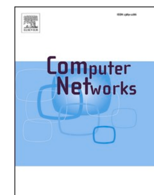
Computer Networks 229 (2023) 109782

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet



Passive operating system fingerprinting revisited: Evaluation and current challenges

Martin Laštovička *, Martin Husák, Petr Velan, Tomáš Jirsík, Pavel Čeleda

Masaryk University, Žerotínovo nám. 617/9, Brno, 601 77, Czech Republic

ARTICLE INFO

Keywords:
OS fingerprinting
Network monitoring
Network management
Cybersecurity
Machine learning
Survey

ABSTRACT

Fingerprinting a host's operating system is a very common yet precarious task in network, asset, and vulnerability management. Estimating the operating system via network traffic analysis may leverage TCP/IP header parameters or complex analysis of hosts' behavior using machine learning. However, the existing approaches are becoming obsolete as network traffic evolves which makes the problem still open. This paper discusses various approaches to passive OS fingerprinting and their evolution in the past twenty years. We illustrate their usage, compare their results in an experiment, and list challenges faced by the current fingerprinting approaches. The hosts' differences in network stack settings were initially the most important information source for OS fingerprinting, which is now complemented by hosts' behavioral analysis and combined approaches backed by machine learning. The most impactful reasons for this evolution were the Internet-wide network traffic encryption and the general adoption of privacy-preserving concepts in application protocols. Other changes, such as the increasing proliferation of web applications on handheld devices, raised the need to identify these devices in the networks, for which we may use the techniques of OS fingerprinting.

1. Introduction

Passive fingerprinting of operating system (OS) is a common task in network management, monitoring, and cybersecurity. It is often a fundamental part of complex tasks and tools as it allows hosts in the network to identify communicating peers and adjust the communication accordingly, e.g., web servers tailor responses based on received User-Agent to avoid interoperability problems [1]. When applied on a network-wide scale, it enables network reconnaissance, asset discovery, cyber situational awareness, and even cyber threats detection. Due to its wide range of applications, OS fingerprinting can be perceived as a matter of course used daily. However, information and communication technology rapidly evolves, and common practices, such as traditional OS fingerprinting methods, may not keep pace with the changes in networks. There is a need to periodically revise the methods and tools and check if they are still relevant in practice.

The specific problem we are approaching in this paper is passive OS fingerprinting via network traffic analysis. We passively collect records of network traffic, e.g., via packet capture or flow (IPFIX) monitoring [2], at an observation point on a communication line. We analyze network communication between the hosts in the network, and we estimate each communicating device's operating system. We do not interact with the devices nor have access to them. Typically, only the packet headers or first few packets should be enough to estimate one or both parties' OS in observed network communication. We observe communication of many devices in our network to build a knowledge base for more precise fingerprints.

Due to the evolution of information technology and communication networks, there is a continuous need to revise the approaches to preserve OS fingerprinting usability. Estimating the OS of a host in the network using its TCP/IP header settings solely is not precise enough for security-related use cases. The wide adoption of network traffic encryption, namely HTTPS, practically prevented payload-based fingerprinting methods, such as User-Agent analysis. From this point of view, the problem is not how to perform OS fingerprinting but on selecting an appropriate and sustainable method for a given use case.

The contributions of this paper are threefold and go beyond the survey of literature. First, we present a survey of the state of the art of passive OS fingerprinting. An overview of use cases is provided to illustrate the number of tasks dependent on passive OS fingerprinting. Second, we evaluate the discussed methods in a series of experiments. Our experiments demonstrate the level of detail and precision of existing

* Corresponding author.
E-mail address: lastovicka@ics.muni.cz (M. Laštovička).

https://doi.org/10.1016/j.comnet.2023.109782
Received 21 September 2022; Received in revised form 3 April 2023; Accepted 15 April 2023
Available online 20 April 2023
1389-1286/© 2023 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

passive OS fingerprinting methods in current network traffic. Third, we identify drawbacks of the existing methods, especially regarding technology evolution in recent years. The experiments suggest which methods are becoming obsolete and which prove to be useful in current networks.

This paper is structured as follows. Background and motivating use cases are outlined in Section 2. Related work is summarized in Section 3. Brief history of OS fingerprinting and traditional methods are presented in Section 4, while Section 5 presents modern approaches, including machine learning and encrypted traffic analysis. The features used in the presented methods are discussed in detail in Section 6. Evaluation and comparison of all the methods is presented in Section 7. Section 8 concludes the paper.

2. Background and motivating use cases

There are many OS fingerprinting applications in network management and cybersecurity domains, often as a part of complex tasks. OS fingerprinting helps identify, enumerate, and map hosts in the network, which then helps in understanding the network and facilitating network operations and security management. OS fingerprinting based on traffic monitoring supports this process on a network-wide scale.

2.1. Network reconnaissance and situational awareness

A high-level motivation for OS fingerprinting is building and maintaining what is known in the literature as cyber situational awareness (CSA) [3]. CSA is a continuous process of perceiving the environment, understanding the processes in it, and projecting future changes of the situation [4]. The perception of the environment is a fundamental process that builds upon data and knowledge from many sources, such as asset management, vulnerability and risk management, anomaly and intrusion detection, and audits. Network reconnaissance, including OS fingerprinting, is an integral part of perception in CSA. Discovering active hosts in the network and identifying their OS is one of the basic steps in gathering information about a network. Such overview is often complemented by network topology mapping, service discovery, and other steps that can often be done using common tools, such as Nmap [89]. The higher the quality of the data on the perception level is, the better we can understand and comprehend the network, anticipate future events and mitigate risks.

2.2. Identification of obsolete, vulnerable, and Rogue devices

Crucial use cases for OS fingerprinting are motivated by operational cybersecurity, cyber defense, and incident response. Herein, we provide three examples of how OS fingerprinting helps cybersecurity operations: detection of *obsolete, vulnerable*, and *rogue* devices.

Detection of obsolete devices is a simple task for OS fingerprinting. An obsolete device with obsolete OS, poses a security risk for the network. Such a device may be severely vulnerable and lacking security updates. Cybersecurity teams may check for obsolete devices, and large-scale OS fingerprinting is a convenient way of doing so. Identifying only the major version of an OS may be sufficient for this task. Such a task is often executed after the end of support of a widely used OS.

Detection of vulnerable devices via OS fingerprinting is a more complex use case than the detection of obsolete devices and require more detailed fingerprinting. Detection of a major version of an OS is often enough to identify an obsolete device but might not be sufficient to detect a vulnerable, yet not obsolete, device. Vulnerability databases can be searched for all the vulnerabilities of a detected OS version. Although this method is not very accurate, it can be used on a large scale to enumerate all the potentially vulnerable devices [5] and save time by reducing the number of vulnerability checks performed by more precise tools. If a new vulnerability appears, OS fingerprinting can be used to estimate the number of devices that could be vulnerable and, thus, provide an estimate of vulnerability's potential impact on an organization and prioritize its handling adequately.

The detection of rogue devices is a task of detecting devices that should not appear in the network. An attacker may install a rogue device in the network to get access to the internal network services while overcoming intrusion detection systems on the network perimeter. Such devices can sniff network traffic, steal data, and compromise other hosts. The detection of a rogue device via OS fingerprinting is possible when the rogue device uses a different OS from known devices in the network, or its OS is not recognized as one of the allowed or supported systems in the network.

There are also numerous highly specialized use cases for OS fingerprinting. A prime example is spam filtering, e.g., as implemented in SpamTitan tool [6]. The tool assumes that most legitimate email servers are based on Linux, and most spam comes from compromised Windows desktops, hence, the tool does not accept e-mails from desktop Windows OS. Similar behavior may be observed in software license servers that do not allow hosts with an unsupported OS to obtain a token to run licensed software locally.

3. Related work

To the best of our knowledge, there is no recent survey paper on passive OS fingerprinting. Historically, however, several survey papers covering passive or active OS fingerprinting were published. The first overview of OS and application fingerprinting techniques was published as a white paper by Allen [7] in 2007. The author discussed similarities between active scanning and passive fingerprinting and illustrate it on Nmap, Xprobe2, and p0f tools. Medeiros et al. [8] conducted a qualitative comparison of four active OS fingerprinting tools (*Nmap, Xprobe2, SinFP*, and *Zion*). *Zion* was shown to outperform other tools, which included the recognition of SYN proxy and *Honeyd*, a TCP/IP stack-emulating honeypot [9]. Herrmann et al. [10] surveyed the device fingerprinting techniques used in network forensics. The survey outlined the use of the active and passive fingerprinting methods to assign the activity to certain actors and answer the forensic question "who did it?".

There are other surveys focused on other areas with a significant overlap with OS fingerprinting. Liu et al. [11] surveyed machine learning techniques for identification of IoT (Internet of Things) devices from passively collected network traces and signal patterns. Identification of a specific IoT device requires more level of detail than OS identification and the techniques can be similar to identifying the device's OS. Sánchez et al. [12] reviewed methods of fingerprinting device behavior to identify individual device or its general type (e.g., desktop, mobile, IoT). Examining the external manifestation of hardware and software equipment is a superset of OS network traffic analysis and covers similar topics. Xu et al. [13] dived into the properties of wireless transmissions and survey methods of device identification based on measurements of the physical and medium access layers. They identified features dependent on the device vendor and transmission software which, in many cases, corresponds to the OS.

A countering task to OS fingerprinting is deception. The administrator may deliberately set a machine to display characteristics of a different OS or application than the actual one. This might be used to hide a machine by masking its identity to prevent attackers from reaching it. A more important use is with the honeypots, hosts, and services in the network that are deliberately left to be exploited by attackers. The honeypots are often emulated, and there is a need to make them look like they run a particular OS or application. Albanese et al. [14] in 2015 used six OS and application fingerprinting tools (*Nmap, SinFP, XProbe, p0f, amap*, and *Nessus*) to fingerprint a masquerading device. The authors illustrated how to deceive the fingerprinting tools with minimal overhead by modifying the fingerprinted hosts' outgoing traffic.

