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

D4.1 Device fingerprinting preliminary results

Contract No. FP7-SEC-285477-CRISALIS

Workpackage	WP 4 - System Discovery
Author	Marco Caselli, Frank Kargl, Dina Hadziosmanovic
Version	1.0
Date of delivery	M12
Actual Date of Delivery	M12
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. Corrado Leita
2229 Route des Cretes
06560 Sophia Antipolis
France

e-mail: corrado_leita@symantec.com
Phone: +33 673 41 91 27

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 (T5.2), 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.

The primary intention of this document is to provide a solid state of the art on fingerprinting mainly focusing on device fingerprinting. Also, we will outline a reference architecture for a fingerprinting tool and describe its modules. Finally, we will define important requirements and challenges when trying to transfer or extend fingerprinting in the industrial control systems (ICS) domain.

1 Introduction

In the ICT field, the fingerprinting is a set of activities that exploit different kinds of information in order to outline or describe devices, software and processes inside a computer network. This 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 hardware, operating systems, application software, and their respective versions or configurations of devices connected to a network. The aim of this activity is to better understand potential vulnerabilities in the network, to analyze attacks, and assess or estimate the risks of chain attacks to derive proper countermeasures.

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

Fingerprinting has an important advantage with respect to a manual information collection in a network management system. In fact, manually collected information are often outdated or otherwise incomplete or wrong. As fingerprinting approaches also exhibit miscategorizations, the two approaches should always be considered complementary and security mechanisms should rely on both information sources where available.

Fingerprinting can be divided into two main categories: active and passive. Active fingerprinting requires the use of tools that actively query the system to obtain a set of information needed to outline the fingerprint. In the second case, passive fingerprinting tries to acquire such information in a less intrusive way, listening to communication. 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 fingerprint implies collecting only the information that is available at that moment on the communication channel. However, performing active scans is not always possible. Moreover, an active operation is always noisier than the passive, thus more likely to be detected. As example, in SCADA systems (Supervisor Control And Data Acquisition), active fingerprinting is not desirable due to

potential computational overload of system's components. Programmable Logic Controllers (PLCs) and Remote Terminal Units (RTUs) are usually tuned on the processes they are controlling and cannot support the 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 the system component cannot satisfy expected requests [1].

Any kind of fingerprinting implies three three basics activites:

- Gathering
- Storing
- Dataset updating

The result of the fingerprinting analysis is mainly based on the type of data that we decide to collect. The most common way to fingerprint ICT systems implies the inspection of specific values in the TCP and IP headers. These are: Initial packet size, Initial TTL, Windows size, Max segment size, Windows scaling value, dont fragment flag, sackOK flag and nop flag. When aggregated, these elements form a 67-bits signature for each computer working in a standard network. Monitoring fields and values of different network protocols is also possible. Basically, we can cover all the ISO/OSI stack by starting from Application layer (Protocol-based fingerprinting) to the Physical one (Signal-based fingerprinting). Besides the ISO/OSI stack we have also the possibility to record completely different information. For example, in specific contexts it could be suitable to use data related to timing and communications patterns [2], [3].

Storing procedures deal mainly with the possibility to use such information in different ways and with numerous analysis software. Beyond this, specific data structures can be suitable just for few tools but, at the same time, be very efficient. This property can be exploited in situations in which the speed of the fingerprinting procedure is essential. For example, some fingerprinting tools relies 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 the success of the analysis. This is because properties of ICT hardware and software change quickly over time. This means that new information has to be added to databases each time unknown fingerprints are shown. This is difficult because: a) new fingerprints have to be verified on large data set and with different networks or systems configurations and b) we always have to pay attention not to create any overlap in the signatures as it could bring our fingerprinting tools in an incongruous state.

2 Reference Model

Due to the fact that specific fingerprinting targets differ in their characteristics and information sources there are various tool on fingerprinting techniques. In this chapter we focus on the standard way-of-working of fingerprinting applications and we analyze a reference architecture that is common to the most used fingerprinting tools.

