Towards Identification of Operating Systems from the Internet Traffic
IPFIX Monitoring with Fingerprinting and Clustering

Petr Matoušek, Ondřej Ryšavý, Matěj Grégr and Martin Vymlátil
Brno University of Technology, Božetěchova 2, Brno, Czech Republic

Keywords: Operating Systems, Identification, Fingerprinting, Clustering, Monitoring, IPFIX.

Abstract: This paper deals with identification of operating systems (OSs) from the Internet traffic. Every packet injected on the network carries a specific information in its packet header that reflects the initial settings of a host's operating system. The set of such features forms *a fingerprint*. The OS fingerprint usually includes an initial TTL time, a TCP initial window time, a set of specific TCP options, and other values obtained from IP and TCP headers. Identification of OSs can be useful for monitoring a traffic on a local network and also for security purposes. In our paper we focus on the passive fingerprinting using TCP SYN packets that is incorporated to a IPFIX probe. Our tool enhances standard IPFIX records by additional information about OSs. Then, it sends the records to an IPFIX collector where network statistics are stored and presented to the network administrator. If identification is not successful, a further HTTP header check is employed and the fingerprinting database in the probe is updated. Our fingerprinting technique can be extended using *cluster analysis* as presented in this paper. As we show the clustering adds flexibility and dynamics to the fingerprinting. We also discuss the impact of IPv6 protocol on the passive fingerprinting.

1 INTRODUCTION

Knowledge of operating systems can be an important information for network security and monitoring. From the point of view of an attacker, the knowledge of the targeting system helps him to choose his strategy and identify where critical files or resources are stored and how can be weaknesses of the system exploited (Sanders, 2011).

Detecting connected hosts and identifying operating systems is also useful for maintaining a site security policy (Lippmann et al., 2013). The policy may specify the types of hosts that are allowed to connect to the local network (LAN). OS fingerprinting can help a network administrator to detect unexpected devices like smart-phones, WiFi access points, or routers using fingerprints of their OSs (Android, Cisco IOS). The knowledge of OSs in the LAN can help a network administrator to find out how many operating systems are currently employed in the LAN, if there are potential vulnerabilities related to unpatched software, which hosts use obsolete systems and should be updated, etc. OS fingerprinting techniques can be also used for NAT detection (Krmicek, 2011).

OS fingerprinting can be defined as "an analysis of certain characteristics and behaviors in network communication in order to remotely identify an OS and its version without having direct access to the system itself" (Allen, 2007). The technique relies on differences among implementations of OSs and TCP/IP stacks that have an impact on certain values in IP or TCP headers of packets generated by these systems. Having a database of typical OS features (called OS signatures), we are able to identify with a certain probability by which OS a packet was sent.

There are two types of OS fingerprinting: passive and active. *Passive fingerprinting* only listens to the packets on the networks. When a packet is received, passive fingerprinting extracts needed values of TCP or IP header fields in order to detect an OS using a OS fingerprint database. If a host does not send an expected type of data like TCP SYNs, DHPC requests, HTTP requests, or does not communicate at all, passive fingerprinting cannot be used. *Active fingerprinting* is a complementary technique that actively sends special packets to a targeted host in order to elicit replies that will reveal an operating system of the target. In comparison to the passive fingerprinting, the active fingerprinting is not transparent from the point of view of network communication. There are several tools implementing both the passive fingerprint-

Matoušek P., Ryšavý O., Grégr M. and Vymlátil M..
Towards Identification of Operating Systems from the Internet Traffic - IPFIX Monitoring with Fingerprinting and Clustering.
DOI: 10.5220/0005099500210027
In *Proceedings of the 5th International Conference on Data Communication Networking* (DCNET-2014), pages 21-27
ISBN: 978-989-758-042-0

ing like p0f, ettercap, or the active fingerprinting, e.g., nmap.

Key factors for successful classification of an OS using the fingerprinting includes a proper selection of OS features to be compared, an up-to-date OS signatures database, and a technique that is used to classify features obtained from a packet header with a fingerprinting database. Common fingerprinting features used for OS identification can be extracted, for example, from a IP header (TTL, Don't Fragment bit, Packet Size), or a TCP header (TCP Windows size, Max Segment Size, Options, etc.) (Lippmann et al., 2013).

