SSH Compromise Detection using NetFlow/IPFIX

Rick Hofstede
r.j.hofstede@utwente.nl

Luuk Hendriks
luuk.hendriks@utwente.nl

Anna Sperotto
a.sperotto@utwente.nl

Aiko Pras
a.pras@utwente.nl

Centre for Telematics and Information Technology (CTIT)
University of Twente, Enschede, The Netherlands

ABSTRACT

Flow-based approaches for SSH intrusion detection have been developed to overcome the scalability issues of host-based alternatives. Although the detection of many SSH attacks in a flow-based fashion is fairly straightforward, no insight is typically provided in whether an attack was successful. We address this shortcoming by presenting a detection algorithm for the flow-based detection of compromises, i.e., hosts that have been compromised during an attack. Our algorithm has been implemented as part of our open-source IDS *SSHCure* and validated using almost 100 servers, workstations and honeypots, featuring an accuracy close to 100%.

Categories and Subject Descriptors

C.2.3 [**Network Operations**]: Network management; Network monitoring

General Terms

Management; Measurement

Keywords

Network measurement; Intrusion detection; SSH; NetFlow; IPFIX

1. INTRODUCTION

Brute-force attacks against SSH daemons are a common type of attack in which attackers perform authentication attempts on a remote machine. These attacks are often performed using dictionaries – lists of frequently used login credentials – and are therefore commonly referred to as *dictionary attacks*. Once compromised, machines can cause serious damage by joining botnets, distributing illegal content, participating in DDoS attacks, mining Bitcoins or other cryptocurrencies, etc. The threat of SSH attacks was recently stressed again by the *Ponemon 2014 SSH Security Vulnerability Report*: 51% of the surveyed companies has been compromised via SSH in the last 24 months [7]. From our own experience, we know that compromises can have various causes, among which are dictionary attacks. For example, in campus networks, such as the network of the University of Twente (UT) with roughly 25k active hosts, we observe approximately 115 dictionary attacks per day, while in a backbone network, such as the Czech National Research and Education Network CESNET, it is not uncommon to observe more than 700 per day. Even more attacks should be expected in the future; several renowned organizations, such as OpenBL[1] and DShield[2], report a tripled number of SSH attacks between August 2013 and April 2014. Research in the area of compromise detection is therefore crucial to reduce the potential damage caused by compromises.

The detection of dictionary attacks and potential compromises can be performed in several ways. First and least error prone is a host-based approach, where log files can be inspected and Intrusion Detection Systems (IDSs) installed. This approach is however hardly scalable in larger networks, as access to every machine is a necessity. Second, a network-based approach can be taken, in which network traffic is inspected at central observation points. Inspection of individual packets would be a challenging task, due to the vast amount of traffic to be inspected. Moreover, the end-to-end encryption of SSH traffic makes inspection of packet payloads a futile process. An alternative approach that can be taken in large and high-speed networks is the analysis of traffic *flows*, exported by means of Cisco's NetFlow or the recent standardization effort IPFIX [4]. In this context, flows are defined as "sets of packets or frames passing an observation point in the network during a certain time interval" [1]. After a flow is considered to have terminated, which is typically based on timeouts, a *flow record* is exported to a central collection device, where it can be analyzed.

Several works have investigated the flow-based detection of SSH attacks [2, 3, 6, 8]. These works have in common that they rely on the observation that SSH traffic related to dictionary attacks is "flat", meaning that every connection is similar in terms of packets, bytes and duration. This signature is used and extended in [8], where the flow-based detection of dictionary attacks in high-speed networks is discussed. Similarly, a three-phase attack model for SSH attacks is proposed in [6], and includes a *scan* phase, *brute-force* phase and *compromise* phase. The key advantage of this model is that it allows for the detection of compromises, i.e., whether an authentication attempt in the brute-force phase has been successful. In a previous work, we have implemented and validated the model by means of the open-source IDS *SSHCure* [3]. Another tool that claims to detect compromises after dictionary attacks is mentioned in [9], but neither a reference to the tool, nor reproducible detection results are provided.

