a) new fingerprints must be verified on large data sets and with different networks or systems configurations and b) there is the need to pay attention not to create any overlap in the signatures as this situation could lead fingerprinting tools to incongruous results.

2 State of the Art

Fingerprinting literature can be divided into four main fields of research. We will describe each of them in the following sections. The first section is a description of numerous approaches to fingerprinting. The second section proposes new techniques to refine the fingerprints. The third section focuses on how fingerprinting can be mitigated or prevented in order to keep a system secure from malicious scans. Finally, the fourth section analyzes possible applications for fingerprinting techniques (e.g. as a support for intrusion detection systems).

2.1 Information sources

Fingerprinting analyzes a wide variety of elements, such as: a) devices (from simple electrical components to complex hardware systems), b) software (both applications and operating systems) and c) communication protocols. Due to this variety, researchers may take advantage of numerous tools and methodologies to get the information they need. Some of these instruments are comprehensive and suitable for many purposes while others are created for specific use cases only.

Most of the researches on TCP/IP stack fingerprinting focuses on how to optimize the standard methodology. This is usually performed by refining stored information about connection negotiation and similar mechanisms implemented by the TCP and the IP protocols. Such analyses take advantage of studies such as [6]. In this paper, the authors present "tcpanaly", a tool for TCP implementations' analysis. Tcpanaly passively inspects TCP segments in order to exploit differences and misbehaviors and uses them to distinguish specific operating system implementations. Although outdated, this work shows how differently the most famous operating systems behave in terms of TCP communications. Furthermore, the work argues how TCP headers are the most valuable source of information. In [7] the authors exploit some of these information. In fact, they succeed to increase the level of confidence in TCP/IP standard fingerprinting by looking at specific characteristics of the protocols implementation. The work is mostly focused on the SYN/SYN-ACK phase of the TCP three-way handshake. Beyond some standard fields of TCP/IP headers, the authors propose to analyze the TCP Timestamp field and a particular behavior of the window size management. Despite the specification

[8], TCP timestamps are not always set to zero and this provides useful information to recognize operating systems. The window size is analyzed in relation to the TCP Timestamp since, sometimes, such size is directly related to that value. The authors conclude explaining how this approach can be extended in many other directions by exploiting the discrepancy of different implementations of the TCP stack due to ambiguities in the TCP protocol specification.

Several authors worked on exploiting implementation differences of specific protocols to define new fingerprints. Works such as [9] explain in detail how it is possible to build a highly-reliable classification technique based on payload packets inspection. In this work, the authors concentrate on traffic flows more than packets. The use of traffic flows simplifies data processing and adds knowledge about the context in which devices to identify work. By exploiting different identification methods, authors succeed to develop a framework for traffic characterization based on packets' payloads. Another practical work on the topic is [10] in which some application-level features of protocols like HTTP, FTP, SMTP are used to improve the recognition of software installed on standard PCs.

On the signal-based fingerprinting, there are several works that try to take advantage of hardware properties and define suitable signatures for ICT systems. In [11] authors show that Ethernet devices can be uniquely identified and tracked using as few as 25 Ethernet frames. This is achieved by analyzing variations in analog signals caused by hardware or manufacturing inconsistencies. In their work, the authors emphasize that analog characteristics are difficult to forge with respect to information provided by the other level of the ISO/OSI stack. This approach exploits a matched filter to profile signals in order to create a signature for the device that produced such signals. This low-level analysis on signals can be combined with standard fingerprinting mechanisms. An alternative approach consists of recording small deviations called clock skews in device hardware [12]. The authors claim that a fingerprinter can use the information contained in TCP headers to estimate a devices clock skew and, thereby, fingerprint a physical device. This mechanism can rely on different properties but the authors focus on deriving the necessary information from the TCP Timestamp fields in NTP communications. This technique can also be applied to different goals (e.g. differentiation between honeynets and real networks, recognition of hosts behind a NAT, etc.).

Several approaches focus on exploiting alternative packet properties, such as timing. For example, in [4] authors present an operating system detection method based on temporal response analysis. Based on this idea they developed a tool, called RING, which is an alternative (or a complementary instrument) to standard fingerprinting tools. The main focus of the work is the process of packet retransmission implemented by the TCP. This mechanism is not strictly specified in [13] thus, there are different implementations. By measuring the delays of packet retransmissions as well as looking at specific flags,

sequence numbers and acknowledge numbers, the authors claim that it is possible to get valuable information about the target behavior. Furthermore, the work concludes by emphasizing that some other TCP transitions can be suitable for temporal response analysis (e.g. the closure of a TCP connection).

Some works explore the possibility to fingerprint communication protocols. This is the case of [5] that specifically introduces the concept of "protocol fingerprint". The starting point of this work focuses on the possibility to classify network traffic relying exclusively on the statistical properties of the flows and, therefore, of the packets. The classification mechanism is based on three simple characteristics of the captured IP packets: the size, inter-arrival time and arrival order. The authors define a fingerprint describing a set of Probability Density Functions (PDF) linked to a set of flows generated by the same known protocol. To classify unknown traffic flows, the authors check if the behavior of the flow is statistically compatible with at least one of the described PDF. Performed tests show promising results even when a larger set of protocols leads to an increase in the error margin.

Finally, there are fingerprinting approaches based on port scanning. Some works as [14] exploit the use of standard network ports by known services to recognize systems and applications. However, there are two common criticisms to this approach: the difficulty to check those ports behind NATs or other network components, and the increasing usage of port randomization by most of the applications.

2.2 Learning techniques

As discussed in the introduction, the creation and maintenance of fingerprints is one of the major problems in fingerprinting. Machine learning is one of the solutions used to improve these activity. The papers discussed in this section contribute to refine the model generation process we will discuss in Section 3. Furthermore, some of the works try to exploit a feedback technique to let the tool automatically and progressively improve its effectiveness.

It is possible to use several different techniques to refine a fingerprinting tool (e.g. Bayesian networks, neural networks, etc.). In [15], for example, the authors use probabilistic learning to develop a Naïve Bayesian classifier to passively infer hosts' operating systems from packet headers. The work relies on the observation fields d_i (TTL, Window SYN packet size, "do not fragment" bit) and the standard Bayesian formulation:

$$P(H_i|D) = \frac{\prod_j P(d_i|H_i)P(H_i)}{P(D)} \tag{2.1}$$

As assumption, the previous fields' values do not interfere with each other. The authors propose two possible training phases. In the first they use the P0F signature file while in the second they collect TCP SYN packets passing into a web server and correlate them with the information about operating systems involved carried by the application headers. The proposed classifier has two main advantages. First, it computes the probabilistic weights based on the training set and this makes it robust against non-conventional TCP stack implementations. Second, such a classifier maintains its confidence degree even in presence of incomplete or conflicting data. Despite these advantages, the work lacks of tests on large datasets. It is worth noting that the authors achieved good results just using four TCP fields while standard fingerprinting relies on an exact match from an exhaustive list of TCP settings. Moreover, the authors were also able to obtain a continuous refinement on the degree of identification confidence without deep-packet inspection.

Application-fingerprinting can also take advantage of such techniques. In [10], authors use three different learning mechanisms: Bayesian inference, clustering and entropy analysis. The first mechanism models a discrete distribution for each analyzed feature. The second takes advantage of the AdaBoost algorithm. This is a meta-algorithm used to incrementally refine a weighted combination of weak classifiers (e.g. Bayesian classifiers). Finally, the Regularized Maximum Entropy algorithm looks for a distribution in the training set that has the maximum entropy and, at the same time, satisfies some specific constraints. The clusterization process relies on several assumptions. First of all, it is worth noting that the result of a comparison is a binary value. This means that a specific flow can be related to an application or be completely unrelated to it. Second, the authors decided to use raw application level data instead of trying to derive some information (e.g. ASCII words of the stream). Finally, just the first bytes of the stream are considered in the comparison. This last characteristic is used to limit the amount of data the machine learning technique has to process as well as speed up the traffic identification. The results indicate that this approach is highly accurate and scales enough to allow online application identification on different kind of links. Among the three learning algorithms, AdaBoost performs the best in several tests showing in some of the cases a considerably better performance than the Naïve Bayesian mechanism.

2.3 Mitigation techniques

Mitigation techniques differ from each other depending on whether we deal with active or passive fingerprinting. The main way to defeat active fingerprinting is the use of firewalls and access control strategies. Passive fingerprinting is more complicated to

mitigate since fingerprinting tools do nothing but listening to network traffic. A possible solution involves the use of hardware or software called "scrubbers" which aims at confusing fingerprinters. As explained in [16] a scrubber is an active mechanism that converts heterogeneous network flows into well-behaved flows that cannot be "interpreted". With this technique, a TCP/IP packet with unequivocal properties related to the operating system that has generated it, is cleaned of such characteristics leaving the content intact. The implementation of a "scrubber" starts by studying suitable traffic modifications at transport layer. Then it focuses on TCP traffic and, finally, on TCP/IP fingerprinting fields. Some examples of implemented scrubbing activities are: reordering of TCP options, modification of the original TCP sequence number and removal of unused combination of TOS bits in IP headers.

Another example of such technology, used to mitigate standard TCP/IP fingerprinting, is proposed in [17]. In this work, the authors discuss the position of the scrubber in the network and finally identify the gateway as the most suitable place where to implement the "scrubbing". Then, they design and implement a scrubber that works at both network and transport layers to convert recognizable traffic. IP-level recognizable characteristics are mainly due to IP header flags and fragment reassembly algorithms. For this reason, the scrubber modify the flags, recalculating the checksums, and reassemble the datagram (to disassemble it again if needed). TCP-level recognizable characteristics are located over TCP header options. Again, the "scrubber" intervenes to reorder them or to change their values accordingly to the RFCs. This strategy let the authors organize heterogeneous communication into sanitized packets that do not reveal clues about hosts' operating systems. Moreover, results show good performance in terms of throughput and scalability.

Some solutions directly focus on specific tools instead of working on fingerprinting general characteristics. Once a software is targeted, these techniques exploit a mitigation approach that confuses the fingerprinting activity that such specific program performs. This is the case of [18] with Nmap. The paper describes different solutions to defeat the tool by behaving like another chosen operating system. There is a long list of practical modifications on standard operating systems that can confuse Nmap. In Linux, the author suggest to use the netfilter module called "IP personality" that changes parameters like TCP Initial Sequence Number, TCP initial Window Size, etc. Moreover, there are useful system patches (such as the "Stealth" provided by Security Technology) that allow to mask the implemented TCP/IP stack. BSD systems have available a patch called Blackhole that implements special mechanisms to respond to TCP and UDP non-conventional traffic. Such customized solutions are usually effective as they are tuned to confuse any attempt of a specific tool to analyze some traffic. However, to obtain that kind of behavior, these techniques rely on a deep knowledge of the targeted software.

Considering the wide range of IT proprietary software and the difficulty to analyze it without the source code (or any other technical documentation), such knowledge is not always achievable.

Some works propose comprehensive solutions based on network architectures designed to avoid or mitigate scans. In [19] the authors introduce an architecture that combines several techniques such as packet scrubbing, advanced packet matching (forcing specific packets to be the same) and tarpitting (purposely delaying network packets). The proposed architecture is based on several modules that successively process all the traffic going in and out of the network. The incoming traffic is instead sequentially "scrubbed", black-or-white-listed (depending on the kind of connection) and finally filtered through a static rule set. The outgoing traffic is cleaned by a packet "scrubber". The fact that all packets are modified in several ways or delayed in time allows the authors to prevent several network scanning techniques and to highly increase the difficulty to obtain accurate information with the remaining ones. Finally, the authors discuss the effect of the proposed architecture against a variety of scanning practices.

Finally, network administrators sometimes need to assess the resistance of their systems against fingerprinting. According to [20] it is possible to verify and assess such property. The author's starting point is an analytical evaluation of existing open source fingerprinting tools. Focusing on active fingerprinting and especially on tools like Nmap and XProbe, the authors present an empirical evaluation of the robustness of specific defensive device configurations to defeat the most discriminative scanning software. The configurations are a combination of different defensive techniques such as "scrubbing" and firewalling. In the conclusions, the author shows comparative tables that assess the performance of each configuration against different fingerprinting technologies and operating systems.

2.4 Applications

As stated in the introduction, fingerprinting can be used to enhance or complement intrusion detection and prevention systems. We divide intrusion detection systems into two main categories: signature-based and anomaly-based. The former relies on well-known malicious patterns that can be recognized in network communications or system behaviors. The latter focuses on detecting activities that significantly differ from common system behaviors.

Fingerprinting can be useful in both strategies. Fingerprints can be a suitable signature for avoiding specific operating systems to access the network. In the second case, fingerprinting could be an additional method to resolve cases where the alerted anomaly

has a high probability to be a false positive. For this reason, the fingerprinting classification algorithm can be a suitable feature inside intrusion detection systems or other security mechanisms.

The approach described in [7] uses passively detected OS fingerprints of end hosts to correctly resolve ambiguities between different network stack implementations. Such ambiguities can be used during reassembly processes of fragmented IP packets and TCP segments to evade IDS detections. In this work, the author provides to IDS sensors additional knowledge about the network stack implementation of the target host and thus improves the chances of detecting the attacker. This feature is implemented through two fingerprint tables. One for the TCP SYN packets and the other for the responses (TCP SYN ACK). The TCP/IP fields checked by the author (Maximum Segment Size, Selective Acknowledgment, Timestamping, and Window Scaling) are combined together to populate the entries of both the tables. At runtime, TCP SYN and SYNACK values of TCP traffic seen by the IDS are kept in a cache. Every time the IDS needs to perform a lookup for a particular IP address to determine the type of operating system, it triggers the cache lookup mechanism that runs a best-match algorithm on all the cache entries. In the conclusions, the author claims to be able to map more than a thousand hosts and identify dozens of different operating systems. However, the implemented mechanism needs pre-compiled and updated SYN ans SYN ACK tables to work properly.

Also intrusion detection systems working at layer one of the ISO/OSI model can exploit some fingerprinting techniques. As discussed before, it is possible to define signal-level fingerprints. This activity can be used to refine the analysis of an attack as showed in [21]. The work is based on the assumption that, in some cases, none of the digital information can be trusted. In this case, the author argues that it is possible to rely on some analogue characteristics of the signals transmitted in the network. Looking at such signals, the author tries to differentiate among the "idealized" signal (the one that carries the information), device-specific variations, and random noise. Once recognized, the device-specific variations create an analogue-fingerprint controlled through specific filters.

Finally, fingerprinting techniques are used to mitigate specific types of cyber-attacks. This is the case of the work described in [22]. In their paper, the authors propose a preliminary architecture that applies spam detection filtering at the router-level using light-weight signatures for spam senders. The authors argue to use TCP headers to develop fingerprint signatures that can be used to identify spamming hosts basing the analysis on the specific operating systems from which emails are sent. An advantage of this mechanism is that it can be used together with other spam filtering techniques to ensure accurate spam detection. Anyway, the authors claim that using simple signatures (e.g. Windows-based signatures) largely fails. This test proves the need for highly

accurate signature selection mechanisms to avoid the risk of misclassification.

2.5 Conclusions

In conclusion, there are several methods and strategies to implement fingerprinting. Despite this, most of the tools rely on the standard TCP/IP stack fingerprinting. This is due to two main factors. First, TCP and IP packets' fields remain the most valuable source of information that is gathered in few distinct bytes. Second, this information is quite simple to extract and parse. This ease is evident looking at the high number of signatures already defined in some fingerprinting tools (e.g. p0f's dataset and Xprobe2's database). Moreover, the process used to compare unknown fingerprints with a dataset's signatures is fast and reliable.

The amount of systems for which a signature still does not exist and the high number of mitigation techniques are two major issues of the standard TCP/IP stack fingerprinting. The continuous development of new operating systems and applications makes the first problem hard to solve. Numerous attempts against this issue rely on automatic learning techniques and try to derive new signatures from specific datasets. However, these results are less accurate than the standard processes of comparison. Fingerprinting mitigation techniques try, instead, to disturb such a comparison processes. Modification and concealing of computer fingerprints are huge obstacles to a proper recognition of a target machine.

Alternative fingerprinting methods mainly focus on time or traffic patterns analysis. However, the accuracy of these methods is still not as high as the standard TCP/IP stack fingerprinting. It is worth noting that the joint use of TCP/IP packets' fields with comprehensive network analyses can overcome some actual limitation of the fingerprinting process.

Finally, several activities regarding ICT security can exploit fingerprinting results and methodologies. In the most common case, fingerprinting techniques can contribute to intrusion detection systems' activities. Also high-level security applications (e.g. anti-spam) can use at their advantage some fingerprinting features. Moreover, there is an extensive use of fingerprinting tools in activities such as vulnerability assessment and penetration testing.

3 Reference Model

In Section 2 we showed that different fingerprinting techniques can involve different kinds of network and host information. We also discussed why TCP/IP standard fingerprinting technique is the most used and widely studied in literature. In this chapter we deepen this technique and we analyze what properties fingerprinters have in common. The conclusion of this chapter is a description of a reference architecture for fingerprinting tools.

3.1 TCP/IP fingerprinting

Standard fingerprints refer to the following information of TCP and IP protocols:

- **Initial packet size** (16 bits): the size of TCP and IP headers together. This size can change due to the variable length of the Options fields of the two headers. This is the reason why fingerprinting tools are interested just in the size of TCP and IP headers of the SYN packet.
- **Initial TTL** (8 bits): the "time to live" is a counter contained in IPv4 and IPv6 header used to prevent a packet from circulating indefinitely. The TTL field is set by the sender of the datagram and reduced by every router on the route to its destination. Usually, the initial value of the TTL is a power of two.
- **Window size** (16 bits): the total number of bytes it is possible to be transmitted over a TCP connection without receiving any acknowledgment
- **Max segment size** (16 bits): the largest amount of data that can be included in a single, non fragmented TCP segment
- **Window scaling value** (8 bits): the parameter, specified in [8], that manages the size of the TCP receive window. This is used to allow TCP work efficiently with high-speed networks and it can be managed in different ways.
- **"don't fragment" flag** (1 bit): this field of the IP header allows an end-system to choose preventing an IP packet from being fragmented by a router.

- **"sackOK" flag** (1 bit): this field of the TCP header allows a receiver host to selectively inform the sender about received segments and to retransmit only those that have actually been lost [23].
- **"nop" flag** (1 bit): the TCP "No Option" flag is used to separate the different options used within the TCP option field

Several fingerprinting tools use these fields together to form a 67-bits signature that describes a device working on an IT network.

Not every TCP/IP packet carries all the information. This is the reason why most of the tools are able to recognize a device only looking at the setup of its connections or at few other specific segments. SYN and SYN-ACK packets are the most valuable sources of information. RST packets are also useful. The rest of the communication flow usually does not carry any of the previously described features (e.g. "sackOK" information are included only in SYN or SYN-ACK segments) and it is often discarded by fingerprinting tools.

3.2 Architecture

Despite different targets (e.g. identification of operating systems, hardware, specific software, etc.) or information sources (e.g. packets, network flows, time, etc.), all fingerprinting tools share several common tasks. For this reason, it is possible to summarize such tasks into a few logical building blocks shared among all fingerprinting tools. Each building block receives as input a result from another. A building block elaborates the provided data by adding new information. Finally the block forwards information or feedback to other components. A reference architecture for a fingerprinting tool is provided in Figure 3.1.

3.2.1 Source

Every fingerprinting tool depends on one or more information sources. A source can be uniform or heterogeneous, depending on the target of the analysis or the overall tool complexity. Fingerprinters usually exploit TCP/IP protocol information. Several works focus on enlarging the standard 67-bits signature or identifying different subsets of characteristics [7] [6]. Furthermore, some tools try to fingerprint different levels of the ISO/OSI stack. We already discussed how Ethernet or application headers are suitable information sources [10]. Finally, network topologies, time patterns, or also network ports usage are valuable sources as well [14].

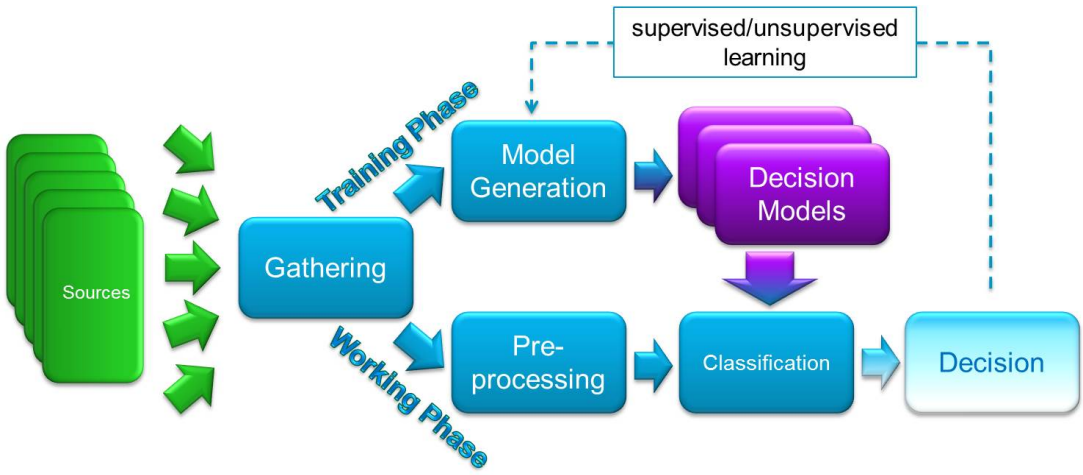

Figure 3.1: Fingerprinting tools' reference architecture

3.2.2 Gathering

The Gathering module implements all the techniques used by a tool to collect the data it needs. It relies directly on the information source and captures the valuable data on the network. Gathering module implementations differ according to the choice to the active or passive paradigm implemented by the fingerprinter. Active tools query the system to obtain a set of information needed to outline the fingerprint. Passive tools try to acquire such information in a less intrusive way, capturing network traffic. Usually, active fingerprinting has more chances to succeed in recognizing the system. This is because active fingerprinting implies collecting all the information necessary for describing a fingerprint while passive fingerprinting implies collecting only the information that is available at that moment on the communication channel. During the active fingerprinting, the Gathering module implements the probing activity of the tool (i.e. by creating suitable packets and sending them to a device in order to study the responses). This is the case of Nmap and XProbe2++. The information collected, once labeled and organized, will form the dataset used by the tool for the analysis. Finally, not all the traffic contains valuable sources of information. For this reason, the Gathering module has to filter out useless or unknown communication and bad traffic.