Classification can be based on a perfect matching when packet features must exactly match one of the OS signatures in database. It is also possible to use an approximate matching where only few features match an OS signature and the best possible (but imprecise) selection of OS is returned. There are also advanced classification methods using machine learning algorithms (Caballero et al., 2007), clustering (Jain, 2010), or fractal geometry and neural networks (Zelinka et al., 2013).

In our work we focus on the passive fingerprinting that is incorporated into IPFIX framework (Claise et al., 2013; Claise and Trammel, 2013) where an IPFIX probe scans incoming packets, extracts relevant features from IP, TCP, or even application headers, and creates an extended IPFIX record (Claise, 2004) with additional information about senders' OSs. Because an IPFIX probe is a passive monitoring device, we are interested only in passive fingerprinting methods. For monitoring purposes, fast on-line processing of incoming packets is preferred over accuracy of the classification. In this paper, we present our technique for fast on-line OS identification using passive fingerprinting. This method was implemented as a plugin into the IPFIX probe. Our classifier uses a special OS signature database where eight OS features of a packet are tested. The database contains signature classes of major operating systems (Windows, Linux, BSD, Mac OS, Android, Palm OS). Classification is made if the perfect match of at least six SYN packet features is found. If the matching is not successful, further check is done using HTTP header. If such header is a part of TCP communication, HTTP User-agent field is extracted. Based on these data, a new signature containing current packet features is generated and automatically added to the OS signature database.

Our results show that this approach is feasible and give good results. During our test we observed that small variations of observed features can causes the system to be classified as *unknown* because the perfect match is required. This can be improved using classification based on cluster analysis. This technique is not dependent on a fixed OS fingerprinting database and perfect matching since the clustering measures *similarity* between classified objects. This paper shows a proposal of this architectures and discusses its pros and cons.

1.1 Contribution

Main contribution of this positional paper is design and implementation of the passive fingerprinting within a scheme of IPFIX monitoring. We present our classification database and discuss our first results that are compared with p0f. Further, we propose an extension of the system using cluster analysis that helps to decrease false negatives. We also discuss an impact of IPv6 protocol on passive fingerprinting because IPv6 and IPv4 headers differ and the same features cannot be used without additional tests and observation of different IPv6 implementations. To our best knowledge, we are not aware of studies describing implementation of OS fingerprinting within IPFIX framework and application of the cluster analysis in OS classification.

1.2 Structure of the Paper

The paper is structured as follows. Section 2 discusses current fingerprinting techniques and results published in recent years. It also proposes open questions and areas to be explored, especially in IPv6 OS fingerprinting. Section 3 shows an architecture of our tool and presents the first results. Section 4 describes a concept of the cluster analysis applied on the area of OS fingerprinting and future steps in our research. The last section summarizes our current work.

2 STATE OF THE ART

Passive and active fingerprinting techniques and tools were explored in previous years. The main motivation of that research was network security and configuration of network-based intrusion detection systems using information about OSs from incoming packets. General principles of OS fingerprinting and common fingerprinting techniques are described in (Allen, 2007).

(Lippmann et al., 2013) focuses on accuracy of passive OS fingerprinting and the evaluation of common tools like nmap, siphon, p0f, and ettercap. It discusses quality of different features, the number

of features to be used for classification and the structure of OS signature databases. This paper also talks about reliability of commonly used features and how can be changed by intermediate network devices or an attacker.

The main limitation of techniques based on OS signatures is to keep database up-to-date when new versions and implementations of OSs appear. Otherwise, modern operating systems remain unrecognized during classification. To overcome this issues, advanced methods for automated generation of OS signatures were explored. One of the first attempt was FiG tool presented by (Caballero et al., 2007). This tool automatically explores a set of candidate queries and applies a machine learning technique to identify the set of valid queries. A similar approach was also proposed by (Schwartzenberg, 2010) where a classifier based on neural networks was introduced. However, these techniques have their limits as proved by (Richardson et al., 2010). In their work, the authors argue that OS signatures that are generated automatically using machine learning techniques are not viable because of over-fitting, indistinguishability, biases in the training data, and missing semantic knowledge of protocols. The authors recommend to add an expert knowledge or manual intervention in the process of putting new OS signatures into the OS database. Otherwise, the accuracy of detection is very low.

