Host Fingerprinting for Web Servers Authentication

Ezio Lefons, Sebastiano Pizzutilo and Filippo Tangorra
Dipartmento di Informatica, Università degli Studi di Bari Aldo Moro, Via Orabona 4, 70125 Bari, Italy

Keywords: Authentication, Clock Skew, Database Security, Fingerprint.

Abstract: Fingerprinting is a biometric technique for computing a unique profile associated to a physical person for authentication purpose. It has been successfully applied also to software entities by using hash functions for integrity checking after downloading. In the paper, we propose a fingerprinting algorithm to identify a machine during a client-server authentication process. In detail, this host identifier can be used for connecting to a database server without using an account storing a plain-text password. After the presentation of experimental results, we show some real scenarios where this solution can be applied.

1 INTRODUCTION

Authentication aims at checking the identity of an individual, which may be a person or a software agent, trying to access a system. The main paradigms for authentication are: (a) *something you know*, based on passwords or secret phrases; (b) *something you have*, based on keys or badges, for example; and (c) *something you are*, based on biometrics techniques (Ben Natan, 2005; Bertino and Sandhu, 2005; Gertz and Jajodia, 2007).

A biometrics technique consists in a measurement of the physical and intrinsic characteristics of an individual, such as his/her fingerprints or iris and/or facial recognition, that allow to create a unique profile of that individual (Liu and Silverman, 2001; Jain et al., 2006; Tang, 2018; Sundararajan et al., 2019). A similar approach is used for creating a unique profile of a software entity on the basis of hash functions, such as MD5. This digest is mainly used for verifying the integrity of a software moving across a network (Deswarte et al., 2004). However, the digest identifies a *class* of software, and not a specific instance. Extending this concept to hardware devices (Alaca and van Oorschot, 2016), a host fingerprint involves a strategy for identifying a single computer running a set of applications (Veysset, 2002). In this case, we are interested in the identification of a single host that is part of a network.

A method for fingerprinting a hardware device is based on clock skew, which is the variation of the signals of the internal clock (Kohno et al., 2005). In this paper, we propose an algorithm for computing such fingerprint in order to identify a host in a client-server communication. In this case, the host to be identified is a computer running a web application server that connects to database server. The benefits of this approach is the possibility to access a database server not by using a password stored as a plain-text but by providing unique identifier computed at run-time.

The method present in literature for computing clock skews is discussed in Section 2, while in Section 3 we explain our proposal, along with experimental results. Section 4 shows some practical applications of our proposal in web-based environments. The final Section concludes the paper with a summary and outlines future work.

2 RELATED WORK

In (Mills, 1992), Mills defines the offset of two clocks as the time difference between them, while defines the skew as the frequency difference between them, computed as the first derivative of offset with time. To this end, a synchronization of the computer's clock with a NTP (Network Time Protocol) server is used for reducing, but not for eliminating, this skew.

A method for exploiting this clock skew for fingerprinting is explained in (Kohno et al., 2005). The authors compute the clock skew of a remote host by starting a network communication with it and compare the timestamps contained in the TCP

Lefons, E., Pizzutilo, S. and Tangorra, F.
Host Fingerprinting for Web Servers Authentication.
DOI: 10.5220/0008942500450051
In *Proceedings of the 6th International Conference on Information Systems Security and Privacy (ICISSP 2020)*, pages 45-51
ISBN: 978-989-758-399-5; ISSN: 2184-4356

header with those of the observer. The correlation between the TCP timestamps and the measured machine's time is based on a linear programming technique, using the Graham's convex hull algorithm, for variable network delay renders simple linear regression insufficient (Graham, 1972; Moon et al., 1999). For the experiment, the authors tested the algorithm using 69 machines in a campus computer laboratory and ran the measurement for 38 days, computing clock skews on 12 and 24 hours intervals. The experiment succeeded in demonstrating the validity of this approach, for the clock skew estimates for any given machine are approximately constant over time and different machines have detectably different clock skews.

The work of these authors has been commented by Fink (Fink, 2007) who, conversely, treats the problem as one of statistics and regression rather than linear programming and optimization. Though similar, the author proposes a solution for computing the sample size required to produce a clock skew that is within a fixed margin of the true population clock skew. The sample size formula has been further validated introducing network delays and analysing correlations with hardware characteristics.

