Passive OS Fingerprinting Methods in the Jungle of Wireless Networks

Martin Lastovicka*†, Tomas Jirsik*†, Pavel Celeda*, Stanislav Spacek*† and Daniel Filakovsky*†
*Masaryk University, Institute of Computer Science, Brno, Czech Republic
†Masaryk University, Faculty of Informatics, Brno, Czech Republic
Email: {lastovicka|jirsik|celeda|spaceks}@ics.muni.cz, filakovsky@mail.muni.cz

***Abstract*—Operating system fingerprinting methods are well-known in the domain of static networks and managed environments. Yet few studies tackled this challenge in real networks, where users can bring and connect any device. We evaluate the performance of three OS fingerprinting methods on a large dataset collected from university wireless network. Our results show that method based on HTTP User-agents is the most accurate but can identify only low portion of the traffic. TCP/IP parameters method proved to be the opposite with high coverage but low accuracy. We also implemented a new method based on detection of communication to OS-specific domains. Its performance is comparable to the two established ones. Next, we discuss the impacts of traffic encryption and embracing new protocols such as IPv6 or HTTP/2.0 on OS fingerprinting. Our findings suggest that OS identification based on specific domain detection is viable and corresponds to the current directions of network traffic evolution, while methods based on TCP/IP parameters and User-agents will become ineffective in the future.**

I. Introduction

The trend of modern networks is to allow users to bring and connect any device to the network. Such freedom is convenient for the users but can introduce new security risks to the network. To protect it from threats, it is necessary to know what devices are currently connected and keep track of changes in the network. Operating system identification is the very first step in understanding the situation in the network. However, most research in this field often aims at static networks with servers and desktops, whereas current network solutions move towards dynamic address assigning to any type of connected device.

Passive OS fingerprinting is a transparent method for OS identification. Each OS has specific settings which leave a fingerprint in packets sent by the system [1]. The passive OS identification analyses packets originating from a system. Based on the fingerprint found in a packet, a particular OS is identified. An alternative approach to the passive OS identification is the active OS identification where OS is determined by the active probing of the system. A comparison of the active and the passive OS detection in term of precision is provided in [2].

We have formulated four use-cases where security managers can benefit from OS fingerprinting in the context of dynamic networks:

- *Unsupported OS* – rapid development of OS versions lead to many users still using an outdated version without security updates. This problem is obvious especially in mobile OS where 9 % of Android users have version older than 4.4 [3].
- *Decision making* – knowledge of what devices are connected helps security administrators to take the right decision and react to incidents more precisely to security incidents [4].
- *BYOD security policy* – Bring Your Own Device is the principle of dynamic networks, but some networks have policy restrictions on which devices can users bring or on using only specific systems.
- *Static networks* – any change of device behind IP address or disallowed OS in the segment can indicate a rogue device and should be investigated. Methods designed for dynamic networks are implicitly capable of such detection.

The results of our work aids to understand behaviour of the passive OS fingerprinting methods in the jungle of wireless networks. Based on our work, an administrator can objectively choose a suitable OS fingerprinting method that fits his needs and use-cases in terms of accuracy, precision and recall.

In our work, we study methods for passive OS fingerprinting, specifically the methods using flow-based passive network monitoring. Passive monitoring was chosen because it can monitor the network continuously and does not depend on any schedule (e.g., daily) of active scans. This is important in dynamic networks where devices behind an IP address can change rapidly and at irregular intervals. The network flow technology allows us to reduce the computational complexity of the monitoring and deploy the fingerprinting methods on the backbone of the network.

We implemented three OS identification methods based on HTTP User-agents parsing, TCP/IP parameters fingerprint [5] and communication with specific domains [6]. Next, we experimented with a combination of the methods so that it would benefit from each one. Finally we describe how we transformed the usually single packet inspection methods into the notion of network flows.

We collected flow data from the Wi-Fi subnets of university campus network and enriched them with logs from DHCP and Radius servers to get a baseline for the experiments. On this dataset, we measure for how many connected devices we

978-1-5386-3416-5/18/$31.00

can identify OS and we calculated the performance metrics (accuracy, precision, recall, f-score) for each method used.

Our main contribution is the comparison of OS fingerprinting methods performance in dynamic networks. We extended the work on a new OS identification method detecting communication with specific domains and we publish our fingerprint databases together with dataset collected from network probes and management server logs. Furthermore, we discuss the pros and cons of the methods according to our results and we identify the future challenges in OS fingerprinting arising from new network protocols and trends in modern networks.

The rest of this paper is structured as follows. Section II reviews current knowledge about OS fingerprinting. In section III we describe how the methods were implemented and how we measure packet header fields used to identify operating system. Collected dataset is presented in Section IV and results of methods on this dataset are summed up in Section V. Discussion of results and methods is in Section VI and conclusions in Section VII.

II. Related Work

The majority of information for an OS fingerprint is collected from TCP/IP packet headers and application protocols. An approach using TCP/IP packet header is well investigated. Lipmann et al. [5] present classifiers capable of identifying 9 classes of OS from packet headers. The classifiers leverage machine learning techniques including cross-validation testing. Tygai et al. [7] employs a SYN packet properties such as TTL, packet length, TCP window size setting and TCP options. Their classifier based on Euclidian distance can correctly identify 95.5 % of operating systems from nearly 2000 SYN packets.

