

# Analyzing Malicious SSH Login Attempts

*Christian Seifert* 2006-09-11

## Introduction

Malicious SSH login attempts have been appearing in some administrators' logs for several years. This article revisits the use of honeypots to analyze malicious SSH login attempts and see what can be learned about this activity. The article then offers recommendations on how to secure one's system against these attacks.

## Using honeypots for research

The [New Zealand Honeynet Alliance](#) is a research organization and member of the [Honeynet Alliance](#), which is dedicated to improving the security of computer systems and networks by researching the behavior, tactics, and tools of black hat hackers through the use of honeypot technology. Honeypots are computer systems whose value lies in their openness to attack and compromise, allowing the researcher to analyze malicious activity on the system.

We have set up such a system at the Victoria University of Wellington to investigate malicious activity that occurs on a university network in New Zealand. This system was a high interaction honeypot that an attacker can interact with like any other system on the network. As far as the attacker is concerned, there should be no discernible difference between the honeypot and other computer systems. However, it is closely monitored through the Honeynet Alliance Roo honeywall that captures all network traffic flowing to and from the honeypot. In addition, system events are recorded on the honeypot itself via its logging facility.

The honeypot ran a standard server configuration of [RedHat 9](#) with a Secure Shell (SSH) server that was accessible via the public Internet. SSH is a program that allows a user to log into another computer over a network via an encrypted channel. After we encountered malicious SSH login attempts in previous setups, we configured our honeypot to allow for additional data collection. We patched the SSH server to record the password along with the account name that was used in the login attempt. The honeypot was brought online on July 11, 2006 and taken offline on August 1, 2006, after 22 full days. The honeypot was attacked numerous times during this period with login attempts on SSH. We take a closer look at the data to determine the tactics of the attackers and to make recommendations to improve security around SSH.

In an additional configuration of the honeypot, which ran from June 28 to July 4, we [added the Sebek module](#) that records key strokes of the attacker once the system has been compromised. We configured several user accounts with commonly used passwords. After a few days, an attacker successfully compromised the system. The analysis of this attack and subsequent attacks are presented in this paper and provide us with further insight into how the malicious SSH login attempts are used to compromise systems.

## Analysis of SSH Login Attempts

This section analyzes the data captured by our honeypot from July 11 to August 1. The analysis is entirely based on data of system log files of the honeypot, in particular the 'messages' log. The 'messages' log captures authentication requests to the SSH server. It captures date, time, the IP address from which the login attempt originated, the result of the request (failure or success), the account name and the password used for the authentication request. Two 'messages' sample log entries are shown below.

```
Jul 13 09:37:59 basta sshd[22308]: PW-ATTEMPT: fritz
Jul 13 09:37:59 basta sshd[22308]: Failed password for root from 10.0.160.14
port 39529 ssh2
Jul 13 09:38:02 basta sshd[22310]: Illegal user fatacunike from 10.0.160.14
Jul 13 09:38:02 basta sshd[22310]: PW-ATTEMPT: fatacunike
Jul 13 09:38:02 basta sshd[22310]: Failed password for illegal user fatacunike
from 10.0.160.14 port 40444 ssh2
```

First, we analyzed the login names that were used on the login attempts. During the sample period, there were 2741 unique account names ranging from common first names, system account names, and common accounts to short alphabetical strings captured by the system logger. Of those, the 15 account names used most often are shown in Table 1. This table shows accounts that usually exist on a system (root, mysql), accounts that are likely to exist on a system (guest, test), as well as common first names (paul). Then Figure 1 shows the distribution of valid and invalid account names that were used.

It comes as no surprise that the invalid account names used far surpass the valid account names. However, we note that 96.30% of all default account names that exist on the honeypot have been used in an attack.

Account Name	Number of login attempts
root	1049
admin	97
test	87
guest	40
mysql	31
info	30
oracle	27
postgres	27
testing	27

webmaster	27
paul	25
web	24
user	23
tester	22
pgsql	21

Table 1. Top 15 account names among 2741 attempts.

