An In-depth Analysis of Linux/Ebury

welivesecurity.com/2014/02/21/an-in-depth-analysis-of-linuxebury/

February 21, 2014

In this blog post, we provide an in-depth analysis of Linux/Ebury - the most sophisticated Linux backdoor ever seen by our researchers. It is built to steal OpenSSH credentials and maintain access to a compromised server.

21 Feb 2014 - 03:35PM

In this blog post, we provide an in-depth analysis of Linux/Ebury – the most sophisticated Linux backdoor ever seen by our researchers. It is built to steal OpenSSH credentials and maintain access to a compromised server.

ESET has been analyzing and tracking an OpenSSH backdoor and credential stealer named Linux/Ebury. The result of this work on the Linux/Ebury malware family is part of a joint research effort with CERT-Bund, the Swedish National Infrastructure for Computing, the European Organization for Nuclear Research (CERN) and other organizations forming an international Working Group.

In this blog post, we provide an in-depth analysis of Linux/Ebury. It is a sophisticated backdoor used to steal OpenSSH credentials and maintain access to a compromised server. According to previous reports, this backdoor has been in the wild for at least two years. Linux/Ebury comes in two different shapes: a malicious library and a patch to the main OpenSSH binaries. The malicious library is a modified version of `libkeyutils.so`. This shared library is loaded by all OpenSSH executables files such as `ssh`, `sshd` and `ssh-agent`. We will describe how the backdoor works and how the `OpenSSH` functionalities are hooked. We will also show how passwords are captured and exfiltrated. Finally, we will provide detailed information on how system administrators can identify infected systems.

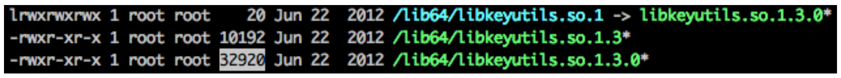
If a system is found to be infected with Linux/Ebury, we strongly recommend re-installing the operating system. It is also very important to consider all credentials used to log into this machine as compromised. All passwords and private OpenSSH keys should be changed

Known variants

Linux/Ebury is noteworthy for multiple reasons. Although this is something common under the Windows operating system, it is the first time we've seen a malicious library being used on POSIX systems. Linux/Ebury also uses innovative tricks to hook functions, discover the address space of the ELF executable that loaded the library and apply patches to its code at runtime. We believe that before using the external library to hook into `OpenSSH` processes, the author of Linux/Ebury used a patch to modify the source code of OpenSSH, thereby adding "new functionalities" to the software. The first variants found were modified binaries left on the disk. The three affected files are `ssh`, `sshd` and `ssh-add`. We have also seen usage of the rpm commands to remove signature from the original `OpenSSH` packages (`openssh-server, openssh-clients`) and modify the RPM database to update the file hashes with those of the malicious files. This will make the output of `rpm --verify openssh-servers` report the files as unmodified. However, the output from `rpm -qi openssh-servers` will clearly show the package is missing its signatures.

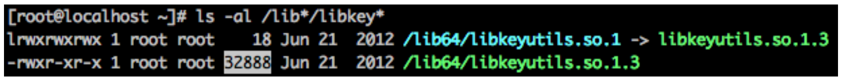
Later variants of Linux/Ebury do not modify the `OpenSSH` files directly. Instead, a shared library that is loaded by all OpenSSH executable files is modified to change the behaviour of the programs. The changes are the same as the patched OpenSSH variants, except that the some functions are hooked and patches to the original code are applied at run time. The shared library that is modified on the system is `libkeyutils.so`. Usually, this file is about 10KB in size. The malware adds approximately 20KB of additional malicious code, making the file weigh in at about 30KB in total.

Here are two examples of how the backdoor is deployed. The first one shows a Linux/Ebury-infected file next to the clean `libkeyutils.so`. The symbolic link is modified to point the rogue version.



```
lrwxrwxrwx 1 root root    20 Jun 22  2012 /lib64/libkeyutils.so.1 -> libkeyutils.so.1.3.0*
-rwxr-xr-x 1 root root 10192 Jun 22  2012 /lib64/libkeyutils.so.1.3*
-rwxr-xr-x 1 root root 32920 Jun 22  2012 /lib64/libkeyutils.so.1.3.0*
```

The second screenshot shows the `libkeyutils.so` is replaced with Linux/Ebury's version.