Our goal is similar to the previous authors—to automatically update an OS signature database without human intervention based on incoming packets. However, our technique is different. First, we explore HTTP application header for OS type specification and if it is found, a set of features from IP and TCP headers is used to form a new OS fingerprint entry that is added to the database. Seconds, the cluster analysis can be applied to identify the best possible match of an OS.

Another part of work concerns on the impact of migration from IPv4 to IPv6 (S.Deering and R.Hinden, 1998). There are several studies describing differences between these two protocols and challenges for OS fingerprinting over IPv6 (Nerakis, 2006). Most of these studies, e.g., (Beck et al., 2007), employ the active fingerprinting using Neighbor Discovery Protocol (T.Narten et al., 2007) for OS identification. However, most of these approaches does not reflect recent changes in standardization of IPv6 extended headers defined by RFC 7045 (Carpenter and Jiang, 2013) and RFC 6564 (Krishnan et al., 2012). (Eckstein, 2011) describes major challenges and limitation of IPv6 OS fingerprinting only theoretically. To our best exploration, there are no thorough experiments showing how IPv6 passive fingerprinting works in real systems. Thus, we decided to extent our tool with IPv6 and to show how IPv6 features in combination with TCP and cluster analysis can be employed for successful OS identification.

3 FINGERPRINTING AND IPFIX

In our research, we apply a technique of the passive fingerprinting on the area of network monitoring using IPFIX framework. The main concept we used is derived from Michal Zalewski's approach implemented in `p0f`[1] that identifies OSs using TCP SYN or SYN+ACK packets. The tool checks selected values from IP headers (TTL, DF bit, ToS), TCP headers (Window size, options), or HTTP headers (User-Agent field) and compares them with OS signatures in a database that currently contains about 300 entries. `P0f` database is composed of TCP signatures, HTTP signatures and MTU signatures. It uses TCP SYN, TCP SYN+ACK, HTTP request, or HTTP response for classification. The result of classification is (i) a perfect match, (ii) a fmatch (small deviation in some values), or (iii) *unknown*.

3.1 OS Signatures

We decided to implement a lightweight version of `p0f` approach. We use only SYN packets and HTTP headers to obtained OS features. Since we are not interested in minor differences between OSs (like versions, kernel numbers, etc.) our database contains only several classes of major OSs. Our set of features includes three IP fields (TTL, DF bit, packet size) and five TCP fields (Max segment size, Selective ACK, Window size, number of NOPs—No operation bytes, and Window scale). Since TTL is decremented by every L3 device, we don't look at a certain value but for an interval. Our signature database was based on `p0f` database, observations published in (Sanders, 2011), and our experiments. The database values for major OS classes are shown in Table 1. Linux values were tested on Ubuntu and Red Hat distributions. We can see that the most important features where OSs differ at most are SYN packet size, number of NOPs, and Win Scale.

We test eight features for OS identification, but the perfect match on all features is not required. Table 2 shows that the higher number of perfectly matched

[1]p0f stands for Passive OS Fingerprinting, see http://lcamtuf.coredump.cx

Table 1: OS signature database.

OS	TTL	DF	Pkt size	Win Size	Win Scale	MSS	SAck	NOPs
Windows XP	64–128	Set	48	variable	0	1440,1460	Set	2
Windows 7	64–128	Set	52	variable	2	1440,1460	Set	3
Windows 8	64–128	Set	52	variable	8	1440,1460	Set	3
Linux	0–64	Set	60	2920–5840,14600	3	1460	Set	1
FreeBSD	0–64	Set	60	65 550	7	1460	Set	1
Mac OS	0–64	Set	64	65 535	4	1460	Not	3
Android 4.x	0–64	Set	52,60	65 535	6	1460	Set	3
Symbian	128–255	Not	44	8 192	0	1460	Not	0
Palm OS	128–255	Not	44	16 348	0	1350	Not	0
NetBSD	0–64	Set	64	32 768	0	1416	Not	5
Open BSD	0–64	Set	64	32 768	0	1440	Set	5

Table 3: Comparison of Win 7 and Win 8 features.

