Enabling SSH Protocol Visibility in Flow Monitoring

Pavel Čeleda, Petr Velan, Benjamin Král
Institute of Computer Science
Masaryk University
Brno, Czech Republic
{celeda,velan,kral}@ics.muni.cz

Ondřej Kozák
Flowmon Networks a.s.
Brno, Czech Republic
ondrej.kozak@flowmon.com

***Abstract*—The network flow monitoring has evolved to collect information beyond the network and transport layers, most importantly the application layer information. This information is used to improve network security and performance by enabling more precise performance analysis and intrusion detection. In this paper, we contribute to this effort by extending flow monitoring with information from the SSH protocol. Firstly, we analyze the SSH protocol to determine which information can be obtained from the connection establishment phase. Based on the analysis, we create an extension to our flow monitoring infrastructure that allows obtaining the selected information. Lastly, we analyze the SSH connections observed in the university campus network and discuss the benefits of performing the detailed SSH protocol analysis. We argue that with a precise recognition of login attempt results it is possible to improve the detection of successful brute-force password attacks. Moreover, we publish an anonymized version of our dataset including the SSH specific information.**

***Index Terms*—SSH, flow, monitoring, network, dataset**

I. Introduction

The network flow monitoring [1] is being enhanced to observe and export application layer information. HTTP protocol related fields are even defined as standard IPFIX entities [2]. Although the increasing usage of encryption hinders this approach, it is possible to gain information even from the encrypted traffic [3], [4]. This paper focuses on the analysis of the Secure Shell (SSH) protocol [5], which is used for critical tasks such as server management and data transfer. To the best of our knowledge, this is the first work that studies the SSH protocol in the context of network flow monitoring.

Our goal is to use information available during the unencrypted phase of the SSH protocol handshake to gain more detailed knowledge about SSH connections. This information can be used to improve network-based detection methods, discover unintentional misconfiguration of the SSH service and audit its security policies. To achieve this goal, we have developed an extension for Flowmon probe [6] to monitor the SSH protocol. This extension gathers information from the SSH connection establishment phase and exports it using new IPFIX information elements. This approach allows us to gain the SSH protocol visibility even in large-scale networks.

We have deployed the SSH flow monitoring on the campus network of the Masaryk University and collected flow data for a month. We analyze the collected data and report on the observed SSH protocol usage status. Moreover, we show that the extended flow data can be used to detect irregular behavior in SSH communication, such as an abnormally high number of authentication attempts. The contribution of this paper is threefold:

- Analysis of the SSH protocol, creation of SSH protocol flow measurement using new IPFIX elements with information from the SSH protocol.
- Deployment of SSH flow monitoring on a campus network and analysis of the collected data.
- Publication of the collected (anonymized) dataset including the SSH specific information.

The organization of the rest of this paper is as follows. Section II discusses the state-of-the-art related to this work. Section III describes the SSH protocol and investigates what information can be extracted from the SSH protocol payload. Section IV evaluates the prototype implementation of the SSH flow monitoring and analyzes the data collected on our campus network. Finally, Section V concludes the paper.

II. Related Work

In recent years, SSH protocol has been studied extensively, especially in relation to security. As SSH allows a user to control and command a remote host, it is often targeted by many attackers who are trying to gain access to the machine. Most frequent of such attacks is a brute-force password attack. However, since SSH communication is encrypted, it is not an easy task to distinguish malicious traffic such as brute-force attack from legitimate traffic.

There are many approaches to detecting attacks on SSH and gaining insight into SSH communication. One of such approaches is based on employing special infrastructure and protocols to enable intrusion detection and deep packet inspection. For example, Sherry et al. [7] propose a system called BlindBox which enables middle-box to monitor encrypted traffic and once a malicious keyword appears in the data then and only then the middle-box will decode the encrypted data stream and perform deep packet inspection. This approach, however, requires cooperation with the communicating parties. Moreover, the infrastructure and proposed protocols make this approach difficult and expensive to set up.

978-3-903176-15-7 © 2019 IFIP

Foroushani et al. [8] take a different approach. They propose a method based on statistical analysis of packet data size and timing. As both are available without decryption, there is no need for special protocols. However, the downside of this system is rather high false positive rate (around 15%) which makes it impractical for real-life scenarios.