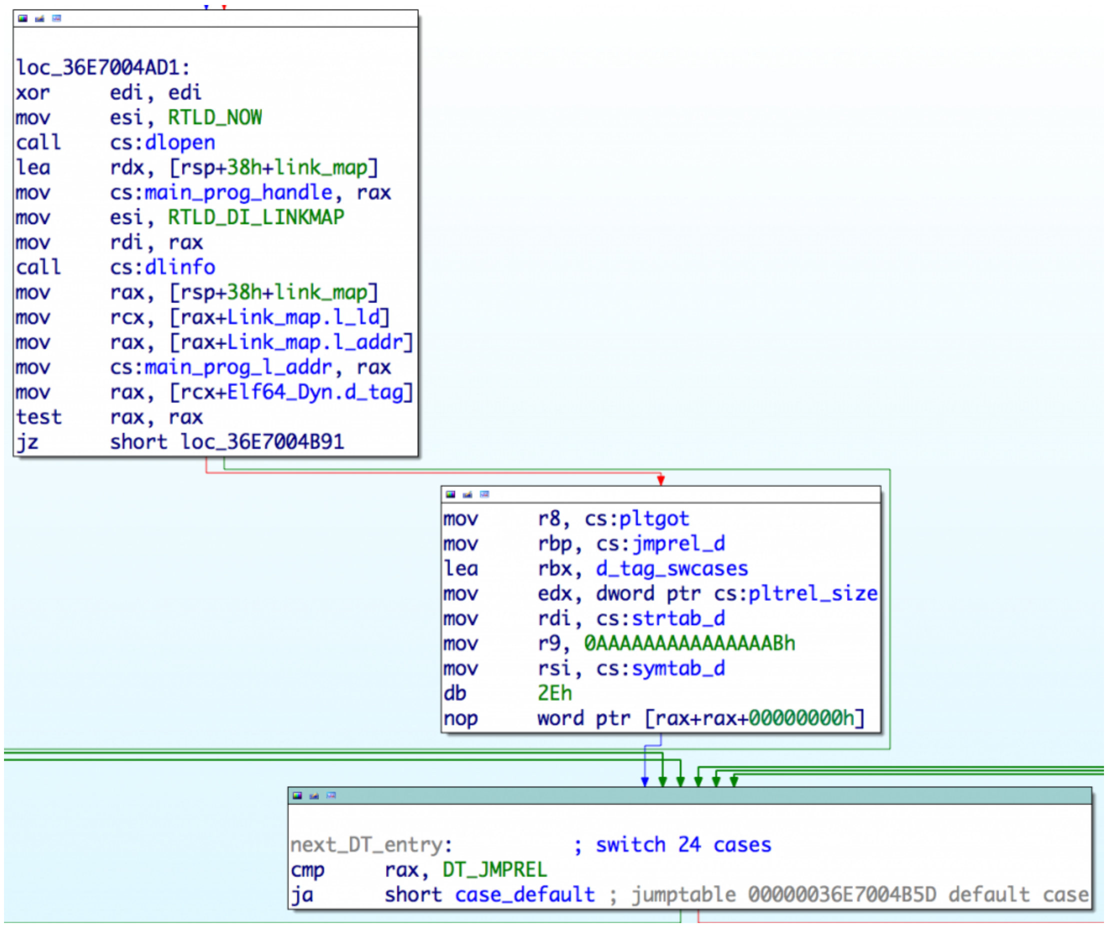


```
[root@localhost ~]# ls -al /lib*/libkey*
lrwxrwxrwx 1 root root    18 Jun 21  2012 /lib64/libkeyutils.so.1 -> libkeyutils.so.1.3
-rwxr-xr-x 1 root root 32888 Jun 21  2012 /lib64/libkeyutils.so.1.3
```

Although placing malicious code inside libraries has already been seen before, this is the first time we have observed this technique being used on the Linux operating system to modify the behaviour of `OpenSSH`.