Recently, researchers paid attention to the analysis of encrypted traffic. This topic is surveyed, for example, by Velan et al. [15]. The

encrypted traffic analysis often aims to identify communication protocols using properties of certain applications or transferred content. These tasks are more complex than fingerprinting but provide pieces of information, such as the OS or application name and version, as a part of the result.

4. History and traditional methods

This section provides an overview of the OS fingerprinting origin and early evolution. It introduces the works that impacted the area the most and the traditional approach to fingerprinting based on TCP/IP header parameters. Apart from the scientific papers, this section also covers the OS fingerprinting tools based on the traditional methods.

4.1. Brief history of OS fingerprinting

The history of passive OS fingerprinting can be traced back to December 1998, when Nmap 2.00 was released [89], introducing active OS detection based on a set of features obtained from Nmap probes' responses. This concept inspired many researchers who switched from active probes to passive monitoring of the existing network traffic and relied on the ordinary traffic to contain enough information to distinguish operating systems. The first results appeared in 2000. Lance Spitzner published the concept of passive fingerprinting [16] using three features from IP header (values of Time To Live (TTL), Do not Fragment (DF), Type of Service (ToS)), and TCP Window size parameter. In the same year, the first versions of *p0f* [17] and *Siphon* [18] tools for passive fingerprinting were released.

The signature databases of available tools were limited at that time, and the tools were rejecting most of the traffic as the parameters did not perfectly fit any record in its database. In 2003, Lippmann et al. [19] reviewed ten TCP/IP features used in open-source tools and proposed their normalization, such as rounding up Time to Live value to the next higher power of two to minimize the influence of monitoring point location on the collected data. They also suggested creating groups of related operating systems often confused by the fingerprinting (e.g., Win 95, Win 98, and Win 98-second edition) to simplify the fingerprinting. Finally, they experimented with machine learning algorithms to identify nine classes of OS using the discussed features. They conclude that OS identification from TCP/IP header features is possible, but a low error rate can only be obtained by merging similar classes.

4.2. TCP/IP methods

Karagiannis et al. [20] proposed a novel view on the information obtained from TCP/IP header parameters. They constructed a communication graph of devices according to source/destination IP addresses and ports, and used protocol. From this graph, they derived profiles of end-hosts behavior, and by analyzing the profile properties, they were able to fingerprint specific devices in the network. The communication profile was further exploited by Siebren Mossel [21] who explored the update procedure of different operating systems. He used network flows as the data source and extracted source ports and destination IP addresses. He paired those features to a dictionary of known versions of OS.

The use of network flow technology proved to be determining factor for many works as it enables fingerprinting in large networks. Martin Vymlátil [22] implemented the export of TCP/IP features used for OS fingerprinting into a flow exporter and continued his work with Matoušek et al. [23]. They use eight TCP/IP features known from tools and previous works and show the feasibility of continuous OS fingerprinting in large networks. They also propose a mechanism for an autonomous update of the signature database. They extract the features from an HTTP request and pair them with the corresponding User-Agent. The User-Agent then serves as ground truth, and if a new combination of feature values is observed, it is stored in the database as a new fingerprint. Finally, the authors experiment with machine learning feature selection and show that their model predicts better on a subset of four features; IP TTL, packet size of the first TCP SYN packet, TCP Window size, and TCP No Operations option. Jirsík and Čeleda [24] experimented with the OS fingerprinting in the flow analysis process for high-speed fingerprinting without further data processing.

The evolution of OS fingerprinting from TCP/IP features continued by its applications in different environments. Tyagi et al. [25] explores the possibilities of unauthorised OS detection in enterprise networks. With seven basic header features, they detect virtual machines run by a malicious actor within the organization network. Osanaiye et al. [26] investigate four features (TTL, Window size, SYN size, and IP DF flag) of machines in a cloud environment to identify their OS and subsequently to detect the true source of a packet during spoofed DDoS attack. Laštovička et al. [27,28] employ three basic features (TTL, Window size, and SYN size) to identify the OS of devices connected to a wireless network. They show passive OS fingerprinting's usability in dynamic wireless networks with mobile devices and many operating systems. Al-Sherari et al. [29] experiment with adding unconventional TCP/IP features (e.g., the size of FIN packet) to improve the OS detection of desktop and mobile devices. They also propose a hybrid approach combining exact matches with fingerprint database and machine learning prediction for feature vectors not matching any known fingerprint.

4.3. Overview of tools

Many theoretical concepts and methods have been introduced, but only a few transformed into recognized tools. One of the very first ones and the best-known tool is *p0f* [17] by Michal Zalewski, released in 2000. It relies on a fingerprint database stored in a text file as a set of signature strings for each operating system. The signature string format is designed to describe various features from multiple protocols and was later adopted by other tools. In 2013, a real-time version of p0f called *k-p0f* [30] was implemented in the PNA (Passive Network Appliance) kernel module to increase throughput as the *k-p0f* intercepts packets directly from the network stack. Even though *p0f* tool was popular, it required manual maintenance of the fingerprint database, and the last version of *p0f v3* was released in 2012 with the last meaningful update of the database on May 21, 2014.

Ettercap tool [31] was originally designed in 2001 as a suite for man-in-the-middle attacks. Apart from the main functionality of packet sniffing and manipulation, it also supports passive OS fingerprinting and generating host profiles. It uses similar features and signature structure as *p0f*, but from the source code [32], it seems the database was last updated in 2004 according to the fingerprint file header and the fact that it does not contain any newer OS versions (i.e., the latest version of some well-known OS are Windows XP, Mac OS X 10.3, or Debian 3 woody).

Similar destiny met other tools mentioned in the literature from that time. The *Siphon Project* [18] by Subterrain Security Group from 2000 disappeared, and its official website no longer exists, leaving only the Github repository for historical and reference purposes. *NetSleuth* [33] was a tool for network monitoring and forensic analysis capable of identifying fingerprints of network devices. Released in 2012 by NetGrab, it promised extensive functionality but disappeared in 2015 with no official information. *PRADS* (Passive Real-time Asset Detection System) [34] by Fjellskål and Wysocki was another attempt for passive OS fingerprinting tool from 2009. It took and extended the *p0f* fingerprint database with recent entries and features. Nevertheless, after the initial activity, the fingerprint database was abandoned on August 16, 2012.

Satori [35] is one of the tools with still active updates. First released in 2004, it was active for about ten years before its end of life. In 2018, its original creator Eric Kollmann revived it as an open-source Python library and continued its development. Due to *Satori* being actively updated (last database update on Sep 2022), we can say *NetworkMiner* fingerprinting abilities are updated as well. *NetworkMiner* [36] is an

open-source tool for network forensic analysis that offers OS fingerprinting. It does not have its own fingerprint database but uses the fingerprints of *p0f, Ettercap*, and *Satori*.

Apart from the open-source tools, OS fingerprinting is often a part of bigger commercial solutions. Active scans dominate this field, but some products offer passive fingerprinting as well. It is a part of security in Juniper *Network and Security Manager Profiler* [37], and in Cisco *IPS Security manager* [38] and Cisco *Meraki firewall* [39]. However, the commercial solutions are closed source, and we cannot evaluate how they perform the fingerprinting and how often the fingerprints are updated. Another approach is represented by AlienVault products, AlienVault *Asset Discovery* [40] and *OSSIM* [41] which rely on the open-source tools on the backend. They used *p0f* as fingerprinting core until July 2012 when they switch to *PRADS*. Passive fingerprinting can also be a part of protection against spam as in SpamTitan solution [6] or can be commercialized as a standalone database of fingerprints like Fingerbank [42]. Fingerbank database was available on Github [43] since 2011 as an open repository of DHCP fingerprints of operating systems. This database currently continues in the form of a public API with a rate limit of 300 requests per hour and a paid access with higher limits.

5. Modern approaches

With TCP/IP headers explored, the research searched for new features to distinguish OS in network traffic. This section covers the works that extract features from application layer protocols, analyze encrypted traffic, and works dedicated to special traffic types. It also covers the

Table 1
Related work on passive OS fingerprinting.