OS	TTL	DF	Pkt size	Win size	MSS	NOPs	Win Scale	SAck
Win 7 Home	128	Set	52	8192	1440,1460	3	8	Set
Win 7 others	64,128	Set	52	8192	1440,1460	3	2	Set
Win 8	128	Set	52	8192	1440,1450	3	2	Set

Table 2: Impact of the number of comparisons.

Matches	W7	W8	XP	Linux	BSD	Unknown
5	10	46	7	43	17	1
6	7	42	6	43	17	9
7	4	16	4	43	14	43
Sample	30	21	9	43	17	4

features is, the higher number of false negatives occurs. It means that some OS remains unidentified if their features differ in one of eight comparisons. In Table 2 we can see the results of comparison when five, six, or seven features were required for the perfect match. The last row shows the real distribution of OSs in our sample. We can see that Window 7 features can be confused with Windows 8 features: in rows 1 and 2 the total count is almost the same but the ratio between these two OSs varies. If we require seven features to be matched, there is a lot of false negatives, mostly for Windows systems. Further analysis showed that differences between Windows 7 and Windows 8 are minimal, especially for Windows 7 Home Edition (32 bits) and Windows 8, see Table 3. This is the reason why classification using cluster analysis can give better results.

3.2 Classification

The classification algorithm is depicted in Figure 1. The algorithm starts with testing a packet type. If it is a SYN packet, the OS features are extracted one by one from the packet header and compared with the OS database. If there is a match, counter cnt_i of each of the matched OSs is increment. After all features

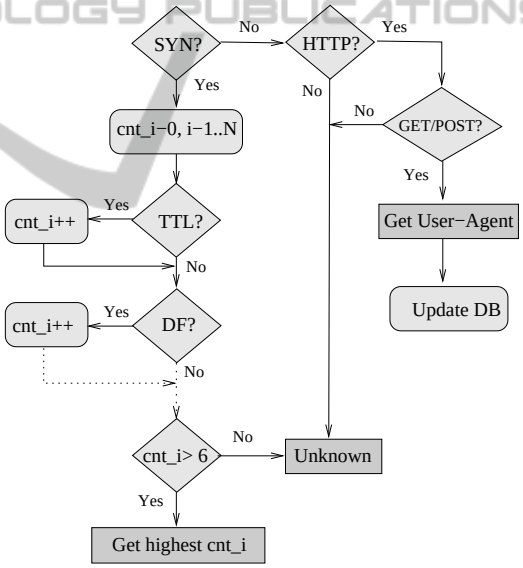

Figure 1: Classification algorithm.

are tested, the best match with the highest score of the counter is selected. If the highest counter score is lower than six, an OS is not identified.

In case of a HTTP packet, its header is examined if it contains GET or POST method. Then, an OS name is retrieved from *User-Agent* field. Further, a new OS signature is generated and added to the OS fingerprinting database to keep the database updated. The output of our classification tool is showed in Figure 2.

Despite the fact that our plugin uses simple OS signature database in comparison to `p0f`, we can see

```
IP: 173.194.113.68      OS: Unknown      ttl: 50  df: 2  synsize 0       win size 0
IP: 173.194.70.125      OS: Unknown      ttl: 44  df: 2  synsize 0       win size 0
IP: 192.168.20.1        OS: MAC OS       ttl: 64  df: 1  synsize 64      win size 65535
IP: 31.13.81.128        OS: Windows 8    ttl: 82  df: 1  synsize 60      win size 14480
IP: 81.91.84.180        OS: Free BSD     ttl: 53  df: 1  synsize 60      win size 5792
IP: 173.194.113.78      OS: Unknown      ttl: 50  df: 2  synsize 60      win size 42540
IP: 68.232.35.139       OS: Free BSD     ttl: 50  df: 1  synsize 60      win size 14480
IP: 185.31.17.185       OS: Free BSD     ttl: 47  df: 1  synsize 60      win size 14280
IP: 173.194.70.84       OS: Unknown      ttl: 43  df: 2  synsize 60      win size 42540
IP: 8.37.70.21  OS: Free BSD    ttl: 45  df: 1  synsize 60      win size 14480  max ss
IP: 23.209.127.139      OS: Free BSD     ttl: 52  df: 1  synsize 60      win size 14480
IP: 23.62.237.104       OS: Free BSD     ttl: 54  df: 1  synsize 60      win size 14480
IP: 23.209.114.110      OS: Free BSD     ttl: 52  df: 1  synsize 60      win size 14480
```

Figure 2: Example of classification of our tool.

