Network-Based Dictionary Attack Detection

3 authors:

Jan Vykopal
Masaryk University
26 PUBLICATIONS   **148** CITATIONS

Tomas Plesnik
Masaryk University
4 PUBLICATIONS   **20** CITATIONS

Pavel Minarík
Masaryk University
8 PUBLICATIONS   **62** CITATIONS

Some of the authors of this publication are also working on these related projects:

Project   KYPO cyber range View project

Project   SABU - Sharing and analysis of security events in Czech Republic View project

Network-based Dictionary Attack Detection

Jan Vykopal[1]   Tomas Plesnik[1]   Pavel Minarik[1,2]

Institute of Computer Science[1]
Botanicka 68a, Brno, Czech Republic
{vykopal, plesnik, minarik}@ics.muni.cz

Mycroft Mind[2], a.s.
Lidicka 28, Brno, Czech Republic
min@mycroftmind.com

Abstract

This paper describes the novel network-based approach to a dictionary attack detection with the ability to recognize successful attack. We analyzed SSH break-in attempts at a flow level and determined a dictionary attack pattern. This pattern was verified and compared to common SSH traffic to prevent false positives. The SSH dictionary attack pattern was implemented using decision tree technique. The evaluation was performed in a large high-speed university network with promising results.

1. Introduction

When we received a report that one of our computer was participating in a phishing scam, we started to analyze the compromised host. Finally, we found out someone broke in the host via secure shell (SSH). The administrator set up a weak password[1], thus a dictionary attack was successful. We also inspected other host logs in our network and concluded such dictionary attacks are very common. Moreover, we deployed new SSH server with a public IP address. Consequently, we encounter new attacks almost every day.

As this real example shows, although there are many advanced authentication schemes such as biometric or hardware tokens, a knowledge-based authentication is still widespread. Unfortunately, human chosen passwords are inherently insecure since a large fraction of the users chooses passwords that come from a small domain. This enables adversaries to attempt to login to accounts by trying all possible passwords, until they find the correct one. This attack is known as a *dictionary attack* [6]. Some studies [8, 1, 9] confirm SSH dictionary attacks are very com-

mon all around the world. In addition, the results of another survey on password selection and memorability [10] say this type of attack was the most successful even if users were forced to select non-trivial passwords.

The paper is organized as follows. Section 2 summarizes the state of the art in this area. Our findings from an analysis of real SSH dictionary attacks in university network are described in Section 3. Validation of the general SSH dictionary attack pattern is presented in Section 4. Section 5 deals with the implementation details. The evaluation is described in Section 6. Finally, Section 7 concludes and discusses future work.

2. Related Work

There are two basic approaches how to detect network anomalies. Considering host-based dictionary attack detection we can use many tools that parse Unix/Linux logs such as *logwatch*[2], or specialized ones such as *DenyHosts*[3]. The former is a pluggable application that goes through system logs for a given period of time and make a report. The latter analyzes the *sshd* server log messages. It keeps tracks of the frequency of attempts from each host and eventually blocks hosts that try to login unsuccessfully many times. This approach is satisfactory at a host level. Recently, a distributed active response architecture appeared [9]. It enables a collaboration of trustworthy hosts. In short, if one host detects a dictionary attack it informs the others about the attacker's IP address and they block the hostile address too.

In contrast to host-based approach, we are not aware of any specialized network-based methods of the dictionary attack detection besides some particular Snort rules. These rules actually describe network traffic at a flow level al-

---

[1]It was the same as the domain name.

[2]http://www.logwatch.org/
[3]http://denyhosts.sourceforge.net/

though Snort was originally developed as a signature-based IDS/IPS operating at layer 7 of the ISO/OSI model. Therefore, it suffers from performance degradation on a fully saturated gigabit line. On the contrary, another approach relies entirely on traffic statistics at layers 3 and 4 of the ISO/OSI model. For example, CAMNEP [7] is an agent-based network IDS. It comprises a few *general* methods of network behavior analysis and profits from the synergy effect. But the authors admit it is not capable of detection the *less visible* attacks. That means the attacks consisting of approximately up to 300 flows. General statistical-based methods are also not capable to distinguish between unsuccessful and successful attack.