Although our work in [3] was based on a generic model for SSH dictionary attacks, promising results were achieved.

[1] `http://www.openbl.org`
[2] `http://www.dshield.org`

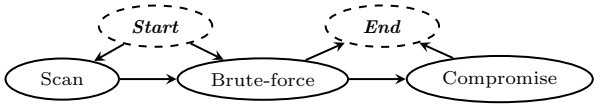


Figure 1: SSH attack phase transitions.

The goal of this work is to improve the performance of the compromise detection. We have analyzed SSH attack tools systematically and derived six scenarios that could be observed upon a compromise. By detecting these scenarios independently, we are able to analyze more types of dictionary attacks, and achieve higher accuracies and detection rates. The contribution of this work is twofold. First, we have refined the detection algorithm presented in [3] to support the six identified compromise scenarios, and implemented it as part of our IDS *SSHCure*. Validation of the algorithm's accuracy is performed based on log files from almost 100 servers, workstations and honeypots. The performance requirement is validated by deploying *SSHCure* on 11 backbone links of CESNET. Second, we provide two publicly available and labeled datasets[3] consisting of anonymized flow data and host log files, to support reproducible research.

The remainder of this paper is structured as follows. In Section 2, we summarize the advances made in our previous works and their shortcomings. Next, in Section 3, we characterize network traffic of various SSH attack tools, daemons and mitigation systems. Based on these findings, we discuss our detection algorithm in Section 4, of which the validation results are presented in Section 5. Finally, we draw our conclusions and state our future work in Section 6.

2. SSH ATTACK MODEL

The work on our detection algorithm and IDS *SSHCure* has started in 2011, when we developed a detection algorithm based on the Hidden Markov Model (HMM) of SSH attacks presented in [6]. That model assumed SSH attacks to feature one or more of the following three attack phases:

- **Scan** – An attacker scans hosts for active daemons.
- **Brute-force** – An attacker performs many authentication attempts on target hosts, using a large number of username/password combinations (dictionary).
- **Compromise**[4] – An attacker has gained access to a target host by using correct login credentials.

The SSH attack state transitions are shown in Figure 1. The only difference with the work in [6] is that attacks can also start in the *brute-force* phase for the following reasons. First, the *scan* phase can have taken place in the past (e.g., before traffic observation has started). Second, our investigation of attack tools (Section 3.2) has shown that attacks can start directly from the *brute-force* phase, as the scan can have been performed by another host or the target may be known in advance by an attacker.

It was also shown in [6] that the attack phases can be clearly observed in a time-series of the number of packets-per-flow (PPF). The *scan* phase is characterized by a low number of PPF, indicative for TCP connections that do not complete their handshake. At some point in time, the attacker starts the *brute-force* phase, trying to authenticate to remote machines. This phase was shown to consist of flows with a significantly larger number of PPF, caused by the authentication procedure of SSH. After a potentially successful login, some residual traffic can be observed between attacker and target as part of the *compromise* phase, where traffic is characterized by having a PPF that is outside the range of what is considered *brute-force* phase traffic.

A purely PPF-based detection algorithm and a first prototype of *SSHCure* were presented in [3]. Although this yielded promising results, a large number of production deployments of *SSHCure* has proven that the designed detection algorithm has various shortcomings that ultimately cause compromises to remain undetected or false alarms to be raised. First, we have discovered characteristic features of the *OpenSSH* daemon that significantly impact the traffic between SSH clients and daemons. Second, the original detection algorithm has proven to be too restrictive in detecting *brute-force* attacks. Given that the *compromise* phase can only be reached via the *brute-force* phase, as shown in Figure 1, undetected *brute-force* attacks yield undetected compromises. Last, attack mitigation systems on various network layers often yield traffic patterns that are similar to the definition of compromise traffic in the original algorithm. We address the shortcomings presented above in Section 3.

