15/03/2024

Security Audit

VulnerableLightApp

Par
NICOLAS , TONY , BENJAMIN , SIMON , ALEXIS

Document tracking sheet

Modification history

Version	Date	Editor	Modification
1.0	15/03/2024	Nicolas	Document creation
1.0	15/03/2024	Tony Benjamin Simon Alexis	Co-writing Proofreading

Summary

01 Present our approach

01.1 Requirement

The aim of this audit is to assess the risks of a VulnerableLightApp web application, using the approach presented below. For each risk identified, recommendations will be made to limit the defect and achieve an acceptable level of security.

01.2 Present our approach

The penetration test methodology is divided into 3 phases:

- The VulnerableLightApp web application was analyzed and placed in an isolated network.

- A network reconnaissance followed by an analysis of available services was carried out.

- Vulnerability scans complemented the previous analysis.

01.3 Goals

The aim of vulnerability testing is to assess the security status of an information system or application at a given moment. The aim is to highlight the security flaws that could actually be exploited by a malicious individual within a deliberately limited timeframe.

01.4 Operating Mode

The tests were carried out on a web application from a test machine.

The following tools were used to perform the tests:

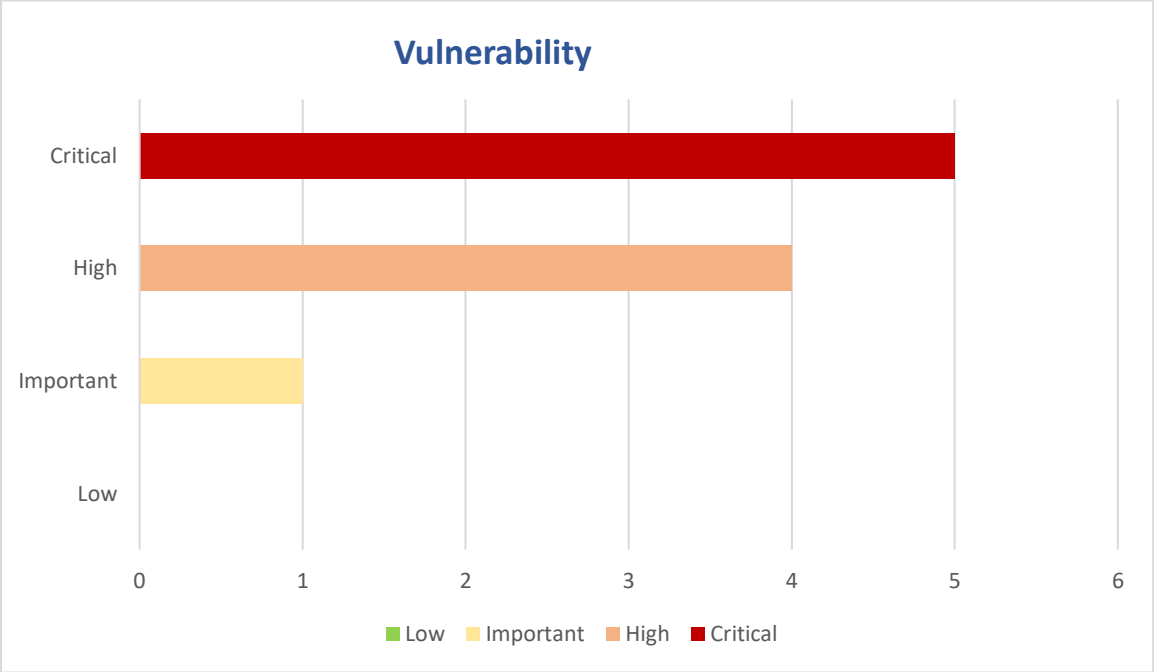
- ZAP
- Swagger

The machine is hard-coded and has no SSH connection.

01.5 Completeness of results

The results were obtained over a limited, predefined test period. This approach is based on a test period and is not intended to be exhaustive. Only functionalities accessible during the audit were analyzed.

02 Technical synthesis



Tests carried out on the Web application revealed **a generally low level of security**. Many security precautions are not implemented or applied.

Vulnerable Light App, presents several critical security vulnerabilities, such as **SQL injections, XML injections and code injections.** No measures are currently taken against these vulnerabilities, which exposes the application.

Cross-Site Scripting was also a vulnerability found during this security audit.