From application protocols, information such as HTTP User-agent, HTTP domain, or DNS queries can be used. Authors of [8] utilize characteristics of DNS queries specific to individual OS, e.g. unique domain names, query patterns, and time intervals. Husak et al. [9] address a current challenge for OS identification from application protocol - network traffic encryption. They present an approach combining TLS fingerprint with a database of HTTP User-agents to identify OS even in encrypted traffic.

Machine learning (ML) techniques are used to generate new OS features and signatures. Aksoy et. al. [10] employ machine learning algorithms to identify packet features suitable for OS detection. They use genetic algorithms for feature subset selection in three ML algorithms. The combination of automatic feature selection and ML algorithms enables the adaptive OS detection and reduces the number of packet features needed for OS classification. Support Vector Machine for remote OS recognition is proposed in work of Zhang et. al. [11]. Their SVM-based OS detection is claimed to be effective in the discovery of new OS signatures.

A significant part of the literature investigates the possibility of OS identification at large scale. For this use case, network flows are mostly used as they significantly reduce the volume of analysed information. Jirsik et. al. [12] proposes a high-performance flow-based system for OS detection with 95.9 % identification rate. Mossel [6] investigates the differences between updates procedures of operating systems at network flow level. The update procedures are thoroughly described for all main OS families. The author concludes, that it is possible to identify main families of OS using the update procedure signatures in network flows. Matousek et al. [13] present an approach using a combination of TCP SYN packet-based identification with additional HTTP header check. They enhance standard IPFIX records by additional information to allow large-scale detection. They further introduce fall-back loop from data collector to the network probe in order to keep OS fingerprint database updated. The limitations of automatic OS fingerprint are discussed by Richardson in [14] and they identify four major challenges for automatic OS detection. The first challenge lies the inability of current tools to find a generalizable and sufficiently discriminative classification rules for different OS versions. Secondly, fully automatic tools cannot easily exploit semantic knowledge of protocols or generate multi-packet probes and attributes. The remaining challenges are the ineffectiveness of the tools in real networks and overfitting of the detection techniques.

The OS identification itself often serves for more complex security challenges. Osanaiye and Dlodlo [15] show how to use TTL values to identify OS of a host in a cloud to discover the origin of a DDoS attack. The OS detection can be leveraged as an identification of vulnerable SDN controllers as shown by Azzouni et al. [16].

Although many OS identification methods have been developed, there exist only few tools for passive OS identification. The best-known tools are *p0f* and *Ettercap*. *p0f* [17] is a passive TCP/IP stack fingerprinting tool. It uses a database of fingerprint descriptions stored in a text file to identify an OS. An advanced version of p0f called *k-p0f* [18] provides increased throughput by implementation on the Linux Kernel level. The k-p0f tool can process up to 450 kpps. *Ettercap* is an open-source tool originally designed for man-in-the-middle attacks. However, one of its main functionalities is OS fingerprinting and generating host profiles.

III. Fingerprinting Methodology

In this section, we describe traffic monitoring and implementation details of OS identification methods. Each method has its own limitation in how much information it can provide about the OS. To unify this information levels, we introduce a hierarchical model of operating systems. The OS hierarchy is based on four levels of details which serve as a common ground to compare the OS identification methods:

1) *Operating system* – formal root of the hierarchical structure.
2) *Vendor* – the most general family of OS based on its vendor. Operating systems from one vendor often share the same kernel or applications and exhibit similar network behaviour.
3) *OS name* – name assigned by the vendor.

4) *Major version* – version of OS that is usually used for its identification.
5) *Minor version* – sub-version of OS; typically carries only minor improvements and changes to the OS.

To monitor network traffic, we use extended network flows [19]. A network flow is an abstraction of a network connection. It is defined as a set of packets or frames passing an observation point in the network during a certain time interval. All packets belonging to a particular flow have a set of common properties called *flow keys*. A basic set of flow keys consists of source and destination IP address, source and destination transport port, and transport protocol.

Network flow contains the basic set of keys usually complemented by statistics about the number of transmitted packets and bytes, timestamps, or TCP flags used. An *extended IP flow* [19] refers to a network flow with basic flow keys that is extended with additional information, usually from application layer. A common information added to a flow is data from HTTP protocol (domain, host, referrer, User-agent) and DNS protocol (DNS query, query type, DNS response). Further protocols, such as email protocols (SMTP, POP3, IMAP), Samba, or SSL/TLS can also be analysed and relevant information added to extended flows. In our work, we capture flows and export them using IP Flow Information Export protocol (IPFIX) [20]. We choose IPFIX as it enables to transport flows with variable length keys and allows to add new information elements. We collect flow extensions for TCP/IP parameters, HTTP User-agent, HTTP hostname, and TLS Service Name Indication (SNI).

The TCP/IP parameters are measured from the first SYN packet recorded in a flow. Specifically, we use extensions taken from IP protocol header – packet Time to Live value and the size of the first packet (IP Total Length). From the TCP header, we measure the TCP Window Size header field.

User-agent string is taken from the corresponding field of HTTP protocol and the probe parses the whole string to extend the flow records by information about operating system and application that generated the string.

The last 32 bytes of HTTP hostname from the packet header are extended to the flow. SNI is a value taken from TLS header field Server Name Indication Extension. This field is not usually used in flow monitoring and has not yet been specified as IANA IPFIX element [21], but it is virtually the same thing as HTTP hostname and will gain more importance with the spread of web traffic encryption.