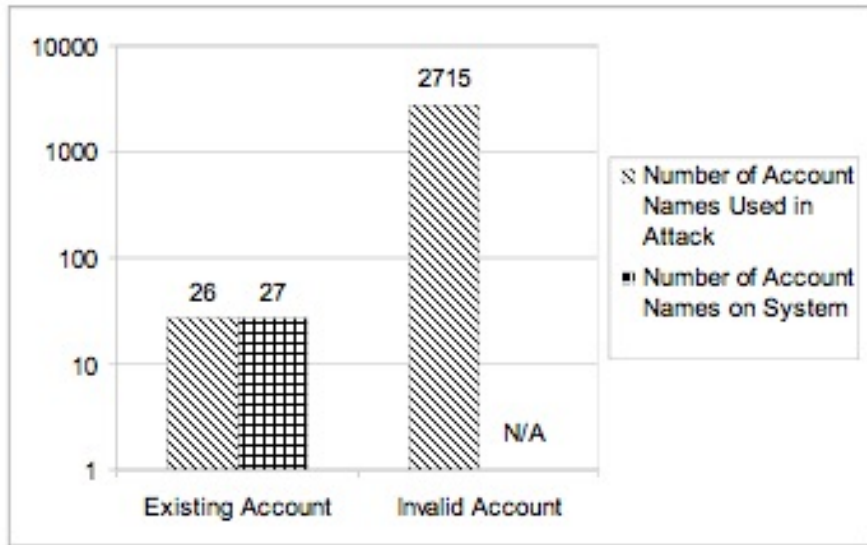


Figure 1. Number of account names, both existing and invalid.

Next, we looked at the passwords used in the login attempts. The attackers tried a range of passwords with most of the account names. In total during our analysis, they attempted to access 2741 different accounts and used 3649 different passwords. Not all passwords were used with all accounts. The passwords ranged from account names, account names with number sequences, number sequences, and keyboard sequences (like 'qwerty'). There were a few more complex passwords used with seemingly random letter and number sequences or common substitution passwords (like r00t or c@t@lin).

Table 2 shows the top 15 passwords used in malicious login attempts.

Password	Number of login attempts
123456	331
Password	106
Admin	47

Test	46
111111	36
12345	34
administrator	28
Linux	23
Root	22
test123	22
1234	21
123	20
Mysql	19
Apache	18
Master	18

**Table 2. Top 15 passwords attempted.**

Then we examined who attacked the honeypot and what strategy these attackers used. There were 23 unique IP addresses involved in the login attempts. The attackers were more or less persistent in their attempts to gain access to the system, as shown in Table 3. Ten of the sources tried less than 50 combinations and then gave up. Five tried harder with approximately 170 login attempts, and eight tried even harder with up to 1450 login attempts. Figure 2 shows the breakdown of login attempts per source address.

Number of Login Attempts	Unique IP Addresses
< 50	10
50 <= x <= 200	5
> 200	8

**Table 3. Frequency range of login attempts.**

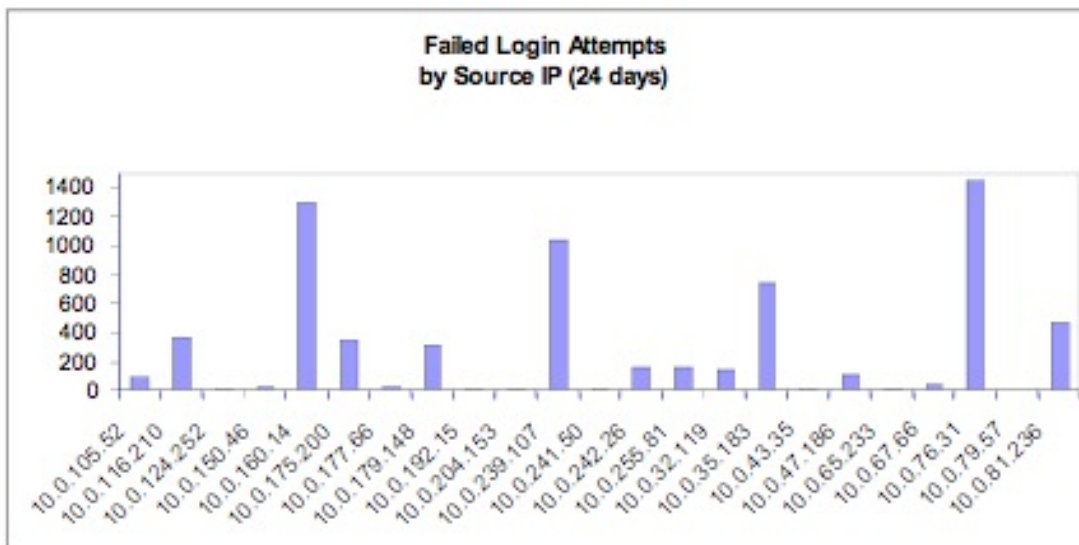


Figure 2. Failed login attempts by source IP.