3. SSH TRAFFIC ANALYSIS

Our experience in analyzing SSH traffic has shown that network traffic between attacker and target can be affected at multiple stages: SSH daemon settings (Section 3.1), attack tools (Section 3.2) and attack mitigation mechanisms (Section 3.3). In the analysis presented in this section, we consider *OpenSSH* as the daemon running on attack targets, as it often comes preinstalled on Linux, BSD and Mac OS X operating systems. To verify whether it is the most used SSH daemon, we have performed a scan on the UT network, which has approximately 25k active hosts. Out of those, more than 700 hosts are running an publicly accessible daemon with a valid identification string, of which 97% identified itself as being *OpenSSH*[5].

3.1 OpenSSH daemon

The *OpenSSH* daemon features several configuration options that should be taken into account for the detection of SSH compromises:

- `LoginGraceTime` defines the time after which the SSH daemon disconnects in case the client does not perform any more authentication attempts. This is done by sending a TCP `FIN` packet to the client. The default value is 2 minutes.
- `MaxAuthTries` defines the maximum number of authentication attempts per connection. The default value is 6. Note that many client tools, such as the OpenSSH client, close the connection already after 3 failed authentication attempts. The allowed number of authentications can be changed by the client as long as the value does not exceed `MaxAuthTries`.

[3]The datasets are available in anonymized form at `http://www.simpleweb.org/wiki/SSH_datasets`

[4]Although we use the attack phases presented in [6], we use the more intuitive *compromise* to denote the *die-off* phase.

[5]Other discovered SSH daemons were *SunSSH*, *Dropbear*, *Cisco SSH*, *Gene6*, *DesktopAuthority*, *SCS* and *WeOnlyDo*.

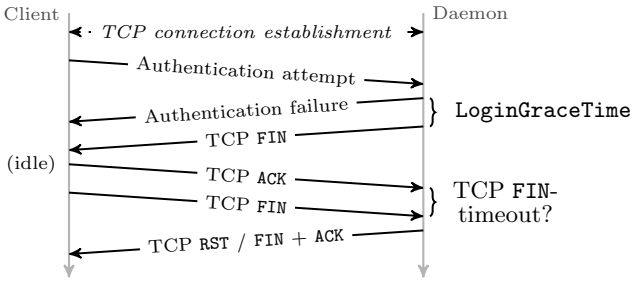


Figure 2: SSH client behavior after `LoginGraceTime`.

- `MaxStartups` defines the maximum number of concurrent, unauthenticated connections. It is defined as the three-tuple `start:rate:full`, with default value 10:30:60. Rate-limiting in the form of dropping connections is then applied with a probability of `rate`/100 when more than `start` connections are unauthenticated. This probability increases linearly up to the moment in which `full` connections are unauthenticated.

Next to these settings, there are TCP settings that affect the network traffic between client and daemon. First, the value of the TCP `FIN`-timeout determines the maximum amount of time a TCP connection remains in the `FIN-WAIT-2` state. In this state, the daemon has initiated a connection termination and received a subsequent TCP `ACK` from the client. As soon as the client also closes the connection by sending a TCP `FIN` packet, the time between this packet and the previous TCP `ACK` packet determines the response of the server; if the TCP `FIN` packet is received before the TCP `FIN`-timeout has taken place, the server replies with a `FIN`+`ACK` packet, while a TCP `RST` packet is sent otherwise. This is shown in Figure 2. Second, the TCP keep-alive interval determines the maximum idle time of a TCP connection. In case of an idle connection, a TCP `ACK` packet is sent without payload. The OpenSSH daemon also has its own keep-alive mechanism, but since the TCP keep-alive mechanism is enabled by default, it is typically disabled.

Existence of users on the target system (i.e., the system where the *OpenSSH* daemon is running) does not affect the network traffic. From the attacker (client) side, no difference can be observed between an attempt using a valid username and an invalid password, and an attempt with an invalid username. Naturally, this is favorable in terms of security, as an attacker cannot determine whether a guessed username exists on the target host and thereby increase the probability of a successful authentication.