2.1 TCP/IP fingerprinting

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

- **Initial packet size** (16 bits): the total size of TCP and IP headers. This size could change due to the variable length of the Options field of the two headers. This is the reason why fingerprinting tools are interested in the initial value.
- **Initial TTL** (8 bits): the "time to live" is a counter attached to IPv4 and IPv6 packets used to prevent a data 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, fingerprinting tools try to guess the initial value of the TTL by calculating the nearest power of two.
- **Window size** (16 bits): the total number of bytes that is allowed to transmit over a TCP connection without receiveng any acknowledgement
- **Max segment size** (16 bits): the largest amount of data that can be included in a single, unfragmented segment
- **Window scaling value** (8 bits): the parameter, specified in [4], manages the use of larger window sizes to let the TCP work efficiently with high-speed networks
- **"don't fragment" flag** (1 bits): by setting the "do not fragment" flag, an end-system can choose to prevent a IP packet from being fragmented by a router.
- **"sackOK" flag** (1 bits): by setting the "sackOK" bit, the TCP data receiver can selectively inform the sender about received segments and let retransmit only those that have actually been lost [5].

- **"nop" flag** (1 bits): the TCP "No Option" flag is used to separate the different options used within the TCP option field

Togheter these fields form a 67-bits signature for each computer working in a standard network.

Device fingerprinting strongly relies on this signature and several tools take advantage of it. Unfortunately, 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 add any knowledge of the previous fields and it is often discarded by fingerprinting tools.

2.2 Architecture

Despite different targets or information sources, fingerprinting activities have several common tasks. For this reason, it is possible to summarize such activities into few logical building blocks shared among all fingerprinting tools. Each building block receives as an input a result from another. A building block elaborates the provided data by adding new information or by checking the content. Finally the block forwards information or feedback to other components. A reference architecture for a fingerprinting tool is provided in Figure 2.1.

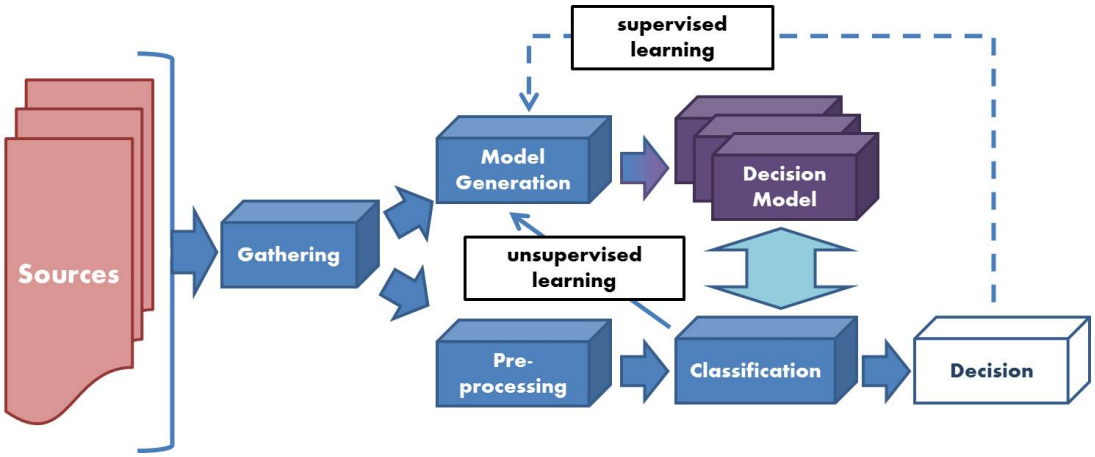

Figure 2.1: Fingerprinting tools' reference architecture

2.2.1 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 identify different subset of characteristics [6] [7]. Furthermore, some tools try to fingerprint different levels of the ISO/OSI stack. In this sense, Ethernet or application headers can be suitable information sources [8]. Finally, network topologies, time patterns, or also network ports usage can be potentially valuable sources as well [9].

2.2.2 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 while dropping the useless ones. Gathering module implementation differs according to the choice to do active or passive fingerprinting. Active tools query the system to obtain a set of information needed to outline the fingerprint. In the second case, passive fingerprinting tries to acquire such information in a less intrusive way, listening to communication. 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 fingerprint implies collecting only the information that is available at that moment on the communication channel. During the active fingerprinting, the Gathering module manages 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 analyses. 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.

