

# Sri Lanka Institute of Information Technology

Web Security - IE2062



Bug Bounty Report 2

PERERA A.P.J

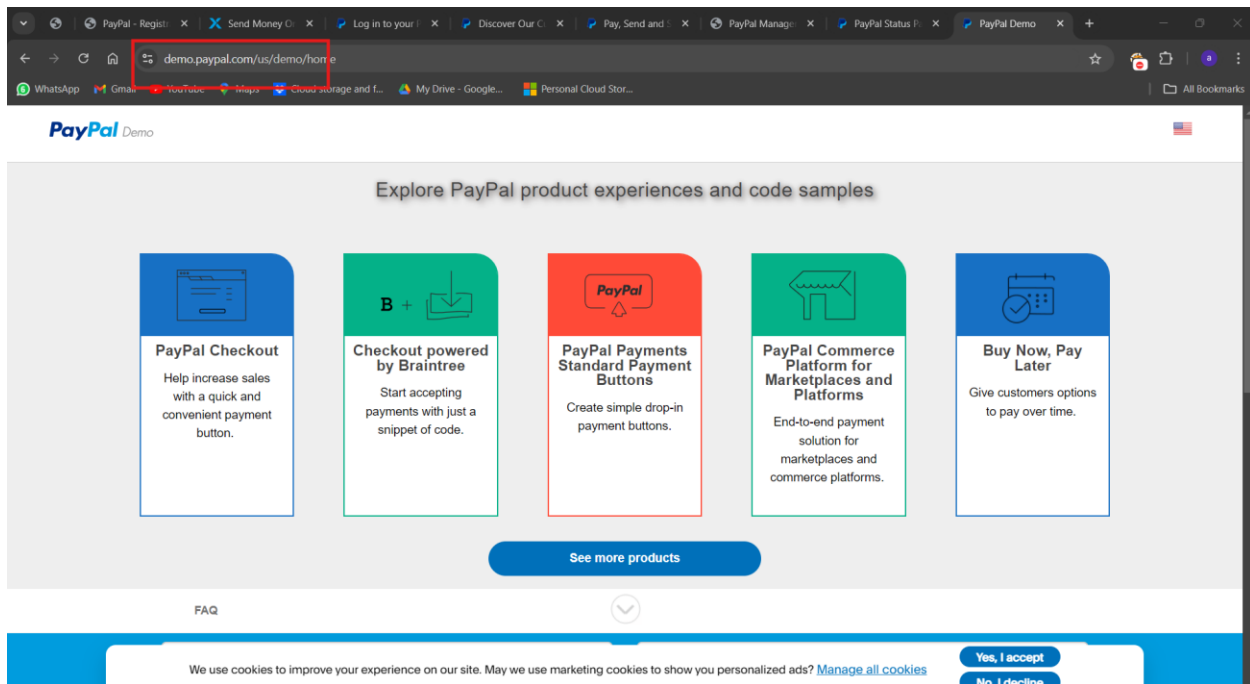
IT22280992

Group Y2S2.CS

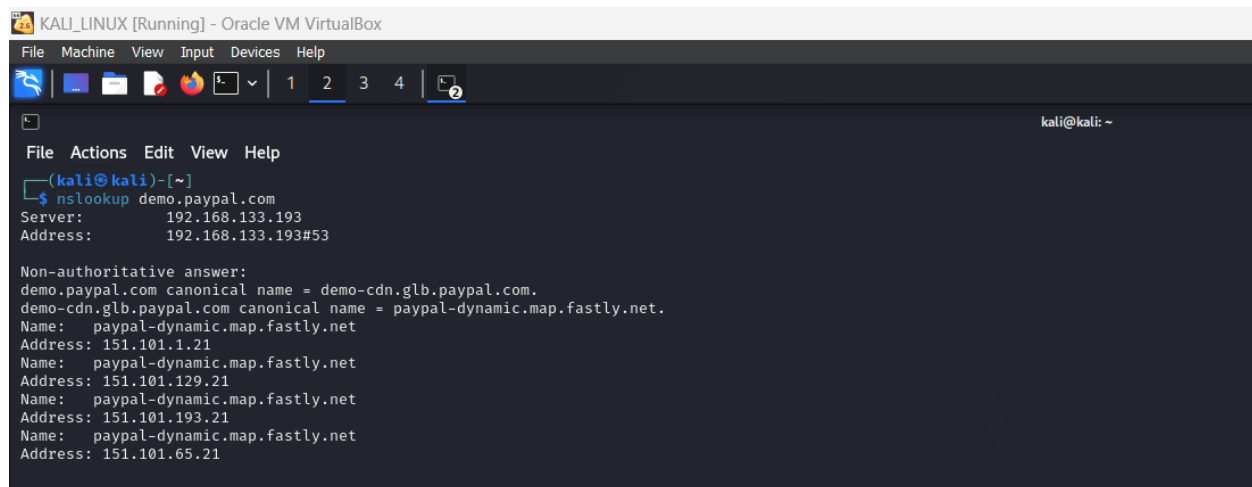
## Table of Content

1	Target :	3
1.1	Nslookup	4
1.2	Nmap	4
1.3	Dimtry	5
2	Using Zap Scanning tool Find the vulnerabilities in <a href="http://demo.paypal.com">http://demo.paypal.com</a>	6
2.1	Vulnerability	7
2.1.1	Vulnerability title	7
2.1.2	Vulnerability description	7
2.1.3	Affected components	8
2.1.4	Impact assessment	10
2.1.5	Steps to reproduce	13
2.1.6	Proof of concept (if applicable)	16
2.1.7	Proposed mitigation or fix	16
2.2	Vulnerability	17
2.2.1	Vulnerability title	17
2.2.2	Vulnerability description	17
2.2.3	Affected components	18
2.2.4	Impact assessment	20
2.2.5	Steps to reproduce	22
2.2.6	Proof of concept (if applicable)	23
2.2.7	Proposed mitigation or fix	23