Year	Work	Data Type	Dataset	Features Count						Methodology*	Level of Detail
				Total	TCP/IP	DNS	HTTP	TLS	Other		
2000	[16]	PCAP	N/A	4	4	–	–	–	–	DB of signatures	OS name
2003	[19]	PCAP	Own data	10	10	–	–	–	–	ML (kNN, Btree, MLP, SVM)	Mixed – OS name, minor version
2003	[44]	PCAP	Own data	1	–	–	1	–	–	DB of signatures	Minor version
2004	[60]	PCAP	Own data	4	4	–	–	–	–	ML (Bayes)	OS name
2005	[55]	PCAP	Own data	1	1	–	–	–	–	Statistical analysis, fourier transformation	Minor version
2007	[20]	PCAP	Own data	5	5	–	–	–	–	Communication graph analysis	Host in a network
2007	[61]	PCAP	Own data	7	7	–	–	–	–	ML (X-Means)	Minor version
2008	[62]	PCAP	Own data	172	–	–	–	–	172	Formal grammars	Minor version but only SIP devices
2009	[63]	Nmap DB	Nmap DB	544	544	–	–	–	–	ML (SVM)	OS name
2010	[64, 65]	PCAP	Dataset [66]	N/A	Y	–	–	–	Y	DB of signatures	Minor version
2010	[67]	PCAP	Own data	22	22	–	–	–	–	ML (J48/C4.5, Jrip, RF, SVM, kNN)	Minor version
2012	[21]	IPFIX	Own data	2	2	–	–	–	–	DB of signatures	OS name
2013	[48]	PCAP	Own data	2	–	2	–	–	–	DB of signatures	OS name
2014	[22]	IPFIX	Own data	8	8	–	–	–	–	DB of signatures	Minor version
2014	[23]	IPFIX	Own data	9	8	–	1	–	–	DB of signatures, ML (k-Means)	Minor version
2014	[29]	PCAP	Own data	9	9	–	–	–	–	DB of signatures, ML (J48/C4.5)	Minor version
2014	[24]	IPFIX	Own data	4	3	–	1	–	–	DB of signatures	Minor version
2015	[25]	PCAP	Own data	7	7	–	–	–	–	DB of signatures	Minor version
2015	[49]	DNS log	Own data	1	–	1	–	–	–	DB of signatures	OS name
2015	[46, 47]	IPFIX	Own data	2	–	–	1	1	–	DB of signatures	TLS client / Minor version
2015	[26]	PCAP	Own data	4	4	–	–	–	–	DB of signatures	Minor version
2016	[52]	PCAP	Own data	53	3	–	–	5	45	ML (SVM, RF, kNN)	OS name
2016	[51, 54]	PCAP	Own data	216	Y	Y	Y	Y	Y	ML (J48, Jrip, Ridor, PART, DT, RF, NB, MLP)	Minor version
2016	[56]	PCAP	Own data	3	–	–	–	–	3	DB of signatures	Specific device
2016	[57]	PCAP	Own data	N/A	–	–	–	–	Y	Frequency-domain analysis	Minor version
2017	[68]	PCAP	Own data	5	5	–	–	–	–	Expectation-Maximization (EM) estimator	Major version
2018	[27]	IPFIX	Dataset [69]	6	3	–	2	1	–	DB of signatures	Minor version
2018	[70]	IPFIX	Dataset [69]	3	3	–	–	–	–	ML (kNN, DT, NB, SVM)	Minor version
2018	[58]	PCAP	Own data	2	–	–	–	–	2	Timing distribution analysis	Specific device
2018	[59]	PCAP	Own data	1	–	–	–	–	1	Timing analysis	Specific device
2019	[53]	IPFIX	Dataset [69]	21	7	–	–	6	8	ML (Gradient Boosting DT)	Major version
2020	[45]	IPFIX	Dataset [71]	12	3	–	2	7	–	ML (DT)	Minor version
2020	[72, 73]	IPFIX	Dataset [69]	3	3	–	–	–	–	ML (SVM, RF, KNN, NB, MLP, LSTM)	OS name
2021	[74]	Nmap DB	Nmap DB	233	233	–	–	–	–	ML (NB, MLP, DT, RF, Bagging, LogisticRegression)	OS name
2021	[75]	PCAP	Own data	23	14	–	1	–	8	ML (NB, DT, SVM, KNN, RF)	OS name
2021	[76]	PCAP	Dataset [68]	7	7	–	–	–	–	Expectation-Maximization	Major version
2021	[50]	PCAP	Own data	1	–	1	–	–	–	DB of signatures	OS name
2022	[91]	IPFIX	Dataset [69]	*	*	–	*	*	*	ML (SVM, RF, NB, AL)	OS name

* DB (Database), kNN (k Nearest Neighbors), MLP (Multi-Layer Perceptron), SVM (Support Vector Machine), RF (Random Forest), NB (Naïve Bayes), DT (Decision Tree), LSTM (Long Short-Term Memory), AL (Active Learning).

applications of machine learning methods as the next step in OS fingerprinting. Finally, overview of all the reviewed work is provided in Table 1.

5.1. Application layer information

The traditional methods focus on the TCP/IP stack features as they are universal for most traffic and are not affected by encryption. However, the application layer contains additional information that allows easier and more precise fingerprinting. The most common choice of application layer protocol to study is HTTP. In 2003, Saumil Shah proposed [44] to use HTTP banners sent by the server for OS identification. He built a database mapping different banner variations to the most probable application server and its operating system. The OS identification of clients often stems from the HTTP User-Agent, which directly states the client OS. It is used as a complement for other methods [24,27,45] to improve identification in unencrypted HTTP connections. Because reading a string for OS identification is not scientifically interesting, some works use User-Agent as ground truth to build models on different features [23,46,47].

The idea of OS identification based on update procedures or OS-specific connections spans multiple protocols. Mossel's work [21] using destination address was already introduced. Matsunaka et al. [48], Chang et al. [49], and Voronov et al. [50] take the OS-specific queries from DNS requests, Laštovička et al. [27,45] use HTTP hostname and TLS Server Name Indication (SNI) fields. In all cases, they build a database of known OS and their related service locations identified either by IP address or a domain name. When a device connects to such a location, the OS is assigned to the device.

The characteristics of DNS queries were studied by Matsunaka et al. [48], who measure their time distribution to identify OS. Features extracted from DNS were also used by Aksoy et al. [51] as a complement to features from other protocols to improve their method's accuracy.

5.2. Encrypted traffic analysis

With the encryption of the network traffic, the amount of information that can be used for OS detection is significantly reduced. Application layer-based methods for OS fingerprinting that are usually more precise than the traditional ones are rendered useless in the encrypted network traffic settings as the information needed to determine the OS (e.g., User-Agent) is not available due to encryption. Even though most of the application data is encrypted, there are still unencrypted parts of the handshake, which can be utilized for OS fingerprinting, such as certificate, SNI, or cipher suites.

Husák et al. [47] build their method for operating system fingerprinting on the assumption that we can observe both encrypted and unencrypted network traffic from the host. The pairing of these two pieces of information enables the OS fingerprinting of the encrypted traffic. They suggest creating a dictionary that links the information from unencrypted network traffic used for OS fingerprint (i.e., User-Agent) with the information observable in the encrypted traffic (i.e., cipher suites lists). The OS detection in the encrypted network traffic is then done by harvesting the cipher suites from the observed encrypted connection and mapping it to the User-Agent in the dictionary based on which the OS is determined. The authors evaluate the cardinality of the relations in the dictionary, and they can identify the OS in 62% of the encrypted connections.

Muehlstein et al. [52] dealt with OS identification from encrypted traffic by adding features extracted from TLS handshake to traditional TCP/IP features and packet timings. They achieved 85% accuracy on data with most of the traffic encrypted. Similarly, Fan et al. [53] use the TLS handshake feature combined with TCP/IP and traffic statistics but achieve up to 96.3% accuracy on a TLS dataset. Laštovička et al. [45] build a classifier solely from TLS *Client hello* features and achieve 93.1% accuracy. Aksoy et al. [51,54] experiment with a general classifier for nine protocols and use up to 216 features extracted from them. That includes encrypted traffic over SSL, SSH, and FTP; however, their reported accuracy of OS identification on these protocols is only 25.0%, 22.5%, and 14.0% respectively.

5.3. Other approaches

Other approaches cover mostly borderline or very specific areas of fingerprinting concerning only selected types of devices. The common topic is measuring time aspects and using advanced analytical methods to infer the device or its OS.

Kohno et al. [55] noticed that operating systems have different default frequencies of a TCP Timestamp option clock that can be used to identify the OS. They further investigate the traffic timings, and, with precise enough measurement, they can identify OS and even a specific device based on its clock skew. The scope of Azzouni et al. [56] work is limited to the identification of OpenFlow controllers, yet their methods are general enough to be considered related to OS fingerprinting. They employ timing analysis of packets, precisely the round-trip time and processing time, together with features of ARP (Address Resolution Protocol) responses. Gurary et al. [57] fingerprint OS of smartphones based on their signal transmission properties. They devise a frequency-domain analysis algorithm that can identify a mobile device's OS from a sample of 30 s of traffic. Shen et al. [58] focus on ICS (Industrial Control Systems) devices and analyze time interval distribution of packets and inter-layer response time to fingerprint them. They complement the analysis with a traffic characteristics profile of each device to detect it and reveal any attacks that tamper with the device or try to impersonate it. Sanchez et al. [59] propose to fingerprint physical devices by their internal clock signals, which depend on their specific hardware. By measuring the timing of responses, they identify a known device from the traffic.

5.4. Machine learning

Machine learning (ML) techniques are mainly used to overcome the drawbacks of traditional methods. Namely, the need for manual creation of a signature database and its maintenance. Machine learning also enables using a higher number of new features and their autonomous selection based on the resulting model's accuracy.

The first experiments with ML methods followed the use of TCP/IP parameters. Lippmann et al. [19] shows its feasibility for a limited number of target classes (i.e., operating systems). Robert Beverly [60] constructed a Bayesian classifier over four basic features to map OS share in network traffic and to count hosts behind NAT devices. Zhang et al. [63] then experiment with an increasing number of features from the Nmap fingerprint database and their processing in an SVM (Support Vector Machine) classifier. The work on creating fingerprints and feature selection continued in Caballero et al. [61], who proposed an approach for automatic fingerprint generation from a TCP header. Richardson et al. [67] reviewed the features used in that time and discussed their influencing factors. They discuss the problems of evaluating the methods on small networks, which result in the automatic tools to bias the training data and the classifiers overfitting. They showed that features' semantics are critical for OS identification and blindly putting features into ML algorithms works only on a very limited dataset. Salah et al. [75] extended the TCP/IP parameters with fingerprinting from IPv6 headers to keep up with the adoption of the protocol.

The capabilities of ML algorithms allowed researchers to use a higher number of features. With the traditional TCP/IP header parameters explored, the attention turned toward other protocols and derived features. Especially, accurate OS fingerprinting from encrypted traffic parameters began to dominate the field. Muehlstein et al. [52] experimented with adding TLS features. Aksoy et al. [51,54] investigate headers of IP, ICMP, TCP, UDP, HTTP, DNS, SSL, SSH, and FTP protocols and use a genetic algorithm to determine the relevant packet header

features. Fan et al. [53] combine TLS handshake with flow statistics and TCP/IP features.