3. Analysis of Real Dictionary Attacks

We analyzed real attacks in the following way. First, we inspected logs on attacked hosts and then identified appropriate traffic in NetFlow data collected at the border of a university network or its subnets. A flow is defined as a unidirectional sequence of packets with some common properties that pass through a network device. These collected flows are exported to an external device, the NetFlow collector. Network flows are highly granular; for example, flow records include details such as IP addresses, packet and byte counts, timestamps, Type of Service (ToS), application ports, input and output interfaces, etc. [4]

3.1. Host-based Analysis

A typical host based analysis means server log inspection. During password authentication, the user supplies a password to the SSH client, which the client transmits securely to the server over the encrypted connection. The server then checks that the given password is acceptable for the target account, and allows the connection if so [2]. In addition, the authentication procedure is logged in Unix/Linux operating systems by default. Note that the attacker with superuser privileges can erase or forge the logs in case of a compromised host. In the case mentioned at the beginning of this paper, the log contains many unsuccessful access events preceding a successful one.

Further, we deployed a new SSH server on our Linux desktop with a public IP address. We set strong passwords for all accounts and monitored access log for 30 consecutive days.

We encountered 911 attempts from 15 IP addresses in total. Table 1 shows the top 12 user account names and the number of failed attempts for each account. The first four places are occupied by the same account names as in Table 1 in [8] and some other names are present in both tables too. On the contrary, the last four places in our table are occupied by first names common in Czech. We inspected the

Table 1. Top failed login attempts

Account name	Number of login attempts
root	104
test	24
admin	13
guest	9
user	7
temp	7
student	6
dino	5
michael	4
martin	4
josefa	4
josef	4

relevant log and found many other Czech names. Hence, we suppose attackers observed a domain name of our computer and consequently chose appropriate dictionary.

3.2. Flow-based Analysis

At the next stage of analysis, we used both software and hardware-accelerated NetFlow probes [3]. We decided for these probes because they ensure no packet loss and reliable NetFlow export. First of all, we searched for reconnaissance activities such as TCP SYN and FIN scanning that are *invisible* in access logs. Only one IP address that performed TCP SYN scanning before the dictionary attack was found. So we focused on this host and discovered other 93 592 TCP SYN probes to 64 251 hosts in our class B network. In a nutshell, TCP scanning is not often preceding to SSH dictionary attacks.

Next, we focused on IP addresses of attackers in relevant time windows known from host based analysis. We derived the following dictionary attack pattern at NetFlow level:

- TCP port of the victim is 22[4], TCP port of the attacker is generally random and greater than 1024,

- many flows (tens or hundreds) from the attacker to the victim in a short time window (5 minutes),

- the flows from the attacker are *small*: from 10 to 30 packets and from 1 400 to 5 000 bytes,

- victims' responses are *small* too (typically the same number of packets and bytes),

- flow duration is *up to 5 seconds*,

- the last flow is different in case of successful attempt.

---

[4]Generally, the pattern is not limited to port 22.

4. Synthesis of Attack Pattern

4.1. Flow Analysis of SSH Traffic

We also analyzed network traffic of various applications that utilize the SSH protocol to eliminate *false positives*. We perform the following scenarios in both Linux and Windows operating system:

- remote login via *ssh*[5] and consequent execution of two commands,
- remote login via *putty* 0.60 and consequent execution of two commands,
- remote copy of a large file (104.1 MB) via *scp*,
- remote copy of 835 files in 315 directories (5.3 MB) via *scp*,
- remote copy of 786 files in 192 directories (1.5 MB) via *WinSCP* 4.0.6 (Build 358)) in SCP and SFTP mode,
- download a large file (104.1 MB) via *sftp*,
- backup 8 000 files in 666 directories via *rsync* 2.6.9.