3.2 Attack Tools

We have analyzed ten uniquely identifiable tools for performing brute-force attacks, ranging from `expect`[6] scripts, to sophisticated applications that try to be invisible for detection algorithms. These tools have been downloaded to our honeypots over several years, complemented by tools that we have found by searching the Web systematically; five tools can be retrieved from their own Web page or Google Code, and four tools can be found on forums and other online communities.

[6] `http://linuxcommand.org/man_pages/expect1.html`

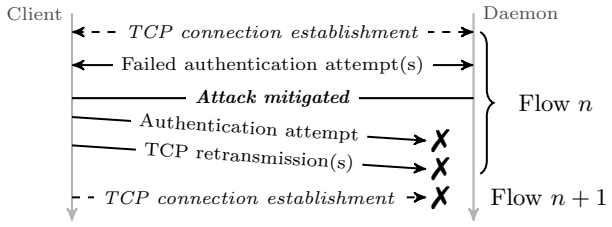


Figure 3: Attacker behavior after mitigation.

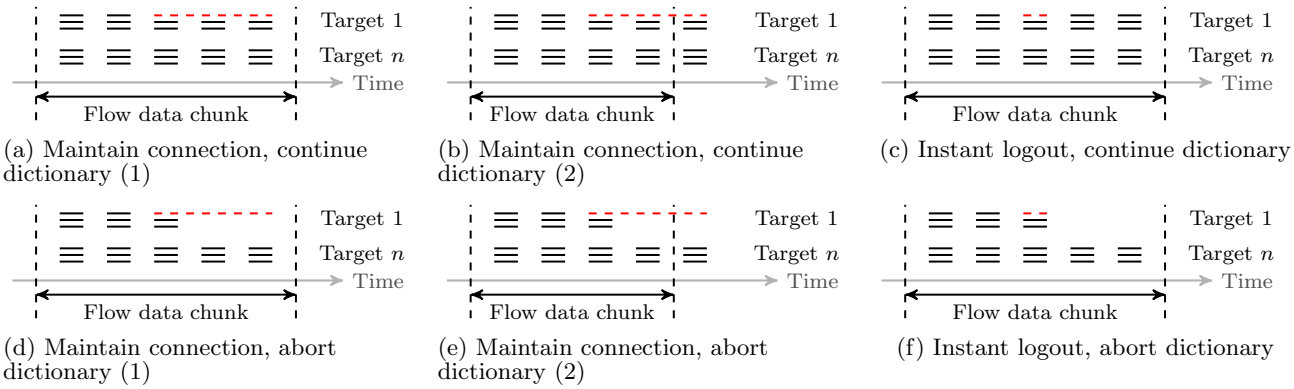
Several observations have been made regarding the analyzed tools. First, the number of authentication attempts per connection varies between 1 and `MaxAuthTries`. Tools that perform a single attempt per connection are typically faster; establishing one connection per attempt avoids waiting for the SSH daemon to report the authentication failure and the client to show a new prompt. Second, the number of PPF per login attempt very much depends on the configuration of the daemon, rather than the attack tool. Third, the attack tools' *action upon compromise*, i.e., the behavior of the attack tool with respect to the connection on which a successful authentication has taken place, was found to be the clearest indicator of a compromise, and is therefore key to our detection algorithm. This finding is crucial, as it makes the detection of compromises independent of the absolute number of PPF, which varies per attack tool and SSH daemon. We have identified four actions upon compromise:

- *Maintain connection, continue dictionary* – The connection with successful authentication is maintained, until the end of the attack. The attacker continues with the attack, also towards the compromised host.
- *Maintain connection, abort dictionary* – The connection with successful authentication is maintained, until the end of the attack. Other attack traffic towards the compromised host is stopped.
- *Instant logout, continue dictionary* – The connection with successful authentication is closed right after the compromise, while the attacker continues to attack both the compromised host and others.
- *Instant logout, abort dictionary* – The connection with successful authentication is closed right after the compromise. Other attack traffic towards the compromised host is stopped.