Also (Polčák and Franková, 2014) explores remote computer identification based on the estimation of clock skew computed from network packets, but measurements were difficult to take, as they needed to analyze network traffic, and required an external reference time to compare with. Salo (Salo, 2007) proposed a solution to this problem by comparing two different clocks: the one used by the CPU and the independent one used to maintain the internal timer. The proposed methodology required a long execution time to generate fingerprint. In (Sanchez-Rola et al., 2018), authors look at code execution time as a way to precisely identify different devices, considering that the time that a computer spends to execute an instruction depends on how many clock cycles the instruction requires, and on the duration of each cycle.

3 METHOD PROPOSAL

Our underlying idea is to observe timestamps taken from the internal clock and computing the offset of a timestamp in reference to the previous one. The assumption is that, if the clock shows a constant skew, then a regular pattern in clock signals is present.

The main differences between this proposal and that present in literature are:

- The skew is locally computed. That is, it is not computed by a fingerprinter through a remote observation of the fingerprint. Conversely, it is computed by a server-side program, observing the timestamps of the internal clock of the host on which the program is running.
- The granularity of the timestamp is the microsecond, in order to bring out the subtle differences in clock signals. In our server-side implementation, we adopted the *microtime* PHP function (Sklar and Trachtenberg, 2003).
- The skew is quickly computed and does not require a long-running observation of the fingerprint. To do so, the algorithm uses a CPU-intensive cycle, which takes few seconds to be completed. The number of cycles is a parameter that has been tuned in an empirical way. We discuss about the number of CPU-cycles in the next implementation sub-section.
- The skew is not computed by comparing the timestamps with those derived from another source (*i.e.*, the observer or a NTP server) but it is self-referential. Therefore, the fingerprint is autonomously computed.

Let T_i be the timestamp observed at the i-th CPU-cycle, for $i = 0,\dots, n \in \mathbb{N}$.

We define

$$offset_i = T_i - T_{i-1},\ \text{for } i = 1,\dots, n, \text{ and}$$

$$O = \{\ offset_i : i \in \{1,\dots, n\} \text{ where } n \in \mathbb{N}\}.$$

Here, an offset is the incremental step in clock signals, that is, the difference between sequential timestamps. Of course, this step can present different values, for the observation process is strongly affected by other processes that may slow down the system and cause abnormal delays (*i.e.,* outliers) in observations. However, if we consider an arbitrarily high value for n, the incremental steps converge to a stable offset.

Let

$$D = \{x_i \mid x_i \in O \text{ and } x_i \neq x_j \text{ if } i \neq j,\ 1 \leq i, j \leq m,\ m \in \mathbb{N}\}$$

be the set of the distinct values of the observed offsets. Of course, $m \leq n$ because the cardinality of D is usually lower than that of O, for numerous repeated offset can be observed.

So, let $f(x_i)$ be the frequency of x_i or the number of times that x_i appears in O. Finally, given the host H, the fingerprint of H is defined as

$$fp_H = k, \text{ such that } k \in D \text{ and } f(k) = max(f(x_i)).$$

For the sake of simplicity, the fingerprint of H is that offset having the highest frequency.

It is worth noting that this approach does not preserve from typical attacks, such as *spoofing* for example, that usually affect also traditional systems. Our proposal is mainly devoted to avoid the attacks aiming at taking possession of a password stored in a plain-text. Indeed, using a fingerprint instead of a text makes it possible to identify a host on the basis of its own characteristic and use it as a run-time computed information.

3.1 Implementation of the Method

As a proof of concept, the source code of the algorithm is publicly available for testing, for it has been released on a repository of open source projects (see https://sourceforge.net/projects/webfingerprint).

The aim of the open source project is to launch a large-scale experimentation for verifying a fingerprint collision. In detail, we are interested in discovering whether two hosts may present the same fingerprint on a wide area network.

In the implementation solution, we introduced two loops: an external loop made by n cycles and an inner loop made by m rounds. This rationale is simulating a human-based sampling of the timestamps that takes a set of observations at regular intervals of time. In our case, after the end of the inner loop, we suspend the clock for s seconds. So, the number of CPU-cycles is given by $n \times m$. After several tries, we fixed $n = 1000$, $m = 10$, and $s = 0.1$ as a compromise between velocity and accuracy. We stress that a low value for the total CPU-cycles implies a large number of outliers. On the other hand, a high value implies a high computation time and a PHP timeout error. Indeed, a computation time that takes more than 5 seconds is not suitable in web-based environments.