To enable the different features of the malware, a constructor function was added in the `libkeyutils.so`. This function is called when the library is loaded. This function detects which binary it was loaded from, applies patches to the original code, and hooks functions from the original import table.

In the most recent variants of Linux/Ebury, strings are obfuscated with a simple `XOR` encryption with a static key. After unpacking, the malware loads various functions it requires using multiple calls to the `dlsym` function. Linux/Ebury then discovers the original executable address space by calling `dlopen`(`NULL, RTLD_NOW`) and passing the returned handle to `dlinfo(handle, RTLD_DI_LINKMAP,` …). This will work even if ASLR is enabled on a system. This behaviour gives Linux/Ebury the ability to walk the import table of an ELF executable and replace the original imported function address in memory. The result is that when the programs`ssh`, `sshd` or `ssh-add` call one of the hooked function, it will be redirected to the malicious libkeyutils.so implementation that can replace the original behaviour. The following code snippet shows the calls to dlopen and dlinfo to find the main program address space and walk the ELF header information:



```
loc_36E7004AD1:
xor     edi, edi
mov     esi, RTLD_NOW
call    cs:dlopen
lea     rdx, [rsp+38h+link_map]
mov     cs:main_prog_handle, rax
mov     esi, RTLD_DI_LINKMAP
mov     rdi, rax
call    cs:dlinfo
mov     rax, [rsp+38h+link_map]
mov     rcx, [rax+Link_map.l_ld]
mov     rax, [rax+Link_map.l_addr]
mov     cs:main_prog_l_addr, rax
mov     rax, [rcx+Elf64_Dyn.d_tag]
test    rax, rax
jz      short loc_36E7004B91
```

```
mov     r8, cs:pltgot
mov     rbp, cs:jmprel_d
lea     rbx, d_tag_swcases
mov     edx, dword ptr cs:pltrel_size
mov     rdi, cs:strtab_d
mov     r9, 0AAAAAAAAAAAAAAABh
mov     rsi, cs:symtab_d
db      2Eh
nop     word ptr [rax+rax+00000000h]
```

```
next_DT_entry:              ; switch 24 cases
cmp     rax, DT_JMPREL
ja      short case_default ; jumptable 00000036E7004B5D default case
```

The logging functions are hooked so that whenever the backdoor is used, nothing gets sent to the logging facility, leaving no trace of the backdoor in the log files on disk. If the backdoor is not in use, logging will behave normally and function calls will get redirected to the original function implementation.

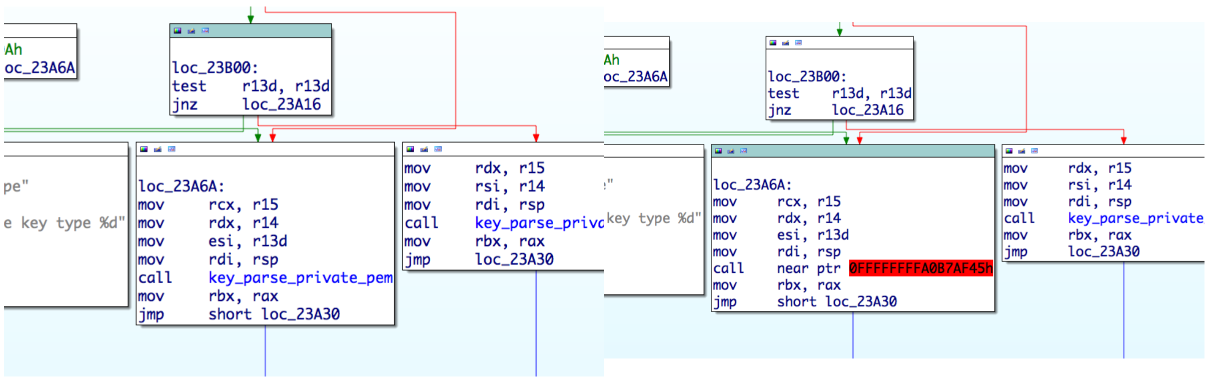
The following functions are hooked when the malicious `libkeyutils.so` is loaded inside the `sshd` process.