2.2.3 Model Generation

The Model Generation component defines how the information is stored and organized. 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 [8] and [10]. 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 is a key element for fingerprinting reliability. For example, P0f provides a command for a signature collision check.

2.2.4 Decision Model

The 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 tool 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 [11]. The latter exploits two different datasets keeping active and passive signatures separated [12].

2.2.5 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 temporary Decision Model of the system under examination. In few cases, such as XProbe2++, the Pre-processing component selects one by one the information needed for the comparison.

2.2.6 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. Most advanced tools add to Classification modules awareness about fingerprinting mitigation techniques. This is used to detect information artificially manipulated against fingerprinting. In particular, a software called “scrubber” is often used to confuse fingerprinting. As explained in [13] a scrubber is an active mechanism that converts heterogeneous network flows into well-behaved flows that cannot be equivocally 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 problem [11].

2.2.7 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 for 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. Moreover, such analysis is important to evaluate further dependencies or information flows among the modules.

3 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 field concentrates on the fundamental topic analyzing new approaches to do fingerprinting. The second proposes new techinques to refine the fingerprints. In the third, papers focus on how the fingerprinting can be mitigated or prevented in order to keep a system secure from malicious scans. Finally the fourth field of research analyzes possible application for fingerprinting techniques (e.g. as a support for intrusion detection systems).

3.1 Information sources

The ICT 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 fact, 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 focus 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 from studies like [7]. In this paper, the authors present "tcpanaly", a tool for TCP implementations' analysis. Tcpanaly passively inspects TCP segments in order to exploit differences and misbehaviours and use them to distinguish specific implementations of operating systems. Altough outdated, this work shows how different the most famous operating systems behave in terms of TCP communications. Furthermore, the work argues how TCP headers are the most valuable sources for such kind of information gathering. In [6] the authors exploit some of these properties. In fact, they succeed in increasing the level of confidence in TCP/IP standard fingerprinting by looking at specific characteristic 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 the TCP/IP fingerprinting, the authors propose to analyze at the TCP Timestamp field and a particular behaviour of the window size

management. Despite the specification [4], the TCP timestamp is not always set to zero in some specific segments and this provides useful information to recognize operating systems. The windows size is analyzed in relation to the TCP Timestamp as, sometimes, such size is directly related to that value. As the authors conclude, this approach can be extended in many other directions by exploiting further ambiguities in the TCP protocol.

Several authors present works on exploiting implementation differences of specific protocols to define fingerprints. Works like [14] explain in detail how it is possible to build a higly-reliable classification technique based on payload packets inspection. Specifically, in this work, the authors concentrate on traffic flows more than packets. The use of traffic flows simplify the data processing and add the necessary knowledge about the context in which the systems they want to identify work. However, exploiting different identification methods, the authors succeed to develop a framework for traffic characterisation based on packets' payloads. Another practical work on the issue is [8] where 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 to define suitable signatures for ICT systems. In [15] authors show that Ethernet devices can be uniquely identified and tracked using as few as 25 Ethernet frames. This is obtained by analyzing variations in analog signals caused by hardware or manufacturing inconsistencies. In their work, the authors emphasize the fact that analog characteristics are difficult to forge with respect to the information provided by the other level of the ISO/OSI protocol stack. This approach exploits a matched filter to profile signals in order to create signature suitable to identify the device the signal originated from. This low-level analysis on signals can be combined with standard fingerprinting mechanisms. An alternative approach implies recording small deviations called clock skews in devices hardware [16]. The authors of the work claims that a fingerprinter can use the information contained within the 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. In the end, the authors show how this technique can be applied to different goals from simply fingerprinting devices such as a differentiation of honeynets and real networks or listing host behind a NAT.

Several approaches focus on exploiting alternative packet properties, such as timing. For example, in [2] authors present operating system detection method based on temporal response analysis. The tool they developed on this idea, called RING, is an alternative or a complementary instrument for fingerprinting with respect to standard tools. The main focus of the work is the process of packet retransmission implemented by the TCP.