Vulnerabilities such as the ability to **generate incorrect tokens** or **local file inclusion** were also found.

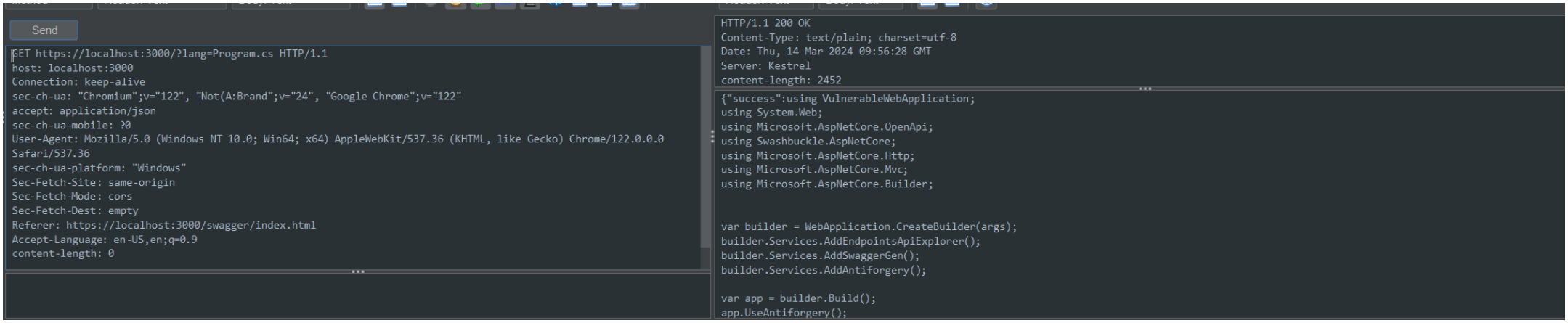
The **detailed list** and proof of these **vulnerabilities** are available in the rest of the document.

02.1 Listing of vulnerability

ID	DESCRIPTION	RISQUE AVERE	PERIMETRE
CWE-22	Path Traversal	Oui	VulnerableLightAPP
CWE-94	Code Injection	Oui	VulnerableLightAPP
CWE-89	SQL Injection	Oui	VulnerableLightAPP
CWE-91	XML Injection	Oui	VulnerableLightAPP
CWE-284	Improper Access Control	Oui	VulnerableLightAPP
CWE-829	Local File Inclusion	Oui	VulnerableLightAPP
CWE-121	Stack-Based buffer overflow	Oui	VulnerableLightAPP
CWE-1270	Generation of incorrect security Tokens	Oui	VulnerableLightAPP
CWE-918	Server-Side Request Forgery (SSF)	Oui	VulnerableLightApp
CW-79	Cross-site Scripting	Oui	VulnerableLightApp