3.2.3 Model Generation

The Model Generation component organizes and stores data. Data provided by the Gathering module is sometimes labeled by humans. For example, in the device fingerprinting training, fingerprints captured by tools are linked to information regarding the system that has generated them. Also, some tools automatically refine the dataset using machine learning techniques as in [10] and [15]. Fingerprinting tools have to always ensure model generation to be consistent and unambiguous. Exploiting few characteristics in generating models can cause overlaps in the dataset. Avoiding identical signatures or ambiguous data structures (e.g. data structures including attributes that refer to variable OS, hardware or software characteristics) is a key element for fingerprinting reliability. For example, P0f provides a command for a signature collision check.

3.2.4 Decision Models

Decision Models are the outputs of the Model Generation phase. The models represent the knowledge of a fingerprinter. They use such knowledge as an input in the classification process. Usually models are organized in signatures and stored in files. Both Nmap and P0f have a signature for each operating system. More advanced tools exploit different kinds of signature at the same time. This is the case of XProbe2++ and SinFP. The former has a variable number of signature items per OS, related to several tests it can perform [24]. The latter exploits two different datasets keeping active and passive signatures separated [25].

3.2.5 Pre-processing

The Pre-processing component defines how the information is organized to be a suitable input for the analysis. Most of the times, fingerprinters exploit the Model Generation module to create a temporary Decision Model of the system under examination. In a few cases, such as XProbe2++, the Pre-processing component selects one by one the information needed for the comparison.

3.2.6 Classification

The Classification module implements the analysis of the collected information with respect to the dataset. This is the last component of the architecture. The classification may imply different kinds of actions. For example, P0f performs simple comparisons among signatures. Nmap and XProbe2++ exploit smarter methods like fuzzy signature matching. Moreover, XProbe2++ refines its classification by using decision trees.

The most advanced tools add to Classification modules awareness about fingerprinting mitigation techniques. This is used to detect information artificially manipulated to avoid fingerprinting. In particular, a software called "scrubber" is often used to confuse fingerprinting. As explained in [16] a scrubber is an active mechanism that converts heterogeneous network flows into well-behaved flows that cannot be interpreted. With this technique, a TCP/IP packet with unequivocal properties related to the operating system that has generated it, is cleaned of such characteristics leaving the content intact. XProbe2++ implements systems to avoid such problems [24].

3.2.7 Decision

Finally, the Decision made by the Classification module gives the input for the manual or automatic refinement process of the dataset. Numerous tools calculate percentages of uncertainty of the provided results.

In the next chapter we map on the reference architecture fingerprinting problems and challenges found in the literature. This is useful to emphasize which building block requires more attention or improvement to be employed in critical infrastructure environments. Moreover, such analysis is important to evaluate further dependencies or information flows among the modules.

4 Tools

Fingerprinting tools presented in this chapter implement some of the techniques and the concepts discussed in Section 2. Described applications typically exploit TCP/IP fingerprinting techniques. In most cases a tool is either active or passive and, therefore, focuses just on one of the two information gathering strategies. However, there are examples of applications that try to take advantage of both.

4.1 Active Fingerprinting Tools

This section describes the most relevant active fingerprinting tools.

4.1.1 Nmap

Written by Gordon Lyon and distributed under the GNU GPL license, Nmap is one of the most important and well known tools for TCP/IP fingerprinting. The tool was developed in October 1998 and the number of implemented tests has been improved over the years. As a consequence, this set of tests significantly increased the number of different systems it could recognize [26].

Nmap sends specially crafted packets to the target host and then analyzes the responses. The main features include: host discovery, port scanning, version detection, OS detection and scriptable interaction with the target. In addition to these, the tool can also provide further information on targets, including reverse DNS names, device types, and MAC addresses.

Nmap works with about 16 TCP, UDP and ICMP probes. The first phase of analysis involves the sending of six different TCP SYN probe packets. They differ based on the window size, window scale, timestamp flag and value, the SACK permitted flag and the options field. The acknowledgment and sequence numbers are random, but saved for use during the analysis. Nmap checks seven fields of a standard TCP/IP fingerprint and performs several other tasks. These seven fields are:

- **TCP Initial Sequence Number (ISN) Sampling**: TCP uses sequence numbers for duplicate detection. Since the possibility of guessing an ISN is also a security concern, modern systems randomly generate this number in different ways.

Nmap spots possible regularities in TCP ISNs in order to recognize the software that generates them.

- **IP ID sequence generation**: similar to the previous test, Nmap checks regularities on the choice of the IP ID sequence value.
- **Options**: this test checks what kind of TCP options operating systems accept and implement. It is worth to note that systems supporting equal options still can be differentiated by their order inside the packets.
- **TCP Sequence/Acknowledgment numbers**: this test checks if the sequence number in the reply is based on the sequence/acknowledgment number given in the initial packet. The result of this test allows to differentiate among some operating systems.
- **TCP RST data checksum**: this test is performed when an end host responds to the probe with an RST packet. Since this data often consists of standard error messages in plain ASCII Nmap checks its content looking for notable information.
- **Non-zero field after the TCP header length**: this field is set only when an explicit congestion notification (ECN) [27] is triggered. The software controls this property because only some operating systems implement it.
- **Non-zero URG pointer**: the test checks the value of this field. Normally the URG pointer can point to a specific section of the payload which contains urgent data. Since SYN packets do not have any payload this pointer is usually set to 0. Anyway, some systems do not properly do that and simply leave garbage.

In the second working phase Nmap uses ICMP probes. The tool checks the following fields in the responses to differentiate among many operating systems:

- **Do not fragment (DF)**: as in the standard TCP/IP fingerprint this test verifies the value of this field.
- **IP ID sequence generation with ICMP**: this test checks whether the IP IDs from ICMP and TCP are based on a single sequence generator or whether they are separate.
- **IP initial TTL guess**: as in the standard TCP/IP fingerprint this test verifies the value of this field.

- **ICMP Response code**: this test checks if the sequence number in the reply is based on the sequence/acknowledgement number given in the initial packet.
- **TCP RST data checksum**: even if the code value of the ICMP echo reply should always be zero some systems send other values. This test evaluates the possible response.

The final protocol used by Nmap is UDP. The test consists of a single UDP probe sent to a closed port. The expected reply is an ICMP port unreachable message in which Nmap checks the following characteristics and compare them with known operating system behaviors.

- **Do not fragment (DF)**: as in the standard TCP/IP fingerprint this test verifies the value of this field.
- **IP initial TTL guess**: as above, this test verifies the value of this field.
- **IP total length**: in an ICMP destination port unreachable message the amount of data included can be arbitrary. Nmap uses this information to differentiate among specific operating systems.
- **Unused port unreachable field nonzero**: the tool checks the last four bytes of an ICMP port unreachable message that should be set to zero according to [28].
- **Returned probe IP total length**: this test controls the integrity of this value.
- **Returned probe IP ID**: operating systems manage differently this field and Nmap uses it to spot them.
- **Integrity of returned probe IP checksum**: this test controls if the checksum is calculated properly.
- **Integrity of returned probe UDP checksum and data**: the UDP header checksum should not be changed in the response but some operating systems do it. Nmap checks this value and also the integrity of the returned data.

Nmap stores a fingerprint in memory using a tree data structure of attributes and values. The matching algorithm for detecting fingerprints is instead relatively simple. Nmap takes a subject fingerprint and tests it against every single reference fingerprint in its database. When the reference fingerprint does have a matching line, they are compared. For a probe line comparison, Nmap examines every individual test from the

subject category line in turn. Whenever a matching test is found, Nmap increments the "PossiblePoints" counter by the number of points assigned to this test. If the test results match, another counter called "NumMatchPoints" is incremented by the test's point value. Once all of the probe lines are tested for a fingerprint, Nmap divides "NumMatchPoints" by "PossiblePoints". The result is a confidence factor describing the probability that the subject fingerprint matches that particular reference fingerprint.

More detailed information can be found in [29].

4.1.2 XProbe2/XProbe2++

Developed and maintained by Ofir Arkin since 2001, the XProbe tool series is a set of active fingerprinting software based on ICMP protocol specifications. The tools are a result of the "ICMP Usage In Scanning Research" [30] project carried out by the Sys-Security Group. XProbe's last release, XProbe2++, exploits different operating systems' fingerprinting techniques such as: fuzzy signature matching, probabilistic guesses, simultaneous multiple matching and a signature database for both TCP/IP and application protocols.

The use of ICMP with respect of TCP can have several advantages. First of all, it makes it possible to differentiate two systems that implement similar TCP stacks. Moreover, in comparison with Nmap, XProbe2 needs less probes and generates a smaller amount of traffic. Another difference between the two software regards the fuzzy logic. XProbe implements this approach since the first release and rates operating systems by giving a score based on how well they match with specific tests. Contrariwise, Nmap takes advantage of an approach based on traditional logic and adds this feature only in responding to a specific request from the user.

Usually, an ICMP error message must contain at least the first eight data bytes of the datagram that caused the error (the so called offending packet). XProbe2++ exploits the fact that several systems quote more than those eight bytes. Moreover, when a host sends back the IP header of the offending packet, it should not change it except for the TTL and the checksum. Since some operating systems do not implement this characteristic correctly, XProbe2++ fingerprints the following fields:

- **IP Total Length**: some systems add or subtract 20 bytes from the original value.
- **IP ID**: in some cases, the bit order is changed.
- **IP Fragmentation flags/offset**: as above some systems change the bit order.
- **IP header checksum**: it can be calculated incorrectly.

- **Type of Service (TOS)**: usually, in source quench messages it has to be equal to the previous packet value while in the other cases it should be 6 or 7 [31]. However, several operating systems behave differently.
- **DF bit**: it should be always set to 0, however sometimes it is quoted from the offending packet.
- **ICMP code**: it is usually the original code field even if some systems reset it to zero.

It is worth noting that Xprobe2++ does not separate tests in different steps and it does not combine the information to form a single signature. The authors choose to implement the analysis within a tree structure. Such a structure is walked through by taking decisions based on the results of the probing. Moreover, XProbe2++ can combine several probes in a single datagram avoiding the possibility that they influence each others' results.

4.1.3 RING

Developed in 2002, RING implements the concept of time pattern fingerprinting. The authors of the tool published their results in [4] describing how it is possible to distinguish among different operating systems by looking at the behavior of their TCP retransmission mechanism implementation. Even if the RFC [13] proposes a retransmission algorithm, such specification is not mandatory. For this reason many operating systems implement some variations of the algorithm.

To have a chance to observe these patterns, RING forces the target to a non-standard communication, where timeout values will be reached. This functionality is implemented simply by setting up a packet filter function that avoids the machine running the tool to close the TCP triple-handshake. At the same time, RING has to collect SYN ACK packets sent by the target evaluating the time elapsed among them.

The stored signatures are time patterns describing the intervals among each retransmission of the target. RING stores information also on the time elapsed until a machine sends the RST packet. Since not every operating system implements the reset of the connection, this is another valuable distinguishing element. Each signature is labeled with the related operating system and its version. From its documentation and provided tests [32] we know that RING's dataset contains a large set of operating systems.

One of the main advantages of RING is the use of just one open port. Differently from other tools, also a target system protected by a firewall is likely to leave at least one port open while filtering all the others. With such a configuration, tools such as Nmap

do not work well since some of their tests would be based on closed ports. Moreover, RING uses only standard TCP packets. This mechanism does not disturb the targeted machine or do not let an intrusion detection system trigger any alert.

On the other hand, in some cases the analysis timespan needed by RING is quite large. This is mainly due to the fact that lots of TCP retransmission implementations differ only after some steps. This is the case of several Windows versions but also complete different operating systems (e.g. the comparison between Windows 2000 and FreeBSD 4.4). To deal with such issue, the authors propose to take advantage of other TCP mechanisms such as the closure of a TCP connection.

4.2 Passive Fingerprinting Tools

This section describes the most relevant passive fingerprinting tools.

4.2.1 P0f3

P0f is one of the most comprehensive passive TCP/IP fingerprinting tools. Michal Zalewski wrote the first version of the software in 2000. After that, William Stearns maintained it over the years until the original author decided to completely rewrite it (first with P0f2 and then with P0f3). Among its characteristics there are: high scalability and fast identification, measurement of system uptime and network hookup, automated detection of connection sharing, and detection of malicious hosts.

P0f3 provides a well-known set of API. This feature offers applications running in the same system a simple way to connect to the tool and get information about the remote hosts. This property can be useful to integrate P0f3 with spam filters, web applications, etc. The tool can work also as a daemon.

P0f3 can use four different detection modes: incoming connection fingerprinting by analyzing SYN packets, outgoing connection fingerprinting by checking SYN-ACK packets, refused outgoing connection fingerprinting by looking at RST packets, and established connection fingerprinting by parsing stray ACK packets. Unfortunately just the first mode is well supported.

The incoming connection fingerprinting relies on the standard TCP/IP fingerprinting. Other than the common fields, the tool looks at:

- **Timestamp**: calculating a correct round-trip-time on specific links needs to include a timestamp in the TCP header. This timestamp can be zero or based on the systems up-time and therefore it differentiates some operating systems.

- **Selective ACK permitted**: systems usually acknowledge TCP segments cumulatively. Since this can lead to inefficiency there is a specification for selective acknowledgment [23]. However, not all the systems implement this functionality.
- **Unrecognized options**: sometimes specific options of the headers are not recognized. This leads most of the fingerprinting tools to misunderstand the packet. P0f3 succeeds in exploiting this information for OS detection.
- **EOL option**: a test checks this TCP option indicating that the end of the option list does not coincide with the end of the TCP header. Just a few operating systems take advantage of such option.
- **TCP option order**: the order of the TCP options within a packet is heavily dependent on the implementation and it is a valuable information for fingerprinting.
- **Packet anomalies**: ad-hoc implementations of TCP and IP levels of the ISO/OSI stack often lead to anomalies in the packets: P0f3 can recognize some of them and use these as instruments to refine the analysis. Among the known anomalies there are:
  - **over-data**: SYN and SYN+ACK packets should not have any payload even if sometimes they do.
  - **over-options**: sometimes systems have options after the EOL one.
  - **Zero IP ID**: similar to Nmaps IP ID test, P0f3 checks whether an IP ID is zero or not to differentiate specific operating systems
  - **IP options**: usually IP options are not set, but some modern systems are starting to use them.
  - **Non-zero URG pointer**: similarly to what Nmap observes, the URG flag should never be set in SYN packets. The content of this pointer can be really useful for operating system detection.
  - **Unused field**: Fields not yet assigned by any RFC or used by any protocol should be always set to zero. Some systems do not clear them.
  - **Non-zero ACK number**: the ACK number in a SYN packet with the ACK flag unset is usually zero and left aside. Some systems however send junk data as in the URG pointer case.
  - **Non-zero second timestamp**: TCP timestamps are used to compute the round-trip time. According to [8] the initial SYN packet should have a zeroed second timestamp but some operating systems behave differently.

  - **Unusual flags**: this test checks the presence of unexpected extra-flags used by some operating systems.

The outgoing connection fingerprinting relies again on the TCP/IP fingerprinting and it exploits some of the tests already described in the incoming connection fingerprinting. P0f3 makes also these further tests:

- **Non-zero ACK number**: this packet is supposed to have a proper ACK number but some operating systems do not follow this specification.
- **Non-zero second timestamp**: similar to the previous test, the tool checks if this value is properly set or not.

The refuse of a connection offers different hints to fingerprinting. This is mainly due to the freedom given by TCP specifications [13]. Moreover, when the connection is not established the value carried within the flag attribute do not have consequences and operating systems freely fill these attributes in different ways. Some distinctive characteristics among standard operating systems are:

- **Connection-refused packet**: a normal connection-refused packet should only be sent when a connection is refused. However, there are implementations that force a system to send it in response to an unexpected ACK packet.
- **"ACK number is zero" error**: P0f3 detects if a system sends a connection-refused packet with an ACK number set to zero.
- **"Non-zero SEQ value error"**: this test checks if the SEQ value of a RST segment is different from zero.
- **Connection dropped**: when the RST flag is set with a non-zero sequence number the acknowledgment number should be zero. However, this is not mandatory and some systems fill the acknowledgment number with a different value.
- **"RST flag and SEQ value are zero" error**: this test checks if the operating system forces the two fields to be coherent or not.

Finally, the Ongoing connection fingerprinting is experimental. In this phase the tool compares the captured packets with six well-know fingerprints that can be found in any phase of a TCP connection.

4.2.2 Ettercap

Ettercap is an open source network security tool for man-in-the-middle attacks. The tool is used for several different purposes such as pentesting, protocol analysis and security auditing. Ettercap also includes several plugins for specific features. Among them, there is one called"finger" which implements some passive operating system fingerprinting techniques.

The purpose of the "finger" plugin is to analyze the passive information coming from a host when it sets or accepts connections towards other hosts. This information is usually enough to detect the operating system and the running services of the related machine. Ettercap selects just SYN and SYN ACK packets to fingerprint systems.

Ettercap's database contains different fingerprints for each type of packet. According to its documentation, Ettercap relies more on SYN fingerprints because the SYN+ACK is always influenced by the SYN (e.g. if a SYN does not contain a SACK option the SYN ACK will not have the SACK option even if supported). For this reason, if the tool receives a SYN+ACK, it always marks the related operating system as temporary and, as soon as it receives also a SYN, confirms the information.

The fingerprinting mechanism is quite simple. The set of fingerprints in the database is sorted according to their values. In this way, the tool maintains two similar fingerprints next to each other. When a SYN or a SYN ACK packet arrives, Ettercap starts matching the captured information with its fingerprint database. If it does not find a perfect match, it returns the last checked value. Due to the above described sorting, the last fingerprint is also the most similar to the one known by the tool.

With respect to other tools, Ettercap is not focused on fingerprinting. For this reason it has limited capabilities to recognize specific versions of several operating systems. Furthermore, the fingerprint database has not been updated since 2004.

4.2.3 Satori

Satori is a passive fingerprinting tool published by Eric Kollmann in 2005. The tool relies on the possibility to use Dynamic Host Configuration Protocol (DHCP) request and discover messages to identify devices at boot time. With respect to TCP/IP fields used by standard fingerprinting, Satori exploits a large amount of information in the "option" field of DHCPv4 and DHCPv6 protocols. Moreover it merges such information with several others retrieved from different Internet protocols (HTTP, ICMP, etc.).

As mentioned above, the possibility to use the DHCP protocol (as well as its predecessor BOOTP) is the main contribution of Satori. DHCP allows to define several parameters within the "options" field. A full list of them is available in [33]. In the

Satori documentation the author argues that most of the operating systems use a unique set of options during a DHCP message exchange. Moreover, some of the options bring already valuable information for the fingerprinting activity (e.g. option 60 "Vendor class identifier"). One of them is the MAC address of the device and this information is also available when the sender is behind a router. It is worth mentioning that Satori has been recently updated to work also with DHCPv6 [34].

What follows is a full list of protocols Satori uses with the DHCP to fingerprint devices:

- **Samba** (SMB): to identify Windows machines
- **Cisco Discovery Protocol** (CDP): used by CISCO systems to share information about connected devices
- **Enhanced Interior Gateway Routing Protocol** (EIGRP): similar to CDP
- **Simple Network Management Protocol** (SNMP): to exploit community names
- **Skinny Client Control Protocol** (Skinny): to listen to session signaling information among devices connected through Cisco systems
- **Universal Plug and Play** (UPnP): to capture information on devices establishing services such as data sharing
- **HyperText Transfer Protocol** (HTTP): to exploit banners
- **Internet Control Message Protocol** (ICMP): to determine devices' base classes
- **Spanning Tree Protocol** (STP): to determine devices' rules (e.g. bridge vs. root nodes)
- **Service Advertising Protocol** (IPX SAP): to profile file and directory servers
- **Open Shortest Path First** (OSPF): to map the network and obtain information about routers

With its coverage, Satori is a comprehensive and efficient passive fingerprinting tool and after eight years its database is still maintained as in the case of most known tools.

4.3 Hybrid Fingerprinting Tools

This section describes the most relevant fingerprinting tools that exploit both active and passive techniques.

4.3.1 SinFP3

SinFP3 is the most famous active-passive fingerprinting tool. Written by Patrice Auffret in 2005, it was re-designed twice in 2006 and 2011. SinFP3 is not just a simple fingerprinting tool but is now a framework for network discovery. The latest version introduces a plugin-based architecture that allows programmers to develop new tools around the framework.

SinFP3 was created to address some limitations of the other fingerprinting tools. There are, in fact, several situations in which most of the fingerprinting methods fail. This is the case of firewalls using Network Address Translation (NAT) and Port Address Translation (PAT) but also software implementing packet normalization techniques. Such situations are becoming increasingly common on the Internet and SinFP3 aims at overcoming this barrier by proposing new ways towards OS fingerprinting.

It is worth noting that, despite its hybrid nature, SinFP3 uses the same dataset of signatures both for active and passive fingerprinting. Differently from other tools, SinFP3 uses an actual SQLlite database to store the data. The information regards just active signatures and, for this reason, it is not possible to simply retrive and compare them in passive mode. The solution adopted by SinFP3 is to modify the signatures on-the-fly when needed. Additionally, the matching algorithm remains the same for both active and passive fingerprinting.