3.2 Experiment

The aim of the experiment is twofold:

- verifying the *collision probability*, and
- verifying the *fingerprint stability*.

The *collision probability* is the probability that two hosts of the same network present the same fingerprint.

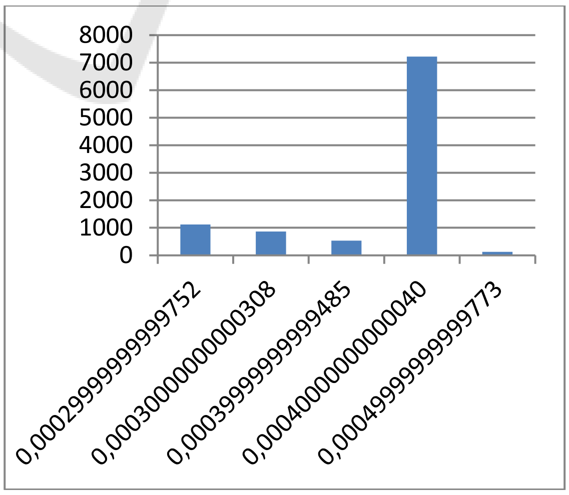
The *fingerprint stability* is the condition that a given fingerprint remains quite stable over time.

For verifying the *collision probability*, the experiment involved a small set of computers of a local network. Each host presents different hardware characteristics. In Figure 1, we report the result of the measurement (relative to 10000 timestamp observations that generate 9999 offsets) on the first host.

The graph shows that only 5 distinct values of offsets have been observed (outliers that do not exceed the threshold frequency of 1% are not considered). The most frequent value has almost the 73% frequency. Similarly, also the second host (see Figure 2) shows the presence of a unique offset having the highest frequency, almost equals to the 69%.

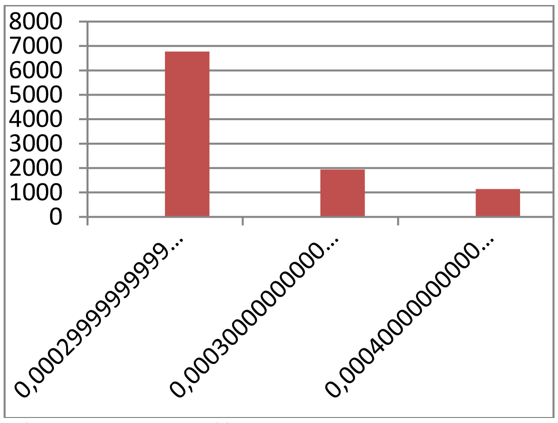
The offset with the highest frequency for the third host is a unique value inside the network and this value is almost equals to the 59% (see Figure 3).

At the end of this first part of the experimental phase, we conclude that the computed fingerprints are unique in the local network and, therefore, these can be safely considered as host identifiers. This measurement has been repeated several times for each host, and the fingerprints have been confirmed, though with slightly different frequencies for each run. Furthermore, we note that the fingerprint is a decimal number with *precision* 18 and *scale* 17. However, only 14 of the 17 decimal digits are significant. So, in absolutely theoretical sense, the collision probability is 1E-14%. On the other hand, in a wide environment, we expect that some values are more frequent than others, for example in systems having similar hardware characteristics. For this reason, as a limit case, some values may never be observed. Therefore, this line of research is currently devoted to verify whether all the values are equi-probable in the range [0.00000000000000000, 0.00099999999999999] over a wide area network.



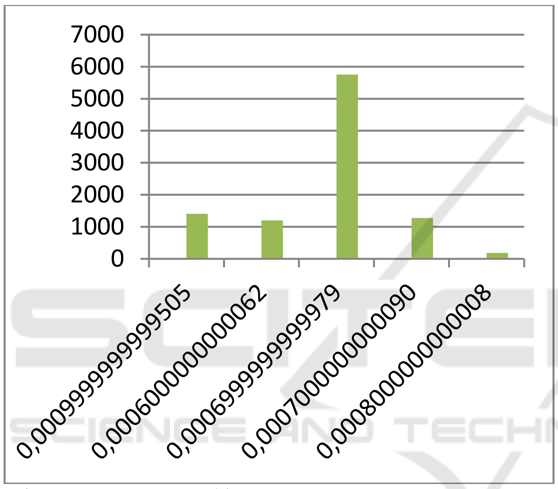
LEGEND:
the x-axis indicates the distinct values of offsets
the y-axis reports the observed frequency of offsets.