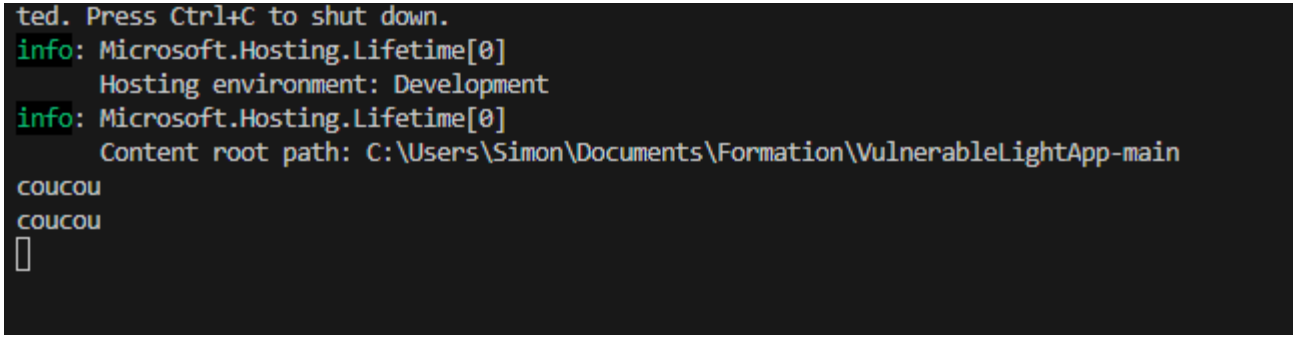
Vulnerability

CWE-22 Path Traversal



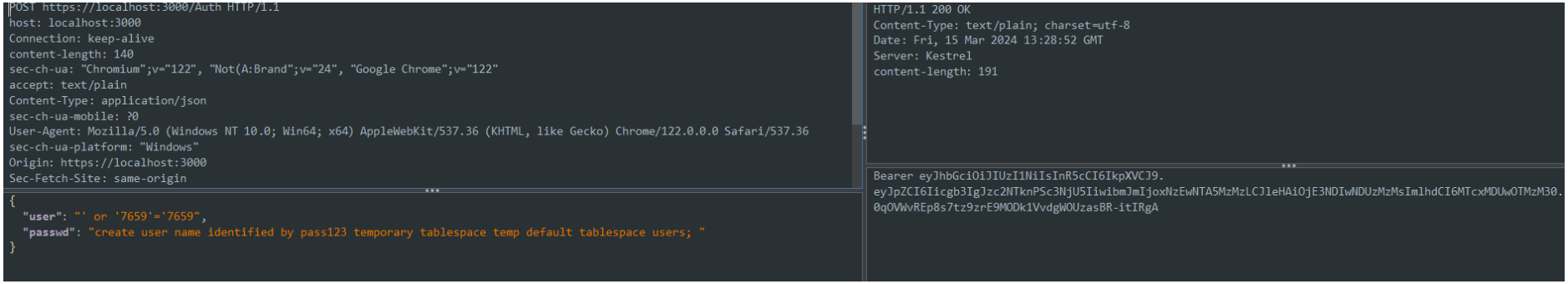
CVSS 5,6 Important	**CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')**		
	The product uses external input to construct a pathname that is intended to identify a file or directory that is located underneath a restricted parent directory, but the product does not properly neutralize special elements within the pathname that can cause the pathname to resolve to a location that is outside of the restricted directory. In many programming languages, the injection of a null byte (the 0 or NUL) may allow an attacker to truncate a generated filename to widen the scope of attack. For example, the product may add ".txt" to any pathname, thus limiting the attacker to text files, but a null injection may effectively remove this restriction		
	Impact	Difficult to exploit	Risk
	High	High	Yes

CWE-94 Code Injection



CVSS 7,8 critical	**CWE-94: Improper Control of Generation of Code ('Code Injection')** The product constructs all or part of a code segment using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the syntax or behavior of the intended code segment. Injection problems encompass a wide variety of issues -- all mitigated in very different ways. For this reason, the most effective way to discuss these weaknesses is to note the distinct features which classify them as injection weaknesses. The most important issue to note is that all injection problems share one thing in common -- i.e., they allow for the injection of control plane data into the user-controlled data plane. This means that the execution of the process may be altered by sending code in through legitimate data channels, using no other mechanism. While buffer overflows, and many other flaws, involve the use of some further issue to gain execution, injection problems need only for the data to be parsed. The most classic instantiations of this category of weakness are SQL injection and format string vulnerabilities		
	Impact	Difficult to exploit	Risk
	High	LOW	Yes