- `audit_log_user_message`
- `audit_log_acct_message`
- `hosts_access`
- `connect`
- `__syslog_chk`
- `write`
- `syslog`
- `popen`
- `hosts_access`
- `crypt`
- `pam_start`

The other functions such as `pam_start` and `crypt` are used to get the password used by a user to authenticate. Interestingly, the `pam_start` hook will place an additional hook for `pam_authenticate` to grab the password. The function connect is hooked so that when the `Xbnd` (documented below) command is used, a call to bind is made with the socket before the real call to connect is made.

Runtime code modification

When hooking from the import table is not an option, Linux/Ebury will modify the code segment by patching specific sections of the program to redirect call instruction to its own implementation. The following figure shows an example where the ssh program calls its own function `key_parse_private_pem` and the execution flow is redirected to malicious code. The address is shown in red because it is in the `libkeyutils`.so address space, outside `ssh`'s. The hook will call the original implementation and log the private key in memory that will later be fetched by the operators.



```
Ah
oc_23A6A

loc_23B00:
test    r13d, r13d
jnz     loc_23A16

pe"
e key type %d"

loc_23A6A:
mov     rcx, r15
mov     rdx, r14
mov     esi, r13d
mov     rdi, rsp
call    key_parse_private_pem
mov     rbx, rax
jmp     short loc_23A30

mov     rdx, r15
mov     rsi, r14
mov     rdi, rsp
call    key_parse_privc
mov     rbx, rax
jmp     loc_23A30
```

```
Ah
oc_23A6A

loc_23B00:
test    r13d, r13d
jnz     loc_23A16

"
key type %d"

loc_23A6A:
mov     rcx, r15
mov     rdx, r14
mov     esi, r13d
mov     rdi, rsp
call    near ptr 0FFFFFFFFA0B7AF45h
mov     rbx, rax
jmp     short loc_23A30

mov     rdx, r15
mov     rsi, r14
mov     rdi, rsp
call    key_parse_private
mov     rbx, rax
jmp     loc_23A30
```

Before attempting to modify the code segment, the program carefully sets a handler to intercept any segmentation fault it may cause. More specifically, the handler will be called if the process receives a SIGSEGV or SIGBUS signal. In the case such signal is caught, Linux/Ebury will simply abort its task and let the `OpenSSH` do its legitimate behavior. The way to recover from a segmentation fault is interesting. Before doing something that could potentially crash the process, sigsetjmp is called to create a snapshot of the current state. Then, if an access violation ever happens, siglongjmp is used in the signal hander to restore to the previous state.

This code patching technique is limited because offsets of code to patch is hardcoded inside the libkeyutils.so trojan. Thus, its effectiveness is limited to the binary targeted by the variant. Typically each `libkeyutils.so` variant will work for 3 to 5 different OpenSSH builds from a specific Linux distribution.

Functionalities

The backdoor is activated by sending specially-crafted data inside of the SSH client protocol version identification string. Here is what the SSH specification has to say about protocol version identification.

```
After the socket is opened, the server sends an identification
string, which is of the form "SSH-<protocolmajor>.<protocolminor>-
<version>\n", where <protocolmajor> and <protocolminor> are
integers and specify the protocol version number (not software
distribution version).  <version> is server side software version
string (max 40 characters); it is not interpreted by the remote
side but may be useful for debugging.
```