This mechanism is not strictly specified in [17] thus, there are different implementations. By measuring the delays of the packet retransmissions as well as looking at specific flags, sequence numbers or acknowledge numbers, the authors claim that it is possible to get valuable information about the target behavior. Furthermore, the work is concluded emphasizing that some other TCP transitions can be suitable for temporal response analysis (e.g. the closure of the connection).

Some works explore the possibility to fingerprint communication protocols. This is the case of [3] that specifically introduces the concept of "protocol fingerprint". The starting point of this work is that it is possible 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 properties of the captured IP packets: the size, inter-arrival time and arrival order. The authors define a fingerprint describing a set Probability Density Functions (PDF) linked to a set of flows generated by the same known protocol. In order to classify unknown traffic flows they check if the behavior of the flow is statistically compatible with at least one of the described PDF. Proposed tests show promising results even if a larger set of protocols could lead to increase the margin of error.

Finally, there are fingerprinting approaches based on port scanning. Some works as [9] exploits the use of standard network ports by known services to recognize systems and applications. However, there are two common criticisms of this approach: the difficulty to check those ports behind NATs and other network components, and the increasing usage of port randomization by most of the applications.

3.2 Learning techniques

As discussed in the introduction, the creation and maintenance of the fingerprints is one of the major problem of fingerprinting. Machine learning is one of the solution used to improve these activities. The papers proposed in this section contribute to refine the "Model Generation" building block outlined in the previous chapter. Furthermore, some of the works try to exploit the Automatic Feedback technique in order to obtain a tool that progressively improve its performance.

To refine a fingerprinting tool several learning techniques, like Bayesian or neural networks, could be useful. In [10], for example, the authors use probabilistic learning to develop a Naïve Bayesian classifier to passively infer a hosts operating system from packet headers. The starting point of the work relies on the observation fields d_i (TTL,

Window SYN packet size, "do not fragment" bit) and the standard Bayesian formulation:

$$P(H_i|D) = \frac{\prod_j P(d_i|H_i)P(H_i)}{P(D)} \tag{3.1}$$

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 transiting into a web server correlating them with the information contained in their application headers about the operating systems involved. The proposed classifier has two main advantages. First, it computes the probabilistic weights based on the training set and this makes it robust to non-conventional TCP stack implementation. Secondly, such classifier maintains its confidence degree even in presence of incomplete or conflicting data. Despite that, the whole work has to be tested on larger dataset. It is worth noting that the authors succeeded in having a good result just using four TCP fields while standard fingerprinting relies on an exact match from an exhaustive list of TCP settings. Moreover, they were able also to obtain a continuous refinement on the degree of identification confidence without deep-packet inspection.

Application-fingerprinting also take advantage of such techniques. In [8], 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 while satisfying some constraints. The clusterization process starts with several assumptions. First of all, it is worth noting that the result of a comparison has a binary value. This means that a specific flow can be related to an application or be completely unrelated to it. The authors decide to use raw application level data instead of trying to derive some information (e.g. ASCII words of the stream). Moreover, 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.

In the results, this work shows how this approach is highly accurate and scales to allow online application identification on differnet kind of links. Among the three learning algorithms, AdaBoost performs the best in several tests showing sometimes a considerable better result than the Naïve Bayesian mechanism.

3.3 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 "scrubber" with the aim of confusing fingerprints. As explained in [13] a scrubber is an active mechanism that converts heterogeneous network flows into well-behaved flows that cannot be equivocally 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 the focus is on TCP traffic and, finally, on TCP/IP fingerprinting fields. Some examples of implemented scrubbing activites are: the reordering of TCP options, the modification of the original TCP sequence number and the 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 [18]. In this work, the authors discuss the position of the scrubber in the network identifing the gatway as the most suitable one. Then, they design and implement a scrubber that work at both the network and transport layers to convert ambiguous traffic. IP-level ambiguites arise mainly in IP header flags and fragment reassembly algorithms. For this reason, the scrubber modify the firsts recalculating the checksum and reassemble the datagram (to disassemble it again if needed). TCP-level ambiguities focus on the options. Again, the "scrubber" intervenes in reordering them or changingtheir values (if they are not rensposible of performance). This strategy let the authors organize heterogeneous communication into sanitized packets that do not reveal clues about the hosts' operating systems. Moreover, the results show good performance in terms of throughput and scalability.

