Blocking Visual Studio Code embedded reverse shell before it's too late

Sep 17, 2023 · 11 min read

· BLUETEAM DEFENSE WINDOWS SPLUNK SOC POWERSHELL

Overview

Visual studio code tunnel

Introduction

Since July 2023, Microsoft is offering the perfect reverse shell, embedded inside Visual Studio Code, a widely used development tool. With just a few clicks, any user with a github account can share their visual studio desktop on the web. VS code tunnel is almost considered a lolbin (Living Of the Land Binary).

I am so glad that my users now have the ability to expose their computer with highly sensitive data right on the web, through an authentication I nor control, nor supervise. My internal network is now accessible from anywhere !

The worse part is that this tunnel can be triggered from the cmdline with the portable version of code.exe. An attacker just has to upload the binary, which won't be detected by any anti-virus since it is legitimate and singed windows binary.

It is therefore something to watch out for.

Search...

ipfyx

Yet another cybersecurity blog

READ MORE

Featured Posts

- Blocking Visual Studio Code embedded reverse shell before it's too late

Recent Posts

- D LINK DNS323 Fan Speed Control
- Bluekeep, why would you still be vulnerable ? SHA2 signing

Categories

DEFENSE 2
ADMIN 1
CYBERSECURITY 1

Tags

POWERSHELL 2
WINDOWS 2
BASH 1
BLUEKEEP 1
BLUETEAM 1
CVE 1
DEFENSE 1
LINUX 1
RDP 1
SOC 1
SPLUNK 1

```
.\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)
* the Microsoft Privacy Statement (https://privacy.microsoft.com/en-US/privacystatem
*

Open this link in your browser https://vscode.dev/tunnel/sauceTomate/C:/Users/ipfyx/I
```

The binary is definitely signed :

POWERSHELL

```
Get-AppLockerFileInformation 'C:\Users\ipfyx\AppData\Local\Programs\Microsoft VS Cod
PublisherName    : O=MICROSOFT CORPORATION, L=REDMOND, S=WASHINGTON, C=US
ProductName      :
BinaryName       :
BinaryVersion    : 0.0.0.0
HasPublisherName : True
HasProductName   : False
HasBinaryName    : False
```

Notice how there are nor ProductName nor BinaryName value, we will use them later.

pfiatde blog post already did a great job at describing what an attacker can do so I won't say much more. Go check his awesome blog post.

Mitigation

But now, as a defender, how do we block or detect its usage ?

Domains blacklist

Microsoft documentation states :

> 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.

Yeah it's not that simple. It will indeed block most users but, from my testing, it won't block an attacker that has already established a tunnel once. Meaning the tunnel can be kept active or restarted at will despite this domain being blacklisted.

From my understanding, VScode contacts ***global.rel.tunnels.api.visualstudio.com*** to get its "clusters" : https://global.rel.tunnels.api.visualstudio.com/api/v1/clusters

JSON
```json
[
  {
    "clusterId": "auc1",
    "uri": "https://auc1.rel.tunnels.api.visualstudio.com",
    "azureLocation": "AustraliaCentral"
  },
  {
    "clusterId": "aue",
    "uri": "https://aue.rel.tunnels.api.visualstudio.com",
    "azureLocation": "AustraliaEast"
...
```

Those domains are also mentioned in another Microsoft documentation :

```
- Dev Tunnels
  - global.rel.tunnels.api.visualstudio.com
  - [clusterId].rel.tunnels.api.visualstudio.com
  - [clusterId]-data.rel.tunnels.api.visualstudio.com
  - *.[clusterId].devtunnels.ms
  - *.devtunnels.ms
```

Blocking those domains will block and cut off any VS code tunnel :

```
*.tunnels.api.visualstudio.com
*.devtunnels.ms
```

Applocker

Applocker is Microsoft application whitelisting technology. When enabled and configured, with default rules for example, everything is blocked by default except for the executables and scripts defined in the preceding rules. Let's pretend we use Microsoft default rules for demo purpose.

To generate them, in *Group Policy Management Editor*, go to *Computer Configuration -> Windows Settings -> Security Settings -> Application Control Policies -> AppLocker*. Right Click on *Create Default Rules*.

Then right click on *AppLocker* and *Export Policy*.

Figure 1: MS Applocker default rules generation

Here is the resulting xml :

XML
```xml
<AppLockerPolicy Version="1">
```

```
      </Conditions>
    </FilePathRule>
...
```

When using those rules, everything that is in Program Files and Windows is allowed, everything else is blocked. Administrators are allowed to execute anything, VSCode for instance. But if you were to define a rule that blocks VSCode, this rule would apply first, since "Deny rules" are applied before "Allowed rules".