A. TCP/IP Parameters

OS identification based on analysis of TCP/IP parameters depends on which header fields are considered. The process of feature selection is a common challenge during OS fingerprinting with TCP/IP stack parameters. The features range from the simple use of Time to Live value to analysis of almost every bit in IP packet header. Especially machine learning models for feature extraction often chose parameters that depend on both communicating parties or the application (e.g., source/destination port numbers, TCP flags, checksum) rather than the OS itself [22]. We select features that depend solely on the OS kernel and can be used directly for its identification. Moreover, we want to use the lowest possible number of parameters while still being able to distinguish different versions of operating systems. Our selected features are listed below:

- Initial SYN packet size (synSize) – the size of the first packet of TCP connection is influenced by the specific implementation of the operating system.
- TCP window size (winSize) – initial value of window size specifies the number of octets the receiving side is currently prepared to receive [23]. Default value comes from the OS settings.
- Time to Live (TTL) – IPv4 protocol value initially set by the OS kernel which is decremented by one on each network element to prevent endless routing of packets. We round the captured value to the next higher power of two so that it is not affected by observation point location as suggested by Lippman et al. [5].

The next step after selecting features is to prepare a database which maps specific parameter values to the operating system. Our approach to creating such fingerprint database is similar to the one suggested by Matousek et al. [13] which maps information gained from HTTP User-agents to TCP/IP parameters. We have processed flow data from the whole university during two months to cover as many different devices as possible, aggregated them into groups with the same triple (synSize, winSize, TTL), and finally assigned the corresponding OS to each triple.

Following this process, we have created a database of 2078 unique mappings from TCP/IP parameters to 51 unique operating systems and their major and minor versions (when available) weighted by their appearance (i.e., number of flows with the same triple) in our network. This weighting called *confidence* is necessary to deal with different operating systems or their versions that send packets with the exact same triple of TCP parameters. We compute confidence as the fraction of the number of flows of a specific OS version compared to the total number of flows with the same triple.

Example of our fingerprint database is listed in Table I. The whole database is publicly available on GitHub [24].

TABLE I
EXAMPLE FINGERPRINT DATABASE

synSize	winSize	TTL	OS	Confidence
52	8192	128	Windows 10.0	55.2 %
52	8192	128	Windows 6.1	31.9 %
52	65535	128	Windows 10.0	74.9 %
60	65535	64	Android 6.0	48.2 %
60	14600	64	Android 4.4	28.4 %
60	29200	64	Ubuntu	20.4 %
64	65535	64	Mac OS X 10.12	26.5 %
64	65535	64	iOS 10.3	10.3 %

The last step in method implementation is the actual OS identification. For a flow with all three parameters included, we match OS from the fingerprint database and assign the

one with the highest confidence. Using this approach, the algorithm can decide even if it gets more possibilities for operating system. Its decision would be the one with the highest probability to be correct, i.e. the highest confidence.

To validate our measurement, we compare our fingerprint DB to databases of p0f and Ettercap. Surprisingly, neither of them uses the size of initial SYN packet and the comparison is limited to only two parameters. The characteristics of main operating systems (Windows, MAC OS X) are the same. However, the p0f and Ettercap DBs generally lack updates (the last update of fingerprints on GitHub was 21 May 2014 for p0f and respectively 26 Oct 2011 for Ettercap). Because of this outdated fingerprint DBs, new systems like Windows 10 or Android 4.4 and higher, that currently dominate the network traffic, are not included and hence not recognized by the tools.

Creating a complete, up-to-date database of fingerprints is a challenging task. Our goal is to create a database containing fingerprints of as many different systems as possible. We decided not to use any strictly managed environment as there would be lack of desired diversity. Instead, we focused on processing data from User-agents which can be done in large scale and covers most currently used systems.

Our fingerprint DB contains confidence rating calculated from the large volume of network traffic. This allows us to deterministically identify OS of a flow even if the DB contains more records for the same triple. However, this approach results in identification biased by current market share of each OS within our university population where an uncommon OS can be overshadowed by a popular one. We identify this fact as a general limit of TCP/IP parameters identification method because every fingerprint database must deal with parameter collisions.

B. HTTP User-Agent

HTTP User-agent is defined in [25] as a string from user originating the HTTP request to provide information about operating system and browser to the server. The purpose of such identification is that a web server can serve content customized for a specific device or software and increase the user experience while browsing web pages. Nowadays, it is typical for web pages to have different versions for mobile and desktop devices.

The User-agent string construction is fully under control of application software independent of the underlying operating system, as User-agent belongs to the application layer of network communication. Moreover, the HTTP/1.1 specification [25] explicitly states that User-agent should not be generated with needless details to prevent user fingerprinting. However, in practice, it is common that applications fill in the operating system name with its major and minor version and often even with the specific build of that version.

To identify operating system from the provided string, a parser has to be implemented. We use commercial Flowmon probes [26] to capture flow data and these probes have built-in capabilities for User-agent parsing. When a packet with User-agent is captured, it is compared to probe's User-agent database and OS name, major and minor version, and build values are assigned to the whole flow. When an unknown pattern of User-agent is encountered, the flow is treated as no User-agent was present at all.

Even if the parser is a part of a commercial product, we believe its results are comparable to open-source libraries for parsing User-agents. Moreover, the probes receive periodical updates which (should) contain updates of the parser database and keep the results up-to-date with new operating systems and their versions.