Figure 1: Values frequency of offsets for host #1.



(LEGEND: see, Figure 1.)

Figure 2: Values frequency of offsets for host #2.



(LEGEND: see, Figure 1.)

Figure 3: Values frequency of offsets for host #3.

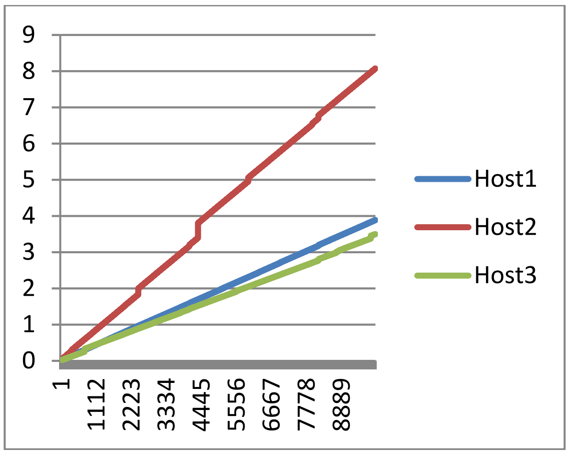


Figure 4: Correlation between number of offsets (x-axis) and CPU-cycles (y-axis).

For further validating the experimentation of the *fingerprint stability*, we modified the computation of the offsets in order to obtain the distance of each timestamp T_i in reference to the initial value T_0. The graph in Figure 4 shows a linear increase of the offsets as correlated to the CPU-cycles for all the hosts. In detail, only the second host showed some biases due to punctual increases of the growth pace. Therefore, we conclude that each offset is incremented each time by approximately the same quantity and, in case of abnormal increments of the offset, surely related to peaks of CPU usage by other processes, this bias is localized and the trend of the function remains constant.

4 APPLICATION

In this Section, we present two working scenarios for the application of a fingerprint-based authentication. The first scenario refers to a typical three-layer web architecture. The second requires the introduction of an Authentication Server acting as a middleware during the login phase.

4.1 Three-layer Architecture

The working scenario does not require any change in a traditional web architecture. We assume that each software agent runs on different hosts (see Figure 5).

The first level is represented by Web Browsers, acting as clients and starting connections towards a Web Server. Here, a Web Application accepts HTTP requests and sends back dynamically-generated web pages. The content of these web pages is created by interacting with a Database Server in order to execute queries and retrieve data. To do so, the Web Application is configured with a database user account, used to connect to the Database Server. This account, usually composed of a username and a password, is stored as a plain-text (Di Tria et al., 2016).

It follows an example of database account used by PHP applications to connect to MySQL databases.

```
<?php
   $db["host"]="111.111.111.111";
   $db["port"]="3306";
   $db["name"]="DBname";
   $db["username"]="root";
   $db["password"]="qwerty";
?>
```

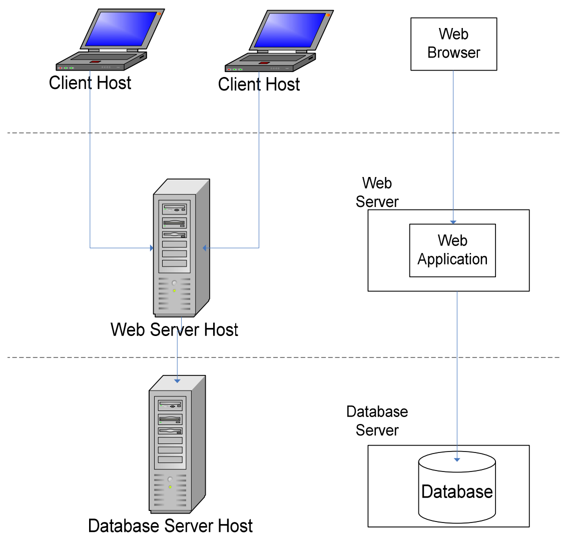


Figure 5: A three-layer web architecture.

This represents a severe security threat, because the database user account is exposed not only to those having a physical access to the host but also to those who success in a brute force attack to the FTP Server, if any.

It is worth noting that, in a simple architecture, both the server processes can be deployed at the same host (*i.e.,* in this case, we can state that `$db[”host”]=”localhost”`).

This does not affect our authentication strategy. Accordingly, the use of a firewall to prevent the exposition of the Database Server does not avoid an authentication strategy.