Some solutions directly focus on specific tools instead of working on the general characteristics of a fingerprint. Once targeted a software, these techniques exploit a mitigation approach focused on confusing the fingerprinting activities that the program performs. This is the case of [19] where the target is Nmap. The paper describes different solutions to defeat the tool by behaving like another chosen operating system. There is a large list of practical modifications on standard operating systems to confuse Nmap. In Linux, the author suggests to use the netfilter module "IP personality" changing parameters like TCP Initial Sequence Number, TCP initial Window Size, etc. Moreover, there are interesting system patches like the "Stealth" provided by Security Technology that allow to masquered the implemented TCP/IP stack. BSD systems propose instead a patch

called Blackhole that implements special mechanisms to respond to TCP and UDP non-conventional traffic. This kind of customized solutions are usually effective as they are tuned to confuse any attempt at the analysis of a specific tool. However, to obtain that kind of behavior, these techniques cannot disregard a deep knowledge of the targeted software. Considering a wide range of 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 [20] the authors introduce an architecture that combines several techniques such as packet scrubbing, advanced packet matching and tarpitting. The proposed architecture is based on several modules that successively process all the traffic coming to and going out of the network. The latter is cleaned by a packet "scrubber". The incoming traffic is instead sequentially "scrubbed", black-or-white-listed (depending on the kind of connection) and finally filtered thorugh a static ruleset. The fact that all the 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 of obtaining accurate information with the remaining ones. Finally, 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 ICT fingerprinting. According to [21] it is possible to verify and assess such property. The work'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 combination of different defensive techniques like "scrubbing" and firewalling. The analysis details also the choice on lettin the previous two to work with specific kinds of traffic or specific header fields. In the results, the author show comparative tables that assess the efficacy of each configurations against different fingerprinting technologies and operating systems.

3.4 Applications

We divide IDS 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 behavior.

ICT 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, ICT 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 "Classification" module of the architecture outlined in the previous chapter can be a suitable feature inside intrusion detection systems or other security mechanisms.

The approach described in [6] uses the passively detected OS fingerprint of the end host to correctly resolve ambiguities between different network stack implementations. In this work, the authors provide to IDS sensors additional knowledge about the network stack implementation of the end host and thus improve 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 authors (Maximum Segment Size, Selective Acknoledgment, Timestamping, and Window Scaling) are combined togheter to populate the entries of both the tables. During runtime, TCP SYN and SYNACK values of TCP traffic seen by the IDS are kept in a cache. Once the IDS needs to perform a lookup for a particular IP address to determine the ype of operating system, it triggers the cache lookup mechanism that runs a best-match algorithm on all the cache entries. In the conclusion, the authors claim 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 describe a signal-level fingerprints and this element can be used to refine the analysis of an attack as showed in [22]. The author of the work starts with the assumption that, in some cases, none of the digital information can be trusted. Since that, it is possible to rely on some analog characteristics of the signals transmitted in the network. Looking at such signals, the author tries to discern among the "idealized" signal (the one that carries the information), device-specific variations, and the random noise. Once recognized, the device-specific variations create an analog-fingerprint controlled through apposite filters.

Finally, fingerprinting techniques are used to mitigate specific sets of cyber-attacks. This is the case of the work described in [23]. 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. In their work, they argue for using TCP headers to develop fingerprint signatures that can be used to identify spamming hosts based on the specific operating system and version from which the email is sent. An advantage of this mechanism is that it can be used in conjunction with a variety of other spam filtering techniques to ensure accurate spam detection. Anyway, the authors claim that using simple signatures (e.g. Windows-based signatures) largely fail. This

fact shows the need for highly accurate signature selection mechanisms to avoid the risk of misclassification.