A closer look at the attackers reveals more information about how the break-in attempts took place. The attackers tried one single account name per login attempt with a password that mostly matched the account name (e.g. test/test) and moved onto the next account name. Other attackers differed in their strategy. On average, attacker 10.0.179.148 tried five passwords before trying a different account. The passwords used in this attack scheme consisted of the number sequences or the account name with a combination of various number sequences (e.g. admin/admin, admin/admin1, admin/admin123, admin/111111). The attacker at 10.0.105.52 concentrated on the root account, attempting all but one of his login attempts (guest) with that account name. The passwords ranged from common passwords to random number and character sequences (e.g. root/!@#, root/123abc, root/default).

Several attackers exhibited behavior that was likely to evade the attention of an IDS by limiting their attacks to just a few attempts. Attacks became more serious by various degrees. First, the number of login attempts increased; second, the login attempts per account increased, and finally, we saw a concentration of login attempts on a particular account, like root. As attempts became more serious, detection of the attacker would become more likely if an intrusion detection system had been deployed. One would think that the attacker's success rate would increase with more attempts and a subsequent increased danger of detection, but we cannot confirm this as our data did not include a survey of account name/password combinations of existing systems.

In an effort to obtain further information about the methodology and tools the attackers used, we examined the variance and average time period between each login attempt per source IP. We assumed that if a tool was used in the attack, we could easily identify its usage by inspecting these values. A small gap between login attempts and a low variance would indicate usage of a tool, whereas larger irregular gaps would indicate the attacker was actually performing the login attempts in person. Figure 3 summarizes this data.

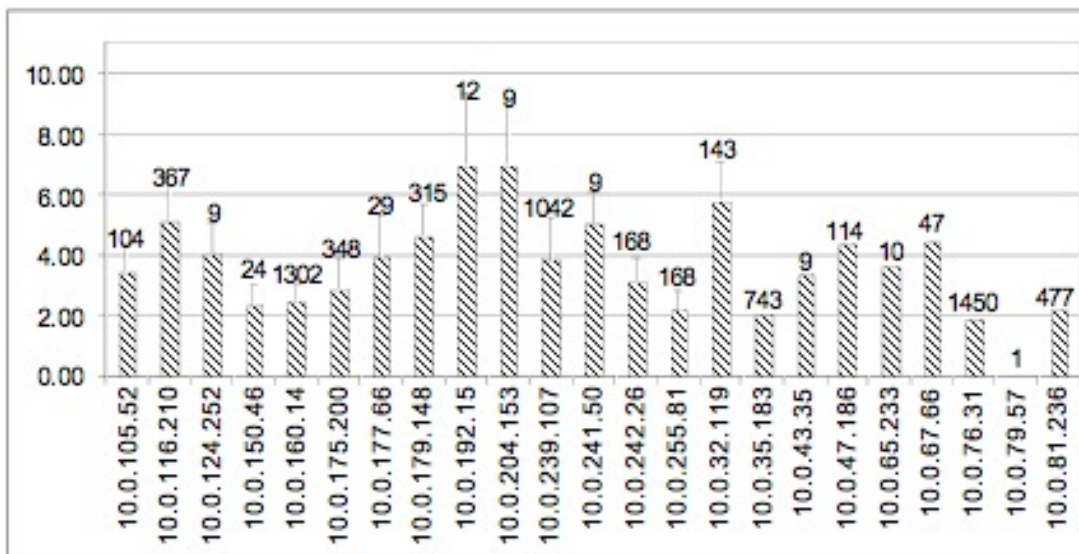


Figure 3. Average period between login attempts (seconds).