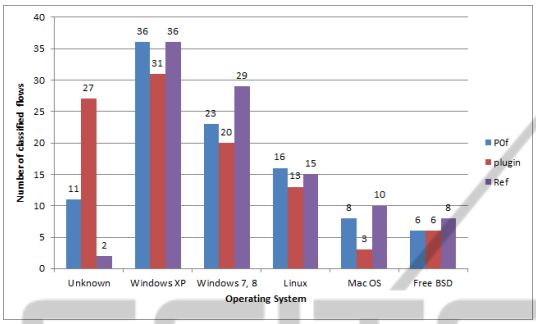

Figure 3: Comparing p0f and our plugin.

that the results are quite similar, see Figure 3. This graph presents the results of OS classification using `p0f` and our tool `plugin`. The third column represents a reference dataset (Ref) with known OSs. The total number of classified flows was 100. As we can see, our plugin has a large number of unclassified flows (false negatives) due to the perfect feature matching that can be enhanced using clustering.

Nevertheless, our first results prove the hypothesis that even small number of signatures classes can help to identify OSs using the passive fingerprinting. In that case we loose more precise information about detected OSs. However, this is not crucial for on-line network monitoring.

3.3 IPFIX Monitoring

The main purpose of this research was to extend the current framework of IPFIX monitoring by information about the operating system. Our classifier is a part of IPFIX probe that collects information about flows (Claise, 2004). Flow records include (i) identification of the flow (source and destination IP address, source and destination port, ToS, protocol, and interface ID), and (ii) statistics about the flow: number of packets, bytes, starting and ending time of the flow, etc. Standard IPFIX supports dynamic structuring of flow records using templates where additional data can be attached.

In our case, we extend IPFIX records by a set specific features extracted from IP or TCP headers, and the name of an identified OS. After a flow expires in the probe cache, an IPFIX record is sent to an IPFIX collector. The collector stores incoming data in its monitoring database. These data can be automatically or manually processed and visualized. A simple example of extended IPFIX records processed by `fbitdump` (Velan, 2012) is shown in Figure 4.

Src IPv4	Operation system	TTL	DF	Win size	Max ss	Nop	Win Scale	Sack OK	SYN packet size
192.168.1.101	Windows 8	64	1	8192	1460	3	8	1	52
77.75.76.72	Windows 8	54	1	5840	1452	3	7	1	52
173.252.113.17	Windows XP	76	1	14600	1452	2	0	1	48
192.168.1.101	Windows 8	64	1	8192	1460	3	8	1	52
77.75.76.3	Windows 8	54	1	4960	1240	3	7	1	52
77.75.76.3	Windows 8	55	1	12400	1240	3	7	1	52

Figure 4: Output of the IPFIX collector.

4 FUTURE WORK

The main drawbacks of OS passive fingerprinting are (i) a need of up-to-date signature database and (ii) the number of false negatives caused by classification based on perfect matching. The first drawback can be minimized using automatic generation of signatures from HTTP headers. The second drawback can be solved using advanced classification methods. Methods based on training neural networks can give good results, however they need to be trained repeatedly for every new OS. From this point of view, an application of the cluster analysis seems to be more promising.

4.1 Data Clustering Using K-means

The cluster analysis is a technique of classifying data into classes called *clusters* based on unsupervised learning. Unlike machine learning that uses supervised classification based on given training data, clustering uses unlabeled data and tries to classify them to clusters based on similarities (Duda et al., 2001). There can be also a hybrid approach called *semi-supervised learning* that uses only a small portion of training data for cluster definition (Chapelle et al., 2006). The goal of data clustering is to discover the natural grouping of a set of objects. An operation of clustering can be described as follows: Given a representation of n objects, find K groups based on a measure of *similarity* such that the similarities between objects in the same group are high while the similarities between objects in different groups are low (Jain, 2010).