Apart from the accuracy, some works focused on the usability of machine learning in large networks. With many different operating systems and lots of training data, the resulting ML models tend to be extensive. However, these complicated models need to process large amounts of data continuously. Shamsi et al. [68,76] propose a non-parametric expectation-maximization estimator to enable Internet-wide OS fingerprinting. Laštovička et al. [70] benchmark well-known ML algorithms to evaluate their performance and accuracy in large networks. They conclude that the decision tree classifier has similar accuracy to other algorithms while needing orders of magnitude lower time to identify the OS.

The works on OS fingerprinting are summarized in Table 1. Most of the works were already referenced in previous sections, namely the ones proposing novel methodology or using novel features. The table summarizes the works in chronological order and presents an overview of data types, datasets, and methodologies used. Further, the table displays which and how many features were used in a particular paper. If a paper only states that it uses a feature from the protocol, but we could not track how many or which features were used, a simple *Y* marks the fact. The features are grouped according to the protocol fields they were taken or derived from. Finally, the level of detail achievable via each method is displayed. It is divided into four categories from the simple OS name (e.g., Android), its major version (e.g., Android 9), and its minor version (e.g., Android 4.2). The last category is specific to the papers dealing with profiles of devices as identification of a specific device inherently contains its operating system.

6. Datasets and fingerprinting features

An interesting phenomenon is that most research works use the researcher's own data, such as the data from a laboratory setup or a live network. There are only a few known datasets and other alternatives, such as the Nmap Feature Database used by Zhang et al. [63]. An insufficient number of available datasets and a short period of time in which they conform to the real environment are well-known issues of cybersecurity and network traffic analysis research. In total, we identified three datasets collected from a live network that were used in works on passive OS fingerprinting. The first dataset, presented by Massicotte et al. [66], was used in two papers by Gagnon and Esfandiari [64,65]. The dataset was published in 2006 and used in works from 2011 and 2012, which raises a question of its relevance to current network traffic. The other two datasets were presented by Laštovička et al. [69,71] to accompany authors' recent works. The datasets and the research works were published between 2018 and 2020 and should accurately represent current real-world network traffic as the datasets contain network flow capture from large-scale campus network including the OS labels from various OS detection techniques up to the minor OS version detail level. The datasets were recently used by Fan et al. [53], Hagos et al. [72,73], and Zhang et al. [91] from different research groups, which indicates the usability of the dataset. Moreover, the usability of the datasets is supported by more the 500 downloads of the datasets.

It is worth noting that the dataset for evaluating passive OS fingerprinting does not have to originate in the networking domain. In essence, any dataset of network traffic that annotates the OS of communicating hosts can be used, for example, datasets from training exercises, Capture the Flag games, and similar events. Such datasets often annotate the hosts, software, and version, but the number and variety of involved hosts are often limited. Thus, such datasets can be used but are not recommended if an alternative is available.

6.1. Fingerprinting features

The works reviewed in the previous sections use a wide range of features, from the simple TCP/IP header fields to features derived from the application layer. Also, the number of features used ranges from one to several hundred. This section discusses the features used for OS fingerprinting and their relation to the actual OS and how they are affected by other phenomena.

To illustrate the feature selection importance, we start with features we encountered in some papers that are outright wrong and should never be used to identify OS. The most obvious one is the source IPv4 address feature. Source IP virtually equals the OS's identifier in a fixed environment with static addresses, and experiments in such environment should reach 100% accuracy. However, if a model trained in these conditions is used in any other or a dynamic network, it has no information value.

Communication and its state affect numerous features used in the literature. The most used ones are transport protocol, destination port, TCP flags, ACK number, and fragment offset from the TCP/IP headers. They are bound to the data transfer itself, and the OS of the packet sender cannot affect them as a different value would disrupt the communication. Similarly, features describing the data from the application layer like HTTP content type or DNS response length (and many others) were used.

6.2. Network and transport layer features

We will discuss in detail the TCP/IP header fields as they are used in most papers. Table 2 provides an overview of all IP header fields together with a short comment on whether or not they are affected by the OS default settings for TCP/IP network stack. The fields marked as *Yes* correspond to OS unique settings and are commonly used in practice. Other features, marked *Partially*, need to be treated carefully as their usage depends on the context of the communication.

The IP Type of Service (ToS) field meaning changed many times since its introduction. The latest updates were made by RFCs 2474 [77] and 3168 [78] in 1998 and 2001, respectively, dividing the ToS into two distinct fields, Differentiated Services Code Point (DSCP) and Explicit Congestion Notification (ECN). Later in 2018, RFC 8436 [79] changed the assignment procedures of the DSCP space affecting the values again. When using the ToS field for OS fingerprinting, such changes need to be considered as old operating systems use different standards and the field has completely different semantics.

The *Total Length* parameter is directly affected by transmitted data. However, it is one of the most used features for OS fingerprinting when applied correctly. The total length of the first packet in TCP communication often referred to as *SYN Size*, does not contain any data part and is influenced solely by OS preferences of TCP Options and padding. *SYN Size* covers only the length of the packet, whereas *Checksum* also corresponds to the header fields' content. Hence, even the initial packet of communication contains the communication-specific and routing information that is not related to OS and still affecting the checksum. It then depends on how the feature is treated and if the fingerprinting

Table 2
IP header features.

Feature	OS Affected	Comment
IP Version	No	Depends on communication
Header Length	No	Depends on options field
DSCP	Partially	Depends on data and OS defaults
ECN	Yes	OS capability of congestion control
Total Length	Partially	Only for selected packets
ID	Yes	OS generated value
Flags	Yes	OS default
Fragment Offset	No	Depends on data transfer
Time To Live	Yes	OS default
Protocol	No	Depends on service
Header Checksum	Partially	Depends on data and OS defaults
Source IP	No	Depends on network topology
Destination IP	Partially	Only for selected values
Options	No	Depends on communication

system can filter out such influence.

The last field from the IP header used for OS fingerprinting depending on the context is the *Destination IP*. Generally, any OS can communicate with any IP address. However, some addresses are dedicated to serving OS-specific content (i.e., update servers) to which others usually do not connect.

The situation with TCP header fields is much clearer. The values are either strictly enforced by the communication (i.e., destination port, flags, acknowledgement) or the sender OS can choose the value from a predefined range. The fields are summarized in Table 3.

6.3. Application layer features

The inspection of higher-level protocols brings new opportunities for OS fingerprinting. They contain a mix of features that depend on the OS, communication, and data. Hence, it is necessary to discuss which phenomena affect each feature and assess its suitability. As there are hundreds of fields in various protocols, we cannot provide an exhaustive assessment of all the features. Instead, we provide an overview of the most used and promising features for passive OS identification which are summarized in Table 4 and discussed in detail below.

The most common use of application layer features is to read the field where OS is explicitly stated. The *HTTP User-Agent* was specifically designed to provide the information about the client OS so that the server can adjust the web page to fit the client. Similarly, other protocols send OS information in their banner and, if it is not encrypted (like in the HTTP case), the OS can be passively identified. Even though the User-Agent field can be easily spoofed, we believe such behavior is not widespread enough to hinder OS fingerprinting statistically, as the ratio of spoofed values ranges between 0,1% to 0,22% [92,93]. Also, other anomalies in User-Agent amount to less than 0,1% of real traffic [94], which makes OS identification based on User-Agent analysis reliable for most fingerprinting use cases.

The OS can be identified from the target of communication using a similar idea to the destination IP OS identification. The device usually has to send a DNS request to translate the hostname to IP address to access the OS-specific content. The *DNS qName* feature can be extracted to serve the OS identification. The communication target can also be monitored in the HTTP and HTTP/2 traffic. The *HTTP hostname* and *TLS Server Name Indication* (SNI) are both sent in cleartext and can be monitored easily.

Analysis of encrypted traffic represents a special use-case in application layer OS fingerprinting. Even though most of the communication is illegible, the parameters from the connection setup provide additional information. Especially the TLS *Client hello* messages are often used. However, the standards change frequently, modifying the semantics and values of the fields. There are significant changes in TLS 1.3 [80] to previous versions. The *Version* of TLS 1.3 client is set to 1.0 in TLS header, then the field *Version* inside the *Client hello* states 1.2, and finally in the *Supported Versions* extension it declares version 1.3. This non-intuitive behavior can confuse an automatically trained OS fingerprinting system, which was not trained on TLS 1.3.

Table 3
TCP header features.

Feature	OS Affected	Comment
Source Port	Yes	OS generated for requests
Destination Port	No	Depends on communication
Seq Number	Yes	OS generated value
Ack Number	No	Depends on communication
Data Offset	No	Depends on options field
Flags	No	Depends on communication
Window Size	Yes	OS default
Checksum	No	Depends on data payload
Urgent Pointer	No	Depends on data payload
Options	Yes	OS default

Table 4
Selected application layer features.

Feature	OS Affected	Comment
HTTP User-agent	Yes	Application generated
SSH Banner	Yes	Application generated
DNS qName	Partially	Only specific values
HTTP Hostname	Partially	Only specific values
TLS SNI	Partially	Only specific values
TLS Version	No	Defined by standard
TLS Cipher suites	Partially	OS generated until TLS 1.2
TLS Supported groups	Partially	OS generated until TLS 1.2

The *Cipher Suites* field specifies a list of supported encryption algorithms. TLS 1.3 defines only five cipher suites to be used, limiting the variability for fingerprinting. The *elliptic_curves* extension was renamed to *supported_groups* and restricted clients to use five elliptic curves and five finite field groups. Moreover, it introduced ordering with the most preferred group first. However, many client implementations intentionally violate the standard by including legacy cipher suites and groups to ensure backward compatibility. Some clients also implement the GREASE (Generate Random Extensions And Sustain Extensibility) [81] mechanism to prevent extensibility failures in the TLS ecosystem. The rapid changes driven by standards and implementations require corresponding updates in fingerprinting systems, but OS identification using these features is still possible.