These actions form an integral part of the detection algorithm presented in Section 4. Actions featuring an instant logout are most prevalent in the considered tools.

3.3 Attack Mitigation Mechanisms

Several SSH attack mitigation mechanisms have been developed over the years, to reduce the risk of compromises during a brute-force attack. These mechanisms exist for both the host-level and the network-level. Host-level mechanisms, usually software-based, scan authentication log files and as soon as the number of failed authentication attempts exceeds a threshold, traffic from the attacker is blocked. Blocking can be performed on several layers. First, on L5, tools like *denyhosts* still allow TCP connections to be established to the target machine, while setting up SSH connections from the attacking host is prohibited. Since no packets



(a) Maintain connection, continue dictionary (1)
(b) Maintain connection, continue dictionary (2)
(c) Instant logout, continue dictionary
(d) Maintain connection, abort dictionary (1)
(e) Maintain connection, abort dictionary (2)
(f) Instant logout, abort dictionary

Figure 4: Various types of compromise flows in a chunk of flow data.

are dropped at the connection-level, no retransmissions or failing connection establishments can be observed. Second, tools like *fail2ban*, *sshdfilter* and *SSHblock* operate on L4 by instructing a local firewall to block traffic from the attacker to the target. If mitigation takes place while a TCP connection is active, retransmissions will occur. Also new TCP connections to the target cannot be established anymore, resulting in `SYN`-only flows. Both situations are shown in Figure 3, where the number of PPF of Flow n deviates from typical brute-force flows, due to the additional packets involved in the retransmission(s). After *Flow n*, there will be at least one `SYN`-only flow (Flow $n+1$). Third and last, tools like *SSHGuard* drop any traffic from the attacker's IP address using a local firewall, i.e., at L3. From a network traffic perspective, the behavior is identical to a L4-block.

Besides host-level mitigation mechanisms, also network-level mechanisms can be in place. These mechanisms are usually operated by packet forwarding devices, performing some sort of traffic blocking, e.g., by means of Access Control Lists (ACLs) or null-routing. Blocking rules can be composed based on blacklists or detections on honeypots, for example. The network traffic after mitigation is similar to host-level mitigation on L3 or L4.

4. DETECTION ALGORITHM

Our three-phase attack model, presented in Section 2, foresees compromises only after the *brute-force* phase; by the nature of SSH, a compromise can only occur after one or more authentication attempts. As such, the operation of the *brute-force* phase detection is essential for detecting compromises. We therefore start describing our *brute-force* phase detection shortly (Section 4.1), before discussing our *compromise* phase algorithm (Section 4.2). In the remainder of this work, we define an *attack* as a set of one of more tuples of attacker and target featuring brute-force behavior.

4.1 Brute-force Phase

Potential *brute-force* phase traffic is selected by considering all hosts sending SSH flows with a number of PPF in the range $r = [11, 51]$ to a daemon, where 11 is the minimum number of packets needed for a single authentication attempt, and 51 the highest number of PPF observed[7] for *brute-force* phase traffic (see Section 3.2). From all selected traffic, we take the most frequently used number of PPF as the *baseline* for identifying deviations for that particular attack. After establishing the *baseline*, we analyze the flow data per tuple of attacker and target; as soon as two or more consecutive flows with the same number of PPF are observed, we consider this an attack in the *brute-force* phase. The higher this threshold, the higher the chance of ruling out benign authentication attempts. Considering that benign authentications would come in groups of three (because the OpenSSH client sets `NumberOfPasswordPrompts` to 3 by default), two consecutive SSH flows with the same number of PPF would already indicate six failed attempts.