In our work, we decided to explore an application of non-hierarchical clustering using K-means method to the OS identification. K-means algorithm finds a partition on a set of n d-dimensional objects such that the squared error between the empirical mean of a cluster and objects in the cluster is minimized. Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of d-dimensional objects that needs to be clustered into set $C = \{c_1, c_2, \ldots, c_K\}$ of K clusters where μ_a is a mean of cluster c_a. Then the squared error between μ_a and the points of clus-

ter c_a is $E(c_a) = \sum_{x_i \in C_a} \|x_i - \mu_a\|^2$ (Jain, 2010). The method requires three parameters for its operation: a number of clusters, a set of initial cluster centers (centroids), and metrics.

In OS identification, the number of clusters and their centers can be extracted from the OS signature database. In our case, signatures from Table 1 can be used. As we see, the table contains the wide range of values for each feature. This would not work properly for euclidean metric because the square error of each dimension (feature) should have the same weight. In this case, features with the small range of values (Set/Unset bits, WinScale, NOPs) would be discriminated against Windows Size or Maximum Segment Size. So, before computing square error, normalization of feature values should be done so that they have the same weight despite the range of their values. Possibly, an additional weight coefficient could be added to some of the most distinguish features.

The OS identification using K-means method can be described as follows:

1. Initialize cluster centroids using OS signature database in Table 1, `p0f` database, or any other OS fingerprinting database.
2. For each incoming SYN packet:
   (a) Compute an square error for every cluster.
   (b) Find out the nearest centroid and select an OS.
   (c) Re-compute the mean value of the cluster.
   (d) Update the database.
3. If a HTTP packet is detected:
   (a) Extract a feature set from the packet.
   (b) Check is such OS cluster exists.
   (c) If the cluster exists, update the mean value of the cluster. Otherwise, create a new cluster.

4.2 Preliminary Results of K-means

Machine learning methods are sensitive to feature selection. As a part of preliminary work on application of machine learning algorithms to OS detection and classification, we have performed an experiment with feature selection for K-means algorithm in order to examine relevance of the proposed feature set used in the previously described fingerprinting method. The experiment consists of evaluating fitness of K-means computed clusters for an exhaustive collection of selected features. The best results are shown in Table 4. The evaluation is based on computation of overall fitness value. For each TCP flow, the operating system is determined using the fingerprinting method. The K-means algorithm computes clusters from these flows according to the selected features. An operating system is identified by finding the most frequent OS label among the flows in the cluster. Each cluster thus corresponds to one of the operating systems. We denote p being the number of flows labeled with the same OS and n being the number of flows with a different OS label, respectively. The fitness is then computed as follows: $fit = p/(p+n)$.

Table 4: Results of K-means clustering experiment.

Feature Set	Fitness
{TTL,SynLen,WinSize,NOPs}	92 %
{TTL,WinSize,NOPs,WinScale}	86 %
{TTL,WinSize,WinScale}	83 %
{TTL,SynLen,WinSize,WinScale}	83 %
{WinSize,NOPs,WinScale}	81 %
{TTL,SynLen,WinSize,NOPs,WinScale}	79 %

As can be seen from the table, the best results were obtained when using four features out of eight proposed for the fingerprinting. Certain combinations of three features give also remarkable results. Nevertheless, feature sets with more than five and less than three items provide poor results. Although these findings represent preliminary results they give a noteworthy hint on finding relevant features for the domain of OS classification.

Our future work will focus on further testing and improvement of our approach using clustering algorithms. It involves following research questions:

- How is this approach accurate in comparison with the first approach and `p0f`?
- Does this approach help to decrease the number of false negatives?
- What kind of features should be preferred in order to receive more precise results?
- How will clusters change in time? Do centroids converge to a stable value?
- Is this approach feasible for special device like routers, printers, etc.?
- What IPv6 features are more suitable for OS identification?

As a part of our future work is the creation of a reference dataset of current OSs under both IPv4 and IPv6.

5 CONCLUSION

The paper applies a method of passive fingerprinting on the area of IPFIX monitoring. This application requires the fast on-line processing of incoming packets

and flexible OS fingerprinting database with minimal intervention of a network administrator. Our first tests show that this approach is feasible. We also implemented cluster analysis using K-means algorithm in order to show that this technique can be successfully applied on the passive OS fingerprinting.