6.4. Derived features

The final type of features is the characteristics measured from the traffic communication patterns rather than from the packet content. They are heavily affected by the hardware of sending machine and the state and current performance of the network. The fingerprinting system must clearly define how it separates the outside influence on the original OS behavior's feature values. Such distinction should be sound both for the laboratory experiment settings and for the real-world network deployment to ensure the method viability.

The prime examples are the features of *packet time interval distribution* and *packet inter-arrival times*. The fingerprinting system needs to fulfill several requirements to measure and use these features. The monitoring probe needs to measure time with sufficient precision as small skews can affect the fingerprinting process. Moreover, it needs to address the problem of a system load of both the monitoring probe and the routing devices on monitored paths. The probe's high system load can delay the processing of some packets invalidating the arrival timings, and similarly, packets could be slowed in a bucket on an overloaded router anywhere on the path from source to destination. To address these issues, many packets must be collected to statistically eliminate the random noise and extract the underlying average values. A different set of timing features used in the literature is affected by the load of the monitored system and the load of the network or monitoring probe. The *server response time* and *round trip time* features add the dependency on how fast the fingerprinted system can handle requests, which is again affected more by the current load than the OS.

7. Evaluation of the presented methods

The previous section presented around 20 features related to the operating system fingerprinting. This section describes an experimental setup to monitor a large operational network, extract ground truth, and evaluate OS fingerprinting methods.

7.1. Experiment setup

For the evaluation of OS fingerprinting methods, we need a dataset with the following requirements. First, the dataset needs to be big enough to capture the variability of the data. In this case, we need many

connections from different operating systems. Second, the dataset needs to be annotated, which means that the corresponding operating system needs to be known for each network connection captured in the dataset. Therefore, we cannot just capture any network traffic for our dataset; we need to be able to determine the OS reliably.

To overcome these issues, we have decided to create the dataset from the traffic of several web servers at our university. This allows us to address the first issue by collecting traces from thousands of devices ranging from user computers and mobile phones to web crawlers and other servers. The ground truth values are obtained from the HTTP User-Agent as proposed and used by [23,46,47,45], which resolves the second of the presented issues. Even though most traffic is encrypted, the User-Agent can be recovered from the web server logs that record every connection's details. By correlating the IP address and timestamp of each log record to the captured traffic, we can add the ground truth to the dataset.

For this dataset, we have selected a cluster of five web servers that host 475 unique university domains for public websites. The monitoring point recording the traffic was placed at the backbone network connecting the university to the Internet.

7.2. Dataset creation

The dataset used in this paper was collected from approximately 8 h of university web traffic throughout a single workday. The logs were collected from Microsoft IIS web servers and converted from W3C extended logging [82] format to JSON. The logs are referred to as *web logs* and are used to annotate the records generated from packet capture obtained by using a network probe tapped into the link to the Internet.

Fig. 1 describes the entire dataset creation process, which consists of seven steps. The packet capture was processed by the Flowmon flow exporter to obtain primary flow data containing information from TLS and HTTP protocols (1). Additional statistical features were extracted using GoFlows [83] flow exporter (2). The primary flows were filtered (3) to remove incomplete records and network scans before merging the data from both exporters (4). *Web logs* were filtered to cover the same time frame as the flow records (5) and then paired (6) with the flow records. The last step was to convert the User-Agent values into the operating system (7) using a Python version of the open-source tool *ua-parser* [84]. We replaced the unstructured User-Agent string in the records with the resulting OS. The details of OS distribution grouped by the OS family are summarized in Table 5. The *Other* OS family contains records generated by web crawling bots that do not include OS information in the User-Agent.

We provide the created dataset with detailed description of the creation process in our university repository [85]. To make it more available for wider audience of scientists, we also published the dataset on Zenodo sharing platform [90].

7.3. OS fingerprinting methods evaluation

Our goal was to experimentally evaluate the accuracy of available OS fingerprinting methods on a dataset from the current traffic of a large operational network. The methods can be split in two groups: 1) machine learning and 2) methods based on manual analysis and a database of signatures.

To evaluate the machine learning methods results, we built a data processing pipeline. The first step is to load the dataset from file while discarding the features not used by the evaluated method. The second step is to filter out the records with a null value in the required features. The numerical features are then standardized using Standard Scaler [86] subtracting the mean and dividing by the standard deviation. The categorical features are encoded using the OneHot encoder, which is necessary for some classifiers and does not introduce any artificial numerical properties. The label with the target class is treated as a string. In the case of identification of OS name, the label is taken directly. We concatenate the OS name, major version, and minor version into one more detailed label to identify minor versions. With the data loaded and preprocessed, the dataset is randomly divided into training and testing sets in the ratio 80:20. The ML model is trained, and its classification score is calculated over the test set. An average classification score of ten independent executions of the process is presented as the result in this paper. Finally, the implementation of each machine learning algorithm is taken from the SciKit tool [87]. We changed the algorithm's parameters only if such a change was explicitly stated in the paper presenting the evaluated method, otherwise, we used the default settings.

To provide a reference point, we let the p0f tool discussed in detail in Section 4.3 identify the operating systems. As p0f works only with PCAP data, we run it over the original packet capture and then paired the results with web log data in the same way as with IP flows. We calculated the accuracy metrics according to Sokolova's [88] multi-class classification microaveraging formulas and summarized the results in Table 7. The accuracy is high due to the fact that in multi-class settings, every true negative classification for each class pushes the resulting average accuracy towards one, and thus, losing the information value on large amounts of samples. The other metrics reflect the fact that p0f was

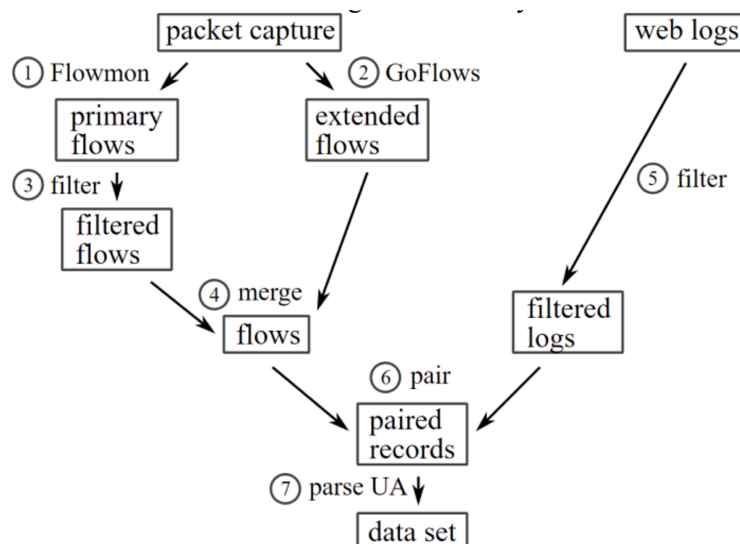


Fig. 1. Dataset creation process. Packet capture is transformed to flows which are merged with web logs to provide ground truth.

lastly updated in 2014 and cannot identify any of the newer OS which dominates the dataset. Lastly, the execution time needed to process the dataset of roughly 22GB of data took 54.7 s on average on a common desktop PC, hence the performance is not an issue.

The first work we aimed to reproduce was Lippmann et al. [19] as it was one of the first works defining the field. The results are summarized in Table 6. The original accuracy results cover the identification after merging classes of OS, which mostly corresponds to OS name. However, we were interested in how the methods performed if the classes were not merged and the identification was considered minor versions. The results for the minor version are in the parenthesis in Table 6. We also noticed the parameters for ML algorithms were limited in the original work. In the year 2003, they needed severe compromises between accuracy and computing power. We repeated the experiment setting the parameters to match the ones described in the paper and then again with the default settings of the SciKit tool.

The execution time of training and testing differs for each ML algorithm. The whole processing pipeline identifying minor version of OS based on the Lippmann's parameters took 15.4 s in the case of k-NN, 3.5 s for decision tree, 120.3 s for MLP, and 144.5 s for SVM on a desktop PC. The higher complexity of MLP and SVM classifiers results in much longer processing time while not resulting in better accuracy metrics as shown above.

Evaluating other works using machine learning algorithms proved to be more complicated. As it allows to use many features, the authors use up to hundreds of them without listing them in their papers or other sources. Table 8 summarizes the results where we were able to identify the features and algorithms used. In the case of Richardson's work [67], we omitted the features of checksums, seq/ack numbers, data offset, and urgent pointer as they are strongly tied to the transmitted data (see Section 6), and we did not include them in the flow export data. The results list the accuracy of evaluated methods on the level of detail of OS name with accuracy on the minor version in parentheses. The focus of many works on ML is to compare different algorithms performance, so we included the measurements in our evaluation.

The second group of works identifies OS by matching the feature values to a previously created signatures database. In most cases, the original database is no longer available (or never was), and evaluating it on a modern dataset is not possible. We decided to replace the database with a Decision Tree classifier trained on the original experiment's features. We used the same processing pipeline as described above to measure the results. As the fingerprinting approach is different, the reproduction is by no means perfect and should be taken only as an approximation of the methods performance. Even with such limitation we believe this comparison can be interesting and provide some insight into the field. The results are presented in Table 9 and cover the works that describe their identification method and list the features used. Most of the works are proof-of-concept type of papers that show OS identification is possible using the selected features but lack any performance metrics.

Table 5
Overview of identified OS traffic.

OS Family	No. of Flows	OS Family	No. of Flows
Other	42,474	Ubuntu	653
Windows	40,349	Fedora	88
Android	10,290	Chrome OS	53
iOS	8840	Symbian OS	1
Mac OS X	5324	Slackware	1
Linux	1589	Linux Mint	1

Table 7
Evaluation of p0f v3.