Let's build a rule to block VScode altogether. The rule applies to Everyone (SID S-1-1-0), therefore to Administrators (SID S-1-5-32-544).

```powershell
Get-AppLockerFileInformation 'C:\Users\ipfyx\AppData\Local\Programs\Microsoft VS Cod
```

In the resulting xml, I replaced Action="Allow" with Action="Deny". I also removed my current VSCode version from *BinaryVersionRange* to match any VScode version.

> Warning : Do not import the following xml in Applocker. Imported alone, this rule will completely block your system. Add a Allow * rule beforehand.

```xml
<AppLockerPolicy Version="1">
        <RuleCollection Type="Exe" EnforcementMode="NotConfigured">
                <FilePublisherRule Id="1dd70b30-eb06-4220-b808-bd8d368624c0" Name="V
                        <Conditions>
                                <FilePublisherCondition PublisherName="O=MICROSOFT C
                                        <BinaryVersionRange LowSection="*" HighSecti
                                </FilePublisherCondition>
                        </Conditions>
                </FilePublisherRule>
        </RuleCollection>
...
```

Included in MS default rules, here is the resulting xml (called vscode.xml from now on) :

```xml
<AppLockerPolicy Version="1">
  <RuleCollection Type="Appx" EnforcementMode="Enabled" />
  <RuleCollection Type="Dll" EnforcementMode="NotConfigured" />
  <RuleCollection Type="Exe" EnforcementMode="Enabled">
    <FilePathRule Id="921cc481-6e17-4653-8f75-050b80acca20" Name="(Default Rule) All
      <Conditions>
        <FilePathCondition Path="%PROGRAMFILES%\*" />
      </Conditions>
    </FilePathRule>
    <FilePathRule Id="a61c8b2c-a319-4cd0-9690-d2177cad7b51" Name="(Default Rule) All
...
```

POWERSHELL
```
Get-AppLockerFileInformation 'C:\Users\ipfyx\AppData\Local\Programs\Microsoft VS Cod
New-AppLockerPolicy : Les règles ne peuvent pas être créées. Les informations de fic
%OSDRIVE%\USERS\ipfyx\DOWNLOADS\VSCODE_CLI_WIN32_X64_CLI\CODE.EXE
```

To be sure, let's test the previously generated vscode rule.

POWERSHELL
```
# Portable code tunnel is blocked but not by the rule
Test-AppLockerPolicy -XmlPolicy .\vscode.xml -Path .\code.exe -User S-1-1-0|fl
FilePath       : C:\Users\ipfyx\Downloads\vscode_cli_win32_x64_cli\code.exe
PolicyDecision : DeniedByDefault
MatchingRule   :

# Code tunnel is blocked but not by the rule
Test-AppLockerPolicy -XmlPolicy .\vscode.xml -Path 'C:\Users\ipfyx\AppData\Local\Pro
FilePath       : C:\Users\ipfyx\AppData\Local\Programs\Microsoft VS Code\bin\code-tu
PolicyDecision : DeniedByDefault
...
```

The *PolicyDecision* is *DeniedByDefault*, which means the rule we just built from the vscode binary does not apply to code-tunnel. If no rules matches, the default behaviour for applocker is to deny any execution. Code-tunnel is denied there, but it would not be if it was placed in the allowed directories. It could be placed there by an attacker or by a legitimate vscode, installed by an admin.

A solution could be to use a Hash Condition.

POWERSHELL
```
# Export
Get-AppLockerFileInformation .\code.exe | New-AppLockerPolicy -RuleType Hash -User S

# Code tunnel is blocked
Test-AppLockerPolicy -XmlPolicy .\vscode-tunnel-hash.xml -Path 'C:\Users\ipfyx\AppDa
FilePath       : C:\Users\ipfyx\AppData\Local\Programs\Microsoft VS Code\bin\code-tu
PolicyDecision : Denied
MatchingRule   : code.exe
```

> Warning : Do not import the following xml in Applocker. Imported alone, this rule will completely block your system. Add a Allow * rule beforehand.

XML
```
<AppLockerPolicy Version="1">
        <RuleCollection Type="Exe" EnforcementMode="NotConfigured">
                <FileHashRule Id="15b4ef38-5f18-484b-bc83-e03d24076a0d" Name="code.e
                        <Conditions>
                                <FileHashCondition>
                                        <FileHash Type="SHA256" Data="0xAC60D7CA817E
                                </FileHashCondition>
```

But this solution is nor resilient nor sustainable since the hash can change at the first update.