The possibility to fingerprint on IPv6 is another interesting feature of SinFP3. The IPv6 headers are the only difference between fingerprinting IPv4 and IPv6. Since that, the author of the tool proposes a fingerprinting schema in which new fields of the IPv6 header are mapped into the old IPv4 ones and thus treated in the same way. The analysis performed on the IPv6 is, therefore, equal, to the one done on the IPv4 except that, for the first, we are interested in ID, TTL and "do not fragment" bit while in the second we look at Flow Label, Hop Limit and Traffic Class. The fingerprinting techniques performed on the TCP remain the same. This method works again for both active and passive fingerprinting.

Each signature of the database is built from three responses of the target machine (two SYN-ACK packets and a RST-ACK packet). The matching algorithm is similar to algorithms implemented in Web search engines. The objective is to find an intersection on multiple domains. The domains are defined by the header field values stored in the database while the intersection represents the common solutions among the three comparisons performed (one for each response).

Thanks to the used signature matching algorithm, the author of the tool claims in [25] that it is almost superfluous to add new signatures to the current version of the database. This feature actually solves one of the major problems of fingerprinting.

Moreover, such an algorithm makes the tool highly resilient to signature modifications derived by intermediate routing or filtering devices.

4.3.2 Shodan

Shodan is a search engine developed by John Matherly in 2009. It can not be strictly considered a fingerprinting tool but it implements some of the ideas and characteristics described in Section 2.

Shodan differentiates from standard search engines because it indexes banner information instead of web page contents. Banners are meta-data servers send back to clients after a request. Protocols such as HTTP, FTP, SSH, Telnet, SNMP and SIP provide numerous information during a client-server communication and Shodan uses such information to fingerprint devices. Finally, it indexes its findings providing a comprehensive database of Internet devices to users.

The website crawls the Internet for any kind of device. However, the project concentrates on SCADA systems and components. For this reason, in 2013, the security community pointed out how easy it is to find critical infrastructures thorough Shodan. An example, is given in [35]. The article provided by CNN Money confirmed that devices such as traffic lights, security cameras and "home automation" systems were available to anyone on the web. Moreover, it stressed how even more critical components (e.g. PLCs supervising plants and power grids) use no or naive security features (e.g. default usernames and passwords).

Beyond the standard search on alphanumerical strings, the database can be browsed through different filters:

- **country**: to find devices located in the given country
- **city**: equal to the previous filter, it narrows the search to a specific city
- **geo**: similar to the country and city filters, it is used to search devices within an area identified by geographic coordinates (latitude, longitude, radius)
- **hostname**: to search for devices with specific hostnames
- **net**: it limits the search to a range of IP addresses or subnets
- **OS**: to look for devices running specific operating systems
- **port**: it allows users to search for specific services
- **before/after**: to filter the search by the data in which devices were indexed

Shodan is not just a simple collection of IP addresses. The Web portal provides some tools to improve researches such as ScanHub that allows to physically locate devices all over the world. Moreover, a large dataset of exploits against different kinds of device is available. Finally Shodan provides also a set of APIs to allow developers build and test pentesting tools on the web. All the resources provided by Shodan are constantly updated by continuously crawling the Internet.

5 ICS fingerprinting

Fingerprinting of ICS devices has not received significant attention by researchers yet. In the literature there are only very few examples of standard fingerprinting methodologies being applied to industrial control system environments. However, not all described concepts and solutions are suitable for ICS environments. As explained in [2], active probing of the network can interfere with the correct operation of some components. For this reason, the authors argue that the use of passive information gathering techniques is always preferable to minimize such risk. But even then, existing passive fingerprinting tools rely on the presence of a dataset with already-known system and device fingerprints.

5.1 Tests

Few of the tools described in the previous chapter provide ICS fingerprints per se but, also in that case, they are not able to correctly fingerprint the devices. We have conducted some preliminary tests that show that, in a ICS environment, P0f only recognizes standard components like windows machines used as SCADA servers or HMI but is not able to identify PLCs or other SCADA/ICS specific devices. Moreover, we tried the most comprehensive standard tools (Nmap and XProbe2++) in a safe environment to evaluate their findings. We run the tests on five different PLCs:

- **SIEMENS SIMATIC S7 1200 (AC/DC/Relay)**
- **SIEMENS SIMATIC S7-1200 (DC/DC/Relay)**
- **ABB AC800M**
- **DATAwatt D05-MCU 60870-5-104**
- **Janitza Electronics UMG 604**

The results of these tests are showed in Section **??**.

5.2 Evaluation

Our tests show that no standard fingerprinting tool was able to correctly fingerprint PLCs. This is mainly due to the fact that fingerprinters' datasets lack information related to ICSs. However, from a practical point of view, this was not the only problem. Some of the tools showed problems related to specific implementation choices. This is the case of two specific tests we made on the SIEMENS SIMATIC S7 1200 (AC/DC/Relay) PLC with Nmap and Xprobe2++: the "lack of integration" problem and the "extra information" problem.

"Lack of integration" problem

Nmap can run several fingerprinting services on different network ports. One of these services works on port 80 and is in charge of recognizing the web server software. Siemens PLCs have a web server application running on port 80 and Nmap succeeded in recognizing it and labeling it as "Siemens Simatic S7-1200 PLC httpd". Despite that, the tool fingerprinted the device as a QEMU because of the result of other fingerprinting services. This test showed limitations Nmap has in solving ambiguities. In this particular case, it would have been enough to discard results with low precision and enforce the high precision result of the web server fingerprint.

"Extra information" problem

This test was useful to evaluate how feasible standard signatures are for ICSs. As the other tools, Xprobe2++ was not able to fingerprint Siemens PLCs due to a lack of information in its dataset. To check that this was the only cause of the result we set up a signature for the Siemens PLC and we added it to the Xprobe2++ database. Despite this new information, the tool was again unable to recognize the same device. Further analyses showed that the field "icmp_echo_tos_bits" of the signature was filled in randomly by the PLC. This operation invalidated the signature enough to be discarded in favor of a different one (in this specific case an "HP JetDirect" printer). This shows that the attributes needed to fingerprint ICS could be different from the attributes used in standard IT networks.

5.3 Challenges

There are a series of characteristics in SCADA/ICS that will make device fingerprinting more challenging compared to regular Internet or company LAN settings.

Active vs. Passive Fingerprinting

ICSs often monitor or control processes in which a failure may have disastrous consequences (or may be otherwise very undesirable). For this reason active probing (like scanning for open ports and then opening arbitrary TCP connections) should generally be avoided. As described in [2], in an ICS environment, it is always preferable to use only passive monitoring techniques to minimize the potential risk induced by active probing. This is less of a concern in general IT networks like cooperate LANs or the Internet, where port scanning and fingerprinting tools like Nmap are widely used. The computational power of devices involved and the non-critical role that these networks have usually allows more invasive techniques of information gathering.

Device Heterogeneity

Device heterogeneity is another challenge in ICS fingerprinting. There are different physical processes for which ICS systems are used. In each of them devices perform a wide range of different actions such as: data collection, sharing of information, coordination control, etc. Finally, ICS vendors usually develop their own devices for all the activities listed above. Creating several unique and reliable fingerprints can be difficult as there is the need to access several different installations in order to reach ubiquitous coverage. In standard IT networks the situation is different. Corporate networks and LANs usually involve personal computers running standard operating systems. Tools like Nmap and P0f provide comprehensive sets of fingerprints for such systems. However, this trend is changing due to the use of smartphones and other embedded devices that are still largely unknown by standard fingerprinting tools.

Proprietary protocols

The usage of proprietary protocols is an issue related to device heterogeneity. ICS devices mainly implement vendor-specific application level protocols. Several of these implementations are not published or documented. Unknown communication protocols cannot be leveraged for device fingerprinting as it is difficult to extract any useful information to recognize network components.

Long-running TCP Sessions

Most popular tools for TCP/IP fingerprinting (both active and passive) takes advantage especially of SYN and SYN-ACK packets exchanged during connection establishment. For passive fingerprinting, there is no assurance to observe such kind of traffic in an

industrial control system within a limited period of time. Typically, once a PLC and a control server set up a connection, the TCP session remains open for a very long time (days or even weeks) and application protocols like Modbus exchange messages over this TCP connection. The only option to trigger a TCP connection setup is to actively connect to the device, which is undesirable as discussed above. This fact severely reduces the applicability of fingerprinting based on observing TCP connection setup packets.

5.4 Opportunities

Long life-cycle of devices

Industrial control systems are used since the '70s and several system operators do not continuously update their hardware and software components [36]. Devices' life cycles are usually long enough to guarantee a fingerprint to remain valid for years. Personal computers and standard operating systems are instead, frequently changed and updated. Such constant changes in hardware and software make increasingly difficult to maintain the signature databases updated.

Stable topology and communications

The number of devices as well as temporal and communication patterns do not usually change in ICS networks [37]. Once established, the process control remains the same and components behave accordingly by continuously performing the same instructions and communications [38]. Fingerprinting can exploit this stability linking such patterns to component signatures. Standard networks do not allow the same. A personal computer's behavior depends on the user and is likely to change over time depending on the user's current needs (e.g. browsing or downloading)

Protocol specification

Some ICS protocols, like Modbus, Profinet and MMS provide a way to query components in order to have information about their hardware and software. As showed in [39] there are a few tools already implemented such as PLCscan and Modbuspatrol. Fingerprinters can exploit such functionalities and enrich signatures with more comprehensive and precise information.

IN thw following section we contextualize this these points looking at the reference model described in 3.

5.5 ICS fingerprinting based on the Reference Model

To the best of our knowledge, only two ICS fingerprinters exist (PLCscan and Modbuspatrol) [39]. However, such tools exploit the "Read Device Identification" function in Modbus allowing users to query a device for information. The tools are biased to a specific protocol, exploiting the last characteristic described in Section 5.4. A comprehensive approach to the development of ICS fingerprinters has to face all the challenges outlined before. For this reason, we use the reference architecture defined in Section 3 to structure the discussion about ICS fingerprinting and to describe the properties each module has to have if used in a industrial environment.

5.5.1 Sources

In ICS environments there are different kinds of information that can be exploited for fingerprinting. As stated in Section 2.1 TCP/IP protocol characteristics are the most valuable ones. However, we identified one main issue. Long TCP sessions do not allow to see many three-way handshake (thus, no SYN and SYN-ACK packets). As those initial packets store the majority of the information needed, this approach seems less effective. Besides using TCP/IP information we can look to the ISO/OSI application layer. Even if application layer protocols might provide useful information, we still have the problem of unknown protocols. As discussed in Section 5.3, numerous proprietary protocols are being used in ICS systems. Such issues can make the exploitation of this information very difficult. However, there are example of protocols that already provide instruments to facilitate fingerprinting (e.g. Modbus). Finally, we believe that using temporal, traffic and communication patterns can be a reasonable solution to implement ICS fingerprinting. The use of this information should not suffer from any of the problems listed before. Moreover, we suggest to exploit the substantial stability and regularity of ICS networks' communication.

5.5.2 Gathering

With respect to standard fingerprinting we propose to reverse the balance in using active and passive techniques. We argue to exploit passive fingerprinting more than active to avoid any interference with the system under analysis. However, we showed that PLCScan and Modbuspatrol actively query devices for information. It is worth noting that, in this case, Modbus provides the function to perform such query thus it is unlikely to cause problems to the infrastructure. In the best situation, the sniffer used to capture network traffic has to be transparent to ICS components. It will not inject any kind

of traffic in the network and it will not send responses to any incoming message. This always guarantees no interferences with ICS operations. Collected information regarding traffic flows has a strong constraint as it relies on the position the sniffer has in the network. For instance, two traffic captures taken from the Field Network and the Process Network are hardly comparable. Thus, the Gathering module has to provide some general information about its position and accordingly label captured data. Fingerprinting based on packets' information does not suffer such a problem and allows a simpler and performing implementation of the module. Finally, not all the traffic contains valuable sources of information. For this reason, the Gathering module has to filter out non-ICS related communication. Moreover, it is useful to remove also bad traffic (e.g. TCP retransmission or duplicate ACK packets) as it is usually impossible to determine the cause and exploit the information.

5.5.3 Model Generation

Signatures are the most widely adopted structures to organize fingerprinting information. However, this method works well only with a fingerprinting methodology that relies on several precise properties. As discussed in Section 3.1 TCP and IP header fields form a 67-bits signature for OS fingerprinting. Such signature can be expanded or narrowed depending on the context and on available information. Datalink or application level information can exploit signatures as well. Querying devices for information usually provides a structured and detailed list of data that does not need any further specification. Beyond this, fingerprinters involving temporal, traffic and communication patterns deal with one comprehensive set of characteristics about an ICS infrastructure that cannot be reduced to a simple signature. Therefore, the use of such characteristics need the definition of a broader concept as well as a comprehensive data structure that outlines architecture, properties, and trends of an ICS infrastructure.

5.5.4 Decision Model

To solve the issue discussed in Section 5.5.3 we introduce the concept of Context Model. The Context Model aggregates information. It represents a comprehensive description of an ICS infrastructure that includes information about the behavior of the whole system has and the operations that single devices or sets of devices implement. Furthermore it concentrates on devices' roles in the system focusing on their characteristics and relationships. We use the Context Model as an intermediate step in creating *Decision Models*. Once extracted by the Gathering module, data is processed through the Context Model to obtain high-level information suitable for the classification phase.

5.5.5 Pre-processing

After the gathering phase, communication information undergoes a further refinement process. This process depends on the structure of the Decision Model and on the classification algorithms used by the fingerprinter. In the most simple case, the Pre-processing module extracts data from the unknown ICS system to build signatures for the comparison. In other cases, extracting high-level information about the infrastructure involves again the use of the Context Model. The Gathering module has two more issues. Firstly, the Classification module does not usually need all the information provided by the Gathering component. For this reason, the Pre-processing works as a filter, deciding what to pass to the next module. Secondly, not all the information needed are always available. In this case, the Pre-Processing module can also decide to drop the information or label it as "incomplete" and forward it anyway.

5.5.6 Classification

The Classification module determines the goal of the fingerprinting. Standard TCP/IP stack fingerprinting implements a set of comparison algorithms in order to recognize operating systems. Other methods fingerprint applications or hardware components. When ICS protocols provide a way to query devices for information, a comprehensive fingerprinting analysis is possible. In other cases, (e.g. exploiting temporal and traffic patterns) the information is not complete enough to detail components. The primary target of ICS fingerprinting is to recognize a component by describing the vendor, the hardware (e.g. device model), and its software. Other possible targets in ICS fingerprinting are:

- Component type identification (e.g differentiate between SCADA servers and PLCs)
- Component role identification (e.g. differentiate between main PLCs and normal PLCs)
- Network topology identification (e.g. differentiate situations in which PLCs communicate only with a SCADA server or schemas in which PLCs coordinate with each other)
- Gathering general information about the process (e.g. ICS working on energy systems perform updates and send messages more often than in water infrastructures)

5.5.7 Decision

The Decision is the output of the ICS fingerprinter. Depending on the exploited information or the complexity of the Decision Model it can be difficult to have a reliable automatic update of the dataset. However, in fingerprinting, the amount of information owned is a key element towards finding matches to unknown devices. Fingeprinting results can be stored as new Decision Models only if there is a clear separation between them and the original dataset. Operating in this way allows the fingerprinter to use new models only in specific cases (e.g. solving ambiguities if the main dataset does not give reliable results).

6 Flow Fingerprinting

6.1 Introduction

In the previous chapter we described opportunities and limitations of fingerprinting on critical infrastructures. Results given by standard fingerprinting tools show that recognizing industrial control devices needs to leverage specific information and properties of the industrial environment. In what follows we propose one fingerprinting approach to leverage these features.

The Flow Fingerprinter attempts to recognize ICS devices by looking at the network traffic and communication patterns.

The tool focuses on the behavior that devices exhibit in the network. It does not aim at recognizing specific hardware and software but focuses on the roles such devices perform in the ICS. The tool models known ICS system and component behaviors and stores the information in its dataset. Then it compare these models with models of unknown ICS systems and components in order to find similarities.

The implementation of a Flow Fingerprinter relies on two hypotheses:

- Similar ICS networks (same used devices, same network topology, etc.) generate similar models
- Similar network devices (with the same role within ICSs) generate similar models

The accuracy of the Flow Fingerprinter depends on the degree of accuracy with which ICS networks are modeled and on the attributes and the metrics chosen to compare two models. In what follows we describe the operational steps of the tool.

6.2 Working phases

The Flow Fingerprinter works through five different phases. These phases follow the reference model described in Chapter 3.

6.2.1 Gathering of information

In the gathering phase the tool extracts all the information it needs from the traffic (either a pcap file or "live traffic"). The Flow Fingerprinter focuses its analysis on a small subset of common protocols of the TCP/IP stack that are:

- Ethernet (Layer 2)
- IP version 4 (Layer 3)
- IP version 6 (Layer 3)
- TCP (Layer 4)
- UDP (Layer 4)

The tool does not parse the headers but store the information regarding the number of network frames that used these protocols and the number of carried bytes.

6.2.2 Model construction

In the model construction phase the tool uses gathered information to build a model of the ICS network and its devices. The data structure chosen to model information is a directed multi-graph. This choice arises from the need to model communication on different layers of the ISO/OSI stack.

The modeling algorithm will map ICS data as follows:

- A **graph** represents an ICS infrastructure or a section of an ICS infrastructure
- A **node** represents a network device on an ICS network
- An **edge** represents a network connection between two devices over one of the protocols mentioned in Section 6.2.1.

For every ICS infrastructure, the tool saves three different models. The first creates a different node for each MAC address (sender or receiver) found in the traffic. The second generates a different node for each IP address (sender or receiver). The third creates a model for each IP address that has an active role in the network (a device that sent at least one frame on the network). We decide to create three different models to have a more comprehensive view of the ICS network topology.

All the models are stored in XML files.

6.2.3 ICS model comparison

Before looking specifically at device roles, in this phase, the tool compares the overall two models. The reason is that if the two models differ there is no assurance to have plausible results for the device comparison. Model comparison relies on standard graph properties (e.g. number of nodes and edges) and measures derived by network's attributes (e.g. bytes and packets exchanged over the network). The tool compares two models using the following metrics:

- Number of packets per second
- Number of bytes per second
- Ratio between the number of nodes and the number of edges
- Ratio between the number of nodes and the number of packets per second
- Ratio between the number of edges and the number of packets per second
- Ratio between the number of nodes and the number of bytes per second
- Ratio between the number of edges and the number of bytes per second
- Ratio between the number of packets and the number of bytes
- Ratio between the number of bytes exchanged over Ethernet connections and the total number of bytes
- Ratio between the number of bytes exchanged over IPv4 connections and the total number of bytes
- Ratio between the number of bytes exchanged over IPv6 connections and the total number of bytes
- Ratio between the number of bytes exchanged over TCP connections and the total number of bytes
- Ratio between the number of bytes exchanged over UDP connections and the total number of bytes

At the end of this working phase, the tool computes the average of the results of the aforementioned metrics to assess the degree of similarity between the models.

6.2.4 Device model comparison

In this phase, the tool looks for similarities between nodes comparing behaviors of unknown devices with information stored in the fingerprinter's dataset (e.g. models within the dataset can be manually or automatically labeled during the training phase). As for the previous case the comparison relies on standard graph properties (e.g. number of neighbors) and network properties (e.g. number of packets sent or received). The tool compares two nodes using the following metrics:

- Ratio between the number of edges and total number of edges in the model
- Ratio between the number of incoming edges and total number of edges in the model
- Ratio between the number of outgoing edges and total number of edges in the model
- Ratio between the number of exchanged packets per second and the average number of packets per second
- Ratio between the number of incoming packets per second and the average number of packets per second
- Ratio between the number of outgoing packets per second and the average number of packets per second
- Ratio between the number of exchanged bytes per second and the average number of bytes per second
- Ratio between the number of received bytes per second and the average number of bytes per second
- Ratio between the number of sent bytes per second and the average number of bytes per second

Moreover, the tool evaluates the aforementioned metrics for each one of the protocols mentioned in Section 6.2.1. Finally, bonuses are given to specific groups of metrics that share the same object (e.g. if all the Ethernet metrics are successful the tool increases the final degree of similarity achieved by the two nodes under comparison).

6.2.5 Result Computation

In the final phase the tool computes the final result of the analysis. Based on the outcomes of phases 3 and 4 two nodes are labeled as "similar" or not. Given model a and model b, for every possible couple of nodes (n_a,n_b), the final similarity result is calculated as follows:

$$Similarity = \#Nodes \cdot ICSModelSimilarity(a, b) \cdot DeviceModelSimilarity(n_a, n_b) \tag{6.1}$$

where $\#Nodes$ is the number of nodes included in the two models. This term is involved in the equation to increase the importance of comparisons that involve a high number of nodes.

6.3 Results

We run several tests to evaluate the Flow Fingerprinter. What follows is a test done on the traffic of a control network of a water treatment and purification plant. The dataset compiled for the flow fingerprinter included some PCAPS provided by ENEL from the cybersecurity laboratory in Livorno (CRISALIS Deliverable 3.1) and a further pcap that belongs to a network controlling a gas storage facility. All the IP addresses have been anonymized for privacy reasons.

The results of the Flow Fingerprinter are showed in Section 9.2

For every unknown node of the network under analysis the fingerprinter provides a set of nodes that match the best. Each fingerprinting attempt shows the two involved sub-networks (result of the first pre-processing step 6.2.3) and the binary results (true/false) of all the metrics used in the device model comparison 6.2.4.