3.5 Conclusions

In conclusion, there are several methods and strategies to implement a fingerprinting process. 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. They concentrate in few bytes distinct and very precise data about software running in the target machine. 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 Xprobe3's database). Moreover, the process used to compare unknown fingerprints with 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 resolve. Numerous attempts to solve the problem rely on automic learning techniques and try to derive new signatures from specific datasets. However, these results are less accurate than the standard process of comparison. Fingerprinting mitigation techniques try, instead, to disturb such comparison process. Modification and occultation of computer fingerprints are huge obstacles to the proper recognition of the target machine.

Alternative methods of fingerprinting 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 fingeprinting results and methodologies. In the most common case, fingerprinting techniques contribute to the successful functioning of intrusion detection systems. Also high-level security applications (e.g. anti-spam) can use at their advantage some fingeprinting features. Moreover, there is an extensive use of fingerprinting tools in activites such as vulnerability assessment and penetration testing.

4 Tools

Fingerprinting tools presented in this chapter implement some of the techniques and the concepts discussed in 3. 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

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 biggest improvement over other software used so far was the number of tests it could do. As a consequence, this set of tests significantly increased the number of different systems it could recognize [24].

During work, 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 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 involve the sending out 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 acknowledgement and sequence numbers are random, but saved for use during the analysis. Nmap checks seven fields of a standard TCP/IP fingerprints and perform several other tasks. This 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 be differentiated by thei order inside the packets.
- **TCP Sequence/Acknowledgement numbers**: 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**: this test is performend 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) [25] 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 field in the responses:

- **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.

- **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 [26].
- **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 structures 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 [27].

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" [28] project carried out by the Sys-Security Group. XProbe last release, XProbe2++, exploits different operating systems fingerprinting techinques like: fuzzy signature matching, probabilistic guesses, simultaneous multiple matching and a signature database for both TCP/IP and application protocols.

Using ICMP instead of TCP can have several advantages. First of all, it makes 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. A further 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 certain 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 has to 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, bits order is changed.
- **IP Fragmentation flags/offset**: as above some systems change the bits order.
- **IP header checksum**: it can be calculated incorrectly.
- **Type of Service (TOS)**: usually, in source quench messages it has to be equal the previous packet value while in the other cases it should be 6 or 7 [29]. 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 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 they influence each others result.

4.1.3 RING

Developed in 2002, RING implements the concept of time pattern fingerprinting. The authors of the tool published their results in [2] 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 [17] proposes a retransmission algorithm, such specification is not mandatory. For this reason many operating system implement some variations of the algorithms.

To have a chance to observe these patterns, RING forces the target to a non-standard communication, where timeouts values will be reached for sure. 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 retrasmission of the target. RING stores information also on the time elapsed until the a machine sends the RST packet. Since not all the operating systems implement the reset of the connection this is another valuable distinguishing element. Each signature is labelled with the related operating system and its version. From its documentation and the tests provided by [30] we know that RING's dataset contains several different 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 unneeded. With such configuration, tools like NMAP, do not work well since some of their tests are based on closed ports. Moreover, RING uses standard only standard TCP packets. This mechanism does not disturb the targeted machine or 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 Free BSD 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

This section describes the most relevant passive fingerprinting tools.

4.2.1 P0f

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 offer to applications running in the same system a simple way to connect to the tool and get information about the remote hosts they are talking to. This property can be useful for integrating it 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 packet, outgoing connection fingerprinting by checking SYN-ACK packets, refuses 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 much on the standard TCP/IP fingerprinting. Other than the common fields, the tool looks at:

- **Timestamp**: to be able to calculate a correct round-trip-time on specific link types it is necessary to include a timestamp in the TCP header. This timestamp can be zero or based on the systems up-time and so 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 acknowledgement [5]. However, not all the systems implement this functionality.

- **Unrecognize options**: sometimes specific options of the headers are not recognized. This lead most of the fingerprinting tools to misunderstand the packet. P0f3 succeed in exploiting this information for OS detection.

- **EOL option**: this test checks this option indicating that the end of the option list does not coincide with the end of the TCP header. Just few systems take advantage of such option.