To sum it up, we did not find any traffic that *fully* corresponds to the attack pattern derived from real dictionary attacks. That means there was observed no flow that meets all requirements above.

In real traffic, we observed a large file upload via SSH: traffic in one direction was formed of large packets whereas in the second direction acknowledgments and transmission of file transfer status look like break-in attempts. Finally, we coped with this issue and details are described in Section 5.

4.2. Simulated Dictionary Attacks

On the contrary, the attack pattern matches all flows during our simulated attacks. We conducted four attacks to validate the derived attack pattern. We created a special account on the SSH server with a known password. We used two dictionaries: the first one contains 100 passwords different to the new valid password and the second dictionary was the same except the last password. This one was replaced by the valid password of the new account. These dictionaries were passed to an Except script coming from the *black hat* community. In terms of particular attacks, we observed that reply flows to the attacker have mostly contained the same number of packets of the same length. This is completely different to common use (non-malicious) of the application discussed above. In addition, the simulation confirms that the last flow differs in case of the successful attack.

---

[5]Version OpenSSH_4.7p1 Debian-8ubuntu1.2, OpenSSL 0.9.8g 19 Oct 2007. *scp* and *sftp* are the same version.
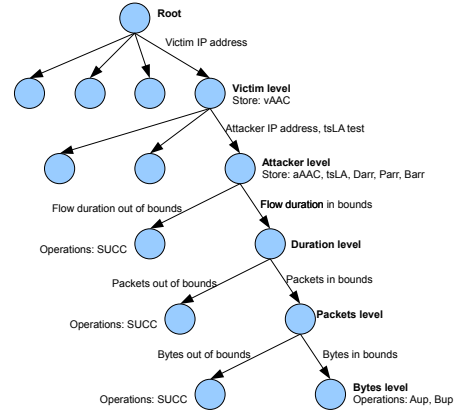

Figure 1. *SSH-D-tree* **diagram.**

5. Implementation

The main idea of the detection algorithm is to tune up key indicators (flow duration, number of packets and number of bytes transferred in victim's reply to the attacker) of the proceeding attack between attacker and its victim and observe significant change of these indicators. Sudden and significant change of the flow followed by the stop of the attack may indicate successful attack. While attacks may vary among different attackers and victims (different SSH protocol implementation, attack strategy, etc.), we need really adaptable approach to detect successful attack. According to these requirements we chose decision tree method to implement the algorithm. Using decision tree, we are able to store network traffic statistics, attack indicators and relevant detection parameters persistently. Decision tree approach is also suitable to handle large data sets.

Our SSH detection decision tree (*SSH-D-tree*) is presented in Figure 1. *SSH-D-tree* uses a subset of attributes of NetFlow record corresponding to SSH server response as the input data. This subset is formally a n-tuple: $source\_ip\_address$, $destination\_ip\_address$, $flow\_start$, $duration$, $packets$, $bytes$. Each component of this n-tuple except of $flow\_start$ forms a level of the *SSH-D-tree*. To avoid false positives, especially scans, we consider only responses having number of packets greater then certain constant. Another false positives may raise from secured data uploading; therefore we use bytes per packet indicator in requests corresponding to given replies. This heuristics helps to filter out SSH server replies which are only TCP acknowledgments actually.

5.1. Detection Algorithm Description

SSH-D-tree starts with a set of input n-tuples ordered by $flow\_start$ and processes individual flows in sequence.

The tree also starts with generic bounds for flow duration, number of packets and number of bytes transferred that fit to all attacks. For each pair $(attacker, victim)$ arrays of attack indicators are build (duration array, packets array and bytes array) until $singleAttackAttempts$ threshold is reached. New bounds for given attacker and victim are calculated using toleration parameters afterwards. From that moment each flow between the bounds is considered as unsuccessful attack and each flow out of bounds is considered as successful attack. More detail and formal explanation of the detection algorithm is available in sections 5.2 and 5.3.