The application of our proposal in this context involves the use of the fingerprint, instead of the traditional database account, for identifying the host that is trying to connect to the Database Server. If the connection originates from the Web Server running on the legitimate host, then the connection is established, otherwise it is rejected. If we wish to use this approach on a traditional MySQL database, we can use the fingerprint as a password. The main benefit is that this fingerprint is not stored in the file system, but it can be computed on the fly. So, we use the following line of code

```
$db[”password”]= fingerprint();
```

where `fingerprint()` is the function that computes the Web Server fingerprint.

For the sake of simplicity, the Web Application computes the fingerprint of the host on which is running and uses it as a password. This requires that legitimate hosts have been previously identified and reviewed, such that a database account for each legitimate host exists in the Database Server.

4.2 Four-layer Architecture

In this working scenario, we introduce the Authentication Server devoted to checking the fingerprint in the authentication process (see Fig. 6).

As in the previous case, the Web Application computes the fingerprint of the host on which it is running, but now contacts the Authentication Server, which checks the given fingerprint against those stored in its own database (*i.e.*, the Fingerprint Database, FDB). If a match is found, then the host is identified and the Authentication Server contacts the Database Server for creating a temporary database account that is returned to the Web Application.

For increasing the account security, both the username and the password can be randomly-generated. This is very similar to the concept of *session token*.

At this point, the Web Application can safely use this *on-demand* database account for the connection to the Database Server. When closing connection, this database account can be removed.

The complete sequence of actions of the different systems is shown in Fig 7.
The Authentication Server introduces a further security level, because, thanks to this responsibility separation and anonymization, the Database Server is unaware of the client identity. So, in case of identity disclosure, due to a brute-force attack to the Fingerprint Database for example, this information leakage cannot be used for connecting to the Database Server with the identity of another person, as it happens in case of stolen or cracked passwords.

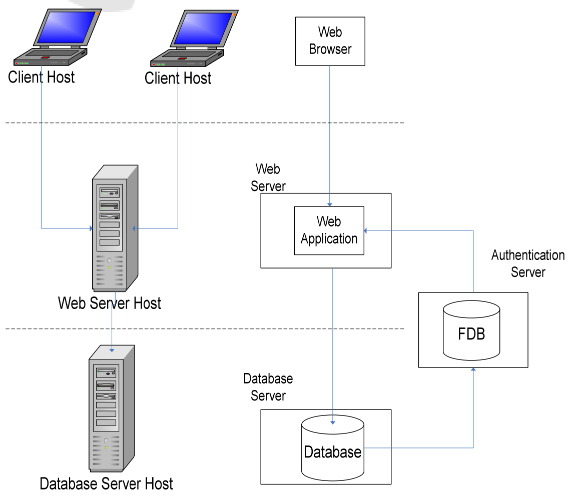


Figure 6: A four-layer web architecture.

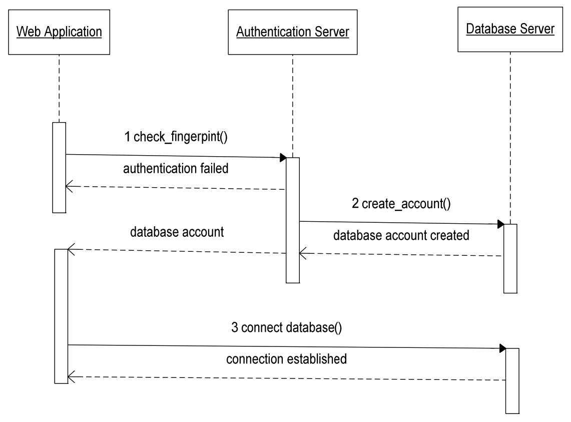


Figure 7: The sequence diagram of the authentication process.

5 CONCLUSIONS AND FUTURE WORK

In the paper, we discussed of a strategy for computing the fingerprint of a host, *i.e.,* a physical device connected to the network.

The proposal is inspired by the method present in literature that is based on clock skew. Analysing the variations of clock signals, it is possible to obtain a unique profile of a host. The works present in literature demonstrated that clock skews are constant for a given host. Our assumption is that this implies a regular patterns in clock signals. So, a clock skew can be quickly mined by analysing a single sequence of timestamps and without any comparison against external sources. This method increases the computation of the fingerprint, by avoiding remote long-time observations.

For concluding, we proposed a novel algorithm for computing a hardware device's fingerprint, in order to use it for authentication purpose in a client-server communication. The fingerprint is computed at run-time in a short interval of time by discovering regular patterns in the difference between timestamps.