6.3.1 Discussion

In the previous test the Flow Fingerprinter was able to correctly fingerprint industrial control devices five out of eight times (62.5% of success). In the same network P0f scored the best among standard fingerprinting tools and was able just to recognize one of the SCADA Servers (12.5%) of success. The Flow Fingerprinter was wrong to recognize two PLCs and a SCADA Server. In two cases (10.0.0.2 and 10.0.0.4) the tool rated the right fingerprint as second best choice.

Figure 6.1 shows an overview of the test results.

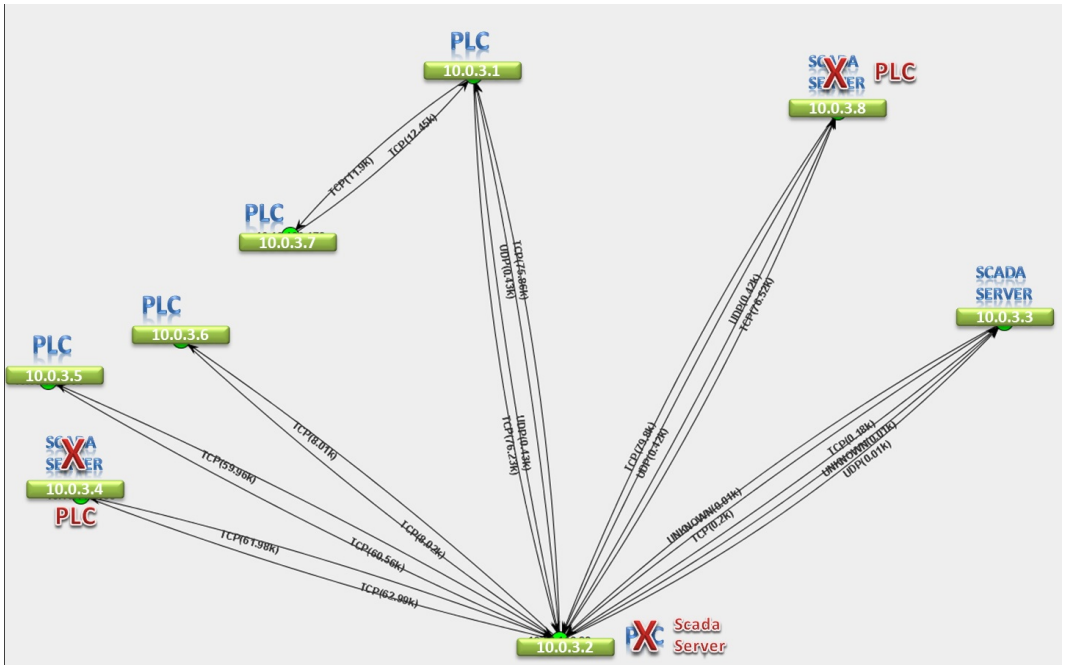

Figure 6.1: Flow Fingerprinter test results

6.4 Conclusions

The Flow Fingerprinter is just one of the possible approaches to ICS fingerprinting. With this example we showed how to leverage the study made in Chapter 3 and Chapter 5. The tool implements all the building blocks of the reference model. Moreover, it tries to exploit advantages and addresses disadvantages of fingerprinting activities in ICS. In this sense, we decided to focus on flow analysis and rely on the stability that ICS networks have both from a communication and a topology point of view. Preliminary tests show that flow analysis is a feasible way to recognize devices within ICSs.

In the future we are going to widen the tests and we aim to integrate the Flow Fingerprinter's analysis with standard techniques.

7 Advanced Metering Infrastructure fingerprinting

Similar to the SCADA case which is presented in Section 5, Advanced Metering Infrastructure (AMI) device fingerprinting is not yet covered by the research community. One main reason for this is a lack of standardization in the AMI domain, as each company chooses proprietary implementations for their smart metering infrastructure. In the following we are going to briefly mention the main characteristics of the AMI environments and their influence on the device fingerprinting process.

7.1 Devices in the Advanced Metering Infrastructure

The purpose of the AMI is to facilitate the data collection from the smart meters deployed at the customer's premises to the head-end. The communication between the smart meters and the head-end can be done directly or via intermediate devices which can be dedicated devices (data concentrators) or even other smart meters (in the case of a mesh communication network). The design of the communication network and the types of smart meters composing an AMI network can differ from operator to operator, so a generic topology or model of the AMI network is very hard to define. Compared to the SCADA network where the number of devices is somewhat fixed, the network topology is known and geographically limited in a small area, the AMI network can have thousands or tens of thousands of devices (mostly smart meters) and its geographical area can be considerable (towns or parts of a large city). Furthermore, as new smart meters and/or data concentrators are installed, the topology of the communication network might change, resulting in a very dynamic environment.

7.2 Difficulties and drawbacks in AMI fingerprinting

As mentioned in the previous section, the AMI is a very dynamic environment and its implementation varies from operator to operator. Usually, each operator has a very uniform AMI environment, having 1-3 different types of devices depending on its requirements, and the communication between them is regulated. At the same, the AMI environment is usually closed off; hence, initial knowledge about the characteristics of

the environment (e.g., communication protocols, communication media, the way communication is initialized, etc) is needed in order to perform analysis such as active or passive fingerprinting.

7.2.1 Active fingerprinting in AMI

Similar with the SCADA case, although the smart meters do not control critical processes, active fingerprinting is not recommended in the AMI environment. The smart meters have limited processing capabilities and the implementation of the communication stack is usually done with a limited memory, so actively probing a smart meter may turn it non-operational. The communication between the smart meter and the head-end can be performed over a communication network that is leased from a third party, such as a GPRS network provided by a mobile network operator. In this case, active fingerprinting requires, beside knowledge of the AMI network, knowledge of the leased network that is used and the authorization of the network operator to perform such analysis.

7.2.2 Passive fingerprinting in AMI

As was presented in CRISALIS deliverable D6.3 the AMI network has a complex topology and may be composed of different networks, which makes it hard to find a single tapping point where to perform full network fingerprinting. Also, spurious traffic might be hard to come by, because in some AMI implementations, in order to save power and bandwidth, the meters may be powered down until they need to speak. The communication pattern is mostly cyclic, the smart meters send the consumption recordings at defined periods of time.

The traffic between the smart meters and the head-end might be fully-encrypted, so in order to extract useful fingerprinting information the encryption communication keys must be known. In this case, even if there is a rogue device in the network, because of the lack of encryption keys, it will be ignored by the head-end and the smart meters. Even if the rogue device uses correct encryption keys, the fingerprinting tools would still be blind without the correct encryption keys.

7.3 AMI fingerprinting tools

To the best of our knowledge, at the time when this report is written, there are no dedicated AMI fingerprinting tools available. The device heterogeneity and proprietary

protocols play a huge role in the fingerprinting process so for not-common protocols specialized tools need to written for each specific environment or even for different models of smart meters. Despite this, classical tools, most of them presented in Section 5, can be used to fingerprint AMI devices, if the communication protocols match, but they need to be adapted so as to extract the information required for fingerprinting. Although AMI fingerprinting might be interesting in the future context of interconnected AMI networks, in the current context it is not a major concern for the end users and for the AMI infrastructure owners as the environment is already protected from unauthorized devices through other means. In other deliverables, we also describe tools that helps in AMI protocol discovery and analysis. Such tools can in the future be the basis for adapting general fingerprinting tools to the specialized environment and the implementation diversity of AMI environments.

8 Conclusions

In this deliverable we presented an overview of standard fingerprinting techniques, tools and methodologies. SCADA fingerprinting is still an unexplored field of research and our approach focused on identifying challenges and opportunities that this environment introduces.

Different constraints arise in performing fingerprinting in Industrial Control Systems. First, we discussed the feasibility of active fingerprinting within Critical Infrastructures. We made an analysis whether active fingerprinting actually poses a threat to SCADA/ICS systems and thus needs to be avoided. Despite several scientific papers claiming the risk of such activity we did some tests at the Enel IdroLab facility without any comprehensive result confirming the threat. We have also investigated which network activities can impact SCADA system functioning. For example, only open and close TCP connections can probably be considered less dangerous than trying to actively communicate with running services.

Device heterogeneity and proprietary protocols are two further constraints of Industrial Control Systems environments. Standard fingerprinting techniques do not always work properly with these limitations. With this deliverable we have opened a discussion on what are the most suitable information sources to use for SCADA fingerprinting (e.g. traffic flows, open ports, etc.). Moreover, depending on the source of information, we have discussed how the fingerprinting process has to go through all the reference architecture's modules provided in Chapter 3.

We tested standard fingerprinting tools on ICS to prove that methods exploited in IT do not suit an industrial environment. Based on that we detailed properties that can be exploited for ICS fingerprinting.

We implemented a prototype of a "Flow Fingerprinter" based on the analysis proposed in this deliverable. The main idea is to recognize SCADA devices typologies and roles by looking at traffic and communication patterns. This method does not need so much information on protocols or implementations. It relies on creating representations of known SCADA systems and their components. Finally, it uses such representations for comparison with unknown systems and devices. The SCADA Flow Fingerprinter implement all the building blocks of the reference model taking into account the discussed constraints.

As the Flow Fingerprinter relies on representations of SCADA systems, we have

opened a discussion with the other contributors of Working Package 4 about how to properly define a "SCADA Context Model". The SCADA Context Model represents the description of a SCADA infrastructure that includes information about the behavior the whole system has and the operations that single devices or sets of devices implement. Furthermore it concentrates on devices roles in the system focusing on their characteristics and relationships. SCADA infrastructures are usually composed by sub-systems and sub-networks working together. For example, it is common to find devices from different vendors in the same SCADA infrastructure, working on a specific task. Recognizing and analyzing sub-systems and sub-networks through the SCADA Context Model allows to narrow the fingerprinting research scope. Looking at the reference model, the Model Generation component will populate the SCADA Context Model and will organize it to become a Decision Model.

9 Appendix

9.1 Standard Fingerprinting Tool Tests

9.1.1 SIEMENS SIMATIC S7 1200 (AC/DC/Relay)

Nmap

Starting Nmap 6.01 (http://nmap.org) at 2013-06-03 14:55 CEST
PORTS: Using top 1000 ports found open (TCP:1000, UDP:0, SCTP:0)
—————— Timing report ——————
hostgroups: min 1, max 100000
rtt-timeouts: init 500, min 100, max 1250
max-scan-delay: TCP 10, UDP 1000, SCTP 10
parallelism: min 0, max 0
max-retries: 6, host-timeout: 0
min-rate: 0, max-rate: 0
——————————————————

NSE: Loaded 17 scripts for scanning.
Initiating Ping Scan at 14:55
Scanning 192.168.1.10 [4 ports]
Packet capture filter (device eth1): dst host 10.0.2.15 and (icmp or icmp6 or ((tcp or udp or sctp) and (src host 192.168.1.10)))
We got a TCP ping packet back from 192.168.1.10 port 80 (trynum = 0)
Completed Ping Scan at 14:55, 0.01s elapsed (1 total hosts)
Overall sending rates: 771.60 packets / s, 29320.99 bytes / s.
mass_rdns: Using DNS server 130.89.2.5
mass_rdns: Using DNS server 130.89.2.4
Initiating Parallel DNS resolution of 1 host. at 14:55
mass_rdns: 13.00s 0/1 [#: 2, OK: 0, NX: 0, DR: 0, SF: 0, TR: 4]
Completed Parallel DNS resolution of 1 host. at 14:55, 13.00s elapsed
DNS resolution of 1 IPs took 13.01s. Mode: Async [#: 2, OK: 0, NX: 0, DR: 1, SF: 0, TR: 4, CN: 0]
Initiating SYN Stealth Scan at 14:55

Scanning 192.168.1.10 [1000 ports]
Packet capture filter (device eth1): dst host 10.0.2.15 and (icmp or icmp6 or ((tcp or udp or sctp) and (src host 192.168.1.10)))
Discovered open port 443/tcp on 192.168.1.10
Discovered open port 80/tcp on 192.168.1.10
Completed SYN Stealth Scan at 14:56, 4.60s elapsed (1000 total ports)
Overall sending rates: 434.90 packets / s, 19133.81 bytes / s.
Initiating Service scan at 14:56
Scanning 2 services on 192.168.1.10
Completed Service scan at 14:56, 12.62s elapsed (2 services on 1 host)
Starting RPC scan against 192.168.1.10
Packet capture filter (device eth1): dst host 10.0.2.15 and (icmp or (tcp and (src host 192.168.1.10)))
Initiating OS detection (try #1) against 192.168.1.10
OS detection timingRatio() == (1370264174.181 - 1370264173.678) * 1000 / 500 == 1.006
Retrying OS detection (try #2) against 192.168.1.10
OS detection timingRatio() == (1370264176.435 - 1370264175.931) * 1000 / 500 == 1.010
NSE: Script scanning 192.168.1.10.
NSE: Starting runlevel 1 (of 1) scan.
Nmap scan report for 192.168.1.10
Host is up, received reset (0.0061s latency).
Scanned at 2013-06-03 14:55:43 CEST for 34s
Not shown: 998 filtered ports
Reason: 998 no-responses
PORT STATE SERVICE REASON VERSION
80/tcp open http syn-ack Siemens Simatic S7-1200 PLC httpd
443/tcp open ssl/http syn-ack Siemens Simatic S7-1200 PLC httpd
Warning: OSScan results may be unreliable because we could not find at least 1 open and 1 closed port
Device type: general purpose
Running (JUST GUESSING): QEMU (90%)
OS CPE: cpe:/o:qemu:qemu
OS fingerprint not ideal because: Missing a closed TCP port so results incomplete
Aggressive OS guesses: QEMU user mode network gateway (90%)
No exact OS matches for host (test conditions non-ideal).
TCP/IP fingerprint:

SCAN(V=6.01%E=4%D=6/3%OT=80%CT=%CU=%PV=Y%G=N%TM=51AC9271%P=i686-pc-linux-gnu)
SEQ(SP=11%GCD=FA00%ISR=9C%TI=I%CI=I%II=I%SS=S%TS=U)
OPS(O1=M5B4%O2=M5B4%O3=M5B4%O4=M5B4%O5=M5B4%O6=M5B4)
WIN(W1=FFFF%W2=FFFF%W3=FFFF%W4=FFFF%W5=FFFF%W6=FFFF)
ECN(R=Y%DF=N%TG=40%W=FFFF%O=M5B4%CC=N%Q=)
T1(R=Y%DF=N%TG=40%S=O%A=S+%F=AS%RD=0%Q=)
T2(R=Y%DF=N%TG=FF%W=0%S=Z%A=S%F=AR%O=%RD=0%Q=)
T3(R=Y%DF=N%TG=FF%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)
T4(R=Y%DF=N%TG=FF%W=0%S=A%A=Z%F=R%O=%RD=0%Q=)
T6(R=Y%DF=N%TG=FF%W=0%S=A%A=Z%F=R%O=%RD=0%Q=)
T7(R=Y%DF=N%TG=FF%W=0%S=Z%A=S%F=AR%O=%RD=0%Q=)
U1(R=N)
IE(R=Y%DFI=O%TG=20%CD=Z)

TCP Sequence Prediction: Difficulty=17 (Good luck!)
IP ID Sequence Generation: Incremental
Service Info: Device: specialized
Final times for host: srtt: 6093 rttvar: 4141 to: 100000

Read from /usr/local/bin/../share/nmap: nmap-os-db nmap-payloads nmap-rpc nmap-service-probes nmap-services.
OS and Service detection performed. Please report any incorrect results at http://nmap.org/submit/
.
Nmap done: 1 IP address (1 host up) scanned in 34.93 seconds
Raw packets sent: 2048 (93.308KB) — Rcvd: 33 (1.876KB)

XProbe2++

Xprobe-ng v.2.1 Copyright (c) 2002-2009 fyodor@o0o.nu, ofir@sys-security.com, meder@o0o.nu

[+] Target is 192.168.1.10
[+] Loading modules.
[+] Following modules are loaded:
[x] ping:icmp_ping - ICMP echo discovery module
[x] ping:tcp_ping - TCP-based ping discovery module
[x] ping:udp_ping - UDP-based ping discovery module
[x] infogather:ttl_calc - TCP and UDP based TTL distance calculation

[x] infogather:portscan - TCP and UDP PortScanner
[x] fingerprint:icmp_echo - ICMP Echo request fingerprinting module
[x] fingerprint:icmp_tstamp - ICMP Timestamp request fingerprinting module
[x] fingerprint:icmp_amask - ICMP Address mask request fingerprinting module
[x] fingerprint:icmp_info - ICMP Information request fingerprinting module
[x] fingerprint:icmp_port_unreach - ICMP port unreachable fingerprinting module
[x] fingerprint:tcp_hshake - TCP Handshake fingerprinting module
[x] fingerprint:tcp_rst - TCP RST fingerprinting module
[x] app:smb - SMB fingerprinting module
[x] app:snmp - SNMPv2c fingerprinting module
[x] app:ftp - FTP fingerprinting tests
[x] app:http - HTTP fingerprinting tests
[+] 16 modules registered
[+] Initializing scan engine
[+] Running scan engine
fingerprint:icmp_tstamp has not enough data
Executing ping:icmp_ping
Executing fingerprint:icmp_port_unreach
[−] icmp_port_unreach::build_DNS_reply(): gethostbyname() failed! Using static ip for www.securityfocus.com in UDP probe
Executing fingerprint:icmp_echo
Executing fingerprint:tcp_rst
fingerprint:tcp_hshake has not enough data
Executing fingerprint:icmp_amask
Executing fingerprint:icmp_info
Executing fingerprint:icmp_tstamp
app:smb has not enough data
Executing app:snmp
ping:tcp_ping has not enough data
ping:udp_ping has not enough data
infogather:ttl_calc has not enough data
Executing infogather:portscan
Executing app:ftp
Executing app:http
[+] Signature looks like:
[+] "HP JetDirect ROM A.03.17 EEPROM A.04.09" (93%)
[+] Generated signature for 192.168.1.10:
fingerprint {

OS_ID =
#Entry inserted to the database by:
#Entry contributed by:
#Date:
#Modified:
icmp_addrmask_reply = n
icmp_addrmask_reply_ip_id = !0
icmp_addrmask_reply_ttl = <255
icmp_echo_code = 0
icmp_echo_df_bit = 1
icmp_echo_ip_id = !0
icmp_echo_reply = y
icmp_echo_reply_ttl = <32
icmp_echo_tos_bits = 0
icmp_info_reply = n
icmp_info_reply_ip_id = !0
icmp_info_reply_ttl = <255
icmp_timestamp_reply = n
icmp_timestamp_reply_ip_id = !0
icmp_timestamp_reply_ttl = <64
icmp_unreach_df_bit = 0
icmp_unreach_echoed_3bit_flags = OK
icmp_unreach_echoed_dtsize = 8
icmp_unreach_echoed_ip_cksum = OK
icmp_unreach_echoed_ip_id = OK
icmp_unreach_echoed_total_len = OK
icmp_unreach_echoed_udp_cksum = OK
icmp_unreach_ip_id = !0
icmp_unreach_precedence_bits = 0
icmp_unreach_reply = n
icmp_unreach_reply_ttl = <255
}
[+] GENERATED FINGERPRINT IS INCOMPLETE!
[+] Please make sure you target is not firewalled and you have specified at least one open TCP port and one closed TCP and UDP ports!
[+] Cleaning up scan engine
[+] Modules deinitialized
[+] Execution completed.