Metric	OS name	Minor version
Accuracy	92.8%	97.6%
Precision	64.1%	6.9%
Recall	49.1%	8.1%
F-score	55.6%	7.5%

8. Conclusion

In this paper, we presented the use cases, methods, and history of passive operating system fingerprinting. We described traditional and modern approaches to the task in detail and discussed the traffic features used in literature from the perspectives of information value and usability. Further, we provided an evaluation of the discussed methods and illustrated their usage and accuracy. We mainly focused on how the fingerprinting methods cope with recent changes and novel trends in network communication.

One of the most important topics we addressed in this paper is the need for OS fingerprinting methods evolution to keep up with the changes in network protocols, new types of devices, and shifts in computing paradigms such as wireless networks, mobile devices, cloud computing, network virtualization, traffic encryption. These examples of rapid development demand fingerprinting methods to be flexible to adapt to them. The usage of machine learning seems to be a solution. We transformed methods relying on legacy signature databases to machine learning models based on the same features during our evaluation. We tested selected methods on web server traffic, and our results suggest that the accuracy of old methods with machine learning is comparable to the current methods. The accuracy of evaluated methods ranges between 80 to 95% when identifying the OS name. When identifying the OS name, major version, and minor version, the spread in accuracy widens to 70–95%.

8.1. Current challenges

We encountered several issues and challenges related to the OS fingerprinting research during the literature review and subsequent experiments. We discussed them throughout the paper; hence, we provide a summary of key findings.

8.1.1. Missing datasets

The methods presented in recent literature are often tailored for one particular type of network as researchers focus on available data, which leads to measurements in the lab environment or the research facility, which severely limits the scope and diversity of the data. The presented methods are fine-tuned to provide the best possible results on the selected data, and the machine learning models are trained on the same

Table 6
Evaluation of Lippmann's work [19].

Classifier*	Accuracy on OS Name (Minor Version)		Accuracy metrics on OS name with default settings (Minor Version)			
	Original Results	Reproduced with Original Parameters	Accuracy	Precision	Recall	f-score
kNN	90.2% (N/A)	94.7% (92.1%)	94.6% (94.3%)	94.0% (90.4%)	94.0% (90.4%)	94.0% (90.3%)
DT	91.3% (N/A)	87.6% (76.0%)	95.8% (94.6%)	95.7% (94.4%)	95.6% (94.3%)	95.7% (94.3%)
MLP	87.5% (N/A)	83.6% (72.3%)	86.4% (75.9%)	86.1% (76.1%)	86.7% (75.5%)	85.7% (70.7%)
SVM	89.1% (N/A)	83.1% (72.3%)	83.3% (72.5%)	83.9% (74.6%)	83.0% (72.6%)	80.4% (66.0%)

* kNN (k Nearest Neighbors), DT (Decision Tree), MLP (Multi-Layer Perceptron), SVM (Support Vector Machine).

Table 8
Evaluation of methods based on machine learning.

Author	Classifier*	Original Results	Reproduced results on OS Name (Minor Version)			
		Accuracy	Accuracy	Precision	Recall	F-score
Beverly [60]	Bayes	N/A	37.7% (0.5%)	49.7% (92.1%)	37.6% (0.6%)	37.4% (0.3%)
Richardson [67] (limited params)	RF	98% (<30%)	94.9% (91.5%)	95.8% (94.6%)	95.8% (94.7%)	95.8% (94.6%)
	DT	N/A	94.4% (92.3%)	95.8% (94.7%)	95.8% (94.6%)	95.8% (94.6%)
	SVM	98% (<30%)	86.2% (74.9%)	84.8% (75.3%)	84.2% (71.9%)	81.8% (65.2%)
	kNN	N/A	91.2% (84.4%)	93.9% (91.2%)	94.0% (91.1%)	94.0% (91.0%)
Laštovička [70]	DT	97.6% (N/A)	84.1% (73.7%)	83.6% (81.1%)	94.3% (73.6%)	81.6% (68.5%)
	kNN	97.6% (N/A)	80.5% (71.6%)	82.9% (79.5%)	83.7% (73.2%)	81.4% (68.6%)
	SVM	97.6% (N/A)	82.5% (73.2%)	85.0% (80.4%)	82.7% (72.2%)	79.9% (66.9%)
	Bayes	81.8% (N/A)	37.2% (21.8%)	50.9% (56.3%)	37.3% (22.4%)	37.0% (23.8%)
Laštovička [45]	DT	93.1% (N/A)	82.1% (73.4%)	81.6% (71.5%)	92.4% (73.4%)	81.6% (71.4%)

* kNN (k Nearest Neighbors), DT (Decision Tree), SVM (Support Vector Machine), RF (Random Forest).

Table 9
Evaluation of methods based on manual analysis.

Author	Accuracy on OS Name (Minor Version)	
	Original Results	Reproduced
Spitzner [16]	N/A	75.8% (70.2%)
Karagiannis [20]	N/A	89.1% (87.4%)
Vymlátil [22]	N/A (89.5%)	89.4% (78.1%)
Matoušek [23]	N/A (91.7%)	89.4% (78.1%)
Jirsík [24]	N/A	84.1% (73.7%)
Tyagi [25]	N/A (95.5%)	94.2% (92.7%)
Osanaiye [26]	N/A	84.4% (73.8%)
Laštovička [27]	80.9% (N/A)	84.1% (73.7%)

type of network they classify. This narrow focus results in the method's poor performance in general use when traffic is different, which reduces their usability in real networks. When such a method is used in another type of network, its accuracy drops, and the method must be retrained on data from the new environment. Such a task is often not trivial as obtaining annotated data from a network is demanding and, in some production networks, even impossible. This leads to the need for fingerprinting methods that can identify OS accurately regardless of the network type.

OS fingerprinting requires detailed information about hosts' OS to train the fingerprinting methods and evaluate them. However, obtaining such data from a large network is a precarious task that leads researchers to estimate the ground truth from available data (i.e., User-Agents or application banners).

8.1.2. Reproducibility

The reproducibility of results is a well-known problem in scientific research. In the case of the OS fingerprinting papers we reviewed, many lacked the description of fingerprinting methods, or the description was insufficient. The source codes and datasets were not provided or no longer available, and the texts of papers often lack the settings of algorithms used.

8.1.3. Comparability

Comparing original results from the papers to others or the reproduced ones is not straightforward. A notable number of OS fingerprinting papers do not provide any accuracy metrics of their experiments and settle down with the possibility of OS identification using the method. Such lack of information leaves questions about how they were evaluated. Furthermore, OS identification's level of detail is often blurry as merging similar systems in one group is very common. This artificial boosting of accuracy leads to some OS being identified to its name with major and minor versions (i.e., Android 4.2), whereas others in the same paper are identified only as an OS family (i.e., Linux). The accuracy metric itself can be very misleading on imbalanced datasets where one target class dominates the dataset. A simple model classifying everything into one class can achieve high accuracy; therefore, other metrics should accompany accuracy to express the method performance more realistically.

8.1.4. Feature acquisition

There is a wide range of features used for OS fingerprinting, including a couple of exotic ones. The connection of features to OS and data is discussed in Section 6, but there are problems with the practicality of some features too. The measurement of them requires a specific location of the monitoring point, unrealistic precision of timing measurement, or additional computations over the raw packet data. Such requirements make it complicated to deploy in a large network and use more processing power than is available on devices commonly used for monitoring. To ensure the fingerprinting methods' usability, the measurement of features should be feasible in any network with standard equipment and should not require extensive computations.

CRediT authorship contribution statement

Martin Laštovička: Conceptualization, Methodology, Software, Validation, Investigation, Writing – original draft, Writing – review & editing, Supervision. **Martin Husák:** Conceptualization, Writing – original draft, Writing – review & editing. **Petr Velan:** Methodology, Software, Validation, Investigation, Data curation, Writing – original draft. **Tomáš Jirsík:** Conceptualization, Writing – original draft, Writing – review & editing. **Pavel Čeleda:** Supervision, Project administration, Funding acquisition, Writing – review & editing.

Declaration of Competing Interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Martin Laštovička, Martin Husák, Petr Velan, Tomas Jirsík, Pavel Čeleda report financial support was provided by Ministry of Education, Youth and Sports of the CR.

Data Availability

The data are publicly available in our university repository and Zenodo platform, and are linked from the paper.

Acknowledgement

This research was supported by ERDF "CyberSecurity, CyberCrime and Critical Information Infrastructures Center of Excellence" (No. CZ.02.1.01/0.0/0.0/16_019/0000822).