The actual Database Servers are based on the something you know paradigm. For this reason, any user or application that needs to interact with it has to be equipped with an account –made by both a username and a password– that, in case of applications, must be stored in a configuration file as a plain text. In order to strengthen the authentication paradigm and adopt the biometrics techniques, a Database Server has to completely revise its authentication mechanism. A ready-to-use solution is exploiting such fingerprints as passwords, which may be computed on the fly.

However, the current real weakness of the strategy proposed in this paper is that the client (*i.e.,* the Web Server) computes its own fingerprint and sends it to the Database Server. Therefore, this strategy is exposed to fake fingerprinting and man-in-the-middle attacks. To improve the connection security, a solution of strong authentication could consist in coupling the fingerprint with Mutual TLS.

Another improved solution consists in a measurement of the Web Server's fingerprint by the Database Server.

So, future work will explore the possibility of a design of a Database Server that is not passive in the authentication process.

Similarly, this strategy can be extended also to Web Browsers in order to identify the final users that interact with a Web Application. This can be done using a client-side computation of the fingerprint on the basis of scripting languages as JavaScript.

ACKNOWLEDGEMENT

A special thanks to Francesco Di Tria, for his valuable technical support in the experiment described in this paper.

REFERENCES

Alaca, F. and van Oorschot, P.C., 2016. Device fingerprinting for augmenting web authentication: classification and analysis of methods. In *ACSAC, 32nd ACM Annual Conference on Computer Security Applications*.

Ben Natan, R., 2005. *Implementing Database Security and Auditing*, Elsevier Digital Press.

Bertino, E. and Sandhu, R., 2005. *Database security - Concepts, Approaches, and Challenges*. IEEE Transactions on Dependable and Secure Computing, 2(1).

Deswarte, Y., Quisquater, J.J., and Saïdane, A., 2004. Remote integrity checking. In *Integrity and internal control in information systems VI*. Springer US.

Di Tria, F., Lefons, E., and Tangorra, F., 2016. Improving Database Security in Web-based Environments. In *2nd International Conference on Information Systems Security and Privacy.*

Fink, R., 2007. *A statistical approach to remote physical device fingerprinting*. Military Communications Conference. IEEE.

Gertz, M., and Jajodia, S., 2007. *Handbook of Database Security: Applications and Trends*. 1st edition 2007, Springer.

Graham, R.L., 1972. *An efficient algorithm for determining the convex hull of a finite planar set*. Information Processing Letters, 1(4).

Jain, A., Bolle, R., and Pankanti, S., (Eds.), 2006. *Biometrics: personal identification in networked society*, Vol. 479. Springer Science & Business Media.

Kohno, T., Broido, A., and Claffy, K.C., 2005. *Remote physical device fingerprinting*. IEEE Transactions on Dependable and Secure Computing, 2(2).

Liu, S., Silverman, M., 2001. *A practical guide to biometric security technology*. IT Professional, 3(1).

Mills, D.L., 1992. *Network Time Protocol (Version 3) Specification*. Implementation and Analysis, Network Working Group, Request for Comments: 1305.

Moon, S.B., Skelly, P., and Towsley, D., 1999. Estimation and removal of clock skew from network delay measurements. In *INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies,* Vol. 1. IEEE.

Polčák, L., and Franková, B., 2014. On reliability of clock-skew-based remote computer identification. In *SECRYPT, 1st International Conference on Security and Cryptography,* IEEE.

Salo. T.J., 2007. Multi-Factor Fingerprints for Personal Computer Hardware. In *MILCOM, the Military Communications Conference,* IEEE.

Sanchez-Rola, I., Santos, I., and Balzarotti. D., 2018. Clock Around the Clock: Time-Based Device Fingerprinting. In *CCS '18, ACM SIGSAC Conference on Computer and Communications Security*. ACM.

Sklar, D., and Trachtenberg, A., 2003. *PHP cookbook*. O'Reilly Media, Inc.

Sundararajan, A., Sarwat, A.I., Pons, A., 2019. *A Survey on Modality Characteristics, Performance Evaluation Metrics, and Security for Traditional and Wearable Biometric Systems*. ACM Comput. Surv. 52(2).

Tang, J., Xu, P., Nie, W., Zhang, Y., Liu, R., 2018. A review of recent advances in identity identification technology based on biological features. In *6th CCF Academic Conference on Big Data*, CCF Big Data.

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