4.2 Compromise Phase

Key to our compromise detection are the four actions that can be observed after a compromise. We have transformed these actions into six scenarios, as shown in Figure 4. The two additional scenarios have been defined to accommodate for the fact that many analysis applications receive and process flow data in fixed-size time bins, as a consequence of which our algorithm has to take into account that attack data may be spread over multiple data chunks. Each of the subfigures shows a flow data chunk, with flows (long dashes) towards targets running an SSH daemon. Short-dashed lines mark a flow with a compromise.

In Figure 4(a), we show that the compromise flow is maintained until the end of the attack, and that other login attempts are observed in parallel towards the same target. A similar scenario is shown in Figure 4(b), but since the end of the attack does not lie within the current data chunk, the compromise flow is characterized by an unterminated TCP connection (i.e., without a TCP `FIN` or `RST` flag set). Similarly to these two scenarios, we show in Figure 4(d) and 4(e) how the compromise flows should be identified in case the attacker aborts its dictionary towards the compromised target: traffic from the same attacker towards other targets reveals the end of the attack. Figure 4(c) and 4(f) show situations where the attack tool performs an instant logout upon compromise. Observe that the compromise in Figure 4(f) may also be very close to the end of the data chunk, which is why compromises classified according to this scenario are checked in the next data chunk again, to verify whether there is no traffic from the attacker towards the compromised target.

[7]The numbers provided in this paper are backed up by measurements and higher than reported by related works, which report maximum values of around 30 [8, 9]. Cisco appliances and Mac OS X are the main cause of these high values.

Table 1: Dataset composition

	Honeypots	Servers	Workstations	Attacks
D1	13	0	0	632
D2	0	76	4	10716

Table 2: Validation results per dataset

	TPR	TNR	FPR	FNR	*Acc*
D1	0.692	0.921	0.079	0.308	0.839
D2	–	0.997	0.003	–	0.997

The compromise detection can be summarized in two steps:

Step 1 – Matching traffic against scenarios. As soon as a *brute-force* phase has been detected between attacker and target, this phase aims at detecting one of the scenarios shown in Figure 4. In case of a match, a compromise is detected. Special care must be taken with unterminated connections, as shown in Figure 4(b) and 4(e), as they may be the result of an attacker stopping its attack without properly closing all connections, instead of a compromise. If this is the case, the SSH daemon will timeout and close the connection after `LoginGraceTime`, as discussed in Section 3.1. At the flow-level, this can be verified by checking the duration of the return flow, i.e. the flow from target to attacker. In case its duration matches the configured `LoginGraceTime`, the attack was stopped without a compromise. Otherwise, we consider the target host to be compromised.

Step 2 – Identification of mitigation mechanisms. After identification of a matching scenario, the traffic is checked for signs of activated mitigation mechanisms. As soon as these mechanisms are activated, at least one of the following situations may apply: 1) Mid-connection mitigation can result in a number of PPF that is higher than the identified *baseline*, due to retransmissions between attacker and target, or 2) new connections have merely a TCP `SYN` flag set and typically consist of three packets, which is a frequently used retry count for establishing TCP connections. Identification of mitigation mechanisms is crucial, as they would trigger false positive compromise detections (i.e., those that feature an instant logout) otherwise.

5. VALIDATION

In this section, we present the validation of our detection algorithm. We start by describing our two datasets in Section 5.1. To be able to evaluate the algorithm, we have implemented it as part of *SSHCure* v2.4.[8] The validation results are discussed in Section 5.2. After that, we evaluate the performance of SSHCure as a system in Section 5.3.

5.1 Datasets

We collected two datasets on the campus network of the UT. The network features a publicly routable /16 IPv4 address block, of which typically 25k addresses are actively being used. Both datasets comprise a period of one month, collected in November and December 2013, and January and February 2014, respectively, consisting of the following data:

- *Honeypot log files* – These have been collected from various low-, medium- and high-interaction honeypots.
- *Workstation & server log files* – These have been collected from workstations and servers, which all have a publicly accessible SSH daemon.
- *Flow data* – Flow data with a sampling rate of 1:1 has been collected at edge routers and, as such, contains all SSH traffic entering and leaving our campus network.