References

[1] R. Fielding, J. Reschke, Hypertext transfer protocol (HTTP/1.1): semantics and content, RFC 7231 (Proposed Standard) (2014). URL http://www.ietf.org/rfc/rfc7231.txt.
[2] R. Hofstede, P. Čeleda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, A. Pras, Flow monitoring explained: from packet capture to data analysis with NetFlow and IPFIX, IEEE Commun. Surv. Tutorials 16 (4) (2014) 2037–2064.
[3] A. Kott, C. Wang, R.F. Erbacher, Cyber Defense and Situational Awareness, 62, Springer, 2015.
[4] M. Husák, T. Jirsík, S.J. Yang, SoK: contemporary issues and challenges to enable cyber situational awareness for network security, in: Proceedings of the 15th International Conference on Availability, Reliability and Security, 2020. ACM.
[5] M. Laštovička, M. Husák, L. Sadlek, Network monitoring and enumerating vulnerabilities in large heterogeneous networks, in: NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium, 2020. IEEE.
[6] Titan H.Q., SpamTitan Passive OS fingerprinting, [cited 2020-09-17] (2018). URL https://www.manula.com/manuals/menlo-park-tech/spamtitan-administrator-guide/1/en/topic/passive-os-fingerprinting.
[7] J.M. Allen, OS and application fingerprinting techniques, SANS Institute InfoSec Reading Room (2007).
[8] J.P.S. Medeiros, A. de Medeiros Brito Júnior, P.S. Motta Pires, A qualitative survey of active TCP/IP fingerprinting tools and techniques for operating systems identification, in: Á. Herrero, E. Corchado (Eds.), Computational Intelligence in Security for Information Systems, Springer, Berlin Heidelberg, 2011, pp. 68–75.
[9] N. Provos, T. Holz, Virtual Honeypots: From Botnet Tracking to Intrusion Detection, 1st Edition, Addison-Wesley Professional, 2007.
[10] D. Herrmann, K.-.P. Fuchs, H. Federrath, Fingerprinting Techniques For Target-Oriented Investigations in Network forensics, Sicherheit 2014– Sicherheit, Schutz und Zuverlässigkeit, 2014.
[11] Y. Liu, J. Wang, J. Li, S. Niu, H. Song, Machine learning for the detection and identification of internet of things devices: a survey, IEEE Internet Things J. 9 (1) (2021) 298–320.
[12] P.M.S. Sánchez, J.M.J. Valero, A.H. Celdrán, G. Bovet, M.G. Pérez, G.M. Pérez, A survey on device behavior fingerprinting: data sources, techniques, application scenarios, and datasets, IEEE Commun. Surv. Tutorials (2021).
[13] Q. Xu, R. Zheng, W. Saad, Z. Han, Device fingerprinting in wireless networks: challenges and opportunities, IEEE Commun. Surv. Tutorialss 18 (1) (2015) 94–104.
[14] M. Albanese, E. Battista, S. Jajodia, A deception based approach for defeating OS and service fingerprinting, in: 2015 IEEE Conference on Communications and Network Security (CNS), 2015, pp. 317–325.
[15] P. Velan, M. Cermák, P. Celeda, M. Drašar, A survey of methods for encrypted traffic classification and analysis, Int. J. Network Manage. 25 (5) (2015) 355–374.
[16] L. Spitzner, Passive fingerprinting, FOCUS on intrusion detection: passive fingerprinting (May 3, 2000) (2000) 1–4.
[17] M. Zalewski, p0f v3, [cited 2022-03-02] (2012). URL http://lcamtuf.coredump.cx/p0f3/.
[18] M. Beddoe, The Siphon project: the passive network mapping tool, [cited 2022-03-02] (2011). URL https://github.com/unmarshal/siphon.
[19] R. Lippmann, D. Fried, K. Piwowarski, W. Streilein, Passive operating system identification from TCP/IP packet headers, in: Workshop on Data Mining for Computer Security, 2003, p. 40.
[20] T. Karagiannis, K. Papagiannaki, N. Taft, M. Faloutsos, Profiling the end host, in: International Conference on Passive and Active Network Measurement, 2007, pp. 186–196. Springer.
[21] S. Mossel, Passive OS detection by monitoring network flows, DLib Magazine (2012).
[22] M. Vymlátil, Detection of Operation Systems in Network Traffic Using IPFIX, Brno University of Technology Thesis, 2014.
[23] P. Matoušek, O. Ryšavý, M. Grégr, M. Vymlátil, Towards identification of operating systems from the internet traffic: ipfix monitoring with fingerprinting and clustering, in: 2014 5th International Conference on Data Communication Networking (DCNET), 2014, pp. 1–7. IEEE.
[24] T. Jirsík, P. Čeleda, Identifying operating system using flow-based traffic fingerprinting. Advances in Communication Networking, Springer International Publishing, 2014, pp. 70–73.
[25] R. Tyagi, T. Paul, B. Manoj, B. Thanudas, Packet Inspection for Unauthorized OS Detection in Enterprises, IEEE Secur Priv 13 (4) (2015) 60–65.
[26] O.A. Osanaiye, M. Dlodlo, TCP/IP header classification for detecting spoofed DDoS attack in Cloud environment, in: IEEE EUROCON 2015 - International Conference on Computer as a Tool, 2015, pp. 1–6.
[27] M. Laštovička, T. Jirsík, P. Čeleda, S. Špaček, D. Filakovský, Passive OS fingerprinting methods in the jungle of wireless networks, in: NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium, 2018, pp. 1–9. IEEE.
[28] M. Laštovička, D. Filakovský, Passive OS fingerprinting prototype demonstration, in: NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium, 2018. IEEE.
[29] T. Al-Shehari, F. Shahzad, Improving operating system fingerprinting using machine learning techniques, Int. J. Comput. Theory Eng. 6 (1) (2014) 57.
[30] J. Barnes, P. Crowley, K-p0F: a High-throughput kernel passive OS fingerprinter, in: Proceedings of the Ninth ACM/IEEE Symposium on Architectures for Networking and Communications Systems, ANCS '13, 2013, pp. 113–114. IEEE.
[31] Ornaghi, Alberto and Valleri, Marco and Escobar, Emilio and Costamagna, Gianfranco and Koeppe, Alexander and Abdulkadir, Ali, Ettercap project, [cited 2022-03-02] (2001). URL https://www.ettercap-project.org/.
[32] A. Ornaghi, M. Valleri, E. Escobar, E. Milam, G. Costamagna, A. Koeppe, Ettercap, [cited 2022-03-02] (2011). URL https://github.com/Ettercap/ettercap.
[33] NetGrab, Netsleuth, [cited 2020-09-04] (2012). URL http://netgrab.co.uk/netsleuth/.
[34] E.B. Fjellskål, K. Wysocki, PRADS - passive real-time asset detection system, [cited 2022-03-02] (2009). URL https://github.com/gamelinux/prads.
[35] E. Kollmann, Satori, [cited 2023-04-03] (2018). URL https://github.com/xnih/satori.
[36] E. Hjelmvik, Networkminer, [cited 2022-03-02] (2007). URL https://www.netresec.com/?page=Networkminer.
[37] Juniper Networks, Inc., Configuring profiler options (NSM Procedure), [cited 2022-03-02] (2013). URL https://www.juniper.net/documentation/en_US/nsm2012.2/topics/task/configuration/firewall-profiler-option-configuring-nsm.html.
[38] Cisco Systems, Inc., User guide for cisco security manager 4.7, [cited 2020-09-17] (2009). URL https://www.cisco.com/c/en/us/td/docs/security/security_management/cisco_security_manager/security_manager/4-7/user/guide/CSMUserGuide/ipsevact.html#100539.
[39] Cisco Systems, Inc., Next-gen firewall, [cited 2022-03-03] (2006). URL https://meraki.cisco.com/technologies/next-gen-firewall.
[40] AT&T Business, Asset discovery, [cited 2022-03-02] (2020). URL https://cybersecurity.att.com/solutions/asset-discovery-inventory.
[41] AT&T Business, AlienVault OSSIM, [cited 2022-03-02] (2019). URL https://cybersecurity.att.com/products/ossim.
[42] Inverse inc., Fingerbank, [cited 2022-03-02] (2014). URL https://fingerbank.org/.
[43] Inverse inc., Fingerbank Github, [cited 2022-03-02] (2014). URL https://github.com/karottc/fingerbank.
[44] S. Shah, HTTP fingerprinting and advanced assessment techniques, BlackHat Asia (2003).
[45] M. Laštovička, S. Špaček, P. Velan, P. Čeleda, Using TLS fingerprints for OS identification in encrypted traffic, in: NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium, 2020. IEEE.
[46] M. Husák, M. Čermák, T. Jirsík, P. Čeleda, Network-based HTTPS client identification using SSL/TLS fingerprinting, in: 2015 10th International Conference on Availability, Reliability and Security, 2015, pp. 389–396. IEEE.
[47] M. Husák, M. Cermák, T. Jirsík, P. Celeda, HTTPS tra ffic analysis and client identification using passive SSL/TLS fingerprinting, EURASIP J. Inf. Secur. 2016 (1) (2016) 6.
[48] T. Matsunaka, A. Yamada, A. Kubota, Passive OS fingerprinting by DNS traffic analysis, in: Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference On, 2013, pp. 243–250. IEEE.
[49] D. Chang, Q. Zhang, X. Li, Study on OS fingerprinting and NAT/Tethering based on DNS log analysis, in: IRTF & ISOC Workshop on Research and Applications of Internet Measurements (RAIM), 2015.
[50] I. Voronov, K. Gnezdilov, Determining OS and applications by DNS traffic analysis, in: 2021 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus), 2021, pp. 72–76. IEEE.
[51] A. Aksoy, M.H. Gunes, Operating system classification performance of TCP/IP protocol headers, in: Local Computer Networks Workshops (LCN Workshops), 2016 IEEE 41st Conference on, 2016, pp. 112–120. IEEE.
[52] J. Muehlstein, Y. Zion, M. Bahumi, I. Kirshenboim, R. Dubin, A. Dvir, O. Pele, Analyzing https encrypted traffic to identify user's operating system, browser and application, in: 2017 14th IEEE Annual Consumer Communications & Networking Conference (CCNC), 2017.
[53] X. Fan, G. Gou, C. Kang, J. Shi, G. Xiong, Identify os from encrypted traffic with tcp/ip stack fingerprinting, in: 2019 IEEE 38th International Performance Computing and Communications Conference (IPCCC), 2019, pp. 1–7. IEEE.
[54] A. Aksoy, S. Louis, M.H. Gunes, Operating system fingerprinting via automated network traffic analysis, in: 2017 IEEE Congress on Evolutionary Computation (CEC), 2017, pp. 2502–2509.
[55] T. Kohno, A. Broido, K.C. Claffy, Remote physical device fingerprinting, IEEE Trans. Dependable Secure Comput. 2 (2) (2005) 93–108.
[56] A. Azzouni, O. Braham, T.M.T. Nguyen, G. Pujolle, R. Boutaba, Fingerprinting OpenFlow controllers: the first step to attack an SDN control plane, in: 2016 IEEE Global Communications Conference (GLOBECOM), 2016, pp. 1–6. IEEE.
[57] J. Gurary, Y. Zhu, R. Bettati, Y. Guan, Operating system fingerprinting. Digital Fingerprinting, Springer, 2016, pp. 115–139.
[58] C. Shen, C. Liu, H. Tan, Z. Wang, D. Xu, X. Su, Hybrid-augmented device fingerprinting for intrusion detection in industrial control system networks, IEEE Wirel. Commun. 25 (6) (2018) 26–31.
[59] I. Sanchez-Rola, I. Santos, D. Balzarotti, Clock around the clock: timebased device fingerprinting, in: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS '18, 2018, pp. 1502–1514. ACM.
[60] R. Beverly, A robust classifier for passive TCP/IP fingerprinting. International Workshop on Passive and Active Network Measurement, Springer, 2004, pp. 158–167.
[61] J. Caballero, S. Venkataraman, P. Poosankam, M.G. Kang, D. Song, A. Blum, FiG: Automatic fingerprint Generation, Carnegie Mellon University, 2007.
[62] H.J. Abdelnur, O. Festor, et al., Advanced Network fingerprinting, in: International Workshop On Recent Advances in Intrusion Detection, Springer, 2008, pp. 372–389.
[63] B. Zhang, T. Zou, Y. Wang, B. Zhang, Remote operation system detection base on machine learning, in: 2009 Fourth International Conference on Frontier of Computer Science and Technology, 2009, pp. 539–542.
[64] F. Gagnon, B. Esfandiari, A hybrid approach to operating system discovery based on diagnosis, Int. J. Network Manage. 21 (2) (2011) 106–119.