A similar approach is taken by Najafabadi et al. [9] who proposed a way to extract features from aggregated network flows such as the number of aggregated flows, average number of packets per flow, or average sizes. The features are also extractable without decryption and can be used by machine learning algorithms to create various classification models.

Another approach using network flows is explored by Hellemons et al. [10]. The method they proposed separates the SSH brute-force attack into three phases and declares several statistical thresholds for each phase. All connections originating from one source that go through at least one of the attack phases are identified as an SSH brute-force attack. This method has been implemented as an intrusion detection system plugin for NfSen. Moreover, the false positives and false negatives rates were observed to be very low. This method has been further improved by Hofstede et al. [11] to detect successful compromises with near 100% accuracy.

However, none of these approaches analyze the SSH data stream. Only the metadata from the data stream or aggregated flow records is extracted and used for analysis. The metadata is usually generic and does not reflect the specifics of SSH protocol. Moreover, there is often none SSH protocol identification in place, and simply all traffic on TCP port 22 (well-known port number for SSH protocol) is used for analysis. An exception is the open source intrusion detection system Bro [12], which implements the SSH protocol identification, and is able to extract data from the SSH protocol that is being transmitted during SSH handshake and encryption setup. This way SSH traffic can be detected and identified across all traffic, and other useful information can be extracted such as encryption and MAC algorithms used in each SSH connection. However, Bro is an intrusion detection system and does not aim at monitoring of high-speed networks.

III. SSH Protocol Monitoring

We have implemented SSH communication monitoring as a software module for Flowmon probe [13], enabling us to monitor SSH communication in large-scale networks. This section begins with description of the essentials of SSH communication setup. The data that can be extracted from SSH communication in plain text is reviewed and our implementation of a module for Flowmon probe is described. Finally, the approach to authentication analysis is outlined, detailing identification of authentication process and outcome.

A. SSH Protocol

SSH is a stateful client-server protocol that consists of three layers - *a transport layer*, *a user authentication layer*, and *a connection layer*. When initiating the connection, the transport layer sets up a secure communication channel on top of TCP/IP. This channel provides integrity, confidentiality, server authentication, and may provide data compression. After a secure channel is established by the transport layer, client authentication is handled by user authentication layer. Finally, one or more SSH application communication channels are established that may be used for various purposes such as providing an interactive command line, port tunneling, or X11 forwarding.

The SSH connection setup can be seen at Figure 1. At the very beginning of communication when the transport layer is being set up, both sides send a handshake message containing information about SSH protocol and software versions in use.

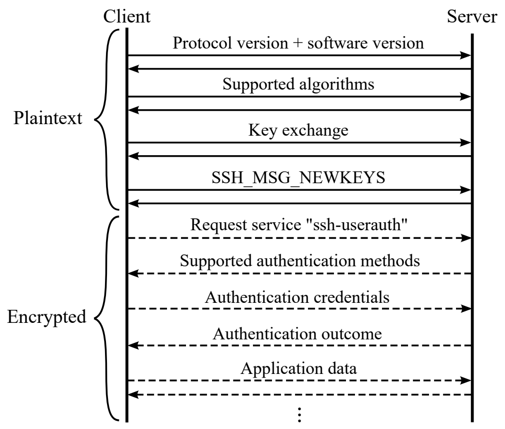
Client
Server
Protocol version + software version
Supported algorithms
Key exchange
SSH_MSG_NEWKEYS
Plaintext
Request service "ssh-userauth"
Supported authentication methods
Authentication credentials
Authentication outcome
Application data
Encrypted

Fig. 1. SSH Transmission Setup.

After the handshake, both client and server inform the other side about supported authentication, encryption, MAC, and compression algorithms. Ten algorithm lists will be sent by each side [14] altogether. The first algorithm in each of client's list that is supported by the server as well will be used by both parties.

After the algorithm negotiation, a symmetrical encryption key is set up using the negotiated key exchange algorithm. Both parties will finish the encryption setup by sending a `SSH_MSG_NEWKEYS` message, informing the other party that the newly exchanged key will be used from now on. From this moment all communication is being encrypted.