CWE-89 SQL Injection



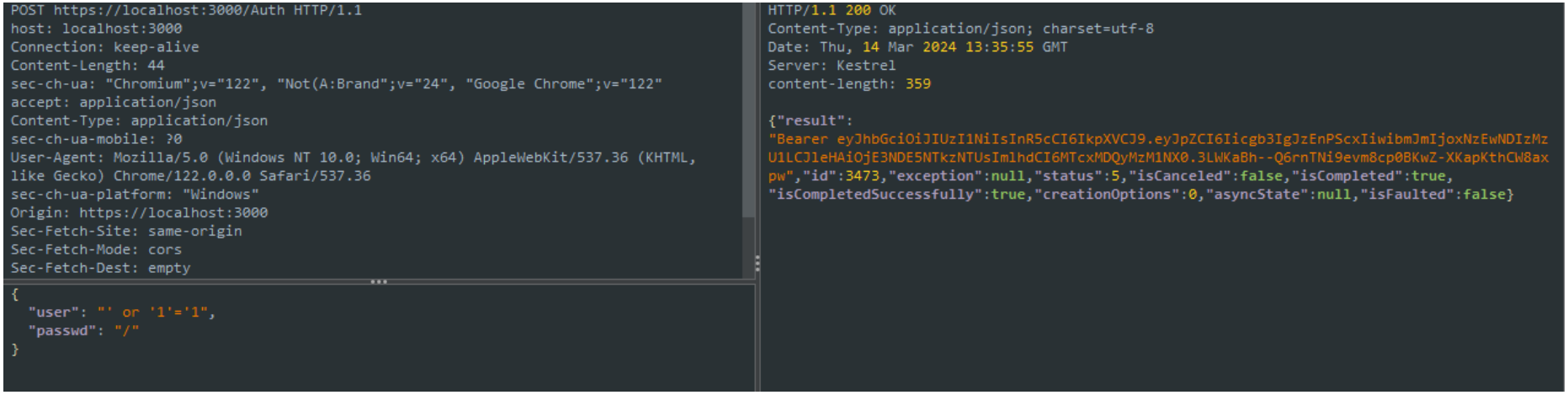
CVSS 8,2 critical	**CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')** The product constructs all or part of a code segment using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the syntax or behavior of the intended code segment. When a product allows a user's input to contain code syntax, it might be possible for an attacker to craft the code in such a way that it will alter the intended control flow of the product. Such an alteration could lead to arbitrary code execution. Injection problems encompass a wide variety of issues -- all mitigated in very different ways. For this reason, the most effective way to discuss these weaknesses is to note the distinct features which classify them as injection weaknesses. The most important issue to note is that all injection problems share one thing in common -- i.e., they allow for the injection of control plane data into the user-controlled data plane. This means that the execution of the process may be altered by sending code in through legitimate data channels, using no other mechanism. While buffer overflows, and many other flaws, involve the use of some further issue to gain execution, injection problems need only for the data to be parsed. The most classic instantiations of this category of weakness are SQL injection and format string vulnerabilities.

Impact	Difficult to exploit	Risk
High	High	Yes

CWE-91 XML Injection

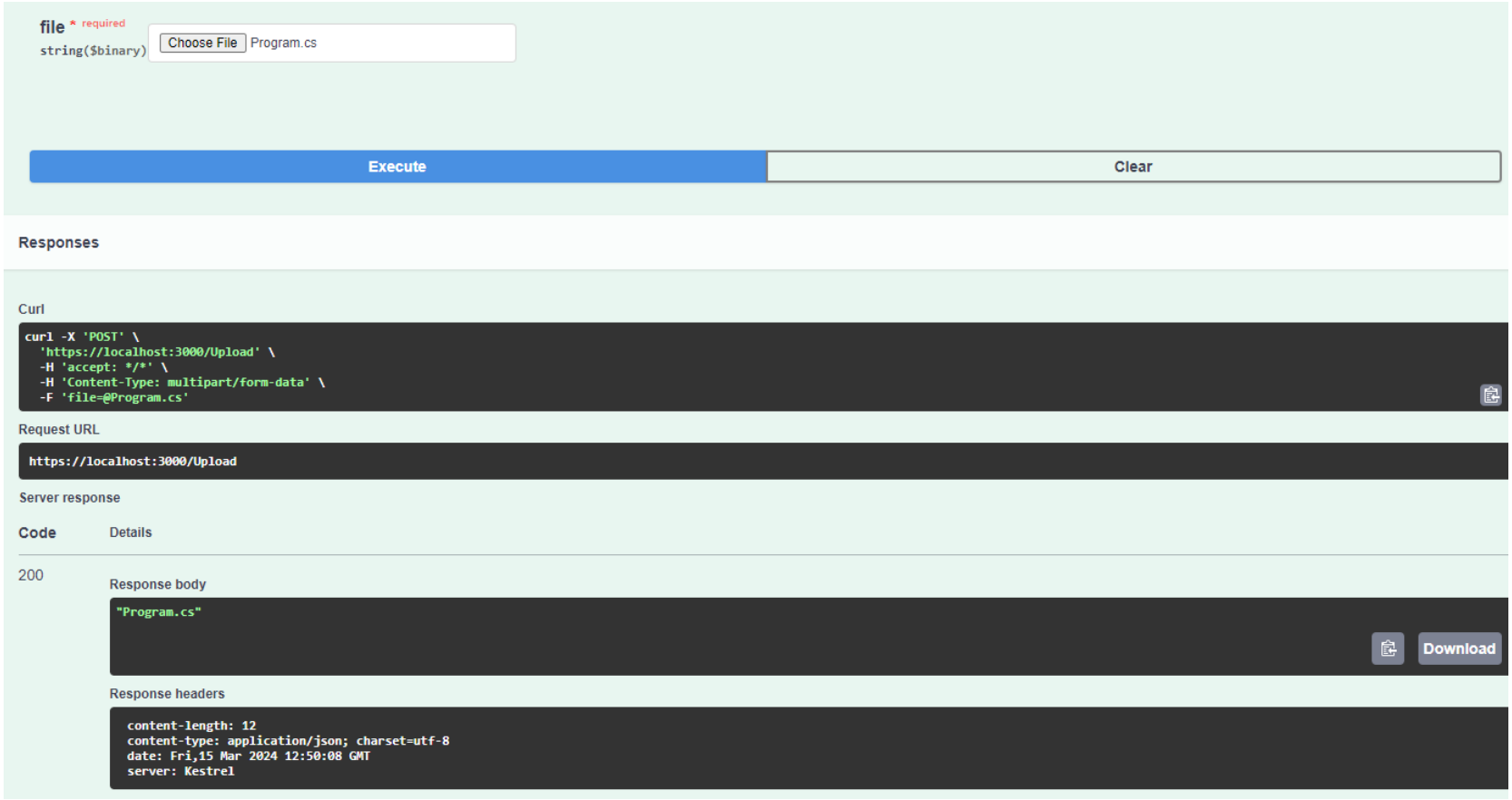
CVSS 7,1 Critical	**CWE-91: XML Injection (aka Blind XPath Injection)** The product does not properly neutralize special elements that are used in XML, allowing attackers to modify the syntax, content, or commands of the XML before it is processed by an end system. Within XML, special elements could include reserved words or characters such as "<", ">", """, and "&", which could then be used to add new data or modify XML syntax.		
	Impact	Difficult to exploit	Risk
	High	High	Yes