9.1.2 SIEMENS SIMATIC S7 1200 (DC/DC/Relay)

Nmap

Starting Nmap 6.01 (http://nmap.org) at 2013-06-03 15:03 CEST
PORTS: Using top 1000 ports found open (TCP:1000, UDP:0, SCTP:0)
———— Timing report ————
hostgroups: min 1, max 100000
rtt-timeouts: init 500, min 100, max 1250
max-scan-delay: TCP 10, UDP 1000, SCTP 10
parallelism: min 0, max 0
max-retries: 6, host-timeout: 0
min-rate: 0, max-rate: 0
————————————————————

NSE: Loaded 17 scripts for scanning.
Initiating Ping Scan at 15:03
Scanning 192.168.1.10 [4 ports]
Packet capture filter (device eth1): dst host 10.0.2.15 and (icmp or icmp6 or ((tcp or udp or sctp) and (src host 192.168.1.10)))
We got a TCP ping packet back from 192.168.1.10 port 80 (trynum = 0)
Completed Ping Scan at 15:03, 0.01s elapsed (1 total hosts)
Overall sending rates: 351.34 packets / s, 13350.90 bytes / s.
mass_rdns: Using DNS server 130.89.2.5
mass_rdns: Using DNS server 130.89.2.4
Initiating Parallel DNS resolution of 1 host. at 15:03
mass_rdns: 13.01s 0/1 [#: 2, OK: 0, NX: 0, DR: 0, SF: 0, TR: 4]
Completed Parallel DNS resolution of 1 host. at 15:03, 13.01s elapsed
DNS resolution of 1 IPs took 13.01s. Mode: Async [#: 2, OK: 0, NX: 0, DR: 1, SF: 0, TR: 4, CN: 0]
Initiating SYN Stealth Scan at 15:03
Scanning 192.168.1.10 [1000 ports]
Packet capture filter (device eth1): dst host 10.0.2.15 and (icmp or icmp6 or ((tcp or udp or sctp) and (src host 192.168.1.10)))
Discovered open port 443/tcp on 192.168.1.10
Discovered open port 80/tcp on 192.168.1.10
Completed SYN Stealth Scan at 15:03, 4.91s elapsed (1000 total ports)
Overall sending rates: 407.20 packets / s, 17915.31 bytes / s.
Initiating Service scan at 15:03

Scanning 2 services on 192.168.1.10
Completed Service scan at 15:03, 12.75s elapsed (2 services on 1 host)
Starting RPC scan against 192.168.1.10
Packet capture filter (device eth1): dst host 10.0.2.15 and (icmp or (tcp and (src host 192.168.1.10)))
Initiating OS detection (try #1) against 192.168.1.10
OS detection timingRatio() == (1370264619.188 - 1370264618.686) * 1000 / 500 == 1.006
Retrying OS detection (try #2) against 192.168.1.10
OS detection timingRatio() == (1370264621.504 - 1370264621.003) * 1000 / 500 == 1.002
NSE: Script scanning 192.168.1.10.
NSE: Starting runlevel 1 (of 1) scan.
Nmap scan report for 192.168.1.10
Host is up, received reset (0.0093s latency).
Scanned at 2013-06-03 15:03:07 CEST for 35s
Not shown: 998 filtered ports
Reason: 998 no-responses
PORT STATE SERVICE REASON VERSION
80/tcp open http syn-ack Siemens Simatic S7-1200 PLC httpd
443/tcp open ssl/http syn-ack Siemens Simatic S7-1200 PLC httpd
Warning: OSScan results may be unreliable because we could not find at least 1 open and 1 closed port
Device type: general purpose
Running (JUST GUESSING): QEMU (90%)
OS CPE: cpe:/o:qemu:qemu
OS fingerprint not ideal because: Missing a closed TCP port so results incomplete
Aggressive OS guesses: QEMU user mode network gateway (90%)
No exact OS matches for host (test conditions non-ideal).
TCP/IP fingerprint:
SCAN(V=6.01%E=4%D=6/3%OT=80%CT=%CU=%PV=Y%G=N%TM=51AC942E%P=i686-pc-linux-gnu)
SEQ(SP=11%GCD=FA00%ISR=9C%TI=I%CI=I%II=I%SS=S%TS=U)
OPS(O1=M5B4%O2=M5B4%O3=M5B4%O4=M5B4%O5=M5B4%O6=M5B4)
WIN(W1=FFFF%W2=FFFF%W3=FFFF%W4=FFFF%W5=FFFF%W6=FFFF)
ECN(R=Y%DF=N%TG=40%W=FFFF%O=M5B4%CC=N%Q=)
T1(R=Y%DF=N%TG=40%S=O%A=S+%F=AS%RD=0%Q=)
T2(R=Y%DF=N%TG=FF%W=0%S=Z%A=S%F=AR%O=%RD=0%Q=)

T3(R=Y%DF=N%TG=FF%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)
T4(R=Y%DF=N%TG=FF%W=0%S=A%A=Z%F=R%O=%RD=0%Q=)
T6(R=Y%DF=N%TG=FF%W=0%S=A%A=Z%F=R%O=%RD=0%Q=)
T7(R=Y%DF=N%TG=FF%W=0%S=Z%A=S%F=AR%O=%RD=0%Q=)
U1(R=N)
IE(R=Y%DFI=O%TG=20%CD=Z)

TCP Sequence Prediction: Difficulty=17 (Good luck!)
IP ID Sequence Generation: Incremental
Service Info: Device: specialized
Final times for host: srtt: 9254 rttvar: 6365 to: 100000

Read from /usr/local/bin/../share/nmap: nmap-os-db nmap-payloads nmap-rpc nmap-service-probes nmap-services.
OS and Service detection performed. Please report any incorrect results at http://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 35.82 seconds
Raw packets sent: 2048 (93.308KB) — Rcvd: 33 (1.876KB)

XProbe2++

Xprobe-ng v.2.1 Copyright (c) 2002-2009 fyodor@o0o.nu, ofir@sys-security.com, meder@o0o.nu

[+] Target is 192.168.1.10
[+] Loading modules.
[+] Following modules are loaded:
[*x*] ping:icmp_ping - ICMP echo discovery module
[*x*] ping:tcp_ping - TCP-based ping discovery module
[*x*] ping:udp_ping - UDP-based ping discovery module
[*x*] infogather:ttl_calc - TCP and UDP based TTL distance calculation
[*x*] infogather:portscan - TCP and UDP PortScanner
[*x*] fingerprint:icmp_echo - ICMP Echo request fingerprinting module
[*x*] fingerprint:icmp_tstamp - ICMP Timestamp request fingerprinting module
[*x*] fingerprint:icmp_amask - ICMP Address mask request fingerprinting module
[*x*] fingerprint:icmp_info - ICMP Information request fingerprinting module
[*x*] fingerprint:icmp_port_unreach - ICMP port unreachable fingerprinting module
[*x*] fingerprint:tcp_hshake - TCP Handshake fingerprinting module
[*x*] fingerprint:tcp_rst - TCP RST fingerprinting module

[x] app:smb - SMB fingerprinting module
[x] app:snmp - SNMPv2c fingerprinting module
[x] app:ftp - FTP fingerprinting tests
[x] app:http - HTTP fingerprinting tests
[+] 16 modules registered
[+] Initializing scan engine
[+] Running scan engine
fingerprint:icmp_tstamp has not enough data
Executing ping:icmp_ping
Executing fingerprint:icmp_port_unreach
[−] icmp_port_unreach::build_DNS_reply(): gethostbyname() failed! Using static ip for www.securityfocus.com in UDP probe
Executing fingerprint:icmp_echo
fingerprint:tcp_hshake has not enough data
Executing fingerprint:tcp_rst
Executing fingerprint:icmp_amask
Executing fingerprint:icmp_info
Executing fingerprint:icmp_tstamp
app:smb has not enough data
Executing app:snmp
ping:tcp_ping has not enough data
ping:udp_ping has not enough data
infogather:ttl_calc has not enough data
Executing infogather:portscan
Executing app:ftp
Executing app:http
[+] Signature looks like:
[+] "HP JetDirect ROM G.06.00 EEPROM G.06.00" (100%)
[+] Generated signature for 192.168.1.10:
fingerprint {
  OS_ID =
  #Entry inserted to the database by:
  #Entry contributed by:
  #Date:
  #Modified:
  icmp_addrmask_reply = n
  icmp_addrmask_reply_ip_id = !0
  icmp_addrmask_reply_ttl = <255

icmp_echo_code = 0
icmp_echo_df_bit = 0
icmp_echo_ip_id = !0
icmp_echo_reply = y
icmp_echo_reply_ttl = <32
icmp_echo_tos_bits = 0
icmp_info_reply = n
icmp_info_reply_ip_id = !0
icmp_info_reply_ttl = <255
icmp_timestamp_reply = n
icmp_timestamp_reply_ip_id = !0
icmp_timestamp_reply_ttl = <64
icmp_unreach_df_bit = 0
icmp_unreach_echoed_3bit_flags = OK
icmp_unreach_echoed_dtsize = 8
icmp_unreach_echoed_ip_cksum = OK
icmp_unreach_echoed_ip_id = OK
icmp_unreach_echoed_total_len = OK
icmp_unreach_echoed_udp_cksum = OK
icmp_unreach_ip_id = !0
icmp_unreach_precedence_bits = 0
icmp_unreach_reply = n
icmp_unreach_reply_ttl = <255
}
[+] GENERATED FINGERPRINT IS INCOMPLETE!
[+] Please make sure you target is not firewalled and you have specified at least one open TCP port and one closed TCP and UDP ports!
[+] Cleaning up scan engine
[+] Modules deinitialized
[+] Execution completed.

9.1.3 ABB AC800M

Nmap

Starting Nmap 6.01 (http://nmap.org) at 2013-05-31 17:06 CEST
PORTS: Using top 1000 ports found open (TCP:1000, UDP:0, SCTP:0)
———— Timing report ————

hostgroups: min 1, max 100000
rtt-timeouts: init 500, min 100, max 1250
max-scan-delay: TCP 10, UDP 1000, SCTP 10
parallelism: min 0, max 0
max-retries: 6, host-timeout: 0
min-rate: 0, max-rate: 0

NSE: Loaded 17 scripts for scanning.
Initiating Ping Scan at 17:06
Scanning 172.16.80.151 [4 ports]
Packet capture filter (device eth0): dst host 10.0.2.15 and (icmp or icmp6 or ((tcp or udp or sctp) and (src host 172.16.80.151)))
We got a TCP ping packet back from 172.16.80.151 port 80 (trynum = 0)
Completed Ping Scan at 17:06, 0.01s elapsed (1 total hosts)
Overall sending rates: 291.59 packets / s, 11080.33 bytes / s.
mass_rdns: Using DNS server 130.89.2.5
mass_rdns: Using DNS server 130.89.2.4
Initiating Parallel DNS resolution of 1 host. at 17:06
mass_rdns: 13.01s 0/1 [#: 2, OK: 0, NX: 0, DR: 0, SF: 0, TR: 4]
Completed Parallel DNS resolution of 1 host. at 17:07, 13.00s elapsed
DNS resolution of 1 IPs took 13.01s. Mode: Async [#: 2, OK: 0, NX: 0, DR: 1, SF: 0, TR: 4, CN: 0]
Initiating SYN Stealth Scan at 17:07
Scanning 172.16.80.151 [1000 ports]
Packet capture filter (device eth0): dst host 10.0.2.15 and (icmp or icmp6 or ((tcp or udp or sctp) and (src host 172.16.80.151)))
Discovered open port 80/tcp on 172.16.80.151
Completed SYN Stealth Scan at 17:07, 24.15s elapsed (1000 total ports)
Overall sending rates: 83.07 packets / s, 3654.11 bytes / s.
Initiating Service scan at 17:07
Scanning 1 service on 172.16.80.151
Completed Service scan at 17:07, 6.07s elapsed (1 service on 1 host)
Starting RPC scan against 172.16.80.151
Packet capture filter (device eth0): dst host 10.0.2.15 and (icmp or (tcp and (src host 172.16.80.151)))
Initiating OS detection (try #1) against 172.16.80.151
OS detection timingRatio() == (1370012860.421 - 1370012859.918) * 1000 / 500 == 1.008

```
Retrying OS detection (try #2) against 172.16.80.151
OS detection timingRatio() == (1370012862.742 - 1370012862.239) * 1000 / 500 ==
1.004
NSE: Script scanning 172.16.80.151.
NSE: Starting runlevel 1 (of 1) scan.
Nmap scan report for 172.16.80.151
Host is up, received reset (0.011s latency).
Scanned at 2013-05-31 17:06:56 CEST for 47s
Not shown: 999 filtered ports
Reason: 999 no-responses
PORT STATE SERVICE REASON VERSION
80/tcp open http syn-ack GoAhead-Webs httpd
Warning: OSScan results may be unreliable because we could not find at least 1 open
and 1 closed port
Device type: general purpose
Running (JUST GUESSING): QEMU (90%)
OS CPE: cpe:/o:qemu:qemu
OS fingerprint not ideal because: Missing a closed TCP port so results incomplete
Aggressive OS guesses: QEMU user mode network gateway (90%)
No exact OS matches for host (test conditions non-ideal).
TCP/IP fingerprint:
SCAN(V=6.01%E=4%D=5/31%OT=80%CT=%CU=%PV=Y%G=N%TM=51A8BCBF%P=i686-
pc-linux-gnu)
SEQ(SP=11%GCD=FA00%ISR=9C%TI=I%CI=I%II=I%SS=S%TS=U)
OPS(O1=M5B4%O2=M5B4%O3=M5B4%O4=M5B4%O5=M5B4%O6=M5B4)
WIN(W1=FFFF%W2=FFFF%W3=FFFF%W4=FFFF%W5=FFFF%W6=FFFF)
ECN(R=Y%DF=N%TG=40%W=FFFF%O=M5B4%CC=N%Q=)
T1(R=Y%DF=N%TG=40%S=O%A=S+%F=AS%RD=0%Q=)
T2(R=Y%DF=N%TG=FF%W=0%S=Z%A=S%F=AR%O=%RD=0%Q=)
T3(R=Y%DF=N%TG=FF%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)
T4(R=Y%DF=N%TG=FF%W=0%S=A%A=Z%F=R%O=%RD=0%Q=)
T6(R=Y%DF=N%TG=FF%W=0%S=A%A=Z%F=R%O=%RD=0%Q=)
T7(R=Y%DF=N%TG=FF%W=0%S=Z%A=S%F=AR%O=%RD=0%Q=)
U1(R=N)
IE(R=Y%DFI=O%TG=40%CD=Z)

TCP Sequence Prediction: Difficulty=17 (Good luck!)
IP ID Sequence Generation: Incremental
```

Final times for host: srtt: 11111 rttvar: 9553 to: 100000

Read from /usr/local/bin/../share/nmap: nmap-os-db nmap-payloads nmap-rpc nmap-service-probes nmap-services.
OS and Service detection performed. Please report any incorrect results at http://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 48.64 seconds
Raw packets sent: 2054 (93.552KB) — Rcvd: 38 (2.116KB)

XProbe2++

Xprobe-ng v.2.1 Copyright (c) 2002-2009 fyodor@o0o.nu, ofir@sys-security.com, meder@o0o.nu

[+] Target is 172.16.80.151
[+] Loading modules.
[+] Following modules are loaded:
[x] ping:icmp_ping - ICMP echo discovery module
[x] ping:tcp_ping - TCP-based ping discovery module
[x] ping:udp_ping - UDP-based ping discovery module
[x] infogather:ttl_calc - TCP and UDP based TTL distance calculation
[x] infogather:portscan - TCP and UDP PortScanner
[x] fingerprint:icmp_echo - ICMP Echo request fingerprinting module
[x] fingerprint:icmp_tstamp - ICMP Timestamp request fingerprinting module
[x] fingerprint:icmp_amask - ICMP Address mask request fingerprinting module
[x] fingerprint:icmp_info - ICMP Information request fingerprinting module
[x] fingerprint:icmp_port_unreach - ICMP port unreachable fingerprinting module
[x] fingerprint:tcp_hshake - TCP Handshake fingerprinting module
[x] fingerprint:tcp_rst - TCP RST fingerprinting module
[x] app:smb - SMB fingerprinting module
[x] app:snmp - SNMPv2c fingerprinting module
[x] app:ftp - FTP fingerprinting tests
[x] app:http - HTTP fingerprinting tests
[+] 16 modules registered
[+] Initializing scan engine
[+] Running scan engine
fingerprint:icmp_tstamp has not enough data
Executing ping:icmp_ping
Executing fingerprint:icmp_port_unreach

[−] icmp_port_unreach::build_DNS_reply(): gethostbyname() failed! Using static ip for www.securityfocus.com in UDP probe
Executing fingerprint:icmp_echo
fingerprint:tcp_hshake has not enough data
Executing fingerprint:tcp_rst
Executing fingerprint:icmp_amask
Executing fingerprint:icmp_info
Executing fingerprint:icmp_tstamp
app:smb has not enough data
Executing app:snmp
ping:tcp_ping has not enough data
ping:udp_ping has not enough data
infogather:ttl_calc has not enough data
Executing infogather:portscan
Executing app:ftp
Executing app:http
[+] Signature looks like:
[+] "HP JetDirect ROM H.07.15 EEPROM H.08.20" (93%)
[+] Generated signature for 172.16.80.151:
fingerprint {
  OS_ID =
  #Entry inserted to the database by:
  #Entry contributed by:
  #Date:
  #Modified:
  icmp_addrmask_reply = n
  icmp_addrmask_reply_ip_id = !0
  icmp_addrmask_reply_ttl = <255
  icmp_echo_code = 0
  icmp_echo_df_bit = 0
  icmp_echo_ip_id = !0
  icmp_echo_reply = y
  icmp_echo_reply_ttl = <64
  icmp_echo_tos_bits = 0
  icmp_info_reply = n
  icmp_info_reply_ip_id = !0
  icmp_info_reply_ttl = <255
  icmp_timestamp_reply = n

icmp_timestamp_reply_ip_id = !0
icmp_timestamp_reply_ttl = <64
icmp_unreach_df_bit = 0
icmp_unreach_echoed_3bit_flags = OK
icmp_unreach_echoed_dtsize = 8
icmp_unreach_echoed_ip_cksum = OK
icmp_unreach_echoed_ip_id = OK
icmp_unreach_echoed_total_len = OK
icmp_unreach_echoed_udp_cksum = OK
icmp_unreach_ip_id = !0
icmp_unreach_precedence_bits = 0
icmp_unreach_reply = n
icmp_unreach_reply_ttl = <255
}
[+] GENERATED FINGERPRINT IS INCOMPLETE!
[+] Please make sure you target is not firewalled and you have specified at least one open TCP port and one closed TCP and UDP ports!
[+] Cleaning up scan engine
[+] Modules deinitialized
[+] Execution completed.

9.1.4 DATAwatt D05-MCU 60870-5-104

Nmap

Starting Nmap 6.01 (http://nmap.org) at 2014-01-14 08:01 EST
PORTS: Using top 1000 ports found open (TCP:1000, UDP:0, SCTP:0)
———— Timing report ————
hostgroups: min 1, max 100000
rtt-timeouts: init 500, min 100, max 1250
max-scan-delay: TCP 10, UDP 1000, SCTP 10
parallelism: min 0, max 0
max-retries: 6, host-timeout: 0
min-rate: 0, max-rate: 0
————————————————————

NSE: Loaded 17 scripts for scanning.
Initiating Ping Scan at 08:01
Scanning 172.16.12.101 [4*ports*]

Packet capture filter (device eth0): dst host 10.0.2.15 and (icmp or icmp6 or ((tcp or udp or sctp) and (src host 172.16.12.101)))
We got a TCP ping packet back from 172.16.12.101 port 80 (trynum = 0)
Completed Ping Scan at 08:01, 0.01s elapsed (1 total hosts)
Overall sending rates: 451.93 packets / s, 17173.20 bytes / s.
mass_rdns: Using DNS server 8.8.8.8
Initiating Parallel DNS resolution of 1 host. at 08:01
mass_rdns: 13.00s 0/1 $[\#: 1, OK: 0, NX: 0, DR: 0, SF: 0, TR: 3]$
Completed Parallel DNS resolution of 1 host. at 08:02, 13.00s elapsed
DNS resolution of 1 IPs took 13.00s. Mode: Async $[\#: 1, OK: 0, NX: 0, DR: 1, SF: 0, TR: 3, CN: 0]$
Initiating SYN Stealth Scan at 08:02
Scanning 172.16.12.101 [1000$ports$]
Packet capture filter (device eth0): dst host 10.0.2.15 and (icmp or icmp6 or ((tcp or udp or sctp) and (src host 172.16.12.101)))
Discovered open port 21/tcp on 172.16.12.101
Discovered open port 23/tcp on 172.16.12.101
Discovered open port 80/tcp on 172.16.12.101
Completed SYN Stealth Scan at 08:02, 5.14s elapsed (1000 total ports)
Overall sending rates: 388.87 packets / s, 17108.05 bytes / s.
Initiating Service scan at 08:02
Scanning 3 services on 172.16.12.101
Service scan Timing: About 66.67% done; ETC: 08:04 (0:00:44 remaining)
Completed Service scan at 08:03, 94.02s elapsed (3 services on 1 host)
Starting RPC scan against 172.16.12.101
Packet capture filter (device eth0): dst host 10.0.2.15 and (icmp or (tcp and (src host 172.16.12.101)))
Initiating OS detection (try #1) against 172.16.12.101
OS detection timingRatio() == (1389704627.127 - 1389704626.625) * 1000 / 500 == 1.006
Retrying OS detection (try #2) against 172.16.12.101
OS detection timingRatio() == (1389704629.526 - 1389704629.025) * 1000 / 500 == 1.000
NSE: Script scanning 172.16.12.101.
NSE: Starting runlevel 1 (of 1) scan.
Nmap scan report for 172.16.12.101
Host is up, received reset (0.026s latency).
Scanned at 2014-01-14 08:01:54 EST for 116s

Not shown: 997 filtered ports
Reason: 997 no-responses
PORT STATE SERVICE REASON VERSION
21/tcp open ftp syn-ack
23/tcp open telnet? syn-ack
80/tcp open http? syn-ack
3 services unrecognized despite returning data. If you know the service/version, please submit the following fingerprints at http://www.insecure.org/cgi-bin/servicefp-submit.cgi
:
==============NEXT SERVICE FINGERPRINT (SUBMIT INDIVIDUALLY)=======
SF-Port21-TCP:V=6.01%I=7%D=1/14%Time=52D5355D%P=i686-pc-linux-gnu%r(NULL,1
SF:C,"220\x20\x20OS-9\x20ftp\x20server\x20ready\r\n")%r(GenericLines,1C,"2
SF:20\x20\x20OS-9\x20ftp\x20server\x20ready\r\n")%r(Help,1B0,"220\x20\x20O
SF:S-9\x20ftp\x20server\x20ready\r\n214-\x20Commands\x20available\x20at\x2
SF:0this\x20site\x20are:\r\n\x20\x20\x20\x20HELP\x20\x20\x20\x20\x20\x20US
SF:ER\x20\x20\x20\x20\x20\x20PASS\x20\x20\x20\x20\x20\x20QUIT\x20\x20\x20\
SF:x20\x20\x20PORT\x20\x20\x20\x20\x20\x20LIST\x20\x20\x20\x20\x20\x20NLST
SF:\x20\x20\x20\x20\x20\r\n\x20\x20\x20\x20CWD\x20\x20\x20\x20\x20\x20\x20
SF:XCWD\x20\x20\x20\x20\x20\x20RETR\x20\x20\x20\x20\x20\x20TYPE\x20\x20\x2
SF:0\x20\x20\x20STOR\x20\x20\x20\x20\x20\x20APPE\x20\x20\x20\x20\x20\x20NO
SF:OP\x20\x20\x20\x20\x20\r\n\x20\x20\x20\x20DELE\x20\x20\x20\x20\x20\x20R
SF:NFR\x20\x20\x20\x20\x20\x20RNTO\x20\x20\x20\x20\x20\x20STRU\x20\x20\x20
SF:\x20\x20\x20MODE\x20\x20\x20\x20\x20\x20PWD\x20\x20\x20\x20\x20\x20\x20
SF:XPWD\x20\x20\x20\x20\x20\r\n\x20\x20\x20\x20MKD\x20\x20\x20\x20\x20\x20
SF:\x20XMKD\x20\x20\x20\x20\x20\x20RMD\x20\x20\x20\x20\x20\x20\x20XRMD\x20
SF:\x20\x20\x20\x20\x20ALLO\x20\x20\x20\x20\x20\x20CDUP\x20\x20\x20\x20\x2
SF:0\x20XCUP\x20\x20\x20\x20\x20\r\n214\x20HELP\x20<command>\x20for\x20inf
SF:ormation\x20about\x20a\x20specific\x20command\r\n")%r(SMBProgNeg,1C,"22
SF:0\x20\x20OS-9\x20ftp\x20server\x20ready\r\n");
==============NEXT SERVICE FINGERPRINT (SUBMIT INDIVIDUALLY)=======
SF-Port23-TCP:V=6.01%I=7%D=1/14%Time=52D5355D%P=i686-pc-linux-gnu%r(NULL,5
SF:F,"\xff\xfb\x01\r\nOS-9/68K\x20V3\.1\x20\x20\x20DATAWATT\x20\x20–\x20M
SF:CU1\x20–\x20\x20DATAWATT\x20-\x2068300\x20\x20\x2014/01/14\x2014:57:58
SF:\r\n\r\nUser\x20name\?:\x20")%r(GenericLines,8B,"\xff\xfb\x01\r\nOS-9/6
SF:8K\x20V3\.1\x20\x20\x20DATAWATT\x20\x20–\x20MCU1\x20–\x20\x20DATAWATT
SF:\x20-\x2068300\x20\x20\x2014/01/14\x2014:57:58\r\n\r\nUser\x20name\?:\x
SF:20\r\nWho\?\r\n\r\nUser\x20name\?:\x20\r\nWho\?\r\n\r\nUser\x20name\?:\
SF:x20")%r(GetRequest,99,"\xff\xfb\x01\r\nOS-9/68K\x20V3\.1\x20\x20\x20DAT