Once the connection is set up, the authentication layer will perform user authentication. First of all, a list of supported authentication methods is sent by the server (such as `password`, `publickey`). The client selects one of the authentication methods offered by the server and sends its name along with all relevant parameters. Server replies with message signifying either successful authentication or failed authentication attempt. However, all this communication is encrypted and therefore cannot be easily read.

B. SSH Flow Monitoring

Since the first part of SSH transmission setup is transmitted in plain text, the content may be freely read by a network flow monitoring probe. Therefore, it is possible to extract the following information from the communication before the encryption setup is finished:

- Client and server SSH protocol version
- Client and server SSH software and version
- Algorithms supported by client and server
  - Key exchange algorithms
  - Server host key algorithms
  - Symmetric encryption algorithms
  - MAC (Message Authentication Code) algorithms
  - Compression algorithms
  - Communication languages

The SSH communication is identified across all traffic regardless of the transport layer port number. The identification is done by looking for a string `SSH-` at the beginning of payload data in one of the first packets in each new network flow, which is always present in the first message client sends to the server to initiate SSH connection.

Once the SSH connection is identified, the detection method goes through several phases, each for one message as it is expected to be sent in order by client or server, as seen in Figure 1. In each of the phases, appropriate data from the message is extracted and saved to the appropriate record in the probe's flow cache. The algorithm lists which are transmitted as strings in the SSH data stream are saved as numerical identifiers in network flow to decrease the network flow message size.

When the flow with SSH application information expires due to active timeout, it is exported to the flow collector. The new flow record that is created in its stead retains all the application level information. Therefore, there are usually many flows records with the SSH information for a single long-lived SSH connection. However, when the flow expires due to inactive timeout, the SSH information cannot be retained. Therefore, when no packets are sent in the connection for an interval longer than the inactive timeout set on the flow probe, the rest of the flow records created from the same connection cannot contain the SSH-specific information.

C. Client Authentication Analysis

The second part of SSH connection initialization, as seen in Figure 1, is encrypted. However, compromise identification is an important component of an authentication attack detection system. Crucial part for compromise identification is the ability to identify the outcome of each authentication attempt. This way, a brute-force attack can be detected. Moreover, if one of the authentication attempts is identified as successful, it can be concluded that the victim device has been breached and attack remediation process can be initiated.

As the message carrying authentication outcome information is encrypted, it cannot be simply read by the intrusion detection system. However, the structure, lenght, and content of each of those two messages are different. As can be seen in Figure 2, the message signifying a successful authentication outcome does not carry any data apart from the message ID. On the other hand, the authentication failure message is embedded with a list of authentication methods that may be used for further authentication attempts. In theory, this list could be empty, but it has been observed that always at least one authentication method will be provided by the server. Because of this message length discrepancy, which is also present after encryption, authentication outcome may be identified.

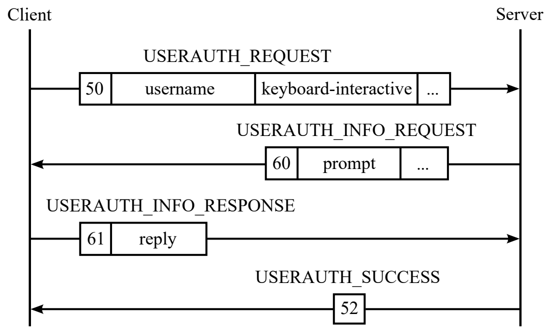

Fig. 2. SSH User Authentication - Example Of Keyboard-Interactive Method.

This approach could be prevented by appending another message `SSH_MSG_IGNORE`, which must be ignored by the other party. This way the overall length of each message could be increased, efficiently erasing the message length discrepancy that facilitates this approach to authentication outcome identification. However, we have not detected any protocol implementation which would use the ignore message to counter this vulnerability.

Moreover, this approach is based on the prerequisite that the SSH communication follows the standard order of messages as seen in Figure 1 and in Figure 2 respectively. If the order of messages is different or the connection is ended abruptly, the detection method will fail to perform the authentication analysis. However, the module for Flowmon probe we have developed is able to identify the phase the SSH connection was in and attaches this information, enabling us to identify problematic SSH connection phases and investigate the cause of the failure. Moreover, the method is able to detect and work with retransmissions and out-of-order packets.