The top five offenders, who are likely to have used tools in their attacks due to the sheer number of attempts, show an average period between attempts of 2 to 4 seconds with a standard deviation that ranges from 0.45 to 1.39 seconds. On the other extreme, we find that the attacker at 10.0.192.15 with an average period between attempts of approximately 7 seconds and a standard deviation of 2.36 seconds - indicating a person might have performed the login attempts. However, the data does not support a clear indication of tool usage as many attack sequences fall in the middle or even below the values of the top offender attack sequences. A possible explanation might be that attack tools were used in most attack sequences and we are only observing network latencies.

Compared to the above attempts, a strong indicator of tool usage is revealed by examination of the account names and passwords used in the attacks from 10.0.255.81 and 10.0.242.26. Although the IP addresses resided in different physical networks and the attacks occurred 4 days apart, the account name/ password combination was identical.

## The "Successful" malicious SSH login attempt

In the previous section, we analyzed the data captured of unsuccessful malicious SSH login attempts. The data gives us some insight into how attackers operate, but leaves many questions unresolved. One of those questions is whether or not tools are used in these attacks.

On July 2, an attacker successfully compromised the honeypot by guessing a valid user account name/password on SSH. The data captured during this incident revealed answers to these open questions.

First we examined the actions of the attacker after the successful compromise of the honeypot. Once a valid account name/password was determined, the attacker logged into the honeypot via SSH and proceeded to download an SSH scanner tool. This tool is described in more detail in the following section, but for now it will be summarized as a tool that allows its user to identify and compromise other SSH servers through password guessing attempts. The installed tool was immediately used to scan a class B network from our honeypot. Due to the

restrictions on outgoing network connections enforced by the Roo honeywall, the SSH scanner did not identify any SSH servers.

After the initial scan, the attacker proceeded to download and install an IRC Bot. IRC Bots are tools that can control a compromised system remotely via IRC chat channels that the compromised system is set to listen to. Using IRC to control a compromised system is much more covert than using SSH directly, as the attacker does not have to directly log into the system anymore. Further, it allows the attacker to control several such systems, also known as Zombies, at the same time.

The conversations in the IRC channel revealed that the Zombies were used to scan class B networks with an SSH scanner just like the one that had been downloaded to our honeypot. During a period of a couple of hours, four class B networks were scanned with the SSH scanner from various IRC Bots, identifying thousands of SSH servers. A scan took approximately 700 seconds to complete. Further, we witnessed the exchange of an account name/password list with 160,324 unique account name/password combinations. The structure of these account names and passwords was very similar to the ones encountered during the attacks on our honeypot system, but more extensive.

## Analysis summary

What do we make of this data captured by our honeypot? SSH is a way to access a computer over a network in a secure, encrypted way and has gained wide acceptance. However, despite its good reputation with respect to security, there are still threats associated with operating SSH on a machine. Password guessing clearly one of these threats, as we have shown in this paper. The mere fact that the SSH server is running and accessible from the Internet attracted 23 sequences of attacks from unique source IPs, with 6,899 login attempts to our honeypot in just 22 days. This equates to roughly one attack and approximately 300 login attempts per day, on average. Some attackers are very serious about performing attacks, executing hundreds of login attempts in a session.

The scanner capture revealed that very powerful tools are used. They are very flexible and can use customized account name/password lists for the attacks. If an attacker wanted to attack a particular domain, they potentially could harvest account names by social engineering and then combine these account names with standard passwords to be used in the attack. Observing the IRC Bot channel, we saw that attackers combined scanning tools with IRC Bot technology to perform scans via Zombies (compromised systems the attackers control via a remote channel). Our performance tests have revealed that the SSH password scanner can scan a class B network in 700 seconds.

Combined with an army of IRC bots, an attacker only needs 525 Zombies to scan the entire IP4 of today's public Internet in just one day. If you have a publicly accessible SSH server, you are very likely to be targeted by one of these attacks.

## Recommendations

There are a number of simple methods to protect against these attacks. The most obvious way is to turn off the daemon service, which on many systems is installed by default. If the

computer system runs as a desktop machine, there is likely no need for remote access via SSH to log into the machine. If this is not an option, there are numerous other options.