5.2. Parameters

SSH-D-tree uses two types of parameters. First type of parameters is static. These parameters specify namely the sensitivity of the detection procedure. The second type of parameters is called dynamic. These are the parameters of the detection procedure depending on current attack progress and characteristics.

- **Static (initial)**
  - **arraySize** – length of attack indicator history influencing bounds for duration, packets, bytes
  - **singleAttackAttempts** – number of attempts from single source to alert attack (length of learning process of the attack indicators between attacker and its victim)
  - **distributedAttackAttempts** – number of attempts from all sources to alert attack
  - **certaintyReductionFactor** – reduction factor of the success of attack
  - **tsDelta** – maximum time difference between two attempts of attack from single source
  - **initialDurationBounds, initialPacketsBounds, initialBytesBounds** – initial bounds of duration, numbers of packets and bytes in SSH server response
  - **deltaDuration, deltaPackets, deltaBytes** – tolerance factors to calculate dynamically changing bounds of expected flow duration, number of packets and number of bytes transferred

- **Dynamic (calculated)**
  - **vAAC, aAAC** – victim/attacker attack counter.
  - **tsLA** – last attack attempt time stamp.
  - **Daar, Paar, Baar** – duration array, packets array, bytes array computed continuously during the attack for each pair $(victim, attacker)$.

5.3. Operations

This section summarizes operations performed during the detection process except bounds check operations which

are obvious.

- **tsLA test** (Last attack attempt time stamp test)
  - `flow_start > tsLA + tsDelta` $\Rightarrow$ drop whole subtree from the given attacker level and create a new one.
- **SUCC** (Successful attack report)
  - `aAAC > singleAttackAttempts` $\Rightarrow$ mark attacker as successful and victim as compromised.
- **Aup** (Attack attempt update)
  - `aAAC = aAAC + 1, vAAC = vAAC + 1`.
  - `aAAC > singleAttackAttempts` $\Rightarrow$ mark attacker as unsuccessful and victim as resisting or divide certainty of successful attacker and compromised victim by `certaintyReductionFactor`.
- **Bup** (Bounds update)
  - update arrays `Daar, Paar, Baar` according to actual values of flow duration, number of packets and number of bytes transferred. Calculate new bounds as the arithmetic mean of values in the array reduced by or increased by tolerance factor respectively.

SSH-D-tree was implemented using *Classification Space* which is a part of *Mycroft Technology*. Short preview of *Mycroft Technology* is available at [5].

6. Evaluation

Although we manually validated the dictionary attack pattern, we evaluated the implementation of the pattern. We automatically analyzed the NetFlow data for the attacks revealed by the host based log inspection described in Section 3. Because there is a general assumption that attacker tries to break-in to the many hosts of a network, we inspected other activities of attackers.

6.1. Test Bed

The source of the NetFlow data was the same as in Section 3. We constantly collect data in NetFlow version 9 format at the border of a university network and subsequently deploy new probes on some subnets inside the network. The whole network is connected to the Internet with 10 gigabit link. The subnets are usually interconnected by a gigabit links. Collected data are stored at a NetFlow collector and on-demand sent to the software implementing the dictionary attack detection.

Table 2. Known attackers and the number of their other SSH dictionary attacks in the same time.

Anonymized IP address	Number of other attacks
10.0.43.12	320
11.0.66.214	170
12.0.145.8	94
13.0.89.2	72
14.0.66.10	69
15.0.104.15	106
16.0.97.196	373
17.0.200.190	45

6.2. Results

As we supposed, all of the previously known attacks were reported. What is more, no other attack was detected for the known host in the given time interval. Further, we detected that all IP addresses trying to break-in to the controlled SSH server[6] conducted also the SSH dictionary attack to other hosts in our network. Table 2 shows the numbers of other attacks for each known attacker ordered by the number of break-in attempts to our test SSH server.