C. Specific Domains

Modern operating systems are configured to do many specific actions upon connecting to a network. These actions include connectivity testing (e.g., *connectivity-check.android.com*) and checking for system updates (e.g., *update.microsoft.com*). This activity can be monitored on the level of DNS communication during name resolving, or as the communication to external servers. We use the latter variant as it is more straightforward to monitor with extended flow data. Both connectivity checks and updates are realized as web traffic over HTTP(S) which can be detected by the request hostname or SNI.

We note that similar identification could be done using IP addresses of the servers. But this approach would be hard to implement and maintain through time as current cloud solutions change the servers IP regularly over time and according to client geographic location. From this point of view, methods based on hostname/SNI are more viable.

We have prepared a database connecting domains to operating systems following two approaches. At first, we tried to find domains of interest manually. We reviewed technical documentation and developer guidelines from the major OS vendor and were looking for details of update process and connectivity check implementations. After that, we focused on blogs and forums where administrators were solving problems with firewall configurations. These proved very useful as blocking of a specific domain disrupts OS functionality and reveals the real domains used.

Secondly, we tried to aggregate traffic over OS from User-agent and hostnames just like in TCP/IP parameters database. This approach does not work directly (as it is flooded by user activity), but it helps us to identify more specific domains that have not been found in the documentation. We manually evaluated their purpose and added the ones connected to an OS activity to the database. We have published this database on Github [24].

Employment of these methods corresponds to current development of network communication. Cloud synchronization and updates will increase in use and ensure this method to be even more relevant. We also believe that the domain names will not change rapidly (e.g., Microsoft uses domain *msftncsi.com* for more than 8 years without a change throughout all Windows versions [27]) and once a mapping is created, it won't need frequent updates.

Moreover, this method overcomes issues with the deployment of IPv6 in end networks and communication encryption. While IPv6 forces changes in TCP/IP parameters semantics (see Section VI) it will have no effect on specific domains as their monitoring is solely on the application layer. Encrypted communication will deny the possibility of packet inspection, but specific domains method can work monitoring SNI field.

D. Combination of Methods

By combining all previously mentioned methods, we implemented a method that benefits from the advantages of the individual methods and is not limited to any specific network layer. Each method is used to identify the OS individually and their results are then combined. We treat the methods equally and the final decision is based on majority voting principle.

In the case when each method has different result or only two methods are able to identify OS and they disagree, the results are taken in the order of User-agent, Specific domains, and TCP/IP parameters. We have decided for this order because TCP/IP parameters are often the same for multiple operating systems and the decision is based on the highest probability to be correct. User-agents are then preferred over specific domains as it is a long-established method.

Each method has different level of details that it could provide about the OS. For their comparison later in this work, we need to set a common level based on the OS hierarchy. The level of detail each method can provide is summarised in Table II. TCP/IP method can provide major and minor version information, but parameters are often the same for multiple versions and can lead to overfitting of identification. Hence, the levels could be omitted and are marked in brackets.

IV. Dataset

The dataset covers data from all subnets of our university wireless network (Eduroam) including buildings of multiple faculties and dormitories. It contains data from the first week in May 2017 and is a combination of three data sources – flow data, Radius logs, and DHCP logs. In total, our dataset covers:

- 79 087 345 flows in IPFIX format,
- activity of 21 746 unique users,
- 253 374 Wi-Fi sessions,
- 25 642 unique MAC addresses (1692 vendor prefixes),
- 6 104 unique IP addresses assigned.

TABLE II
OS identification methods level of detail

Method	Vendor	OS name	Major version	Minor version
User-agent	✓	✓	✓	✓
TCP/IP parameters	✓	✓	(✓)	(✓)
Specific domains	✓	✓	×	×
Combination	✓	✓	✓	✓

Flow data represent our primary source of information and contains extension fields discussed in Section III. Our flow monitoring probes are located at the backbone network connecting the university to the Internet, hence our visibility covers the communication from and to university. As we are interested in dynamic networks, we have filtered the traffic so that it contains only traffic originating from (i.e., source IP address is from) our wireless subnets. The opposite direction flows (target IP in our subnet) ate not interesting for OS identification of a host and were omitted.

Logs from DHCP servers then enrich the sessions by device MAC address and device name. As the network is dynamic and we have no control over connected devices, we have to derive the ground truth for OS identification from these logs. A large portion of devices comes with a pre-installed operating system with default device name. In many cases, it is hard or impossible for a common user to change the device name. For example, Android devices use string "android-<android_id>", Apple products use "<user>-iPhone", "<user>-iPad", and Microsoft uses default name "Windows-Phone" for its mobile products.

All connections to our wireless network must be authenticated with username and password. We have collected logs from Radius servers, which provide the authentication service. From these logs, sessions are created as 4-tuple (*id, assignedIP, startTime, endTime*), where id is a simple auto-incrementing counter. Sessions then serve as a ground truth to measure methods coverage. We have removed the user identity from dataset due to privacy reasons.

In a wireless network, a device can freely connect and disconnect and we cannot control how long the device is connected to the network. The session duration is a key factor influencing the number of flows to decide from during OS identification. Sessions duration ranged from a few seconds to over 19 hours with the average of 31.8 minutes and median 8.6 minutes. The cumulative distribution function for duration time is depicted on Fig. 1.