http://www.openssh.com/txt/ssh-rfc-v1.5.txt — T. Ylonen

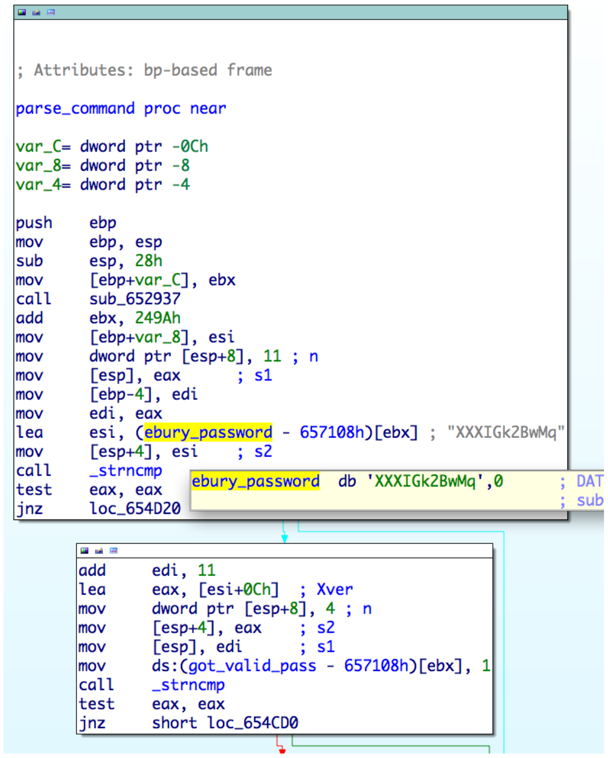
The `<version>` portion could be anything and should not be interpreted by the SSH server. In the case of a Linux/Ebury backdoor connection, the `<version>` contains a hexadecimal string of twenty-two (22) characters or more. It embeds an eleven (11) character password that is first encrypted with the client IP address and then encoded as a hexadecimal string; optionally a four (4) byte command may be encrypted and encoded as well after the password.

Note that the protocol version identification is also sent before the encryption handshake is done, making it possible to flag potential intrusion detection from a network trace.

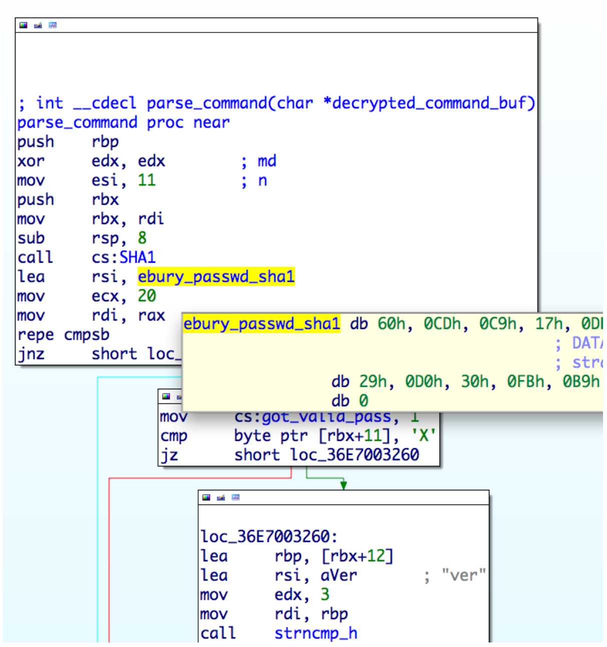
An example `SSH` protocol version to start a root shell looks like this:

```
SSH-2.0-fb54c28ba102cd73c1fe43
```