SF:AWATT\x20\x20–\x20MCU1\x20–\x20\x20DATAWATT\x20-\x2068300\x20\x20\x20
SF:14/01/14\x2014:58:08\r\n\r\nUser\x20name\?:\x20GET\x20/\x20HTTP/1\.0\r\
SF:nWho\?\r\n\r\nUser\x20name\?:\x20\r\nWho\?\r\n\r\nUser\x20name\?:\x20")
SF:%r(HTTPOptions,9D,"\xff\xfb\x01\r\nOS-9/68K\x20V3\.1\x20\x20\x20DATAWAT
SF:T\x20\x20–\x20MCU1\x20–\x20\x20DATAWATT\x20-\x2068300\x20\x20\x2014/0
SF:1/14\x2014:58:13\r\n\r\nUser\x20name\?:\x20OPTIONS\x20/\x20HTTP/1\.0\r\
SF:nWho\?\r\n\r\nUser\x20name\?:\x20\r\nWho\?\r\n\r\nUser\x20name\?:\x20")
SF:%r(RTSPRequest,9D,"\xff\xfb\x01\r\nOS-9/68K\x20V3\.1\x20\x20\x20DATAWAT
SF:T\x20\x20–\x20MCU1\x20–\x20\x20DATAWATT\x20-\x2068300\x20\x20\x2014/0
SF:1/14\x2014:58:18\r\n\r\nUser\x20name\?:\x20OPTIONS\x20/\x20RTSP/1\.0\r\
SF:nWho\?\r\n\r\nUser\x20name\?:\x20\r\nWho\?\r\n\r\nUser\x20name\?:\x20")
SF:%r(RPCCheck,97,"\xff\xfb\x01\r\nOS-9/68K\x20V3\.1\x20\x20\x20DATAWATT\x
SF:20\x20–\x20MCU1\x20–\x20\x20DATAWATT\x20-\x2068300\x20\x20\x2014/01/1
SF:4\x2014:58:26\r\n\r\nUser\x20name\?:\x20\x80\.\.\(r\xfe\.\.\.\.\.\.\.\.
SF:\.\.\.ane,\x20R\.Doggett\x86\xa0\.\x97\—\.\.\.\.\.\.\.\.\.\.\.\.\.\.\.\
SF:.\.\.\.\.")%r(DNSVersionBindReq,3,"\xff\xfb\x01")%r(DNSStatusRequest,6D
SF:,"\xff\xfb\x01\r\nOS-9/68K\x20V3\.1\x20\x20\x20DATAWATT\x20\x20–\x20MC
SF:U1\x20–\x20\x20DATAWATT\x20-\x2068300\x20\x20\x2014/01/14\x2014:58:32\
SF:r\n\r\nUser\x20name\?:\x20\.\.\.\.\.\.\.\.\.\.\.\.\.\.\.")%r(Help,79,"\xf
SF:f\xfb\x01\r\nOS-9/68K\x20V3\.1\x20\x20\x20DATAWATT\x20\x20–\x20MCU1\x2
SF:0–\x20\x20DATAWATT\x20-\x2068300\x20\x20\x2014/01/14\x2014:58:37\r\n\r
SF:\nUser\x20name\?:\x20HELP\r\nWho\?\r\n\r\nUser\x20name\?:\x20")%r(SSLSe
SF:ssionReq,3,"\xff\xfb\x01")%r(Kerberos,3,"\xff\xfb\x01")%r(SMBProgNeg,11
SF:A,"\xff\xfb\x01\r\nOS-9/68K\x20V3\.1\x20\x20\x20DATAWATT\x20\x20–\x20M
SF:CU1\x20–\x20\x20DATAWATT\x20-\x2068300\x20\x20\x2014/01/14\x2014:58:46
SF:\r\n\r\nUser\x20name\?:\x20\.\.\.\xa4MBr\.\.\.\.\x08\x20\x08\.,\x20L\.C
SF:rane,\x20R\.Doggett@\.\.\.\.\.\.\.\.\.\.\.\.\.\.@\.\.\.\.\.\.\x81\.\.PC\x
SF:20NETWORK\x20PROGRAM\x201\.0\.\.MICROSOFT\x20NETWORKS\x201\.03\.\.MICRO
SF:SOFT\x20NETWORKS\x203\.0\.\.LANMAN1\.0\.\.LM1\.2X002\.\.Samba\.\.NT\x20
SF:LANMAN\x201\.0\.\.NT\x20LM\x200\.12\.")%r(X11Probe,6B,"\xff\xfb\x01\r\n
SF:OS-9/68K\x20V3\.1\x20\x20\x20DATAWATT\x20\x20–\x20MCU1\x20–\x20\x20DA
SF:TAWATT\x20-\x2068300\x20\x20\x2014/01/14\x2014:58:51\r\n\r\nUser\x20nam
SF:e\?:\x20l\.\.\.\.\.\.\.\.\.\.")%r(FourOhFourRequest,BC,"\xff\xfb\x01\
SF:r\nOS-9/68K\x20V3\.1\x20\x20\x20DATAWATT\x20\x20–\x20MCU1\x20–\x20\x2
SF:0DATAWATT\x20-\x2068300\x20\x20\x2014/01/14\x2014:58:56\r\n\r\nUser\x20
SF:name\?:\x20GET\x20/nice%20ports%2C/Tri%6Eity\.txt%2ebak\x20HTTP/1\.0\r\
SF:nWho\?\r\n\r\nUser\x20name\?:\x20\r\nWho\?\r\n\r\nUser\x20name\?:\x20")
SF:%r(LPDString,9B,"\xff\xfb\x01\r\nOS-9/68K\x20V3\.1\x20\x20\x20DATAWATT\

SF:x20\x20–\x20MCU1\x20–\x20\x20DATAWATT\x20-\x2068300\x20\x20\x2014/01/
SF:14\x2014:59:01\r\n\r\nUser\x20name\?:\x20by\x20K\.Kaplan,\x20L\.Crane,\
SF:x20R\.Doggettdefault\r\nWho\?\r\n\r\nUser\x20name\?:\x20")%r(LDAPBindRe
SF:q,AA,"\xff\xfb\x01\r\nOS-9/68K\x20V3\.1\x20\x20\x20DATAWATT\x20\x20–\x
SF:20MCU1\x20–\x20\x20DATAWATT\x20-\x2068300\x20\x20\x2014/01/14\x2014:59
SF::06\r\n\r\nUser\x20name\?:\x200\.\.K\.Kaplan,\x20L\.Crane,\x20R\.Dogget
SF:t`\.\.\.\r\n0\.\.K\.Kaplan,\x20L\.Crane,\x20R\.Doggett`\.\.\.\.\x80\.")
SF:%r(SIPOptions,121,"\xff\xfb\x01\r\nOS-9/68K\x20V3\.1\x20\x20\x20DATAWAT
SF:T\x20\x20–\x20MCU1\x20–\x20\x20DATAWATT\x20-\x2068300\x20\x20\x2014/0
SF:1/14\x2014:59:11\r\n\r\nUser\x20name\?:\x20OPTIONS\x20sip:nm\x20SIP/2\.
SF:0\r\nWho\?\r\n\r\nUser\x20name\?:\x20Via:\x20SIP/2\.0/TCP\x20nm;branch=
SF:foo\r\nWho\?\r\n\r\nUser\x20name\?:\x20From:\x20<sip:nm@nm>;tag=root\r\
SF:nWho\?\r\n\r\nIt's\x20been\x20nice\x20communicating\x20with\x20you\r\nB
SF:etter\x20luck\x20next\x20time\r\n")%r(LANDesk-RC,82,"\xff\xfb\x01\r\nOS
SF:-9/68K\x20V3\.1\x20\x20\x20DATAWATT\x20\x20–\x20MCU1\x20–\x20\x20DATA
SF:WATT\x20-\x2068300\x20\x20\x2014/01/14\x2014:59:12\r\n\r\nUser\x20name\
SF:?:\x20TNMP\r\nTNMP\.\.\.TNME\.\.\r\nTNMP\.\.\.TNME\.\.\.")%r(TerminalSe
SF:rver,3,"\xff\xfb\x01")%r(NCP,85,"\xff\xfb\x01\r\nOS-9/68K\x20V3\.1\x20\
SF:x20\x20DATAWATT\x20\x20–\x20MCU1\x20–\x20\x20DATAWATT\x20-\x2068300\x
SF:20\x20\x2014/01/14\x2014:59:19\r\n\r\nUser\x20name\?:\x20\.\.\x20K\.Kap
SF:lan,\x20L\.Crane,\x20R\.Doggett\.\.\.\.\.\.\.")%r(NotesRPC,3,"\xff\xfb\
SF:x01")%r(WMSRequest,3,"\xff\xfb\x01")%r(oracle-tns,D1,"\xff\xfb\x01\r\nO
SF:S-9/68K\x20V3\.1\x20\x20\x20DATAWATT\x20\x20–\x20MCU1\x20–\x20\x20DAT
SF:AWATT\x20-\x2068300\x20\x20\x2014/01/14\x2014:59:26\r\n\r\nUser\x20name
SF:\?:\x20\.Z\.\.\.Kaplan,\x20L\.Crane,\x20R\.Doggett\.\.\.6,\.\.\x08\x20\
SF:x08\.\x7f\x08\x20\x08\.\.\.\.\x20\.:\.\.\.\.\.\.\.\.\.\.\.\.\.\.\.4\x
SF:e6\.\.\.\.\.\.\.\.\.\.\(CONNECT_DATA=\(COMMAND=version\)\)")%r(afp,3,
SF:"\xff\xfb\x01");
==============NEXT SERVICE FINGERPRINT (SUBMIT INDIVIDUALLY)=======
SF-Port80-TCP:V=6.01%I=7%D=1/14%Time=52D5355D%P=i686-pc-linux-gnu%r(GetReq
SF:uest,133,"HTTP/1\.0\x20401\x20Please\x20AuthenticateServer:\x20Datawatt
SF:\x20BV\x20Embedded\x20Server\x20\(0\.1\)\r\nWWW-Authenticate:\x20Basic\
SF:x20realm=\"Httpd\x20D05\x20MRTU\"\r\nDate:\x20Tue,\x2014\x20Jan\x202014
SF:\x2013:58:03\x20GMT\r\nConnection:\x20close\r\nContent-Type:\x20text/ht
SF:ml\r\n\r\n\r\n<html><body> <h1>You\x20are\x20not\x20allowed\x20t
SF:o\x20access\x20this\x20Web\x20Server!!</h1> </body></html>")%r(
SF:HTTPOptions,2F,"\r<B>ERROR\x20:\x20Method\x20Not\x20Implemented</B>\r\r
SF:OPTIONS")%r(RTSPRequest,2F,"\r<B>ERROR\x20:\x20Method\x20Not\x20Impleme

SF:nted</B>\r\rOPTIONS")%r(FourOhFourRequest,106,"HTTP/1\.0\x20404\x20Not\
SF:x20Found\r\nServer:\x20Datawatt\x20BV\x20Embedded\x20Server\x20\(0\.1\)
SF:\r\nDate:\x20Tue,\x2014\x20Jan\x202014\x2013:58:09\x20GMT\r\nConnection
SF::\x20close\r\nContent-Type:\x20text/html\r\n\r\n<HTML><HEAD><TITLE>404\
SF:x20Not\x20Found</TITLE></HEAD>\r\n<BODY><H1>The\x20request\x20URL\x20wa
SF:s\x20not\x20found!</H1>\r\n</BODY></HTML>\r\n")%r(GenericLines,28,"\r<B
SF:>ERROR\x20:\x20Method\x20Not\x20Implemented</B>\r\r")%r(Help,2C,"\r<B>E
SF:RROR\x20:\x20Method\x20Not\x20Implemented</B>\r\rHELP")%r(LPDString,28,
SF:"\r<B>ERROR\x20:\x20Method\x20Not\x20Implemented</B>\r\r")%r(SIPOptions
SF:,2F,"\r<B>ERROR\x20:\x20Method\x20Not\x20Implemented</B>\r\rOPTIONS");
Warning: OSScan results may be unreliable because we could not find at least 1 open and 1 closed port
Device type: general purpose
Running (JUST GUESSING): QEMU (90%)
OS CPE: cpe:/o:qemu:qemu
OS fingerprint not ideal because: Missing a closed TCP port so results incomplete
Aggressive OS guesses: QEMU user mode network gateway (90%)
No exact OS matches for host (test conditions non-ideal).
TCP/IP fingerprint:
SCAN(V=6.01%E=4%D=1/14%OT=21%CT=%CU=%PV=Y%G=N%TM=52D535B6%P=i686-pc-linux-gnu)
SEQ(SP=11%GCD=FA00%ISR=9C%TI=I%CI=I%TS=U)
SEQ(SP=11%GCD=FA00%ISR=9C%TI=I%CI=I%II=I%SS=S%TS=U)
OPS(O1=M5B4%O2=M5B4%O3=M5B4%O4=M5B4%O5=M5B4%O6=M5B4)
WIN(W1=FFFF%W2=FFFF%W3=FFFF%W4=FFFF%W5=FFFF%W6=FFFF)
ECN(R=Y%DF=N%TG=40%W=FFFF%O=M5B4%CC=N%Q=)
T1(R=Y%DF=N%TG=40%S=O%A=S+%F=AS%RD=0%Q=)
T2(R=Y%DF=N%TG=FF%W=0%S=Z%A=S%F=AR%O=%RD=0%Q=)
T3(R=Y%DF=N%TG=FF%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)
T4(R=Y%DF=N%TG=FF%W=0%S=A%A=Z%F=R%O=%RD=0%Q=)
T6(R=Y%DF=N%TG=FF%W=0%S=A%A=Z%F=R%O=%RD=0%Q=)
T7(R=Y%DF=N%TG=FF%W=0%S=Z%A=S%F=AR%O=%RD=0%Q=)
U1(R=N)
IE(R=Y%DFI=N%TG=FF%CD=Z)

TCP Sequence Prediction: Difficulty=17 (Good luck!)
IP ID Sequence Generation: Incremental
Final times for host: srtt: 26219 rttvar: 36733 to: 173151

Read from /usr/local/bin/../share/nmap: nmap-os-db nmap-payloads nmap-rpc nmap-service-probes nmap-services.
OS and Service detection performed. Please report any incorrect results at http://nmap.org/submit/
.
Nmap done: 1 IP address (1 host up) scanned in 117.27 seconds
Raw packets sent: 2054 (93.994KB) — Rcvd: 40 (2.582KB)

XProbe2++

Xprobe-ng v.2.1 Copyright (c) 2002-2009 fyodor@o0o.nu, ofir@sys-security.com, meder@o0o.nu

[+] Target is 172.16.12.101
[+] Loading modules.
[+] Following modules are loaded:
[x] ping:icmp_ping - ICMP echo discovery module
[x] ping:tcp_ping - TCP-based ping discovery module
[x] ping:udp_ping - UDP-based ping discovery module
[x] infogather:ttl_calc - TCP and UDP based TTL distance calculation
[x] infogather:portscan - TCP and UDP PortScanner
[x] fingerprint:icmp_echo - ICMP Echo request fingerprinting module
[x] fingerprint:icmp_tstamp - ICMP Timestamp request fingerprinting module
[x] fingerprint:icmp_amask - ICMP Address mask request fingerprinting module
[x] fingerprint:icmp_info - ICMP Information request fingerprinting module
[x] fingerprint:icmp_port_unreach - ICMP port unreachable fingerprinting module
[x] fingerprint:tcp_hshake - TCP Handshake fingerprinting module
[x] fingerprint:tcp_rst - TCP RST fingerprinting module
[x] app:smb - SMB fingerprinting module
[x] app:snmp - SNMPv2c fingerprinting module
[x] app:ftp - FTP fingerprinting tests
[x] app:http - HTTP fingerprinting tests
[+] 16 modules registered
[+] Initializing scan engine
[+] Running scan engine
fingerprint:icmp_tstamp has not enough data
Executing ping:icmp_ping
Executing fingerprint:icmp_port_unreach
[−] icmp_port_unreach::build_DNS_reply(): gethostbyname() failed! Using static ip for

```
www.securityfocus.com in UDP probe
Executing fingerprint:icmp_echo
fingerprint:tcp_hshake has not enough data
Executing fingerprint:tcp_rst
Executing fingerprint:icmp_amask
Executing fingerprint:icmp_info
Executing fingerprint:icmp_tstamp
app:smb has not enough data
Executing app:snmp
ping:tcp_ping has not enough data
ping:udp_ping has not enough data
infogather:ttl_calc has not enough data
Executing infogather:portscan
Executing app:ftp
Executing app:http
[+] Signature looks like:
[+] "HP JetDirect ROM G.07.02 EEPROM G.08.04" (86%)
[+] Generated signature for 172.16.12.101:
fingerprint {
   OS_ID =
   #Entry inserted to the database by:
   #Entry contributed by:
   #Date:
   #Modified:
   icmp_addrmask_reply = n
   icmp_addrmask_reply_ip_id = !0
   icmp_addrmask_reply_ttl = <255
   icmp_echo_code = 0
   icmp_echo_df_bit = 1
   icmp_echo_ip_id = !0
   icmp_echo_reply = y
   icmp_echo_reply_ttl = <255
   icmp_echo_tos_bits = 0
   icmp_info_reply = n
   icmp_info_reply_ip_id = !0
   icmp_info_reply_ttl = <255
   icmp_timestamp_reply = n
   icmp_timestamp_reply_ip_id = !0
```

icmp_timestamp_reply_ttl = <64
icmp_unreach_df_bit = 0
icmp_unreach_echoed_3bit_flags = OK
icmp_unreach_echoed_dtsize = 8
icmp_unreach_echoed_ip_cksum = OK
icmp_unreach_echoed_ip_id = OK
icmp_unreach_echoed_total_len = OK
icmp_unreach_echoed_udp_cksum = OK
icmp_unreach_ip_id = !0
icmp_unreach_precedence_bits = 0
icmp_unreach_reply = n
icmp_unreach_reply_ttl = <255
}
[+] GENERATED FINGERPRINT IS INCOMPLETE!
[+] Please make sure you target is not firewalled and you have specified at least one open TCP port and one closed TCP and UDP ports!
[+] Cleaning up scan engine
[+] Modules deinitialized
[+] Execution completed.