CWE-284 Improper Access Control



CVSS 5,9 High	**CWE-284 Improper Acccess Control** The product does not restrict or incorrectly restricts access to a resource from an unauthorized actor Access control involves the use of several protection mechanisms such as: • Authentication (proving the identity of an actor) • Authorization (ensuring that a given actor can access a resource), and • Accountability (tracking of activities that were performed)		
	Impact	Difficult to exploit	Risk
	High	Low	Yes

CWE-829 Local File Inclusion

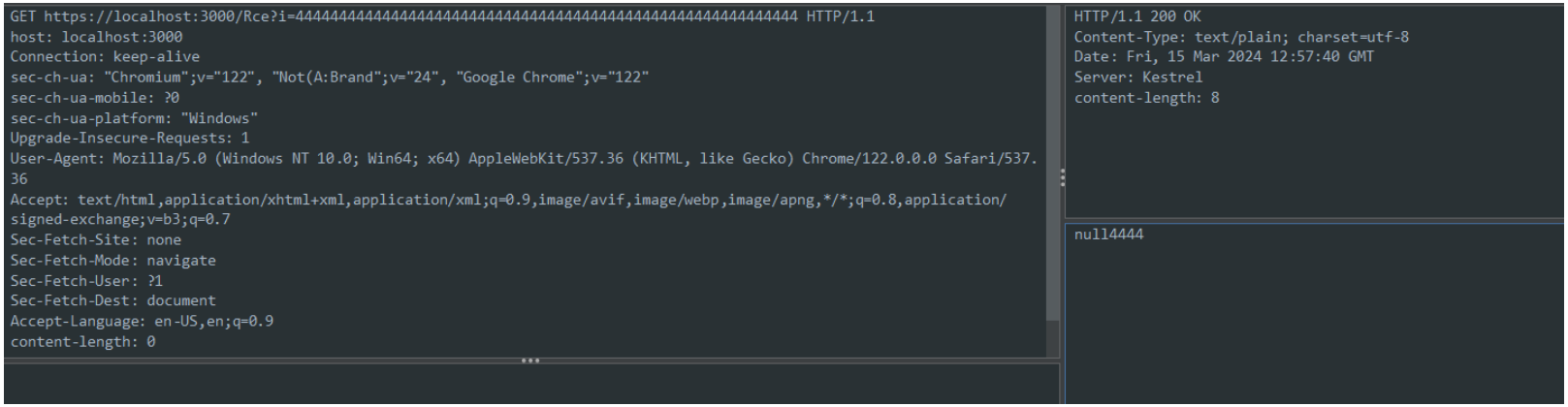


<table>
<tr><td rowspan="3">CVSS 6,3

High</td><td colspan="3">CWE-829: Inclusion of Functionality from Untrusted Control Sphere</td></tr>
<tr><td colspan="3">The product imports, requires, or includes executable functionality (such as a library) from a source that is outside of the intended control sphere.