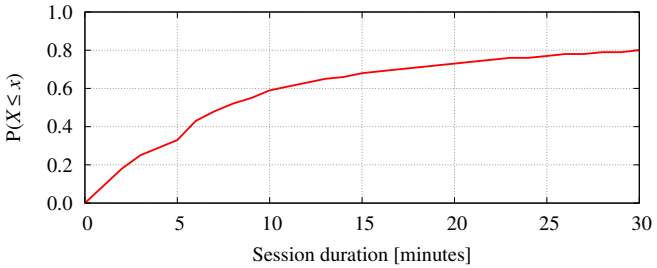

Fig. 1. Cumulative distribution function of session duration

V. Results

We have implemented the methods described in Section III and used them to identify OS of each Wi-Fi session from our dataset. The identification does not work on a single flow but takes every flow from the session with OS identified into

account. For a session containing more flows, OS is assigned to each flow and final result for the session is decided based on majority voting principle. This principle is used because some devices exhibit fingerprints of multiple OS during one session.

Similarly, the combination method can get into a situation where one session contains fingerprints of multiple operating systems (i.e., User-agent method result conflicts with TCP/IP parameters). This scenario occurred in 26.52 % of sessions and the conflicts were solved as described in Section III-D.

A. OS Identification Coverage

Our first experiment regarding OS identification is to measure how many sessions can the methods evaluate. To evaluate a session, at least one flow in the session needs to carry required information for a method to work. The User-agent method needs a HTTP request with a UA containing OS information. This requirement is fulfilled by 64.3 % of the sessions from the dataset. Specific domain method requires a HTTP(S) request on a domain from its dictionary. This is present in 78.1 % sessions. TCP stack method is the most generic and requires just a TCP connection, from which the parameters can be measured which is covered in 88.4 % of sessions. Our combination method needs at least one of the previous methods to have results and can identify OS in 93.4 % of the sessions. Summary of the coverage of OS identification methods in the dataset is depicted on Fig. 2.

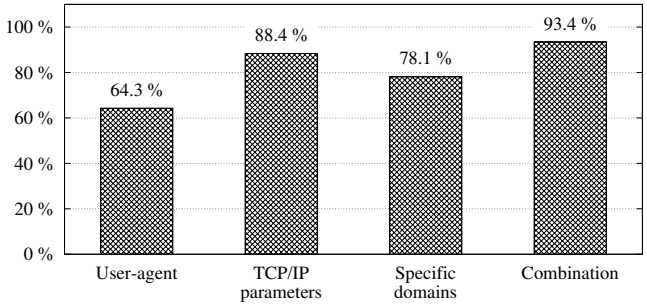

Fig. 2. Coverage of OS identification methods

Combination method was able to identify OS in more sessions than other methods which was caused by the presence of HTTP flows without TCP/IP parameters in our dataset. We have identified cause of this problem in the flow export of initial SYN packets with flags CE (Congestion Window Reduced, ECN-Echo) set, which caused the exported to think it is not an initial SYN packet and to skip filling the extended fields. We have reported this issue to the flow exporter vendor which confirmed our findings and implemented fix in the next version of the exporter.

B. OS Identification Accuracy

The second experiment explores how good the methods are in OS identification. In the terms of statistical analysis, our methods fall into multi-class classifier category with l non-overlapping classes. We have computed performance measures of accuracy, precision, recall, and f-score for each method according to [28]. As the distribution of operating systems is not uniform and there are big differences in their appearances in the dataset, we have decided to use the micro-averaging technique to favour the bigger classes.

Our dataset contains ground truth for OS identification extracted from DHCP logs (see Section IV), which is on the level of details of OS name from the OS hierarchy. It is also the highest level of detail the specific domains method could achieve and hence all methods performance is measured on this level of detail to ensure a fair comparison.

We have calculated confusion matrix with true positive (TP), false positive (FP), true negative (TN), and false negative (FN) values for each of the l classes (OS names) by comparing classification result to the ground truth. Then we computed the performance measures from the following equations:

$$AverageAccuracy = \frac{1}{l} \cdot \sum_{i=1}^{l} \frac{TP_i + TN_i}{TP_i + TN_i + FP_i + FN_i}$$

$$Precision_\mu = \frac{\sum_{i=1}^{l} TP_i}{\sum_{i=1}^{l} (TP_i + FP_i)}$$

$$Recall_\mu = \frac{\sum_{i=1}^{l} TP_i}{\sum_{i=1}^{l} (TP_i + FN_i)}$$

$$F - score_\mu = \frac{2 \cdot Precision_\mu \cdot Recall_\mu}{Precision_\mu + Recall_\mu}$$

Exact performance measures for individual methods are listed in Table III, their comparison is then depicted on Fig. 3.

TABLE III
MICRO AVERAGING FOR MULTI-CLASS CLASSIFIER PERFORMANCE MEASURES

Method	Accuracy	Precision	Recall	F-score
User-agent	0.9189	0.9812	0.6063	0.7495
TCP/IP parameters	0.8088	0.5249	0.4643	0.4927
Specific domains	0.8402	0.6286	0.4907	0.5512
Combination	0.8582	0.6587	0.6041	0.6302

User-agent method shows generally best results as applications generating HTTP requests are usually honest about its operating system. The significant drop in recall compared to other measures is caused by the high number of sessions without usable User-agent (i.e., no or encrypted User-agent sent, or it contains information only about the application and not OS) which causes many false negatives.