Once the backdoor password is verified, the `sshd` process will allow any password to work during a password authentication.
If `PermitRootLogin`, `PasswordAuthentication` or `PermitEmptyPassword` is disabled, it will enable them to make sure it works. It will also disable all logging of the successful session creation. It will be like nothing happened.In version 1.3.1 and newer, a SHA-1 sum of the password is kept in the binary instead of an eleven (11) character string (basic security practice by the Linux/Ebury's authors). This makes guessing of the password impractical unless you have a packet capture of the operators logging in successfully. Furthermore, the password is different from one variant to another. This leads us to think the operators have a database of infected servers with corresponding backdoor passwords so they can activate each backdoor successfully.



```
; Attributes: bp-based frame

parse_command proc near

var_C= dword ptr -0Ch
var_8= dword ptr -8
var_4= dword ptr -4

push    ebp
mov     ebp, esp
sub     esp, 28h
mov     [ebp+var_C], ebx
call    sub_652937
add     ebx, 249Ah
mov     [ebp+var_8], esi
mov     dword ptr [esp+8], 11 ; n
mov     [esp], eax      ; s1
mov     [ebp-4], edi
mov     edi, eax
lea     esi, (ebury_password - 657108h)[ebx] ; "XXXIGk2BwMq"
mov     [esp+4], esi    ; s2
call    _strncmp
test    eax, eax
jnz     loc_654D20

ebury_password  db 'XXXIGk2BwMq',0        ; DAT
                                          ; sub

add     edi, 11
lea     eax, [esi+0Ch]   ; Xver
mov     dword ptr [esp+8], 4 ; n
mov     [esp+4], eax     ; s2
mov     [esp], edi       ; s1
mov     ds:(got_valid_pass - 657108h)[ebx], 1
call    _strncmp
test    eax, eax
jnz     short loc_654CD0
```



```
; int __cdecl parse_command(char *decrypted_command_buf)
parse_command proc near
push    rbp
xor     edx, edx        ; md
mov     esi, 11         ; n
push    rbx
mov     rbx, rdi
sub     rsp, 8
call    cs:SHA1
lea     rsi, ebury_passwd_sha1
mov     ecx, 20
mov     rdi, rax
repe cmpsb
jnz     short loc_

ebury_passwd_sha1 db 60h, 0CDh, 0C9h, 17h, 0DI
                                          ; DAT/
                                          ; stri
                  db 29h, 0D0h, 30h, 0FBh, 0B9h
                  db 0

mov     cs:got_valid_pass, 1
cmp     byte ptr [rbx+11], 'X'
jz      short loc_36E7003260

loc_36E7003260:
lea     rbp, [rbx+12]
lea     rsi, aVer       ; "ver"
mov     edx, 3
mov     rdi, rbp
call    strncmp_h
```

The main purpose of the Linux/Ebury module is to steal credentials. The stolen credentials are most likely used to infect more servers. There is no code in Linux/Ebury to propagate itself; the backdoor is probably spread manually by an attacker or installed through deployment scripts.

Credential Stealing

Credentials are intercepted at multiple locations, when they are typed or used by the victim:

- **Password from successful login to the infected server**: Whenever someone logs in a system infected with Linux/Ebury, the `sshd` daemon will save the password and send it to the exfiltration server.
- **Any password login attempt to the infected server**: Even if the attempt is unsuccessful, the username and password used will be sent to the operators. They will be formatted differently, however, allowing the operators to differentiate these invalid credentials from the valid ones.
- **Password on successful login from the infected server**: When someone uses the ssh client on an infected server, Linux/Ebury will intercept the password and sent it to its exfiltration server.
- **Private key passphrase**: When the ssh client on an infected server prompts the user for an private key passphrase, the passphrase will be sent to the remote exfiltration server.

- **Unencrypted private key**: When a private key is used to authenticate to a remote server, the unencrypted version is intercepted by the malware. Unlike passwords, it will not send the key to the exfiltration server. Instead, it will store it memory and wait for the operators to fetch the key with the Xcat command.
- **Private keys added to the OpenSSH agent with ssh-add**: The keys added to an OpenSSH agent are also intercepted by the malware. Both the unencrypted key itself and the passphrase typed by the user will be logged.

Whatever the credential type, Linux/Ebury will add all relevant information for the operators to be able to use it, like the username, the target IP address and its OpenSSH listening port.

Password exfiltration

When a password is intercepted by Linux/Ebury, the information is sent to a remote server via a crafted DNS request sent to a specific IP address. The malware creates a regular A record request that will be sent on UDP port 53. The domain name requested contains encrypted and hexadecimal-encoded data as well as an IP address. The data has the following format:

```
<hexadecimal-encoded data>.<IP address>
```

The `<hexadecimal-encoded data>` contains the credentials previously described. It is `XOR` encrypted with the 4-byte static key 0x000d5345 before being hexadecimal-encoded.

The IP address included in the query depends on the type of stolen credentials. If the credentials are for the infected server itself, the client IP address is used. Otherwise, the remote IP address where the infected server is connecting to is used.

We believe the malware authors chose to send packets that look like legitimate DNS requests over UDP port 53 to avoid being blocked by firewalls. It is very common to whitelist DNS requests in firewall configurations because blocking them could disrupt name resolution.