From the performance point of view, our *SSH-D-tree* is able to process approximately 2 500 flows per second on COTS (cost of-the-shelf) hardware. Whole one day university traffic is processed in less than one hour including overhead (data deliver, pre-filtering, sorting, etc.). Therefore we are able to deploy *SSH-D-tree* on-line.

Last, but not least, we detected numerous other previously unknown SSH dictionary attacks in our network. Unfortunately, we could check the access logs only in case of four hosts. The log inspection confirms that all of these attacks really occurred.

7. Conclusion

We presented the SSH dictionary attack pattern at a flow level derived from real attacks conducted to hosts in multigigabit university network. We validated the attack pattern itself and its implementation including the ability to recognize successful and unsuccessful attacks by simulated attacks and analysis at a host level. The implementation of the pattern profits from the network based approach: it is inherently scalable, transparent for network traffic, OS-independent and demands constant maintenance effort (not proportional to the number of monitored hosts).

We also identified the following possible *false negatives*. If the attacker hits the valid password at the very beginning

---

[6]We consider IPs with more than 1 attempt.

of the attack, the detection procedure is not successful, because the threshold $singleAttackAttempts$ is usually initialized to 20. However, it is very unlikely scenario and we can still detect attacks consisting of tens or a few hundreds of attempts. Next, the method is unable to classify whether the attack is successful or not, if the change of volume characteristics of the last flow fits dynamically computed bounds. Anyway, the attack is reported. Considering *false positives*, we encountered only one case: the attack pattern matches SSH traffic generated by Nagios monitoring application.

In our future work, we will focus to classify not only attacking IP addresses and their victims, but rather the certain malicious flows. Further, the method should also provide some reasons for the decision that the attack occurred (e. g., distributed dictionary attack).

Acknowledgment

References

[1] E. Alata, V. Nicomette, M. Kaaniche, M. Dacier, and M. Herrb. Lessons learned from the deployment of a high-interaction honeypot. In *EDCC '06: Proceedings of the Sixth European Dependable Computing Conference*, pages 39–46, Washington, DC, USA, 2006. IEEE Computer Society.

[2] D. J. Barrett and R. E. Silverman. *SSH, The Secure Shell: The Definitive Guide*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2001.

[3] P. Celeda, M. Kovacik, T. Konir, V. Krmicek, P. Springl, and M. Zadnik. Flowmon probe. In *Network Studies: Selected Technical Reports*. PB tisk, s.r.o., http://www.cesnet.cz/doc/techzpravy/2006/flowmon-probe/, 2007.

[4] B. Claise. Cisco Systems NetFlow Services Export Version 9. RFC 3954 (Informational), Oct. 2004.

[5] Mycroft Mind, a.s. Mycroft Mind, a.s. Technology Profile, http://www.mycroftmind.com/technology.

[6] B. Pinkas and T. Sander. Securing passwords against dictionary attacks. In *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security*, pages 161–170, New York, NY, USA, 2002. ACM.

[7] M. Rehak, M. Pechoucek, K. Bartos, M. Grill, P. Celeda, and V. Krmicek. CAMNEP: An intrusion detection system for high-speed networks. *Progress in Informatics*, (5):65–74, March 2008.

[8] C. Seifert. Analyzing malicious ssh login attempts, http://www.securityfocus.com/infocus/1876, 2006.

[9] J. L. Thames, R. Abler, and D. Keeling. A distributed active response architecture for preventing ssh dictionary attacks. In *IEEE Southeastcon 2008*, pages 84–89, 2008.

[10] J. Yan, A. Blackwell, R. Anderson, and A. Grant. Password memorability and security: Empirical results. *IEEE Security and Privacy*, 2(5):25–31, 2004.