TCP/IP parameters method exhibits the worst performance from tested methods. We identify the main cause in two areas – Apple products sharing the same parameters, and Windows and Android devices using many different parameters. The first issue causes that most sessions with OS from Apple family (MAC OS X, iOS, Darwin) are identified as MAC

OS X since this method alone has no way to distinguish between them. On the other hand, Windows and Android devices communicate with many parameters configurations and their traffic dominance hides other operating systems in classification.

The new method using detection of specific domains proves to be capable of OS identification at large scale and even surpass the established TCP method. It can distinguish OS with small market share but suffers from the Apple issues discussed above. All Apple products communicate with a similar set of domains and even their applications installed on different OS (e.g., iTunes) download updates from the same domains. We can generalise this statement to all vendors as it is not economic to maintain distinct update server for every product they develop.

Our combination method embraces all positives and negatives from previous methods. It is able to identify OS in the highest portion of sessions and performance measures are better than TCP and specific domain methods. However, it is not as precise as the User-agent method which is the result of each method having the same weight during identification and flows without UA pushing the decision towards a wrong one.

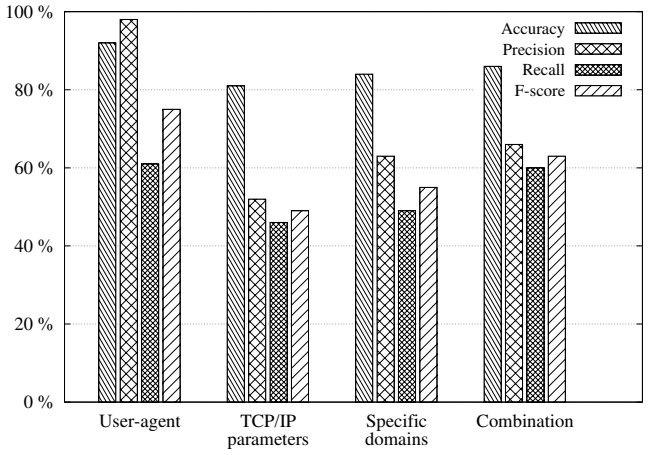

Fig. 3. Micro averaging for multi-class classifier performance measures

Besides the performance of the methods, we can look at the situation in our Wi-Fi network according to the identified operating system. Fig. 4 shows the market share of vendors based on results from the combination method. Mobile devices such as phones and tablets dominate the dynamic network (Android 56.18 %, MAC OS X 30.07 %) and traditional operating systems currently have decreasing popularity (Windows 4.48 %). *Unknown* means that the method could not evaluate the session and *Other* category is the rest of operating systems with low market share (e.g., BlackBerry).

VI. Lessons Learned and Discussion

In our experiment, we had to tackle several problems during the data gathering and aggregation. The first part of this section discusses these challenges and drafts possible solutions that might be useful for future work. The second part of this section presents possibilities for future use of tested methods and foreshadows obstacles, that will arise from the deployment of new network communication protocols.

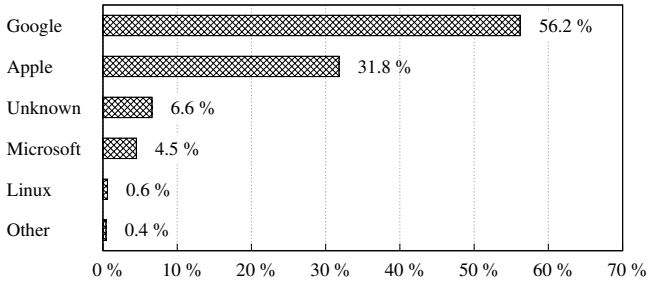

Fig. 4. Operating system usage share of our network grouped by vendor

A. Data Collection

The first challenge belongs to the data gathering and correlation, as fingerprinting methods need input from multiple sources (i.e., network flow metadata and application logs) and their timestamps are not always precisely synchronised. Establishing ground truth requires a lot of manual work in combining data from these sources, even with the use of centralized logging server. The efficiency would be greatly improved if this process was automated.

Finding the method for establishing ground truth poses the second challenge. Our method of using DHCP log parameters of device name works well for Apple and Google operating systems, that use easily identifiable device name by default and its change by a user is discouraged or impossible. On the other hand, Microsoft and Linux operating systems allow simple device name change and thus these devices couldn't be identified as easily. The second parameter we tried to use was MAC address, but it proved not to be helpful for ground truth establishment. Unfortunately, we were unable to find any meaningful mapping of Network Interface Controller vendors to operating systems of the devices due to the usage of same hardware family by multiple products with different operating systems. Ground truth establishment in large scale thus remains an open research problem for now.

The third major challenge is ensuring that all gathered extended flow data is correct and complete. These properties are often difficult to achieve on a vast network, as the monitoring probes might get congested and return incomplete information. We have experienced problems with some missing extended flow fields in approximately 3 % sessions. It is important to use probe configuration that will allow it to collect necessary flow parameters under usual network load.

The last obstacle we identified is coping with NAT (Network Address Translation) technology. We have not encountered this problem since we do not use any address translation on our Wi-Fi network. In a general network, NAT will confuse all three fingerprinting methods since it will be considered as a

singular device. The solution of this problem might be a focus for future research which will combine results from research on NAT detection with OS identification.