IV. Experimental Evaluation

We have deployed the SSH flow monitoring at the perimeter of the campus network of the Masaryk University. The flow data was collected for the entire August 2018 and includes basic flow records as well as flow records with SSH information. Since the percentage of SSH traffic containing the application layer information is relatively small (only 0.36% of all TCP connections), the impact on the monitoring performance is rather negligible. To confirm this assumption, we have monitored the resources consumed by the SSH flow monitoring and verified that there is no significant performance hit in comparison to the flow monitoring that was deployed already. Although the number of flows with ssh information is small (0.12% of all flows), they represent 14.31% of total observed traffic volume. This is probably due to a periodic data synchronization using the SSH protocol.

Flows are usually split by an active or inactive timeout, as described in Subsection III-B. This is particularly important for long-lived SSH connections that can be active for weeks at a time. Therefore, we carefully differentiate between flows and connections in this section. An SSH connection is usually reported in multiple flow records. Moreover, all flows up to the first inactive timeout contain the application layer information.

The rest of this section analyzes the captured SSH flows. First, it reports on statistical properties such as non-standard port numbers, client and server protocol and software versions, and used key exchange algorithms. Second, a brief analysis of SSH authentication attempts is performed. We show that most of the unsuccessful authentication attempts are not part of brute-force password attacks, as one might expect, but rather a benign part of service infrastructure. Last, we provide a description of the dataset created during the experiment.

A. SSH Protocol Statistics

The processing of SSH traffic is not limited to a specific port; therefore, it is possible to observe SSH traffic on non-standard ports. We have found that only 87.19% of SSH connections use the standard port, i.e., port 22. Top 10 non-standard ports account for more than 86% of the rest of the connections. Perhaps surprisingly, port 80 is being used for SSH traffic as well. This is probably done in order to avoid overly restrictive firewall policies. The number of connections for each of the top 10 non-standard ports is depicted in Figure 3.

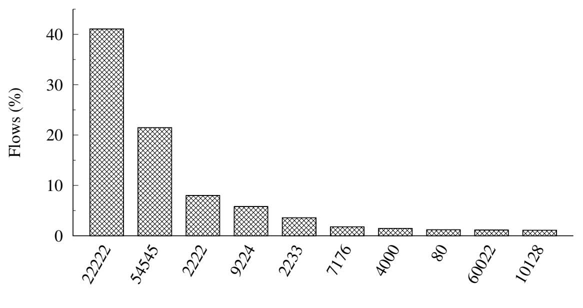

Fig. 3. Top 10 Non-Standard SSH Ports.

The SSH protocol can be observed in multiple versions. SSH-2 was published as an IETF Internet Standard; however, different versions were available much earlier. SSH-2 is incompatible with SSH-1; moreover, SSH-1 is not considered to be secure and should be phased out. Nonetheless, there is still a small number of clients and servers using the outdated SSH-1 protocol, as shown in Table I.

TABLE I
SSH PROTOCOL VERSIONS.

Client Version	% of Flows	Server Version	% of Flows
2.0	99.985	2.0	98,689
1.99	0.014	1.99	1,307
1.5	0.001	1.5	0,004
1.33	0.000	2. 0	0.000

98% of flows where client communicated using protocol 1.99 was generated by *3.2.9 SSH Secure Shell for Windows* client application. Client applications reporting protocol version 1.99 on the server side are Cisco-1.25, Cisco-2.0 (86.4%) and different versions of OpenSSH (12.9%). The rest of the traffic is generated by a number of other clients, each with only a tiny share.

We have also analyzed the different software versions observed in the SSH traffic. First, we filtered out software with less than 10 flows, since the version strings were meaningless, probably randomly generated, and possibly part of some network scan. The rest of the records were stripped of the version number, and only the software names were aggregated. The results for client and server software differ significantly. Table II shows the top 10 most common client and server software that was found in the dataset. The shown client software represents 98.4% of total flows. The server software is more diversified than the client software, and the top 10 represents only 94.1% of the total flows.

TABLE II
SSH SOFTWARE IMPLEMENTATIONS.