There are 2 ways by which Linux/Ebury can choose a server where the DNS packets are sent. First, it can be set explicitly by the operator when sending an Xver command. The second method uses an algorithm to generate a domain name dynamically. This domain name will be queried for its A and TXT records. The TXT record will be used to verify that it is under the control of the operators using public key cryptography. Details about the domain generation algorithm and the verification processed will be published later.

Extra commands

Three commands are available to the malicious actors to ease the management of the compromised server. A command is appended to the backdoor password before being encrypted. When the backdoor identifies a command, it interprets it instead of starting a shell. Here is a list of commands that can be processed by the backdoor:

- `Xcat`: print all the passwords, passphrases, and keys saved in the shared memory and exit.
- `Xver`: print the installed Linux/Ebury version and exit. Xver also accept an optional four (4) byte argument. If present, it will set the exfiltration server IP address to the given one.
- `Xbnd`: `Xbnd` takes a four (4) byte argument. When creating a tunnel to a remote host, bind the client socket to the specified interface IP address.

Version tracking

The authors of Linux/Ebury have a good practice of leaving version number inside their binaries. This allows operators to know what version is installed on each system. It also helped researchers understand the chronology of events and sort samples more easily.

For example, since version 1.3.2, Linux/Ebury will not send any information to a remote server if an interface is in promiscuous mode. An interface is set to promiscuous mode when software like tcpdump is capturing the traffic on a network interface. The authors probably added this feature in reaction to an article from cPanel about Linux/Ebury that suggested running tcpdump to monitor DNS requests and notice exfiltration data as an indicator of compromise.

Indicators of Compromise (IOCs)

We will provide two means of identifying the presence of the Linux/Ebury SSH backdoor. The easiest way to identify an infected server relies on the presence of a feature added by the malware to the ssh binary. A longer process involves inspection of the shared memory segments used by the malware.

The command `ssh -G` has a different behaviour on a system with Linux/Ebury. A clean server will print

```
ssh: illegal option -- G
```

to `stderr` but an infected server will only print the typical “usage” message. One can use the following command to determine if the server he is on is compromised:

```
$ ssh -G 2>&1 | grep -e illegal -e unknown > /dev/null && echo
"System clean" || echo "System infected"
```

Linux/Ebury relies on POSIX shared memory segments (SHMs) for interprocess communications. The current version uses large segments of over 3 megabytes of memory with broad permissions allowing everyone to read and write to this segment.

Other processes could legitimately create shared memory segments with broad permissions. Make sure to validate that `sshd` is the process that created the segment like we show below.

One can identify large shared memory segment with broad permissions by running the following as root:

```
# ipcs -m
------ Shared Memory Segments --------
key        shmid      owner     perms      bytes     nattch
0x00000000 0          root      644        80        2
0x00000000 32769      root      644        16384     2
0x00000000 65538      root      644        280       2
0x000010e0 465272836  root      666        3282312   0
```

Then to look for the process that created the shared memory segment, use:

```
# ipcs -m -p
------ Shared Memory Creator/Last-op PIDs --------
shmid      owner      cpid       lpid
0          root       4162       4183
32769      root       4162       4183
65538      root       4162       4183
465272836  root       15029      17377
```

If the process matches sshd:

```
# ps aux | grep &lt;pid&gt;
root     11531  0.0  0.0 103284   828 pts/0    S+   16:40   0:00 grep 15029
root     15029  0.0  0.0  66300  1204 ?        Ss   Jan26   0:00 /usr/sbin/sshd
```

An `sshd` process using shared memory segments larger than three (3) megabytes (3,145,728 bytes) and with broad permissions (666) is a strong indicator of compromise.

Network-based indicators

We are providing simple snort rules in order to easily pinpoint malicious activity in large networks. The Internet being a wild place these have greater chances of triggering false positives. Use wisely.