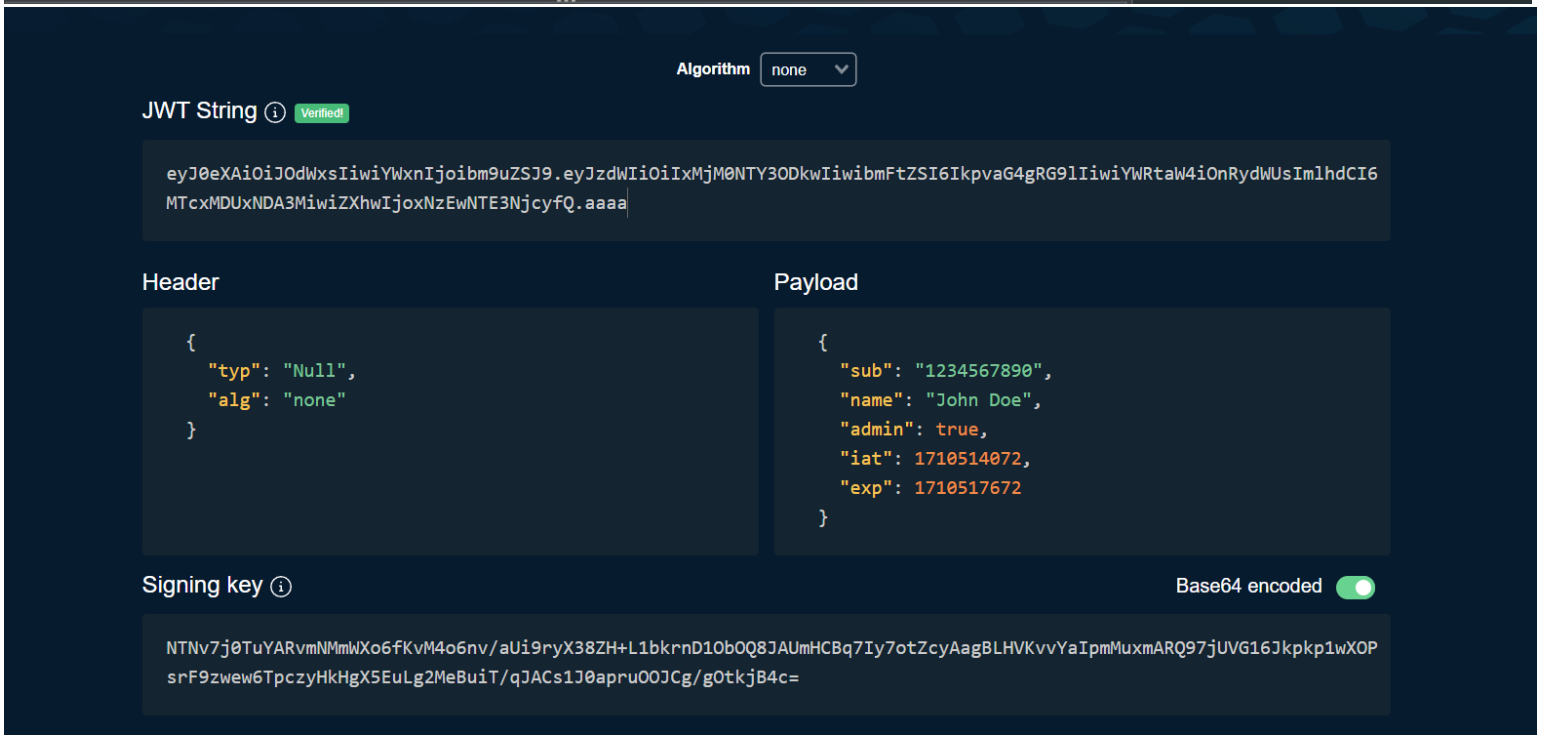
This might lead to many different consequences depending on the included functionality, but some examples include injection of malware, information exposure by granting excessive privileges or permissions to the untrusted functionality, DOM-based XSS vulnerabilities, stealing user's cookies, or open redirect to malware (CWE-601).</td></tr>
<tr><td>Impact</td><td>Difficult to exploit</td><td>Risk</td></tr>
<tr><td>High</td><td>high</td><td>Yes</td></tr>
</table>