- **TCP options' order**: order in a packet is heavily dependant on the implementation and it is a valuable information for fingerprinting.

- **Packet anomalies**: ad-hoc implementations of the 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.
  - **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 [4] the initial SYN packet should have a zeroed second timestamp.
  - **Unusual flags**: this set check the presence of unexpected extra-flags.

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**: since this packet is supposed to have a proper ACK number, P0f3 checks if it is zero anyway.
- **Non-zero second timestamp**: similar to the previous test, the tool checks if this value is set.

Connection refusing fingerprinting offers different hints. This is mainly due to two reasons. First, the connection is not established anyhow and strange flags do not have many consequences. Second, the freedom given by TCP specifications [17]. Among the characteristics controlled by P0f3 there are:

- **Connection-refused packet**: a normal connection-refused packet should only be sent when a connection is refused. However, there are rare cases where a system sends it in response to an unexpected ACK packet.
- **"ACK number is zero" error**: p0f detects if a system sends a connection-refused packets with an ACK number set to zero.
- **"Non-zero SEQ value error"**: this test checks if the SEQ value is different from zero.
- **Connection dropped**: when the RST flag is set with a non-zero sequence number the acknowledgement number should be zero. However, this is not mandatory and some systems behave differently.
- **"RST flag and SEQ value are zero" error**: this test checks if the two fields are coherent.

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 can also include several plugins for specific features. Among

them, there is "finger" which implements some passive operating system fingerprinting techniques.

The main functioning schema is to analyze the passive information coming from a host when it sets or accepts connections with 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.

The used database contains different fingerprints for each type of packet. According to its documentation, Ettercap relies more on SYN fingerprint because the SYN+ACK is always influenced by the SYN (e.g. if a SYN does not contain a SACK the SYN ACK will not have the SACK option even if supported). For this reason, if the tool receives a SYN+ACK, 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 near with each other. When a valuable 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, such last fingerprint is also the nearest one known by the tool.

With respect to the other tools, Ettercap is not focused on fingerprinting. For this reason it has limited possibilities to recognize complex systems. Furthermore, fingerprint database has not been updated since 2004.

4.3 Hybrid

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 develo 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 Addres Translation (NAT) and Port Address Translation (PAT) but also software implementing packet normalization techniques. Such situations

are becoming increasingly common on the Internet and SinFP wants to overcome this barrier by proposing new ways toward the OS fingerprinting.

It is worth noting that, despite is hybrid nature, SinFP3 uses the same dataset of signatures both for active and passive fingerprinting. Differently from the other, the tool 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 it is 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 fingerpriting 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 fragmented" bit while in the second we look at Flow Lable, 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 engine. The target is to find an intersection on multiple domains. The domaiins are defined by the header fields' 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 [12] that it is almost superfluous to add new signatures to the current version of the database. This feature actually solve one of the major problems of the fingerprinting. Moreover, such algorithm makes the tool highly resilent to signature modifications derived by intermediate routing or filtering devices.

5 SCADA fingerprinting

Fingerprinting of SCADA and ICS devices has not received significant attention by researchers yet. In the literature there are only very few examples of standard fingerprintig methodologies being applied in industrial environments. However, not all described concepts and solutions are suitable for SCADA environments. As explained in [1], 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. None of the tools described in the previous chapter provide such information per se. We have conducted some preliminary tests that show that, in a SCADA 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. There is also no strong evidence that standard TCP/IP fingerprinting methodologies would even work for PLCs and RTUs.

5.1 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

SCADA often monitor or control processes in systems where a failure may have disastrous consequences (or may be otherwise very undesirable). For this reason a generic active probing of systems (like scanning for open ports and then opening arbitrary TCP connections) should generally be avoided. As described in [1], in an SCADA 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

Devices heterogeneity is another challenge in SCADA fingerprinting. There are different physical processes for which SCADA 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, SCADA 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.

Proprietary protocols

The usage of proprietary protocols is an issue related to device heterogeneity. SCADA devices mainly implement vendor-specific application level protocols. Several of this 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) take 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 SCADA 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 outlined above. This fact severely reduces the applicability of fingerprinting based on TCP connection setup.