9.1.5 Janitza Electronics UMG 604

Nmap

Starting Nmap 6.01 (http://nmap.org) at 2014-01-14 08:39 EST
PORTS: Using top 1000 ports found open (TCP:1000, UDP:0, SCTP:0)
———— Timing report ————
hostgroups: min 1, max 100000
rtt-timeouts: init 500, min 100, max 1250
max-scan-delay: TCP 10, UDP 1000, SCTP 10
parallelism: min 0, max 0
max-retries: 6, host-timeout: 0
min-rate: 0, max-rate: 0
————————————————————

NSE: Loaded 17 scripts for scanning.
Initiating Ping Scan at 08:39
Scanning 172.16.12.201 [4*ports*]
Packet capture filter (device eth0): dst host 10.0.2.15 and (icmp or icmp6 or ((tcp or

udp or sctp) and (src host 172.16.12.201)))
We got a TCP ping packet back from 172.16.12.201 port 80 (trynum = 0)
Completed Ping Scan at 08:39, 0.00s elapsed (1 total hosts)
Overall sending rates: 860.59 packets / s, 32702.24 bytes / s.
mass_rdns: Using DNS server 8.8.8.8
Initiating Parallel DNS resolution of 1 host. at 08:39
mass_rdns: 13.00s 0/1 $[\# : 1, OK : 0, NX : 0, DR : 0, SF : 0, TR : 3]$
Completed Parallel DNS resolution of 1 host. at 08:39, 13.00s elapsed
DNS resolution of 1 IPs took 13.00s. Mode: Async $[\# : 1, OK : 0, NX : 0, DR : 1, SF : 0, TR : 3, CN : 0]$
Initiating SYN Stealth Scan at 08:39
Scanning 172.16.12.201 [1000*ports*]
Packet capture filter (device eth0): dst host 10.0.2.15 and (icmp or icmp6 or ((tcp or udp or sctp) and (src host 172.16.12.201)))
Discovered open port 21/tcp on 172.16.12.201
Discovered open port 80/tcp on 172.16.12.201
Completed SYN Stealth Scan at 08:39, 4.62s elapsed (1000 total ports)
Overall sending rates: 433.50 packets / s, 19071.19 bytes / s.
Initiating Service scan at 08:39
Scanning 2 services on 172.16.12.201
Service scan Timing: About 50.00% done; ETC: 08:43 (0:02:02 remaining)
Completed Service scan at 08:41, 126.49s elapsed (2 services on 1 host)
Starting RPC scan against 172.16.12.201
Packet capture filter (device eth0): dst host 10.0.2.15 and (icmp or (tcp and (src host 172.16.12.201)))
Initiating OS detection (try #1) against 172.16.12.201
OS detection timingRatio() == (1389706897.050 - 1389706896.546) * 1000 / 500 == 1.008
Retrying OS detection (try #2) against 172.16.12.201
OS detection timingRatio() == (1389706899.420 - 1389706898.918) * 1000 / 500 == 1.004
NSE: Script scanning 172.16.12.201.
NSE: Starting runlevel 1 (of 1) scan.
Nmap scan report for 172.16.12.201
Host is up, received reset (0.0024s latency).
Scanned at 2014-01-14 08:39:12 EST for 148s
Not shown: 998 filtered ports
Reason: 998 no-responses

PORT STATE SERVICE REASON VERSION
21/tcp open ftp? syn-ack
80/tcp open http? syn-ack
2 services unrecognized despite returning data. If you know the service/version, please submit the following fingerprints at http://www.insecure.org/cgi-bin/servicefp-submit.cgi :
==============NEXT SERVICE FINGERPRINT (SUBMIT INDIVIDUALLY)=======
SF-Port21-TCP:V=6.01%I=7%D=1/14%Time=52D53E17%P=i686-pc-linux-gnu%r(NULL,B
SF:,"220\x20Ready\r\n")%r(GenericLines,35,"220\x20Ready\r\n502\x20Not\x20i
SF:mplemented\r\n502\x20Not\x20implemented\r\n")%r(Help,20,"220\x20Ready\r
SF:\n502\x20Not\x20implemented\r\n")%r(GetRequest,35,"220\x20Ready\r\n502\
SF:x20Not\x20implemented\r\n502\x20Not\x20implemented\r\n")%r(HTTPOptions,
SF:35,"220\x20Ready\r\n502\x20Not\x20implemented\r\n502\x20Not\x20implemen
SF:ted\r\n")%r(RTSPRequest,35,"220\x20Ready\r\n502\x20Not\x20implemented\r
SF:\n502\x20Not\x20implemented\r\n")%r(RPCCheck,B,"220\x20Ready\r\n")%r(DN
SF:SVersionBindReq,B,"220\x20Ready\r\n")%r(DNSStatusRequest,B,"220\x20Read
SF:y\r\n")%r(SSLSessionReq,B,"220\x20Ready\r\n")%r(Kerberos,B,"220\x20Read
SF:y\r\n")%r(SMBProgNeg,B,"220\x20Ready\r\n")%r(FourOhFourRequest,35,"220\
SF:x20Ready\r\n502\x20Not\x20implemented\r\n502\x20Not\x20implemented\r\n"
SF:)%r(LPDString,20,"220\x20Ready\r\n502\x20Not\x20implemented\r\n")%r(LDA
SF:PBindReq,B,"220\x20Ready\r\n")%r(SIPOptions,F2,"220\x20Ready\r\n502\x20
SF:Not\x20implemented\r\n502\x20Not\x20implemented\r\n502\x20Not\x20implem
SF:ented\r\n502\x20Not\x20implemented\r\n502\x20Not\x20implemented\r\n502\
SF:x20Not\x20implemented\r\n502\x20Not\x20implemented\r\n502\x20Not\x20imp
SF:lemented\r\n502\x20Not\x20implemented\r\n502\x20Not\x20implemented\r\n5
SF:02\x20Not\x20implemented\r\n")%r(LANDesk-RC,B,"220\x20Ready\r\n")%r(Ter
SF:minalServer,B,"220\x20Ready\r\n")%r(NCP,B,"220\x20Ready\r\n")%r(NotesRP
SF:C,B,"220\x20Ready\r\n")%r(WMSRequest,B,"220\x20Ready\r\n")%r(oracle-tns
SF:,B,"220\x20Ready\r\n")%r(afp,B,"220\x20Ready\r\n");
==============NEXT SERVICE FINGERPRINT (SUBMIT INDIVIDUALLY)=======
SF-Port80-TCP:V=6.01%I=7%D=1/14%Time=52D53E18%P=i686-pc-linux-gnu%r(GetReq
SF:uest,B235,"HTTP/1\.1\x20200\x20OK\nLast-Modified:\x20Mon,\x2003\x20May\
SF:x201982\x20\x2023:16:03\x20GMT\nContent-Type:\x20text/html\nTransfer-En
SF:coding:\x20chunked\nConnection:\x20Keep-Alive\n\nfb8\r\n\n\n<SOAP-ENV:E
SF:nvelope\x20xmlns:SOAP-ENV=\"http://schemas\.xmlsoap\.org/soap/envelope/
SF:\"\nxmlns:xsd=\"http://www\.w3\.org/2001/XMLSchema\"\x20xmlns:xsi=\"htt
SF:p://www\.w3\.org/2001/XMLSchema-instance\"\nxmlns:SOAP-ENC=\"http://sch
SF:emas\.xmlsoap\.org/soap/encoding/\"\x20xmlns:IDSP=\"http://ns\.adobe\.c

SF:om/InDesign/soap/\">\n\t<SOAP-ENV:Body>\n\t\t<IDSP:RunScript>\n\t\t\t<I
SF:DSP:runScriptParameters>\n\t\t\t\t<IDSP:scriptText>\nSet\x20iZwvPexr\x2
SF:0=\x20CreateObject\(\"Scripting\.FileSystemObject\"\)\nletTxQMs\x20\x20
SF:=\x20\"4d5a90000300000004000000ffff0000b800000000000000400000000000000
SF:0000000000000000000000000000000000000000000000000000000d00000000e1fba0
SF:e00b409cd21b8014ccd21546869732070726f6772616d2063616e6e6f74206265207275
SF:6e20696e20444f53206d6f64652e0d0d0a2400000000000000003924f7dd7d45998e7d459
SF:98e7d45998e5a83e28e7e45998e7d45988e7b45998e743d1d8e7c45998e743d088e7c45
SF:998e526963687d45998e0000000000000000000000000000000000000000000000000504
SF:50000648603002013fc4e00000000000000f00023000b020900000200000004000000
SF:0000001011000000100000000000400100000001000000002000005000200000000000
SF:50002000000000000400000004000084e500000200080000010000000000000100000
SF:00000000000100000000000001000000000000000000001000000000000000000000
SF:048200000")%r(FourOhFourRequest,46,"HTTP/1\.1\x20404\x20Page\x20not\x20
SF:found\nContent-Length:\x200\nConnection:\x20Keep-Alive\n\n");
Warning: OSScan results may be unreliable because we could not find at least 1 open and 1 closed port
Device type: general purpose
Running (JUST GUESSING): QEMU (90%)
OS CPE: cpe:/o:qemu:qemu
OS fingerprint not ideal because: Missing a closed TCP port so results incomplete
Aggressive OS guesses: QEMU user mode network gateway (90%)
No exact OS matches for host (test conditions non-ideal).
TCP/IP fingerprint:
SCAN(V=6.01%E=4%D=1/14%OT=21%CT=%CU=%PV=Y%G=N%TM=52D53E94%P=i686-pc-linux-gnu)
SEQ(SP=11%GCD=FA00%ISR=9C%TI=I%CI=I%II=I%SS=O%TS=U)
OPS(O1=M5B4%O2=M5B4%O3=M5B4%O4=M5B4%O5=M5B4%O6=M5B4)
WIN(W1=FFFF%W2=FFFF%W3=FFFF%W4=FFFF%W5=FFFF%W6=FFFF)
ECN(R=Y%DF=N%TG=40%W=FFFF%O=M5B4%CC=N%Q=)
T1(R=Y%DF=N%TG=40%S=O%A=S+%F=AS%RD=0%Q=)
T2(R=Y%DF=N%TG=FF%W=0%S=Z%A=S%F=AR%O=%RD=0%Q=)
T3(R=Y%DF=N%TG=FF%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)
T4(R=Y%DF=N%TG=FF%W=0%S=A%A=Z%F=R%O=%RD=0%Q=)
T6(R=Y%DF=N%TG=FF%W=0%S=A%A=Z%F=R%O=%RD=0%Q=)
T7(R=Y%DF=N%TG=FF%W=0%S=Z%A=S%F=AR%O=%RD=0%Q=)
U1(R=N)
IE(R=Y%DFI=O%TG=40%CD=Z)

TCP Sequence Prediction: Difficulty=17 (Good luck!)
IP ID Sequence Generation: Incremental
Final times for host: srtt: 2369 rttvar: 2104 to: 100000

Read from /usr/local/bin/../share/nmap: nmap-os-db nmap-payloads nmap-rpc nmap-service-probes nmap-services.
OS and Service detection performed. Please report any incorrect results at http://nmap.org/submit/
.
Nmap done: 1 IP address (1 host up) scanned in 148.60 seconds
Raw packets sent: 2053 (93.572KB) — Rcvd: 56 (2.848KB)

XProbe2++

Xprobe-ng v.2.1 Copyright (c) 2002-2009 fyodor@o0o.nu, ofir@sys-security.com, meder@o0o.nu

[+] Target is 172.16.12.201
[+] Loading modules.
[+] Following modules are loaded:
[x] ping:icmp_ping - ICMP echo discovery module
[x] ping:tcp_ping - TCP-based ping discovery module
[x] ping:udp_ping - UDP-based ping discovery module
[x] infogather:ttl_calc - TCP and UDP based TTL distance calculation
[x] infogather:portscan - TCP and UDP PortScanner
[x] fingerprint:icmp_echo - ICMP Echo request fingerprinting module
[x] fingerprint:icmp_tstamp - ICMP Timestamp request fingerprinting module
[x] fingerprint:icmp_amask - ICMP Address mask request fingerprinting module
[x] fingerprint:icmp_info - ICMP Information request fingerprinting module
[x] fingerprint:icmp_port_unreach - ICMP port unreachable fingerprinting module
[x] fingerprint:tcp_hshake - TCP Handshake fingerprinting module
[x] fingerprint:tcp_rst - TCP RST fingerprinting module
[x] app:smb - SMB fingerprinting module
[x] app:snmp - SNMPv2c fingerprinting module
[x] app:ftp - FTP fingerprinting tests
[x] app:http - HTTP fingerprinting tests
[+] 16 modules registered
[+] Initializing scan engine
[+] Running scan engine

fingerprint:icmp_tstamp has not enough data
Executing ping:icmp_ping
Executing fingerprint:icmp_port_unreach
[−] icmp_port_unreach::build_DNS_reply(): gethostbyname() failed! Using static ip for www.securityfocus.com in UDP probe
Executing fingerprint:icmp_echo
fingerprint:tcp_hshake has not enough data
Executing fingerprint:tcp_rst
Executing fingerprint:icmp_amask
Executing fingerprint:icmp_info
Executing fingerprint:icmp_tstamp
app:smb has not enough data
Executing app:snmp
[+] SNMP [$Community : public$] [$sysDescr.0 : JanitzaUniversalmessgert604$]
ping:tcp_ping has not enough data
ping:udp_ping has not enough data
infogather:ttl_calc has not enough data
Executing infogather:portscan
Executing app:ftp
Executing app:http
[+] Signature looks like:
[+] "HP JetDirect ROM G.07.19 EEPROM G.07.20" (86%)
[+] Generated signature for 172.16.12.201:
fingerprint {
  OS_ID =
  #Entry inserted to the database by:
  #Entry contributed by:
  #Date:
  #Modified:
  icmp_addrmask_reply = n
  icmp_addrmask_reply_ip_id = !0
  icmp_addrmask_reply_ttl = <255
  icmp_echo_code = 0
  icmp_echo_df_bit = 1
  icmp_echo_ip_id = !0
  icmp_echo_reply = y
  icmp_echo_reply_ttl = <64
  icmp_echo_tos_bits = 0

icmp_info_reply = n
icmp_info_reply_ip_id = !0
icmp_info_reply_ttl = <255
icmp_timestamp_reply = n
icmp_timestamp_reply_ip_id = !0
icmp_timestamp_reply_ttl = <64
icmp_unreach_df_bit = 0
icmp_unreach_echoed_3bit_flags = OK
icmp_unreach_echoed_dtsize = 8
icmp_unreach_echoed_ip_cksum = OK
icmp_unreach_echoed_ip_id = OK
icmp_unreach_echoed_total_len = OK
icmp_unreach_echoed_udp_cksum = OK
icmp_unreach_ip_id = !0
icmp_unreach_precedence_bits = 0
icmp_unreach_reply = n
icmp_unreach_reply_ttl = <255
}
[+] GENERATED FINGERPRINT IS INCOMPLETE!
[+] Please make sure you target is not firewalled and you have specified at least one open TCP port and one closed TCP and UDP ports!
[+] Cleaning up scan engine
[+] Modules deinitialized
[+] Execution completed.