CWE 121 Stack Based buffer Overflow



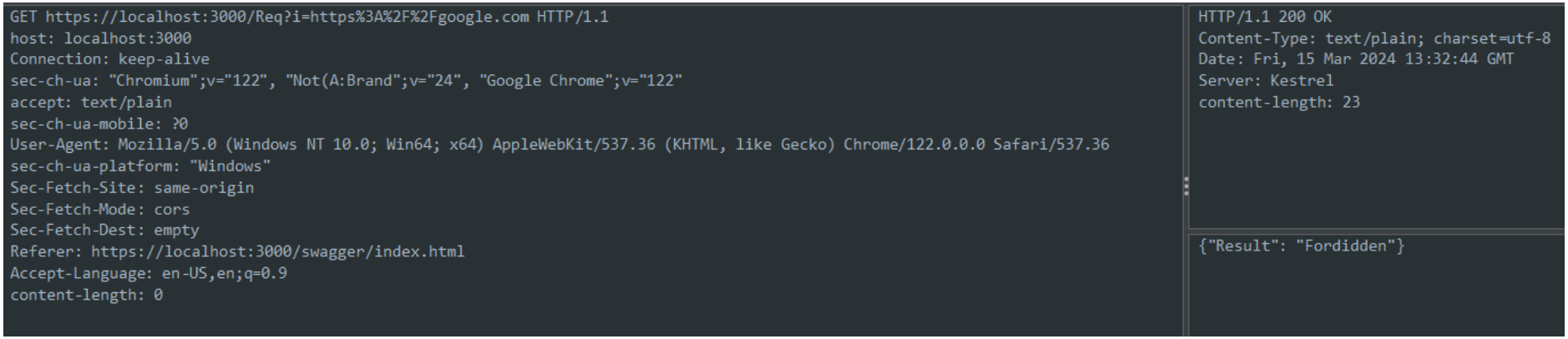
CVSS 7,7 Critical	**CWE-121: Stack-based Buffer Overflow** A stack-based buffer overflow condition is a condition where the buffer being overwritten is allocated on the stack (i.e., is a local variable or, rarely, a parameter to a function).		
	Impact	Difficult to exploit	Risk
	Strong	high	Yes

CWE 1270 Generation of incorrect Security tokens



CVSS 5,8 High	**CWE-1270: Generation of Incorrect Security Tokens** The product implements a Security Token mechanism to differentiate what actions are allowed or disallowed when a transaction originates from an entity. However, the Security Tokens generated in the system are incorrect. Systems-On-a-Chip (SoC) (Integrated circuits and hardware engines) implement Security Tokens to differentiate and identify actions originated from various agents. These actions could be "read", "write", "program", "reset", "fetch", "compute", etc. Security Tokens are generated and assigned to every agent on the SoC that is either capable of generating an action or receiving an action from another agent. Every agent could be assigned a unique, Security Token based on its trust level or privileges. Incorrectly generated Security Tokens could result in the same token used for multiple agents or multiple tokens being used for the same agent. This condition could result in a Denial-of-Service (DoS) or the execution of an action that in turn could result in privilege escalation or unintended access.		
	Impact	Difficult to exploit	Risk
	High	Low	Yes

CWE-918 Server Side Request Forgery (SSF)



```
GET https://localhost:3000/Req?i=https%3A%2F%2Fgoogle.com HTTP/1.1
host: localhost:3000
Connection: keep-alive
sec-ch-ua: "Chromium";v="122", "Not(A:Brand";v="24", "Google Chrome";v="122"
accept: text/plain
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/122.0.0.0 Safari/537.36
sec-ch-ua-platform: "Windows"
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://localhost:3000/swagger/index.html
Accept-Language: en-US,en;q=0.9
content-length: 0
```

```
HTTP/1.1 200 OK
Content-Type: text/plain; charset=utf-8
Date: Fri, 15 Mar 2024 13:32:44 GMT
Server: Kestrel
content-length: 23
```

```
{"Result": "Fordidden"}
```