This first rule matches against the SSH Client Protocol field that the backdoor uses to connect to a victim. Any external host trying to connect to the backdoor on properly identified SSH ports will trigger the alert.

```
alert tcp $EXTERNAL_NET any -&gt; $HOME_NET $SSH_PORTS (msg:"Linux/Ebury SSH backdoor activty"; content:"SSH-2.0"; isdataat:20,relative; pcre:"/^SSH-2\.0-[0-9a-f]{22,46}/sm"; reference:url,/2014/02/21/an-in-depth-analysis-of-linuxebury/; classtype:trojan-activity; sid:1000001; rev:1;)
```

The following Snort rule for detecting Linux/Ebury infected machines sending harvested credentials to a exfiltration server has been provided by CERT-Bund.

```
alert udp $HOME_NET any -&gt; $EXTERNAL_NET 53 (msg:"Linux/Ebury SSH backdoor data exfiltration"; content:"|12 0b 01 00 00 01|"; depth:6; pcre:"/^\x12\x0b\x01\x00\x00\x01[\x00]{6}.[a-f0-9]{6,}(([\x01|\x02|\x03]\d{1,3}){4}|\x03::1)\x00\x00\x01/Bs"; reference:url,/2014/02/21/an-in-depth-analysis-of-linuxebury/; reference:url,https://www.cert-bund.de/ebury-faq; classtype:trojan-activity; sid:1000002; rev:1;)
```

Conclusion

The Linux/Ebury malware clearly is a complex threat with many interesting features such as code hooking, advanced POSIX exception handling and various ways of hiding itself on an infected system. Based on data we captured, we estimate that thousands of systems are currently infected with Linux/Ebury. We feel it is important to specify that we have not witnessed a weakness in the OpenSSH software itself. The Linux/Ebury trojan propagates using the stolen credentials collected by the operators. With the administrator's credentials, there is no need for a "0-day" exploit, although how the operators bootstrapped the whole credential stealing operation is still a mystery.

File Details

Trojanized `libkeyutils.so` file SHA-1 hashes.

`09c8af3be4327c83d4a7124a678bbc81e12a1de4` – Linux/Ebury – Version 1.3.2
`2e571993e30742ee04500fbe4a40ee1b14fa64d7` – Linux/Ebury – Version 1.3.4
`39ec9e03edb25f1c316822605fe4df7a7b1ad94a` – Linux/Ebury – Version 1.3.2
`3c5ec2ab2c34ab57cba69bb2dee70c980f26b1bf` – Linux/Ebury – Version 1.3.2
`471ee431030332dd636b8af24a428556ee72df37` – Linux/Ebury – Version 1.2.1
`5d3ec6c11c6b5e241df1cc19aa16d50652d6fac0` – Linux/Ebury – Version 1.3.3
`74aa801c89d07fa5a9692f8b41cb8dd07e77e407` – Linux/Ebury – Version 1.3.2
`7adb38bf14e6bf0d5b24fa3f3c9abed78c061ad1` – Linux/Ebury – Version 1.3.2
`9bb6a2157c6a3df16c8d2ad107f957153cba4236` – Linux/Ebury – Version 1.3.2
`9e2af0910676ec2d92a1cad1ab89029bc036f599` – Linux/Ebury – Version 1.3.3
`adfcd3e591330b8d84ab2ab1f7814d36e7b7e89f` – Linux/Ebury – Version 1.3.2
`bf1466936e3bd882b47210c12bf06cb63f7624c0` – Linux/Ebury – Version 1.3.2
`d552cbadee27423772a37c59cb830703b757f35e` – Linux/Ebury – Version 1.3.3
`e14da493d70ea4dd43e772117a61f9dbcff2c41c` – Linux/Ebury – Version 1.3.2
`f1ada064941f77929c49c8d773cbad9c15eba322` – Linux/Ebury – Version 1.3.2

References

CERT-Bund: Ebury FAQ – https://www.cert-bund.de/ebury-faq

cPanel – http://docs.cpanel.net/twiki/bin/view/AllDocumentation/CompSystem

Dr. Web – http://news.drweb.com/show/?i=3600&lng=en&c=5

Dr.Web – http://news.drweb.com/?i=3332&lng=en

Gunderson, Steinar – http://plog.sesse.net/blog/tech/2011-11-15-21-44_ebury_a_new_ssh_trojan.html

SANS https://isc.sans.edu/diary/SSHD+rootkit+in+the+wild/15229

WebHosting Talk – http://www.webhostingtalk.com/showthread.php?t=1235797

21 Feb 2014 - 03:35PM