Our future work will focus on implementation of different cluster analysis algorithms and their evaluation on real data and comparison with existing passive approaches. In addition, we will apply this approach on IPv6 communication to show how to identify OSs using the passive fingerprinting from IPv6/TCP communication.

ACKNOWLEDGMENTS

Acknowledgment will be completed in the camera-ready version of the paper due to the blind review. Research in this paper was supported by project "Modern Tools for Detection and Mitigation of Cyber Criminality on the New Generation Internet", no. VG20102015022 granted by Ministry of the Interior of the Czech Republic and project "Research and application of advanced methods in ICT", no. FIT-S-14-2299 supported by Brno University of Technology.

REFERENCES

Allen, J. M. (2007). OS and Application Fingerprinting Techniques. Infosec reading room, SANS Institute.

Beck, F., Festor, O., and Chrisment, I. (2007). IPv6 Neighbor Discovery Protocol based OS fingerprinting. Technical report, INRIA.

Caballero, J., Venkataraman, S., Poosankam, P., Kang, M. G., Song, D., and Blum, A. (2007). FiG: Automatic fingerprint generation. *Department of Electrical and Computing Engineering*, page 27.

Carpenter, B. and Jiang, S. (2013). *Transmission and Processing of IPv6 Extension Headers*. IETF RFC 7045.

Chapelle, O., Schölkopf, B., and Zien, A., editors (2006). *Semi-Supervised Learning*. MIT Press, Cambridge, MA.

Claise, B. (2004). *Cisco Systems NetFlow Services Export Version 9*. IETF RFC 3954.

Claise, B. and Trammel, B. (2013). *Information Model for IP Flow Information Export (IPFIX)*. IETF RFC 7012.

Claise, B., Trammel, B., and Aitken, P. (2013). *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information*. IETF RFC 7011.

Duda, R., Hart, P., and Stork, D. (2001). *Pattern classification*. Pattern Classification and Scene Analysis: Pattern Classification. Wiley.

Eckstein, C. (2011). OS fingerprinting with IPv6. Infosec reading room, SANS Institute.

Jain, A. K. (2010). Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, 31(8):651–666.

Krishnan, S., Woodyatt, J., Kline, E., Hoagland, J., and Bhatia, M. (2012). *A Uniform Format for IPv6 Extension Headers*. IETF RFC 6564.

Krmicek, V. (2011). *Hardware-Accelerated Anomaly Detection in High-Speed Networks*. PhD. Thesis, Masaryk University, Brno, Czech Republic.

Lippmann, R., Fried, D., Piwowarski, K., and Streilein, W. (2013). Passive Operating System Identification from TCP/IP Packet Headers. In *Proceedings Workshop on Data Mining for Computer Security (DMSEC)*.

Nerakis, E. (2006). IPv6 Host Fingerprint. Thesis, Naval Postgraduate School, Monterey, California.

Richardson, D. W., Gribble, S. D., and Kohno, T. (2010). The Limits of Automatic OS Fingerpritn Generation. In *Proceedings of AISec'10*, Chicago, Illinois, USA.

Sanders, C. (2011). *Practical Packet Analysis*. No Starch Press, 2nd edition.

Schwartzenberg, J. (2010). Using machine learning techniques for advanced passive operating system fingerprinting. Msc. theses.

S.Deering and R.Hinden (1998). *Internet Protocol, Version 6 (IPv6) Specification*. RFC 2460.

T.Narten, E.Nordmark, W.Simpson, and H.Soliman (2007). *Neighbor Discovery for IP version 6 (IPv6)*. RFC 4861.

Velan, P. (2012). Processing of a Flexible Network Traffic Flow Information. Msc. thesis, Masaryk University, Fakulty of Informatics, Brno, Czech Republic.

Zelinka, I., Merhaut, F., and Skanderova, L. (2013). Investigation on operating systems identification by means of fractal geometry and os pseudorandom number generators. In *International Joint Conference CISIS'12-ICEUTE'12-SOCO'12 Special Sessions Advances in Intelligent Systems and Computing*, volume 189, pages 151–158. Springer.