Client Software	% of Flows	Server Software	% of Flows
OpenSSH	37.935	OpenSSH	91.827
libssh2	23.289	Cisco	1.680
check_ssh	18.107	libssh	0.238
libssh	10.016	dropbear	0.243
PuTTY	2.510	HomeSSH	0.020
Go	2.196	ROSSSH	0.033
paramiko	2.171	conker	0.032
WinSCP	1.022	mod_sftp	0.004
zabbix_agent	0.741	FlowSsh	0.012
Granados	0.331	Zyxel	0.001
nsssh2	0.057	Comware	0.003
FileZilla	0.007	CerberusFTPServer	0.000

Version numbers were stripped from software strings before aggregating.

We can see that the OpenSSH dominates both the client and server software version. However, it is especially dominant on the servers, which is not surprising given that it is the default SSH implementation for virtually every GNU/Linux distribution. The client software is more balanced. The SSH protocol is implemented in Windows software, such as WinSCP and PuTTY. Moreover, SSH often needs to be called from within scripting and programming languages and specific libraries, such as paramiko, libssh, and libssh2, exist for this reason. Server monitoring tools such as Zabbix and Nagios provide their own clients for checking the SSH status as well.

Key exchange, data encryption, and MAC algorithms are important security aspects of each SSH session. Tables III and IV shows all key exchange and encryption algorithms that were used for the observed connections. We can see elliptic curve Diffie-Hellman key agreement is being used quite often (45.7% of flows), the original Diffie-Hellman algorithm is used only slightly more often (51.2%), and no key exchange algorithm is used in the rest of the cases. Moreover, the use of the deprecated SHA-1 algorithm in the key exchange phase is only 12.2%.

The AES encryption algorithm variants are the dominant algorithms used for encrypting the SSH transport layer. The

TABLE III
SSH KEY EXCHANGE ALGORITHMS.

Key Exchange Algorithm	% of Flows
diffie-hellman-group-exchange-sha256	38.154
curve25519-sha256@libssh.org	20.617
curve25519-sha256	16.844
ecdh-sha2-nistp256	8.080
diffie-hellman-group14-sha1	6.723
diffie-hellman-group1-sha1	5.523
(null)	3.114
diffie-hellman-group-exchange-sha1	0.754
ecdh-sha2-nistp521	0.141
ecdh-sha2-nistp384	0.045
diffie-hellman-group16-sha512	0.004

most common is AES with keysize of 128 bits in counter (CTR) mode. The other observed ciphers are chacha20, 3des, arcfour, and blowfish. Out of these four, only the chacha20 has significant usage. The use of arcfour is not generally encouraged, however, it might still be enabled due to its high performance.

TABLE IV
SSH ENCRYPTION ALGORITHMS.

Algorithm	% of Flows	Algorithm	% of Flows
aes128-ctr	66.454	aes256-cbc	0.306
chacha20-poly1305@*	23.000	3des-cbc	0.010
aes128-cbc	4.829	aes192-cbc	0.001
aes256-ctr	2.373	aes192-ctr	0.001
aes128-gcm@*	2.074	arcfour	0.001
(null)	0.592	blowfish-cbc	0.000
aes256-gcm@*	0.356	none	0.000

* . . . openssh.com implementation of the algorithm

OpenSSH version 7.8 was released during our experiment on August 23. We have observed the use of this version on the same day. This allows us to easily identify systems that are regularly updated. Moreover, by observing client and server software versions, we can perform or enhance fingerprinting of operating systems. It also allows us to recognize network devices from vendors with their own SSH daemon implementation, such as Cisco and Mikrotik.

B. Client Authentication Analysis

One of the most important information that can be extracted from an SSH connection is how many login attempts were performed and whether they were successful. Our data contain this information for most of the flows where it is possible to distinguish between successful and unsuccessful authentication attempt. Most login attempts start with a query to determine available authentication methods, which is not distinguishable from a regular authentication attempt. Therefore, even if the use gives a valid password on the first try, it is reported as two login attempts.

In some cases, the outcome of an authentication attempt cannot be determined with absolute certainty. When this happens, the flow record contains this information together with the reason why the analysis failed. We analyze only the successful and failed authentication attempts that were reported with absolute certainty.

Figure 4 shows the number of observed authentication attempts for both successful and failed login cases. We can see that most successful authentications succeed on one of the first three attempts. However, in some of the observed cases as much as 13 login attempts were performed. Since most SSH daemons refuse a connection after only a handful of failed login attempts, we believe that the high numbers of login attempts were caused by using SSH keys. When a user adds multiple keys to her local SSH agent, all of them are tried upon a new connection unless a concrete one is specified and until a correct one is found.