CVSS 6,5 High	**CWE-918: Server-Side Request Forgery (SSRF)** The web server receives a URL or similar request from an upstream component and retrieves the contents of this URL, but it does not sufficiently ensure that the request is being sent to the expected destination. By providing URLs to unexpected hosts or ports, attackers can make it appear that the server is sending the request, possibly bypassing access controls such as firewalls that prevent the attackers from accessing the URLs directly. The server can be used as a proxy to conduct port scanning of hosts in internal networks, use other URLs such as that can access documents on the system (using file:///), or use other protocols such as gopher:// or tftp://, which may provide greater control over the contents of requests.		
	Impact	Difficult to exploit	Risk
	High	Low	Yes

CWE – 79 Cross Site Scripting

CVSS 7,7 Critical	**CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')** The product does not neutralize or incorrectly neutralizes user-controllable input before it is placed in output that is used as a web page that is served to other users Cross-site scripting (XSS) vulnerabilities occur when: 1. Untrusted data enters a web application, typically from a web request. 2. The web application dynamically generates a web page that contains this untrusted data. 3. During page generation, the application does not prevent the data from containing content that is executable by a web browser, such as JavaScript, HTML tags, HTML attributes, mouse events, Flash, ActiveX, etc. 4. A victim visits the generated web page through a web browser, which contains malicious script that was injected using the untrusted data. 5. Since the script comes from a web page that was sent by the web server, the victim's web browser executes the malicious script in the context of the web server's domain. 6. This effectively violates the intention of the web browser's same-origin policy, which states that scripts in one domain should not be able to access resources or run code in a different domain. There are three main kinds of XSS: • **Type 1: Reflected XSS (or Non-Persistent)** - The server reads data directly from the HTTP request and reflects it back in the HTTP response. Reflected XSS exploits occur when an attacker causes a victim to supply dangerous content to a vulnerable web application, which is then reflected back to the victim and executed by the web browser. The most common mechanism for delivering malicious content is to include it as a parameter in a URL that is posted publicly or e-mailed directly to the victim. URLs constructed in this manner constitute the core of many phishing schemes, whereby an attacker convinces a victim to visit a URL that refers to a vulnerable site. After the site reflects the attacker's content back to the victim, the content is executed by the victim's browser.

- **Type 2: Stored XSS (or Persistent)** - The application stores dangerous data in a database, message forum, visitor log, or other trusted data store. At a later time, the dangerous data is subsequently read back into the application and included in dynamic content. From an attacker's perspective, the optimal place to inject malicious content is in an area that is displayed to either many users or particularly interesting users. Interesting users typically have elevated privileges in the application or interact with sensitive data that is valuable to the attacker. If one of these users executes malicious content, the attacker may be able to perform privileged operations on behalf of the user or gain access to sensitive data belonging to the user. For example, the attacker might inject XSS into a log message, which might not be handled properly when an administrator views the logs.
- **Type 0: DOM-Based XSS** - In DOM-based XSS, the client performs the injection of XSS into the page; in the other types, the server performs the injection. DOM-based XSS generally involves server-controlled, trusted script that is sent to the client, such as Javascript that performs sanity checks on a form before the user submits it. If the server-supplied script processes user-supplied data and then injects it back into the web page (such as with dynamic HTML), then DOM-based XSS is possible.

Once the malicious script is injected, the attacker can perform a variety of malicious activities. The attacker could transfer private information, such as cookies that may include session information, from the victim's machine to the attacker. The attacker could send malicious requests to a web site on behalf of the victim, which could be especially dangerous to the site if the victim has administrator privileges to manage that site. Phishing attacks could be used to emulate trusted web sites and trick the victim into entering a password, allowing the attacker to compromise the victim's account on that web site. Finally, the script could exploit a vulnerability in the web browser itself possibly taking over the victim's machine, sometimes referred to as "drive-by hacking."

In many cases, the attack can be launched without the victim even being aware of it. Even with careful users, attackers frequently use a variety of methods to encode the malicious portion of the attack, such as URL encoding or Unicode, so the request looks less suspicious.

Impact	Difficult to exploit	Risk
High	Low	Yes