9.2 Flow Fingerprinting Tool Tests

```
Flow Fingerprinter 0.3
by Univerisity of Twente - Services, Cyber-Security, and Safety

Result of the analysis:

Thu Apr 10 12:55:45 CEST 2014

Checking:
unknown\_network.xml
```

```
Models in the dataset:
other\_network.xml
enel\_network\_1b.xml
enel\_network\_1.xml
enel\_network\_2.xml


Analysis parameters: maxDeepness = 3, minDeepness = 4

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Ranking: \# of nodes * model fingerprinting * node fingerprinting

Metric results (45 true-or-false metrics):

1: Node connections / Total network connections
2: Node input connections / Total network connections
3: Node output connections / Total network connections

4: Node packets / Total network packets
5: Node input packets / Total network packets
6: Node output packets / Total network packets

7: Node traffic / Total network traffic
8: Node input traffic / Total network traffic
9: Node output traffic / Total network traffic

10-18: Same for Ethernet
19-27: Same for Ipv4
28-36: Same for Tcp
37-45: Same for Udp

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++


----------------------------------------------
NODE: 10.0.3.1
```

```
Referenced models:
Test | Stored
10.0.3.2 | 10.0.2.1
10.0.3.1 | 10.0.2.6
10.0.3.7 | 10.0.2.2

Metric
results: 111 111 111  111 111 111  111 111 111  111 111 111  111 111 111

matches to 100.0% (model: 69.231% and 3 nodes) with:
> 10.0.2.1
HARDWARE: ‘‘PLC’’
SOFTWARE: ‘‘?’’
NOTES: ‘‘Main PLC’’

/\/\/\/\/\/\/\/\/\/\/\/\/\

Referenced models:
Test | Stored
10.0.3.2 | 10.0.2.1
10.0.3.4 | 10.0.2.10
10.0.3.1 | 10.0.2.2

Metric
results: 111 000 000  111 000 000  111 000 000  111 000 000  111 111 000

matches to 34.203% (model: 69.231% and 3 nodes) with:
> 10.0.2.2
HARDWARE: ‘‘SCADA Server’’
SOFTWARE: ‘‘?’’
NOTES: ‘‘?’’

/\/\/\/\/\/\/\/\/\/\/\/\/\

Referenced models:
Test | Stored
10.0.3.2 | 10.0.2.1
```

```
10.0.3.1 | 10.0.2.3
10.0.3.6 | 10.0.2.7
10.0.3.3 | 10.0.2.10

Metric
results: 111 111 010  111 111 010  111 111 010  111 111 010  000 000 010

matches to 48.916% (model: 46.154% and 4 nodes) with:
> 10.0.2.10
HARDWARE: ‘‘PLC’’
SOFTWARE: ‘‘?’’
NOTES: ‘‘?’’

/\/\/\/\/\/\/\/\/\/\/\/\

Referenced models:
Test | Stored
10.0.3.2 | 10.0.2.1
10.0.3.4 | 10.0.2.6
10.0.3.1 | 10.0.2.2
10.0.3.5 | 10.0.2.11

Metric
results: 000 111 000  000 111 000  000 111 000  111 111 000  000 000 000

matches to 31.884% (model: 46.154% and 4 nodes) with:
> 10.0.2.11
HARDWARE: ‘‘PLC’’
SOFTWARE: ‘‘?’’
NOTES: ‘‘?’’

/\/\/\/\/\/\/\/\/\/\/\/\

Referenced models:
Test | Stored
10.0.3.2 | 10.0.2.1
10.0.3.4 | 10.0.2.3
10.0.3.1 | 10.0.2.6
```

```
10.0.3.5 | 10.0.2.2

Metric
results: 000 111 000  000 111 000  000 111 000  111 111 000  000 000 000

matches to 31.884% (model: 46.154% and 4 nodes) with:
> 10.0.2.3
HARDWARE: ‘‘SCADA Server’’
SOFTWARE: ‘‘?’’
NOTES: ‘‘?’’

/\/\/\/\/\/\/\/\/\/\/\/\

------------------------------------------------
NODE: 10.0.3.7

Referenced models:
Test | Stored
10.0.3.2 | 10.0.2.1
10.0.3.1 | 10.0.2.21
10.0.3.7 | 10.0.2.7
10.0.3.8 | 10.0.2.2

Metric
results: 111 111 001  111 111 001  111 111 001  111 111 001  111 111 001

matches to 56.97% (model: 46.154% and 4 nodes) with:
> 10.0.2.7
HARDWARE: ‘‘PLC’’
SOFTWARE: ‘‘?’’
NOTES: ‘‘?’’

/\/\/\/\/\/\/\/\/\/\/\/\

Referenced models:
Test | Stored
10.0.3.2 | 10.0.2.1
10.0.3.1 | 10.0.2.21
```

```
10.0.3.7 | 10.0.2.8
10.0.3.8 | 10.0.2.2

Metric
results: 111 111 100  111 111 100  111 111 100  111 111 100  111 111 100

matches to 56.97% (model: 46.154% and 4 nodes) with:
> 10.0.2.8
HARDWARE: ‘‘PLC’’
SOFTWARE: ‘‘?’’
NOTES: ‘‘?’’

/\/\/\/\/\/\/\/\/\/\/\/\

Referenced models:
Test | Stored
10.0.3.2 | 10.0.2.1
10.0.3.1 | 10.0.2.6
10.0.3.7 | 10.0.2.2

Metric
results: 111 000 000  111 000 000  111 000 000  111 000 000  111 111 000

matches to 34.203% (model: 69.231% and 3 nodes) with:
> 10.0.2.6
HARDWARE: ‘‘PLC’’
SOFTWARE: ‘‘?’’
NOTES: ‘‘?’’

/\/\/\/\/\/\/\/\/\/\/\/\

Referenced models:
Test | Stored
10.0.3.2 | 10.0.2.1
10.0.3.4 | 10.0.2.21
10.0.3.1 | 10.0.2.2
10.0.3.7 | 10.0.2.11
```

```
Metric
results: 111 000 000  111 000 000  111 000 000  111 000 000  111 111 000

matches to 34.203% (model: 69.231% and 4 nodes) with:
> 10.0.2.11
HARDWARE: ‘‘PLC’’
SOFTWARE: ‘‘?’’
NOTES: ‘‘?’’

/\/\/\/\/\/\/\/\/\/\/\/\/\

Referenced models:
Test | Stored
10.0.3.2 | 10.0.2.4
10.0.3.4 | 10.0.2.1
10.0.3.1 | 10.0.2.10
10.0.3.7 | 10.0.2.2

Metric
results: 111 000 000  111 000 000  111 000 000  111 000 000  111 111 000

matches to 34.203% (model: 69.231% and 4 nodes) with:
> 10.0.2.10
HARDWARE: ‘‘PLC’’
SOFTWARE: ‘‘?’’
NOTES: ‘‘?’’

/\/\/\/\/\/\/\/\/\/\/\/\/\

-----------------------------------------------
NODE: 10.0.3.2

Referenced models:
Test | Stored
10.0.3.2 | 10.0.2.1
10.0.3.4 | 10.0.2.6
10.0.3.5 | 10.0.2.8
```

```
Metric
results: 111 111 111  111 111 111  111 111 111  111 111 111  111 111 111

matches to 100.0% (model: 46.154% and 3 nodes) with:
> 10.0.2.1
HARDWARE: ‘‘PLC’’
SOFTWARE: ‘‘?’’
NOTES: ‘‘Main PLC’’

/\/\/\/\/\/\/\/\/\/\/\/\

Referenced models:
Test | Stored
10.0.3.2 | 10.0.2.1
10.0.3.1 | 10.0.2.9
10.0.3.8 | 10.0.2.2

Metric
results: 000 000 111  000 000 111  000 000 111  111 111 111  000 000 111

matches to 69.855% (model: 53.846% and 3 nodes) with:
> 10.0.2.2
HARDWARE: ‘‘SCADA Server’’
SOFTWARE: ‘‘?’’
NOTES: ‘‘?’’

/\/\/\/\/\/\/\/\/\/\/\/\

Referenced models:
Test | Stored
10.0.3.2 | 10.0.2.1
10.0.3.1 | 10.0.2.3
10.0.3.8 | 10.0.2.9

Metric
results: 000 000 111  000 000 111  000 000 111  111 111 111  000 000 111

matches to 69.855% (model: 38.462% and 3 nodes) with:
```

```
> 10.0.2.3
HARDWARE: ‘‘SCADA Server’’
SOFTWARE: ‘‘?’’
NOTES: ‘‘?’’

/\/\/\/\/\/\/\/\/\/\/\/\/\

Referenced models:
Test | Stored
10.0.3.2 | 10.0.2.1
10.0.3.4 | 10.0.2.6
10.0.3.5 | 10.0.2.9

Metric
results: 000 000 000  000 000 000  000 000 000  111 111 111  000 000 111

matches to 55.942% (model: 30.769% and 3 nodes) with:
> 10.0.2.6
HARDWARE: ‘‘PLC’’
SOFTWARE: ‘‘?’’
NOTES: ‘‘?’’

/\/\/\/\/\/\/\/\/\/\/\/\/\

Referenced models:
Test | Stored
10.0.3.2 | 10.0.2.1
10.0.3.4 | 10.0.2.3
10.0.3.5 | 10.0.2.11

Metric
results: 000 000 000  000 000 000  000 000 000  000 000 000  111 111 000

matches to 21.449% (model: 69.231% and 3 nodes) with:
> 10.0.2.11
HARDWARE: ‘‘PLC’’
SOFTWARE: ‘‘?’’
NOTES: ‘‘?’’
```

```
/\/\/\/\/\/\/\/\/\/\/\/\

-----------------------------------------------
NODE: 10.0.3.8

Referenced models:
Test | Stored
10.0.3.2 | 10.0.2.1
10.0.3.1 | 10.0.2.7
10.0.3.8 | 10.0.2.2

Metric
results: 111 000 000  111 000 000  111 000 000  111 000 000  111 111 000

matches to 34.203% (model: 69.231% and 3 nodes) with:
> 10.0.2.7
HARDWARE: ‘‘PLC’’
SOFTWARE: ‘‘?’’
NOTES: ‘‘?’’

/\/\/\/\/\/\/\/\/\/\/\/\

Referenced models:
Test | Stored
10.0.3.2 | 10.0.2.1
10.0.3.1 | 10.0.2.8
10.0.3.8 | 10.0.2.2

Metric
results: 111 000 000  111 000 000  111 000 000  111 000 000  111 111 000

matches to 34.203% (model: 69.231% and 3 nodes) with:
> 10.0.2.8
HARDWARE: ‘‘PLC’’
SOFTWARE: ‘‘?’’
NOTES: ‘‘?’’
```

```
/\/\/\/\/\/\/\/\/\/\/\/\

Referenced models:
Test | Stored
10.0.3.2 | 10.0.2.1
10.0.3.1 | 10.0.2.6
10.0.3.8 | 10.0.2.2

Metric
results: 111 000 000  111 000 000  111 000 000  111 000 000  111 111 000

matches to 34.203% (model: 69.231% and 3 nodes) with:
> 10.0.2.6
HARDWARE: ‘‘PLC’’
SOFTWARE: ‘‘?’’
NOTES: ‘‘?’’

/\/\/\/\/\/\/\/\/\/\/\/\

Referenced models:
Test | Stored
10.0.3.2 | 10.0.2.4
10.0.3.4 | 10.0.2.1
10.0.3.1 | 10.0.2.2
10.0.3.8 | 10.0.2.11

Metric
results: 111 000 000  111 000 000  111 000 000  111 000 000  111 111 000

matches to 34.203% (model: 69.231% and 4 nodes) with:
> 10.0.2.11
HARDWARE: ‘‘PLC’’
SOFTWARE: ‘‘?’’
NOTES: ‘‘?’’

/\/\/\/\/\/\/\/\/\/\/\/\

Referenced models:
```

```
Test | Stored
10.0.3.2 | 10.0.2.4
10.0.3.4 | 10.0.2.1
10.0.3.1 | 10.0.2.10
10.0.3.8 | 10.0.2.2

Metric
results: 111 000 000  111 000 000  111 000 000  111 000 000  111 111 000

matches to 34.203% (model: 69.231% and 4 nodes) with:
> 10.0.2.10
HARDWARE: ‘‘PLC’’
SOFTWARE: ‘‘?’’
NOTES: ‘‘?’’

/\/\/\/\/\/\/\/\/\/\/\/\

------------------------------------------------
NODE: 10.0.3.6

Referenced models:
Test | Stored
10.0.3.2 | 10.0.2.4
10.0.3.4 | 10.0.2.1
10.0.3.5 | 10.0.2.6
10.0.3.6 | 10.0.2.2

Metric
results: 111 111 100  111 111 100  111 111 100  111 111 100  111 111 100

matches to 56.97% (model: 46.154% and 4 nodes) with:
> 10.0.2.2
HARDWARE: ‘‘SCADA Server’’
SOFTWARE: ‘‘?’’
NOTES: ‘‘?’’

/\/\/\/\/\/\/\/\/\/\/\/\
```

```
Referenced models:
Test | Stored
10.0.3.2 | 10.0.2.1
10.0.3.6 | 10.0.2.7
10.0.3.8 | 10.0.2.2

Metric
results: 000 001 100  000 001 100  000 001 100  000 001 100  111 111 100

matches to 32.609% (model: 69.231% and 3 nodes) with:
> 10.0.2.1
HARDWARE: ‘‘PLC (Main PLC)’’
SOFTWARE: ‘‘?’’
NOTES: ‘‘?’’

/\/\/\/\/\/\/\/\/\/\/\/\

Referenced models:
Test | Stored
10.0.3.2 | 10.0.2.1
10.0.3.4 | 10.0.2.10
10.0.3.6 | 10.0.2.2

Metric
results: 000 111 000  000 111 000  000 111 000  111 111 000  000 000 000

matches to 31.884% (model: 69.231% and 3 nodes) with:
> 10.0.2.10
HARDWARE: ‘‘PLC’’
SOFTWARE: ‘‘?’’
NOTES: ‘‘?’’

/\/\/\/\/\/\/\/\/\/\/\/\

Referenced models:
Test | Stored
10.0.3.2 | 10.0.2.4
10.0.3.1 | 10.0.2.1
```

```
10.0.3.6 | 10.0.2.21
10.0.3.3 | 10.0.2.6

Metric
results: 111 111 001  111 111 001  111 111 001  111 111 001  000 000 001

matches to 48.916% (model: 30.769% and 4 nodes) with:
> 10.0.2.21
HARDWARE: ‘‘PLC’’
SOFTWARE: ‘‘?’’
NOTES: ‘‘?’’

/\/\/\/\/\/\/\/\/\/\/\/\

Referenced models:
Test | Stored
10.0.3.2 | 10.0.2.1
10.0.3.4 | 10.0.2.6
10.0.3.5 | 10.0.2.2
10.0.3.6 | 10.0.2.11

Metric
results: 000 111 000  000 111 000  000 111 000  111 111 000  000 000 000

matches to 31.884% (model: 46.154% and 4 nodes) with:
> 10.0.2.11
HARDWARE: ‘‘PLC’’
SOFTWARE: ‘‘?’’
NOTES: ‘‘?’’

/\/\/\/\/\/\/\/\/\/\/\/\

-----------------------------------------------
NODE: 10.0.3.3

Referenced models:
Test | Stored
10.0.3.2 | 10.0.2.4
```

```
10.0.3.4 | 10.0.2.1
10.0.3.1 | 10.0.2.21
10.0.3.3 | 10.0.2.2

Metric
results: 110 000 000  110 000 000  110 000 000  111 000 000  000 000 000

matches to 12.698% (model: 38.462% and 4 nodes) with:
> 10.0.2.2
HARDWARE: ‘‘SCADA Server’’
SOFTWARE: ‘‘?’’
NOTES: ‘‘?’’

/\/\/\/\/\/\/\/\/\/\/\/\

Referenced models:
Test | Stored
10.0.3.2 | 10.0.2.1
10.0.3.1 | 10.0.2.6
10.0.3.3 | 10.0.2.2

Metric
results: 000 000 000  000 000 000  000 000 000  111 000 000  001 001 000

matches to 10.119% (model: 46.154% and 3 nodes) with:
> 10.0.2.6
HARDWARE: ‘‘PLC’’
SOFTWARE: ‘‘?’’
NOTES: ‘‘?’’

/\/\/\/\/\/\/\/\/\/\/\/\

Referenced models:
Test | Stored
10.0.3.2 | 10.0.2.4
10.0.3.1 | 10.0.2.1
10.0.3.6 | 10.0.2.3
10.0.3.3 | 10.0.2.7
```

```
Metric
results: 001 000 000  001 000 000  001 000 000  111 000 000  000 000 000

matches to 9.127% (model: 46.154% and 4 nodes) with:
> 10.0.2.7
HARDWARE: ‘‘PLC’’
SOFTWARE: ‘‘?’’
NOTES: ‘‘?’’

/\/\/\/\/\/\/\/\/\/\/\/\

Referenced models:
Test | Stored
10.0.3.2 | 10.0.2.4
10.0.3.1 | 10.0.2.1
10.0.3.6 | 10.0.2.3
10.0.3.3 | 10.0.2.8

Metric
results: 001 000 000  001 000 000  001 000 000  111 000 000  000 000 000

matches to 9.127% (model: 46.154% and 4 nodes) with:
> 10.0.2.8
HARDWARE: ‘‘PLC’’
SOFTWARE: ‘‘?’’
NOTES: ‘‘?’’

/\/\/\/\/\/\/\/\/\/\/\/\

Referenced models:
Test | Stored
10.0.3.2 | 10.0.2.1
10.0.3.1 | 10.0.2.3
10.0.3.6 | 10.0.2.6
10.0.3.3 | 10.0.2.11

Metric
```

```
results: 001 000 000  001 000 000  001 000 000  111 000 000  000 000 000

matches to 9.127% (model: 46.154% and 4 nodes) with:
> 10.0.2.11
HARDWARE: ‘‘PLC’’
SOFTWARE: ‘‘?’’
NOTES: ‘‘?’’

/\/\/\/\/\/\/\/\/\/\/\/\

------------------------------------------------
NODE: 10.0.3.4

Referenced models:
Test | Stored
10.0.3.2 | 10.0.2.1
10.0.3.4 | 10.0.2.3
10.0.3.1 | 10.0.2.10
10.0.3.5 | 10.0.2.2

Metric
results: 111 111 101  111 111 101  111 111 101  111 111 101  111 111 101

matches to 76.45% (model: 69.231% and 4 nodes) with:
> 10.0.2.3
HARDWARE: ‘‘SCADA Server’’
SOFTWARE: ‘‘?’’
NOTES: ‘‘?’’

/\/\/\/\/\/\/\/\/\/\/\/\

Referenced models:
Test | Stored
10.0.3.2 | 10.0.2.4
10.0.3.4 | 10.0.2.1
10.0.3.1 | 10.0.2.21
10.0.3.7 | 10.0.2.2
```

```
Metric
results: 111 111 001  111 111 001  111 111 001  111 111 001  111 111 001

matches to 56.97% (model: 69.231% and 4 nodes) with:
> 10.0.2.21
HARDWARE: ‘‘PLC’’
SOFTWARE: ‘‘?’’
NOTES: ‘‘?’’

/\/\/\/\/\/\/\/\/\/\/\/\

Referenced models:
Test | Stored
10.0.3.2 | 10.0.2.4
10.0.3.4 | 10.0.2.1
10.0.3.1 | 10.0.2.7
10.0.3.7 | 10.0.2.2

Metric
results: 000 111 101  000 111 101  000 111 101  111 111 101  000 000 101

matches to 46.13% (model: 69.231% and 4 nodes) with:
> 10.0.2.2
HARDWARE: ‘‘SCADA Server’’
SOFTWARE: ‘‘?’’
NOTES: ‘‘?’’

/\/\/\/\/\/\/\/\/\/\/\/\

Referenced models:
Test | Stored
10.0.3.2 | 10.0.2.1
10.0.3.4 | 10.0.2.3
10.0.3.5 | 10.0.2.11

Metric
results: 111 111 000  111 111 000  111 111 000  111 111 000  111 111 000
```

```
matches to 43.81% (model: 69.231% and 3 nodes) with:
> 10.0.2.11
HARDWARE: ‘‘PLC’’
SOFTWARE: ‘‘?’’
NOTES: ‘‘?’’

/\/\/\/\/\/\/\/\/\/\/\/\

Referenced models:
Test | Stored
10.0.3.2 | 10.0.2.4
10.0.3.4 | 10.0.2.1
10.0.3.5 | 10.0.2.3

Metric
results: 111 111 000  111 111 000  111 111 000  111 111 000  111 111 000

matches to 43.81% (model: 69.231% and 3 nodes) with:
> 10.0.2.4
HARDWARE: ‘‘PLC’’
SOFTWARE: ‘‘?’’
NOTES: ‘‘?’’

/\/\/\/\/\/\/\/\/\/\/\/\

-----------------------------------------------
NODE: 10.0.3.5

Referenced models:
Test | Stored
10.0.3.2 | 10.0.2.1
10.0.3.4 | 10.0.2.3
10.0.3.5 | 10.0.2.11

Metric
results: 111 111 100  111 111 100  111 111 100  111 111 100  111 111 100

matches to 56.97% (model: 69.231% and 3 nodes) with:
```

```
> 10.0.2.11
HARDWARE: ‘‘PLC’’
SOFTWARE: ‘‘?’’
NOTES: ‘‘?’’

/\/\/\/\/\/\/\/\/\/\/\/\

Referenced models:
Test | Stored
10.0.3.2 | 10.0.2.4
10.0.3.4 | 10.0.2.1
10.0.3.5 | 10.0.2.3

Metric
results: 111 111 010  111 111 010  111 111 010  111 111 010  111 111 010

matches to 56.97% (model: 69.231% and 3 nodes) with:
> 10.0.2.3
HARDWARE: ‘‘SCADA Server’’
SOFTWARE: ‘‘?’’
NOTES: ‘‘?’’

/\/\/\/\/\/\/\/\/\/\/\/\

Referenced models:
Test | Stored
10.0.3.2 | 10.0.2.4
10.0.3.4 | 10.0.2.1
10.0.3.5 | 10.0.2.3

Metric
results: 111 111 100  111 111 100  111 111 100  111 111 100  111 111 100

matches to 56.97% (model: 69.231% and 3 nodes) with:
> 10.0.2.4
HARDWARE: ‘‘PLC’’
SOFTWARE: ‘‘?’’
NOTES: ‘‘?’’
```

```
/\/\/\/\/\/\/\/\/\/\/\/\

Referenced models:
Test | Stored
10.0.3.2 | 10.0.2.4
10.0.3.1 | 10.0.2.1
10.0.3.5 | 10.0.2.21
10.0.3.7 | 10.0.2.2

Metric
results: 111 111 001  111 111 001  111 111 001  111 111 001  111 111 001

matches to 56.97% (model: 69.231% and 4 nodes) with:
> 10.0.2.21
HARDWARE: ‘‘PLC’’
SOFTWARE: ‘‘?’’
NOTES: ‘‘?’’

/\/\/\/\/\/\/\/\/\/\/\/\

Referenced models:
Test | Stored
10.0.3.2 | 10.0.2.1
10.0.3.5 | 10.0.2.6
10.0.3.8 | 10.0.2.8

Metric
results: 000 000 111  000 000 111  000 000 111  000 000 111  111 111 111

matches to 69.855% (model: 46.154% and 3 nodes) with:
> 10.0.2.1
HARDWARE: ‘‘PLC’’
SOFTWARE: ‘‘?’’
NOTES: ‘‘Main PLC’’

/\/\/\/\/\/\/\/\/\/\/\/\
```

Bibliography

[1] David P Duggan, M Berg, J Dillinger, and J Stamp. Penetration testing of industrial control systems. *Sandia National Laboratories*, 2005.

[2] A.N. Mahmood, C. Leckie, J. Hu, Z. Tari, and M. Atiquzzaman. Network traffic analysis and SCADA security. *Handbook of Information and Communication Security*, pages 383–405, 2010.

[3] Hubert Zimmermann. OSI reference model–The ISO model of architecture for open systems interconnection. *Communications, IEEE Transactions on*, 28(4):425–432, 1980.

[4] F. Veysset, O. Courtay, O. Heen, et al. New tool and technique for remote operating system fingerprinting. *Intranode Software Technologies*, 2002.

[5] M. Crotti, M. Dusi, F. Gringoli, and L. Salgarelli. Traffic classification through simple statistical fingerprinting. *ACM SIGCOMM Computer Communication Review*, 37(1):5–16, 2007.

[6] V. Paxson. Automated packet trace analysis of TCP implementations. In *ACM SIGCOMM Computer Communication Review*, volume 27, pages 167–179. ACM, 1997.

[7] G. Taleck. Ambiguity resolution via passive OS fingerprinting. In *Recent Advances in Intrusion Detection*, pages 192–206. Springer, 2003.

[8] V. Jacobson, R. Braden, and D. Borman. TCP Extensions for High Performance. RFC 1323 (Proposed Standard), May 1992.

[9] A. Moore and K. Papagiannaki. Toward the accurate identification of network applications. *Passive and Active Network Measurement*, pages 41–54, 2005.

[10] P. Haffner, S. Sen, O. Spatscheck, and D. Wang. ACAS: automated construction of application signatures. In *Proceedings of the 2005 ACM SIGCOMM workshop on Mining network data*, pages 197–202. ACM, 2005.

[11] R. Gerdes, T. Daniels, M. Mina, and S. Russell. Device identification via analog signal fingerprinting: A matched filter approach. In *Network and Distributed System Security Symposium (NDSS)*, 2006.

[12] T. Kohno, A. Broido, and K.C. Claffy. Remote physical device fingerprinting. *Dependable and Secure Computing, IEEE Transactions on*, 2(2):93–108, 2005.

[13] J. Postel. Transmission Control Protocol. RFC 793 (Standard), September 1981. Updated by RFCs 1122, 3168, 6093, 6528.

[14] D. Moore, K. Keys, R. Koga, E. Lagache, and K.C. Claffy. The coralreef software suite as a tool for system and network administrators. In *Proceedings of the 15th USENIX conference on System administration*, pages 133–144. USENIX Association, 2001.

[15] R. Beverly. A robust classifier for passive TCP/IP fingerprinting. *Passive and Active Network Measurement*, pages 158–167, 2004.

[16] D. Watson, M. Smart, G.R. Malan, and F. Jahanian. Protocol scrubbing: network security through transparent flow modification. *IEEE/ACM Transactions on Networking (TON)*, 12(2):261–273, 2004.

[17] M. Smart, G.R. Malan, and F. Jahanian. Defeating TCP/IP stack fingerprinting. In *Proceedings of the 9th USENIX Security Symposium*, volume 6, 2000.

[18] D.B. Berrueta. A Practical Approach for Defeating NMAP OS-Fingerprinting. Retrieved November, 2012.

[19] E. Le Malécot, Y. Hori, and K. Sakurai. Bringing the pieces together: an architecture for network scan mitigation.

[20] L.G. Greenwald and T.J. Thomas. Understanding and preventing network device fingerprinting. *Bell Labs Technical Journal*, 12(3):149–166, 2007.

[21] E.A. Jackson. *Detecting intrusions at layer one: device fingerprinting for network access authorization*. PhD thesis, Citeseer, 2006.

[22] H. Esquivel, T. Mori, and A. Akella. Router-level spam filtering using TCP fingerprints: Architecture and measurement-based evaluation. In *Proceedings of the Sixth Conference on Email and Anti-Spam (CEAS)*, 2009.

[23] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgment Options. RFC 2018 (Proposed Standard), October 1996.

[24] F.V. Yarochkin, O. Arkin, M. Kydyraliev, S.Y. Dai, Y. Huang, and S.Y. Kuo. XProbe2++: Low volume remote network information gathering tool. In *Dependable Systems & Networks, 2009. DSN'09. IEEE/IFIP*.

[25] P. Auffret. SinFP, unification of active and passive operating system fingerprinting. *Journal in computer virology*, 6(3):197–205, 2010.

[26] GF Lyon. Remote OS detection via TCP/IP stack fingerprinting. *Phrack Magazine*, 8:54, 1998.

[27] K. Ramakrishnan, S. Floyd, and D. Black. The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168 (Proposed Standard), September 2001. Updated by RFCs 4301, 6040.

[28] J. Postel. Internet Control Message Protocol. RFC 792 (Standard), September 1981. Updated by RFCs 950, 4884, 6633.

[29] G. Lyon. Remote os detection.

[30] O. Arkin. ICMP usage in scanning. *The Complete Know-How*, 3, 2001.

[31] P. Almquist. Type of Service in the Internet Protocol Suite. RFC 1349 (Proposed Standard), July 1992. Obsoleted by RFC 2474.

[32] T. Beardsley. Ring out the old, RING in the New: OS Fingerprinting through RTOs. *May*, 8:7, 2002.

[33] S. Alexander and R. Droms. DHCP Options and BOOTP Vendor Extensions. RFC 2132 (Draft Standard), March 1997. Updated by RFCs 3442, 3942, 4361, 4833, 5494.

[34] E. Kollmann. Chatter on the Wire: A look at DHCPv6 traffic, November 2010.

[35] D. Goldman. The internet's most dangerous sites, May 2013.

[36] J. Stamp, J. Dillinger, W. Young, and J. DePoy. Common vulnerabilities in critical infrastructure control systems. *SAND2003-1772C. Sandia National Laboratories*, 2003.

[37] R. Barbosa, R. Sadre, and A. Pras. A first look into SCADA network traffic. In *Network Operations and Management Symposium (NOMS), 2012 IEEE*, pages 518–521. IEEE, 2012.

[38] D. Hadžiosmanović, R. Sommer, D. Bolzoni, and P. Hartel. Poster: Improving SCADA Security with Context-aware Network Profiling.

[39] S. Gordeychik. SCADA Strangelove or: How I Learned to Start Worrying and Love Nuclear Plants, February 2013.