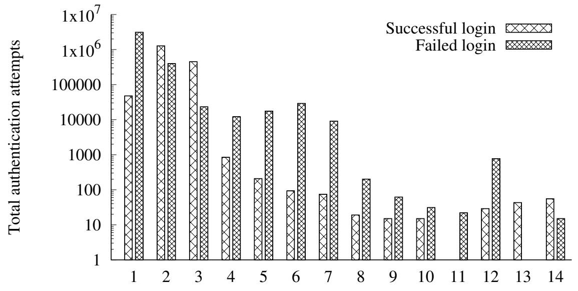

Fig. 4. Authentication Attempts per SSH Connection.

The high number of unsuccessful login attempts is not caused by attackers and users that gave up after a single authentication attempt. It is caused by periodic checks performed by monitoring tools such as Zabbix and Nagios. Table V shows client software of 99.7% of all flows that contained zero authentication attempts and were not successful. We can see that a third of the flows are caused by check_ssh module of the Nagios monitoring system and that more than a half of the connections were initiated by a software using the libssh library. OpenSSH was used only in 3% of connections which is a significant difference in comparison with Table II.

TABLE V
UNSUCCESSFUL SSH CLIENTS WITH 0 AUTHENTICATION ATTEMPTS.

Client Software	% of Flows	Client Software	% of Flows
libssh2	39.746	Terminal	0.413
check_ssh	34.909	Granados	0.366
libssh	17.847	paramiko	0.340
OpenSSH	3.001	PuTTY	0.077
Go	1.603	WinSCP	0.017
zabbix_agent	1.429		

Version numbers were stripped from software strings before aggregating.

C. Dataset Description

We have published the dataset [15] that was used in the analysis in this paper. It contains flow records with SSH fields for August 2018. The traffic was observed at the perimeter of the Masaryk University campus network. To preserve privacy, the IP addresses were anonymized using a salted SHA-256 hash function. SSH connections made using IPv4 and IPv6 are stored separately. This was done mostly to simplify the processing and to allow separate analysis in the future. For example, IPv6 traffic is less likely to contain SSH scans. The information contained in the dataset is described in Table VI.

TABLE VI
DATASET ELEMENTS.

Basic Flow Elements	SSH Elements
Flow Start Timestamp	SSH Client Version
Flow End Timestamp	SSH Server Version
Source IP address (Anon.)	SSH Client Application
Source Transport Port	SSH Key Exchange Algorithm
Destination IP Address (Anon.)	SSH Host Key
Destination Transport Port	SSH Client Encryption Alg.
Transport Protocol	SSH Server Encryption Alg.
Number of Packets	SSH Client MAC Alg.
Number of Bytes	SSH Server MAC Alg.
TCP Flags	SSH Client Compression Alg.
	SSH Server Compression Alg.
	No. of Authentication Attempts
	Authentication Attempts Result

V. CONCLUSIONS

In this paper, we have shown that information from SSH traffic can be obtained using passive flow monitoring. Even though SSH protocol encrypts its transport layer, much useful information remains in its initialization phase that is valuable for security purposes. Moreover, even though connection initialization is encrypted as well, it is possible to determine, in most cases, whether an authentication attempt was successful or not from the size of the SSH messages.

Apart from the analysis of the SSH protocol, we have evaluated the SSH flow monitoring software on the real campus network. We have shown how the extended flow data could be used for SSH security management:

- *Port usage* – traditional flow data analysis tools rely on port classification (SSH port 22). Some administrators use different port numbers to hide SSH services (security through obscurity). We perform deep packet inspection to identify SSH usage on non-standard ports.
- *Protocol versions* – major SSH version in use is 2.0 today. Other versions are suspicious (obsolete and insecure).
- *Cryptography audit* – SSH transport relies on multiple algorithms to protect the communication. We provide visibility on used algorithms to perform compliance testing. The measured data can be further used to fingerprint clients and servers.
- *Software implementations* – software version string indicates the name of the application and the capabilities of an implementation. The measured data can be used to detect vulnerable clients and servers using CVE (Common Vulnerabilities and Exposures).
- *Client authentication* – SSH servers are the target of many brute-force password attacks. Most detection methods report on SSH connection attempts. Our proposed detection method understands traffic even when it is encrypted and reports on successful or failed login.