GPO

Visual Studio has the so wanted features :

```
- Dev Tunnels - controls test functionality
```

But as of today (1.8.82), VScode doesn't. You can force automatic updates though ! UpdateMode_default would be my goto.

```powershell
gc 'C:\Users\ipfyx\AppData\Local\Programs\Microsoft VS Code\policies\en-us\VSCode.adı
```

```xml
<?xml version="1.0" encoding="utf-8"?>
<policyDefinitionResources revision="1.0" schemaVersion="1.0">
        <displayName />
        <description />
        <resources>
                <stringTable>
                        <string id="Application">Visual Studio Code</string>
                        <string id="Supported_1_67">Visual Studio Code &gt;= 1.67</s
                        <string id="Category_updateConfigurationTitle">Update</strin
                        <string id="UpdateMode">UpdateMode</string>
...
```

Detection

Process

Looking for code-tunnel execution

A simple detection could be :

```
index=win sourcetype="XmlWinEventLog" EventCode=4688 code tunnel cmdline="*code*.exe
| stats min(_time) as time_min max(_time) as time_max count as occurence values(NewPı
```

host	SubjectUser	cmdline
sauteTomate	ipfyx	"c:\Users\ipfyx\AppData\Local\Programs\Microsoft VS Code\bin\code-tunnel" tunnel --accept-server-license-terms --name totallyNotSuspicious

```
index=win sourcetype="XmlWinEventLog" EventCode=4688 tunnel accept server license te
| stats min(_time) as time_min max(_time) as time_max count as occurence values(NewP
```

host	SubjectUser	cmdline
sauteTomate	ipfyx	"c:\Users\ipfyx\AppData\Local\Programs\Microsoft VS Code\bin\c0de.exe" tunnel --accept-server-license-terms --name totallyNotSuspicious
sauteTomate	ipfyx	"c:\Users\ipfyx\AppData\Local\Programs\Microsoft VS Code\bin\code-tunnel" tunnel --accept-server-license-terms --name totallyNotSuspicious

If you use sysmon, you can switch to EventCode=1 and Sysmon sourcetype instead of EventCode 4688.

Looking for suspicious child process

Great idea from lobas :

```
IOC: Process tree: code.exe -> cmd.exe -> node.exe -> winpty-agent.exe
```

A straightforward search would be :

```
index=win sourcetype="XmlWinEventLog" EventCode=4688 code (cmd OR powershell) Parent
| table _time, host, NewProcess, ParentProcess
```

But once again, *code.exe* could be renamed

We could use the search above to look for suspicious child process from our process with *accept-server-license-terms* option :

```
index=win sourcetype="XmlWinEventLog" EventCode=4688 (cmd OR powershell) ParentProce
    [search index=win sourcetype="XmlWinEventLog" EventCode=4688 tunnel accept serve
    | table host NewProcessId
    | rename NewProcessId as ParentProcessId
    | format]
| table _time, host, NewProcess, ParentProcess
```

_time	host	NewProcess

09-18	sauteTomate	C:\Windows\System32\WindowsPowerShell\v1.0\powers

> EDIT 20230924 : license terms agreement is not mandatory. And even if it were, this could be bypass by creating the file `license_content.json` with the right content (see below). All we have left is detecting the string "*.exe*tunnel*" in a commandline, which is subject to false positive.

Applocker

> EDIT 20230924

Since we defined some Applocker rules, we got some log from it ! We could just search for that weird publisher value :

```
index=win sourcetype="XmlWinEventLog" EventCode=8004 event_provider="Applocker" "O=M:
| table _time, host, FilePath, PolicyName, RuleName Fqbn
```

_time	host	FilePath
2023-09-18 11:35:54	sauteTomate	%OSDRIVE%\USERS\IPFYX\APPDATA\LOCAL\PROG VS CODE\BIN\CODE.EXE

Got you ! There should not be that many binary signed without ProductName right ? Let's count them quickly in System32 for example :

```powershell
gci -Recurse C:\Windows\System32 -include *.exe |Get-AppLockerFileInformation -Error

Count    : 20
```

Sigh. 20 binary names to masquerade as...

File creation

Another great idea from lobas :

```
IOC: File write of code_tunnel.json which is parametizable, but defaults to: %UserProfil
e%\.vscode-cli\code_tunnel.json
```

```
PS > gc C:\Users\ipfyx\.vscode\cli\license_consent.json
{"consented":true}
```

I do not have sysmon EventCode 11 yet, so I will leave the SPL search as an exercise for the user. Watching for those files creation inside the UserProfile directory is not enough since it can be changed with the *--cli-data-dir* option.

```powershell
.\code.exe tunnel help
...
GLOBAL OPTIONS:
      --cli-data-dir <CLI_DATA_DIR>  Directory where CLI metadata should be stored [
```

Web traffic monitoring

If you haven't blocked the domains I mentioned earlier, watch out for any HTTP traffic toward those domains.

Conclusion

A GPO parameter would be awesome but it is yet to be seen. An Applocker hash rule is not sustainable. Moreover, your non-domain-joined computers would not be concerned by neither of them. Therefore, for now, we are only left to blocking those two bad boys :

```
*.tunnels.api.visualstudio.com
*.devtunnels.ms
```