[8] *SSHCure* is available at `http://sshcure.sf.net`

The exact composition of the datasets is shown in Table 1, which shows the number of honeypots, servers and workstations of which the log files comprise the dataset, and the number of attacks identified.

The datasets have been chosen carefully to reflect two completely different types of systems. On the one hand, Dataset D1 solely consists of data from honeypots, i.e., systems set up for being compromised (very easily). On the other hand, Dataset D2 is made up of data from mostly servers, which are likely configured with very strong passwords and attack mitigation mechanisms. In addition, we deliberately selected workstations, servers and honeypots that are operated by different persons, to ensure that our results are not biased by similar configurations.

Since the collected log files are the one and only proof of whether login attempts have succeeded or not, we consider this data the *ground-truth* for the validation of this work. More precisely, we consider all successful authentications after more than six login attempts and have no idle period of more than one hour to be compromises, and exclude all logins from hosts in the IP address range of the UT. Note that in a benign situation, six attempts will typically be carried by two flows (due to `NumberOfPasswordPrompts` being set to 3 in the configuration of the OpenSSH SSH client), while up to six flows may be observed during an attack, in case an attacker performs only one authentication attempt per connection. Since our goal is to validate the detection of compromises, we consider only those attacks that show *brute-force* phase behavior.

5.2 Detection Accuracy

We have compared the log files to *SSHCure*'s detection results based on the following metrics:

- *True Positives* (TP), *False Positives* (FP): Attacks correctly and incorrectly identified to feature a compromise, respectively.
- *True Negatives* (TN), *False Negatives* (FN): Attacks correctly and incorrectly identified to not feature a compromise, respectively.

We evaluate our detection algorithm by means of its accuracy (*Acc*), which is a measure for the number of attacks that has been classified correctly and is defined as follows:

$$Acc = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}} \quad (1)$$

The results of our evaluation are listed in Table 2, where the percentages of the evaluation metrics are used. For example, the *True Positive Rate* (TPR) is the percentage of attacks correctly identified to feature a compromise.

Although the number of correctly classified attacks in D1 is very high, yielding an accuracy of 84%, we have to face incorrect classifications as well. On the one hand, FPs are mainly a result of the sensitivity of the *instant logout, continue attack* scenario (Figure 4(c)). In this scenario, we observe deviations in the number of PPF from the identified

baseline. We have found to need a slightly higher sensitivity to obtain the results in Table 2, than for servers, for example. The sensitivity may be reduced for honeypots, which will reduce the FPs, while the number of FNs increases. On the other hand, the FNs are caused by the result of the nature of honeypots, and show how the characteristics of the dataset limit our approach; the easy-to-guess credentials of honeypots can result in compromises from the first authentication attempt or even compromises after every attempt. As such, the *baseline* for identifying deviations in the number of PPF, i.e., compromises, cannot be established reliably.

The explanation for incorrectly classified attacks in D1 is confirmed by the evaluation results of D2, where no compromises are captured. We assume that this is due to the low number of workstations compared to servers considered, as server administrators typically have more system administration skills than the average workstation user, typically resulting in stronger login credentials. However, a much higher accuracy (close to 100%) is achieved for D2, due to the fact that only 0.3% of the attacks is incorrectly classified.

5.3 SSHCure Performance

We are aware of many successful deployments of *SSHCure*, in networks at different scales: campus networks, hosting companies, ISPs, and Computer Security Incident Response Teams (CSIRTs) up to government-level. We also closely collaborate with CESNET, which uses *SSHCure* on a central flow collector where flow data from all peering links is collected. Given that we aim at deploying *SSHCure* in high-speed networks, we have evaluated whether *SSHCure* is able to analyze CESNET's SSH traffic in real-time, i.e., every data chunk should be processed before the next data chunk arrives. In January 2014, where up to 29.9 GB of SSH traffic per five minute data chunk has been transferred, SSHCure was able to do so for the vast majority of data chunks. Only in situations with many large and concurrent attacks, SSHCure was not able to finish in time on our measurement system, after which it automatically skipped the next data chunk to not overload the collection system.