B. OS Detection Obstacles

All tested methods for passive OS fingerprinting will have to adapt to emerging new standards of network communication. We identify the following trends and protocols, that will have to be supported: IPv6, encrypted communication through TLS and HTTP/2.0 or QUIC [29] by Google. All of them will affect each of the methods in some way, although not necessarily to the same extent.

User-agent fingerprinting method will be the most affected when the above-mentioned protocols are commonly used. Because the current trend in network communication is to encrypt all data transfers with TLS by default, the User-agent field will not be readable during data transfer. In HTTP/2.0 [30], the encryption was not made mandatory, but browser developers have explicitly stated that it will be supported only in encrypted format [31]. The QUIC protocol explicitly requires encryption of content and only few implementations send UAID (User Agent equivalent) in the first unencrypted client hello message. This indicates that User-agent method's usability will be declining soon.

TCP/IP parameters method will not be affected by encryption, but it will have to be adapted for concurrent use of IPv4 and IPv6. The diminishing address pool and the ever-growing number of IoT devices that need public IP addresses will require extended usage of IPv6 protocol. In IPv6, the TTL parameter equivalent, Hop Limit, is suggested by Router Advertisement messages for all connected devices and the IPv4 header field Total Length was changed to Payload Length with slightly different semantics. Similarly, the QUIC protocol is based on UDP over which it establishes connection. The parameters of the first packet could be measured, but it would require changes to monitoring probes and new studies of such parameters. It follows that new fingerprinting parameters should be identified for IPv6 and current fingerprint databases should be extended to cover both versions of IP protocol and possibly to cover both TCP and UDP parameters.

Specific domains method will remain functional with encrypted traffic and unaffected by underlying network protocols as the SNI field remains in all new protocols. It will even be able to identify new operating system versions from already known vendors if the domain remains unchanged. However, to keep its database up to date it is necessary to monitor any changes in specific domains done by the vendor. If such process is in place, we believe that this method will be the most reliable and accurate in the near future.

All these challenges push OS fingerprinting method toward decreasing level of details about the OS. It can be expected that only OS name (or even only vendor) will be distinguishable by the methods. While this situation helps protect user privacy, it conflicts with the use-case of unsupported OS version detection.

VII. Conclusion

In this paper, we implemented two well-known methods for OS identification and expanded work on a new method which identifies OS based on communication with update or connectivity check servers. Finally, we developed a method combining the three methods in a fashion that they have equal weight in the final decision. Source codes, fingerprint databases and dataset are published on Github [24].

We have tested all methods on traffic from a large dynamic network to evaluate how well are different approaches able to identify OS in such environment. To the best of our knowledge, this is the first study of passive fingerprinting methods performance in an environment, where users can bring any device and connect it to the network without control or interference from the researchers. Moreover, using this environment we obtained traces from modern devices which is often problematic for researchers.

The results show great differences between TCP/IP and User-agents method caused by their requirements on input data. In our dataset, TCP/IP parameters method relying on network and transport layer can identify 24 % more sessions than application layer based User-agent method. Specific domains method relies on application layer too but works on both encrypted and unencrypted communication which helps it to identify OS in 14 % more sessions than the UA method.

Performance measures of accuracy, precision, recall and f-score were computed for each method. The overall best method proved to be the one with the most requirements on input data – User-agents. Its precision over 98 % shows it works well in any environment if the input contains what it needs. The accuracy of all methods is over 80 %, but other measures show that the rate of false positives and false negatives is quite high for all of them.

Finally, we discuss the challenges in large network monitoring and obtaining ground truth for OS fingerprinting. We also discuss the impacts of new Internet protocols on OS identification methods and identify the specific problems that will have to be solved.

For the future work, we plan to develop a method for automatic updates for the Specific domains identification. Similarly to TCP/IP parameters updates from flows with User-agents [13], it should be able to continuously monitor the network and update its database with new patterns. This work will ensure that the method won't end up outdated in the future.

Acknowledgement

This research was supported by the Security Research Programme of the Czech Republic 2015 - 2020 (BV III / 1 VS) granted by the Ministry of the Interior of the Czech Republic under No. VI20172020070 Research of Tools for Cyber Situation Awareness and Decision Support of CSIRT Teams in the Protection of Critical Infrastructure.

Martin Laštovička is Brno Ph.D. Talent Scholarship Holder – Funded by the Brno City Municipality.

REFERENCES

[1] J. M. Allen, "Os and application fingerprinting techniques," *SANS Institute InfoSec Reading Room*, 2007.

[2] F. Gagnon and B. Esfandiari, "A hybrid approach to operating system discovery based on diagnosis," *International Journal of Network Management*, vol. 21, no. 2, pp. 106–119, 2011.

[3] Google Inc., "Dashboards: Platform Versions," [cited 2017-08-25]. [Online]. Available: https://developer.android.com/about/dashboards/index.html

[4] A. Kott, C. Wang, and R. F. Erbacher, *Cyber defense and situational awareness*. Springer, 2015, vol. 62.

[5] R. Lippmann, D. Fried, K. Piwowarski, and W. Streilein, "Passive operating system identification from tcp/ip packet headers," in *Workshop on Data Mining for Computer Security*, 2003, p. 40.

[6] S. Mossel, "Passive os detection by monitoring network flows," 2012.