5.2 Opportunities

Long life-cycle of devices

SCADA systems are used since the '70s and several system operators do not continuously update their hardware and software components [31]. 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 SCADA networks [32]. Once established, the process control remains the same and components behave accordingly by continuously performing the same instructions and communications [33]. Fingerprinting can exploit this stability linking such patterns to component signatures. Standard networks do not allow the same. Personal computer's behavior depends on the user and is likely to change over time depending on his current needs (e.g. browsing or downloading)

Protocol specification

Some SCADA protocols, like Modbus and Profinet, provide a way to query components in order to have information about their hardware and software. As showed in [34] 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.

5.3 SCADA fingerprinting based on the Reference Model

To the best of our knowledge, only two SCADA fingerprinters exist (PLCscan and Modbuspatrol) [34]. 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 Sec. 5.2. A comprehensive approach to the development of SCADA fingerprinters has to face all the challenges outlined before. For this reason, we use the reference architecture defined in Sec. 2 to structure the discussion about SCADA fingerprinting and to describe the properties each module has to have if used in a industrial environment.

5.3.1 Sources

In SCADA environments there are different kind of information that can be exploited for fingerprinting. As stated in Sec. 3.1 TCP/IP protocol characteristics are the most valuable. 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 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 stated in Sec. 5.1, numerous proprietary protocols are being used in SCADA systems. Such issue 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 SCADA 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 SCADA networks' communication.

5.3.2 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 SCADA 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 SCADA 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 problem and allows a simpler and more 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-SCADA 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.

5.3.3 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 Sec. 2.1 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 SCADA infrastructure that cannot be reduced to a simple signature. Therefore, the use of such characteristics need the definition a broader concept as well as a comprehensive data structure that outlines architecture, properties, and trends of an SCADA infrastructure.

5.3.4 Decision Model

To solve this issue we introduce the concept of Context Model. The Context Model is an aggregator of information. It represents a comprehensive description of an 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. 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.

5.3.5 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 SCADA 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.

5.3.6 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 SCADA 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 SCADA fingerprinting is to recognize a component by describing the vendor, the hardware (e.g. device model), and its software. Other possible targets in SCADA 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 PLC coordinate with each other)
- gathering general information about the process (e.g. SCADA working on energy systems perform updates and send messages often than in water infrastructures)

5.3.7 Decision

The Decision is the output of the SCADA fingerprinter. Depending on to 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 toward 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 Final Remarks

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 have started making 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 Enel IdroLab facility without any comprehensive result confirming the threat. We are currently investigating 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. An extensive evaluation of this issue will be provided in D4.4.

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 2.

We are currently implementing a prototype of a "Flow Fingerprinting" based on the analyses 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.

In D4.4 we are planning to extend all the analyses and tests we have performed and reported in this document. We will explore the feasibility of developing a theoretical framework for implementing fingerprinting in SCADA and ICS environments. Finally, we will discuss Flow Fingerprinting as well as any other fingerprinting solution suitable for industrial systems.

Bibliography

[1] 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.

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

[3] 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.

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

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

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

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

[8] 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.

[9] 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.

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

[11] 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*.

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

[13] 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.

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

[15] 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.

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

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

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

[19] D.B. Berrueta. A practical approach for defeating nmap os-fingerprinting. retrieved november, 2012.

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

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

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

[23] 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.

[24] GF Lyon. Remote os detection via tcp/ip stack fingerprinting. *Phrack Magazine*, 8:54, 1998.

[25] 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.

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

[27] G. Lyon. Remote os detection.

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

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

[30] T. Beardsley. Ring out the old, ring in the new: Os fingerprinting through rtos. *May*, 8:7, 2002.

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

[32] 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.

[33] D. Hadziosmanovic, R. Sommer, D. Bolzoni, and P. Hartel. Poster: Improving scada security with context-aware network profiling.

[34] S. Gordeychik. Scada strangelove or: How i learned to start worrying and love nuclear plants, February 2013.