The dataset used for the evaluation of the SSH flow monitoring was made publicly available at [15]. Therefore, our analysis can be not only verified but also extended by other researchers. We believe that a number of other interesting observations can be derived from our data.

In our future work, we would like to leverage SSH flows to perform client and server fingerprinting and clustering to identify patterns in SSH communication of malicious users. It may be that malicious users use software with distinct parameters and implementation such as a unique combination of advertised software version and encryption algorithms. The data collected could be used as input for machine learning methods to filter new connections based on the patterns identified in previous malicious traffic.

ACKNOWLEDGMENT

This research was supported by the 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. 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," *Communications Surveys Tutorials, IEEE*, vol. 16, no. 4, pp. 2037–2064, 2014.

[2] Internet Assigned Numbers Authority. (2017, Sep.) IP Flow Information Export (IPFIX) Entities. [Online]. Available: https://www.iana.org/assignments/ipfix/ipfix.xhtml

[3] P. Velan, M. Čermák, P. Čeleda, and M. Drašar, "A Survey of Methods for Encrypted Traffic Classification and Analysis," *International Journal of Network Management*, vol. 25, no. 5, pp. 355–374, 2015.

[4] B. Anderson and D. McGrew, "Identifying Encrypted Malware Traffic with Contextual Flow Data," in *Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security*, ser. AISec '16. New York, NY, USA: ACM, 2016, pp. 35–46.

[5] T. Ylonen and C. Lonvick, "The Secure Shell (SSH) Protocol Architecture," RFC 4251 (Proposed Standard), Internet Engineering Task Force, Jan. 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4251.txt

[6] Flowmon Networks, "Flowmon Probe." [Online]. Available: https://www.flowmon.com/en/products/flowmon/probe

[7] J. Sherry, C. Lan, R. A. Popa, and S. Ratnasamy, "BlindBox: Deep Packet Inspection over Encrypted Traffic," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 213–226, Aug. 2015.

[8] V. A. Foroushani, F. Adibnia, and E. Hojati, "Intrusion Detection in Encrypted Accesses with SSH Protocol to Network Public Servers," in *2008 International Conference on Computer and Communication Engineering*, May 2008, pp. 314–318.

[9] M. M. Najafabadi, T. M. Khoshgoftaar, C. Calvert, and C. Kemp, "Detection of SSH Brute Force Attacks Using Aggregated Netflow Data," in *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, Dec 2015, pp. 283–288.

[10] L. Hellemons, L. Hendriks, R. Hofstede, A. Sperotto, R. Sadre, and A. Pras, "SSHCure: A Flow-Based SSH Intrusion Detection System," in *Dependable Networks and Services*, R. Sadre, J. Novotný, P. Čeleda, M. Waldburger, and B. Stiller, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 86–97.

[11] R. Hofstede, L. Hendriks, A. Sperotto, and A. Pras, "SSH Compromise Detection Using NetFlow/IPFIX," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 5, pp. 20–26, Oct. 2014.

[12] J. Amann, J. Azoff, T. Fleury, V. Grigorescu, S. Hall, V. Paxson, A. Sharma, J. Siwek, A. Slagell, R. Sommer, and et al., "The Bro Network Security Monitor." [Online]. Available: http://www.bro.org/

[13] O. Kozák, "Advanced SSH Traffic Analysis," Master's Thesis, Masaryk University, Faculty of Informatics, Brno, May 2018. [Online]. Available: https://is.muni.cz/th/uf67z/

[14] T. Ylonen and C. Lonvick, "The Secure Shell (SSH) Transport Layer Protocol," RFC 4253 (Proposed Standard), Internet Engineering Task Force, Jan. 2006, updated by RFC 6668. [Online]. Available: http://www.ietf.org/rfc/rfc4253.txt

[15] P. Čeleda, P. Velan, B. Král, and O. Kozák, "Dataset: Enabling SSH Protocol Visibility in Flow Monitoring," http://dx.doi.org/10.5281/zenodo.1412596, Sep. 2018.