In addition to evaluating *SSHCure*'s processing performance, we have compared its detection results to the OpenBL SSH blacklist. OpenBL deploys more than 40 sensors that report which host has performed brute-force login attempts on port 22. With only a single instance of SSHCure deployed at the UT, we already achieve a coverage of OpenBL of up to 3% per day over January 2014, defined as the share of IP addresses blacklisted by OpenBL that was also reported by SSHCure. By deploying SSHCure in the CESNET network, we even achieve a coverage of up to 7% per day with a single sensor. In addition, 14–37% of the attacker IP addresses reported by SSHCure at UT was not (yet) blacklisted by OpenBL, while in the CESNET network, this percentage was 47–95%. We therefore envision SSHCure to be used as a complementary sensor for SSH blacklisting.

6. CONCLUSIONS

This paper has presented a detection algorithm for the flow-based detection of hosts compromised during SSH dictionary attacks. We show that accuracies close to 100% can be achieved, while deployment in backbone networks is still feasible. Given the potential damage caused by compromised hosts, we believe that a flow-based compromise detection is a must-have for any larger-scale network.

In spite of the achieved detection results, we are aware of at least two situations that impair the detection of compromises. First, we are aware of attack tools that perform *flow stretching*, a technique that inserts random data in SSH packets such that hardly two flows in an attack appear similar. The consequence is that attacks are rarely detected – neither by our algorithm, nor by related works. Second, we have observed that retransmissions can cause false positive detections, especially from locations far-away from the observation point. This is because retransmissions are not exported in flow data, except for increased packet and byte counters. To overcome this, we plan to define and implement new IPFIX Information Elements for the explicit export of TCP control information. In addition, we plan to investigate how to detect stealthy, distributed SSH attacks based on flow data only; works that have described these attacks yet, such as [5], still rely on other data sources, such as syslog. Although we can already detect distributed attacks, they still need to hit certain thresholds to be reported – an assumption that does not always hold for stealthy attacks.

7. ACKNOWLEDGMENTS

This work is partly funded by FLAMINGO, a Network of Excellence project (ICT-318488) supported by the European Commission under its Seventh Framework Programme. Special thanks go to Václav Bartoš (CESNET), CERT-UT, ICTS-UT, and the OpenBL project, for their valuable contributions to the research process.

8. REFERENCES

[1] 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), September 2013.

[2] M. Drašar. Protocol-Independent Detection of Dictionary Attacks. In *Proceedings of EUNICE'13*, 2013.

[3] L. Hellemons, L. Hendriks, R. Hofstede, A. Sperotto, R. Sadre, and A. Pras. SSHCure: A Flow-Based SSH Intrusion Detection System. In *Proceedings of AIMS'12*, 2012.

[4] 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*, 2014.

[5] M. Javed and V. Paxson. Detecting Stealthy, Distributed SSH Brute-Forcing. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & Communications Security*, 2013.

[6] A. Sperotto, R. Sadre, P.-T. de Boer, and A. Pras. Hidden Markov Model modeling of SSH brute-force attacks. In *Proceedings of DSOM'09*, 2009.

[7] Venafi, Inc. Ponemon 2014 SSH Security Vulnerability Report. Technical report, Venafi, 2014.

[8] J. Vykopal. *Flow-based Brute-force Attack Detection in Large and High-speed Networks*. PhD thesis, Masaryk University, Brno, Czech Republic, 2013.

[9] J. Vykopal, T. Plesnik, and P. Minarik. Network-based Dictionary Attack Detection. In *Proceedings of ICFN'09*, 2009.