- Use the `/etc/hosts.allow` and `/etc/hosts.deny` files found on most Unix and Linux system to restrict daemon access to specific hosts.
- Install a firewall to restrict access to the SSH server from only designated machines and networks. This works particularly well if administration of a machine from an internal network necessitates remote access to that machine.
- Restrict the SSH server to only authenticate particular users or groups.
- Move the listening port of the SSH server from 22 to some other unused port. While this would not prevent attackers from connecting to the server and start guessing password, it will significantly reduce the likelihood of finding your SSH daemon, as attackers use standard SSH clients and attack tools that assume the SSH server is running on its standard port 22.
- Use an alternate authentication method besides simple passwords. More on this below. If this is not an option, ensure that a strong, complex password or passphrase is used.

SSH provides an alternate authentication method which successfully mitigates password guessing attacks. This authentication method is based on cryptographic keys, or so-called private key and public key. The public key is placed onto the server and acts as a custom lock for access to your account. This lock can only be opened with the corresponding private key. Once you provide this key, you gain access. Password guessing attacks would fail as attackers cannot guess or generate such a private key. All modern SSH servers are configured by default to support this authentication method. However, they usually fail back to password-based authentication in case the incorrect private key is provided, opening the door for password guessing attacks once again. The server needs to instead be configured to accept key-based authentication *only* for this mitigation strategy to be successful.

Setting up SSH with cryptographic keys is very simple and takes only a few minutes. Previous articles written by Brian Hatch have addressed [SSH User Identities](#) for secure access between an individual user and an SSH server. For more information on the host key generated by each server, the [SSH Host Key Protection](#) article may also be of use. Then, if SSH crypto keys are in use, the reader may want to further example [SSH and ssh-agent](#) to make it easier and faster to login via SSH.

In some instances, password-based authentication or access to an SSH server cannot be disabled. In those cases, other measures need to be taken. We have seen that attackers guess accounts and have good knowledge of existing system accounts and accounts one can commonly find on computer systems. If the attacker is able to guess an account name that exists on the system – on our honeypot this was achieved for 96.30% of the default account names of the RedHat honeypot system – the attacker already has one foot in the door. As such, we recommend not making use of easily guessable account names, such as common first names. Don't use 'Peter', 'Ian', or 'Mark' but rather create account names that contain a combination of first and last name, like 'seifer\_chr'. This can usually be achieved by the administrator who controls the assignment of account names.

In addition, we have seen that the 'root' account is the most often used account name for attacks, as it commonly exists on computer systems. We recommend that remote access to that account simply be disabled. Rather, an administrator should 'su' (superuser) gain access

to this account first via a regular user account.

Attackers commonly attempted to guess accounts that exist on most systems by default, like ftp and mysql. Access to the shell can only be obtained for those accounts if a shell is associated with the account. For those accounts like ftp or mysql that simply exist to run a service on the machine, no shell is necessary and should be disabled, effectively barring remote access with these accounts via SSH.

In addition to having account names that cannot be guessed, it is important that the users' passwords are strong. We have seen that the passwords used in the attacks often match account names or account names with number sequences. We assume that attackers select these passwords as they are most "successful" in malicious login attempts. This implies that at least some users set their passwords to these easily guessable strings. The only way an administrator of a system can prevent users from choosing such passwords is by installing various tools, like [passwd+](#), that force users to choose strong passwords.

Attackers are using tools to perform password guessing and login attempts, such as the captured Scanner, [QT](#), and [55hb](#). However, despite these tools, the minimum average time of login attempts was around two seconds due to an artificial delay on unsuccessful login attempts that was incorporated into the SSH server, as well as various network delays. While this provides protection against brute force attacks, only a few attempts and guesses are necessary on weak account names and passwords for them to be compromised. Security measures described above should be installed in order to practice security in breadth and depth.

## Future work

Our analysis was based on data captured by our honeypot. We are not able to determine how successful these attacks are against systems that can be found on the web. We would have to compare the account name/password combinations used in the attacks to account name/password combinations that exist on real systems in order to determine success rate. Further, we proposed moving the listening port of the SSH server to some other unused port. We need to set up a system with such a configuration to assess its effectiveness.

## About the author

[Christian Seifert](#) is a member of the New Zealand HoneyNet Alliance.

## Acknowledgement

Thanks to Jamie Riden for providing additional mitigation strategies on password guessing and references to actual SSH password guessing tools.

## Reprints or translations

Reprint or translation requests require [prior approval](#) from SecurityFocus.

© 2006 SecurityFocus

[Privacy Statement](#)

Copyright 2006, SecurityFocus