[7] R. Tyagi, T. Paul, B. Manoj, and B. Thanudas, "Packet inspection for unauthorized os detection in enterprises," *IEEE Security & Privacy*, vol. 13, no. 4, pp. 60–65, 2015.

[8] T. Matsunaka, A. Yamada, and A. Kubota, "Passive os fingerprinting by dns traffic analysis," in *Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference On*. IEEE, 2013, pp. 243–250.

[9] M. Husák, M. Čermák, T. Jirsík, and P. Čeleda, "HTTPS traffic analysis and client identification using passive SSL/TLS fingerprinting," *EURASIP Journal on Information Security*, vol. 2016, no. 1, p. 6, Feb 2016. [Online]. Available: https://doi.org/10.1186/s13635-016-0030-7

[10] A. Aksoy, S. Louis, and M. H. Gunes, "Operating system fingerprinting via automated network traffic analysis," in *2017 IEEE Congress on Evolutionary Computation (CEC)*, June 2017, pp. 2502–2509.

[11] B. Zhang, T. Zou, Y. Wang, and B. Zhang, "Remote operation system detection base on machine learning," in *2009 Fourth International Conference on Frontier of Computer Science and Technology*, Dec 2009, pp. 539–542.

[12] T. Jirsík and P. Čeleda, "Identifying operating system using flow-based traffic fingerprinting," in *Advances in Communication Networking*. Springer International Publishing, 2014, pp. 70–73.

[13] P. Matouek, O. Ryav, M. Grgr, and M. Vymltil, "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)*, Aug 2014, pp. 1–7.

[14] D. W. Richardson, S. D. Gribble, and 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.

[15] O. A. Osanaiye and 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 (EUROCON)*, Sept 2015, pp. 1–6.

[16] A. Azzouni, O. Braham, T. M. T. Nguyen, G. Pujolle, and R. Boutaba, "Fingerprinting openflow controllers: The first step to attack an sdn control plane," in *Global Communications Conference (GLOBECOM), 2016 IEEE*. IEEE, 2016, pp. 1–6.

[17] M. Zalewski, "p0f v3," [cited 2014-04-15]. [Online]. Available: http://lcamtuf.coredump.cx/p0f3/

[18] J. Barnes and 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*, ser. ANCS '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 113–114. [Online]. Available: http://dl.acm.org/citation.cfm?id=2537857.2537875

[19] R. Hofstede, P. Čeleda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, and A. Pras, "Flow Monitoring Explained: From Packet Capture to Data Analysis With NetFlow and IPFIX," *IEEE Communications Surveys Tutorials*, vol. 16, no. 4, pp. 2037–2064, Fourthquarter 2014.

[20] B. Claise, B. Trammell, and P. Aitken, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information," RFC 7011 (INTERNET STANDARD), Internet Engineering Task Force, Sep. 2013. [Online]. Available: http://www.ietf.org/rfc/rfc7011.txt

[21] IANA, "IP Flow Information Export (IPFIX) Entities," [cited 2017-08-16]. [Online]. Available: https://www.iana.org/assignments/ipfix/ipfix.xhtml

[22] A. Aksoy and M. H. Gunes, "Operating system classification performance of tcp/ip protocol headers," in *Local Computer Networks Workshops (LCN Workshops), 2016 IEEE 41st Conference on*. IEEE, 2016, pp. 112–120.

[23] J. Postel, "Transmission Control Protocol," RFC 793 (INTERNET STANDARD), Internet Engineering Task Force, Sep. 1981, updated by RFCs 1122, 3168, 6093, 6528. [Online]. Available: http://www.ietf.org/rfc/rfc793.txt

[24] Github, "Source codes and dataset." [Online]. Available: https://github.com/CSIRT-MU/PassiveOSFingerprint

[25] R. Fielding and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content," RFC 7231 (Proposed Standard), Internet Engineering Task Force, Jun. 2014. [Online]. Available: http://www.ietf.org/rfc/rfc7231.txt

[26] Flowmon Networks, "Netflow, a new era of network traffic monitoring," [cited 2017-09-20]. [Online]. Available: https://www.flowmon.com/en/solutions/use-case/netflow-ipfix

[27] Microsoft, "Network Connectivity Status Indicator and Resulting Internet Communication in Windows 7 and Windows Server 2008 R2," [cited 2017-08-16]. [Online]. Available: https://technet.microsoft.com/en-us/library/ee126135%28WS.10%29.aspx

[28] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Information Processing & Management*, vol. 45, no. 4, pp. 427–437, 2009.

[29] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, J. Bailey, J. Dorfman, J. Roskind, J. Kulik, P. Westin, R. Tenneti, R. Shade, R. Hamilton, V. Vasiliev, W.-T. Chang, and Z. Shi, "The quic transport protocol: Design and internet-scale deployment," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17. New York, NY, USA: ACM, 2017, pp. 183–196. [Online]. Available: http://doi.acm.org/10.1145/3098822.3098842

[30] M. Belshe, R. Peon, and M. Thomson, "Hypertext Transfer Protocol Version 2 (HTTP/2)," RFC 7540 (Proposed Standard), Internet Engineering Task Force, May 2015. [Online]. Available: http://www.ietf.org/rfc/rfc7540.txt

[31] IETF HTTP Working Group, "HTTP/2 Frequently Asked Questions," [cited 2017-09-03]. [Online]. Available: https://http2.github.io/faq/