[65] F. Gagnon, B. Esfandiari, A hybrid approach to operating system discovery based on diagnosis theory, in: 2012 IEEE Network Operations and Management Symposium, IEEE, 2012, pp. 860–865.
[66] F. Massicotte, F. Gagnon, Y. Labiche, L. Briand, M. Couture, Automatic evaluation of intrusion detection systems, in: 2006 22nd Annual Computer Security Applications Conference (ACSAC'06), IEEE, 2006, pp. 361–370.
[67] D.W. Richardson, S.D. Gribble, T. Kohno, The limits of automatic OS fingerprint generation, in: Proceedings of the 3rd ACM workshop on Artificial intelligence and security, ACM, 2010, pp. 24–34.
[68] Z. Shamsi, D.B. Cline, D. Loguinov, Faulds: a non-parametric iterative classifier for Internet-wide OS fingerprinting, in: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, 2017, pp. 971–982.
[69] M. Laštovička, T. Jirsík, P. Čeleda, S. Špaček, D. Filakovský, PassiveOSFingerprint, [cited 2022-02-03] (2018). URL https://github.com/CSIRT-MU/PassiveOSFingerprint.
[70] M. Laštovička, A. Dufka, J. Komárková, Machine learning fingerprinting methods in cyber security domain: which one to use?, in: 2018 14th International Wireless Communications & Mobile Computing Conference (IWCMC), 2018, pp. 542–547. IEEE.
[71] M. Laštovička, S. Špaček, P. Velan, P. Čeleda, Dataset using TLS fingerprints for OS identification in encrypted traffic (2019). doi:10.5281/zenodo.3461771. URL http://doi.org/10.5281/zenodo.3461771.
[72] D.H. Hagos, M. Løland, A. Yazidi, Ø. Kure, P.E. Engelstad, Advanced passive operating system fingerprinting using machine learning and deep learning, in: 2020 29th International Conference on Computer Communications and Networks (ICCCN), 2020, pp. 1–11. IEEE.
[73] D.H. Hagos, A. Yazidi, Ø. Kure, P.E. Engelstad, A machine-learning based tool for passive os fingerprinting with tcp variant as a novel feature, IEEE Internet Things Journal 8 (5) (2020) 3534–3553.
[74] R. Pérez-Jove, C.R. Munteanu, A.P. Sierra, J.M. Vázquez-Naya, Applying artificial intelligence for operating system fingerprinting, Eng. Proc. 7 (1) (2021) 51.
[75] S. Salah, M. Abu Alhawa, R. Zaghal, Desktop and mobile operating system fingerprinting based on ipv6 protocol using machine learning algorithms, Int. J. Secur. Netw. 17 (1) (2022).
[76] Z. Shamsi, D.B. Cline, D. Loguinov, Faulds: a non-parametric iterative classifier for internet-wide os fingerprinting, IEEE/ACM Trans. Network. 29 (5) (2021) 2339–2352.
[77] K. Nichols, S. Blake, F. Baker, D.L. Black, RFC 2474: definition of the differentiated services field (DS Field) in the IPv4 and IPv6 Headers (Dec. 1998). URL https://tools.ietf.org/html/rfc2474.
[78] K.K. Ramakrishnan, S. Floyd, D.L. Black, RFC 3168: the Addition of explicit congestion notification (ECN) to IP (Sep. 2001). URL https://tools.ietf.org/html/rfc3168.
[79] G. Fairhurst, RFC 8436: update to IANA registration procedures for Pool 3 values in the differentiated services field codepoints (DSCP) registry (Aug. 2018). URL https://tools.ietf.org/html/rfc8436.
[80] E. Rescorla, The transport layer security (TLS) protocol version 1.3, RFC 8446 (2018). URL https://tools.ietf.org/html/rfc8446.
[81] D. Benjamin, RFC 8701: applying generate random extensions and sustain extensibility (GREASE) to TLS extensibility (Jan. 2020). URL https://tools.ietf.org/html/rfc8701.
[82] Microsoft Documentation, W3C logging, [cited 2022-03-02] (2018). URL https://docs.microsoft.com/en-us/windows/win32/http/w3c-logging.
[83] G. Vormayr, J. Fabini, T. Zseby, Why are my flows different? A tutorial on flow exporters, IEEE Commun. Surv. Tutorialss 22 (3) (2020) 2064–2103.
[84] Google Inc., uap-python: a python implementation of the UA Parser, [cited 2022-03-02] (2015). URL https://github.com/ua-parser/uap-python.
[85] M. Laštovička, M. Husák, P. Velan, T. Jirsík, P. Čeleda, OS fingerprinting dataset, [cited 2022-03-02] (2021). URL https://is.muni.cz/www/lastovickam/public/Dataset_OS_Fingerprinting.zip.
[86] Scikit-learn, Standard Scaler, [cited 2022-03-02] (2007). URL https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html.
[87] Scikit-learn, Scikit-learn: machine learning in Python, [cited 2022-0302] (2007). URL https://scikit-learn.org/stable/index.html.
[88] M. Sokolova, G. Lapalme, A systematic analysis of performance measures for classification tasks, Inf. Process Manag. 45 (4) (2009) 427–437.
[89] G.F. Lyon, Nmap network scanning: the official Nmap project guide to network discovery and security scanning, Insecure. Com LLC (US) (2008).
[90] M. Laštovička, M. Husák, P. Velan, T. Jirsík, P. Čeleda, Passive operating system fingerprinting revisited - network flows dataset (2023). 10.5281/zenodo.7635138. URL 10.5281/zenodo.7635138.
[91] D. Zhang, Q. Wang, Z. Wei, S. Chen, An Operating system identification method based on active learning, in: 2022 International Conference on Electrical, Computer and Energy Technologies (ICECET), 2022, pp. 1–6.
[92] M. Grill, M. Rehák, Malware detection using http user-agent discrepancy identification, in: 2014 IEEE International Workshop on Information Forensics and Security (WIFS), IEEE, 2014.
[93] A. Adhikari, Device identification from network traffic measurements-A HTTP user agent based method, Aalto Univ. School Electr. Eng. (2012).
[94] J. Chen, G. Gou, G. Xiong, An analysis of anomalous user agent strings in network traffic, in: 2019 IEEE 21st International Conference on High Performance Computing and Communications, IEEE, 2019.

Martin Laštovička obtained his Ph.D. in Informatics at the Faculty of Informatics, Masaryk University, Czech Republic. He works as the head of the cybersecurity operations group in CSIRT-MU. His research topic lies in network traffic analysis and practical applications of machine learning to build Cyber Situational Awareness through the identification of network entities and their relationships. His focus is to apply research outputs to real-world data and enhance operations of the CSIRT-MU team.

Tomáš Jirsík obtained his Ph.D. in Informatics from the Faculty of Informatics, Masaryk University, Czech Republic. He is currently a senior cybersecurity data analyst at Cisco Systems. Previously, he was a senior researcher at the Institute of Computer Science at Masaryk University and a member of the Computer Security Incident Response Team of Masaryk University (CSIRT-MU). His research focus lies in network traffic analysis with a specialization in host profiling. His research further includes network segmentation approaches via machine learning and host fingerprinting in network traffic.

Martin Husák is a researcher at the Institute of Computer Science at Masaryk University, a member of the university's security team (CSIRT-MU), and a contributor to The Honeynet Project. His Ph.D. thesis addressed the problem of early detection and prediction of network attacks using information sharing. His research interests are related to cyber situational awareness and threat intelligence with a special focus on the effective sharing of data from honeypots and network monitoring.

Pavel Čeleda is an associate professor at Masaryk University. He received a Ph.D. degree in Informatics from the University of defence, Brno. His main research interests include traffic analysis, situational awareness, and cybersecurity testbeds for research and education. The research topics are subject of many projects, collaborations, and Ph.D. dissertations. He is a principal investigator of the KYPO cyber range project and co-PI of the C4e center of excellence.

Petr Velan obtained his Ph.D. in Informatics at the Faculty of Informatics, Masaryk University, Czech Republic. He is currently a senior researcher at the Institute of Computer Science at Masaryk University and a member of the Computer Security Incident Response Team of Masaryk University (CSIRT-MU), where he participates on cybersecurity research projects. His research focus lies on the network traffic monitoring and analysis with a specialization in network flow monitoring.