1 Target : <http://demo.paypal.com>



## 1.1 Nslookup

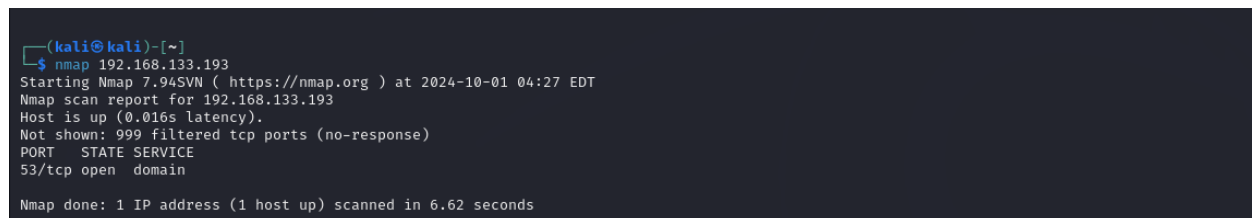


The screenshot shows a Kali Linux terminal window titled "KALI\_LINUX [Running] - Oracle VM VirtualBox". The terminal displays the output of the `nslookup demo.paypal.com` command. The output shows the server and address as 192.168.133.193. It then provides a non-authoritative answer for the domain, listing several IP addresses and canonical names for `demo.paypal.com` and `demo-cdn.glb.paypal.com`.

```
KALI_LINUX [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
kali@kali: ~
File Actions Edit View Help
(kali@kali)-[~]
$ nslookup demo.paypal.com
Server:      192.168.133.193
Address:     192.168.133.193#53

Non-authoritative answer:
demo.paypal.com canonical name = demo-cdn.glb.paypal.com.
demo-cdn.glb.paypal.com canonical name = paypal-dynamic.map.fastly.net.
Name:   paypal-dynamic.map.fastly.net
Address: 151.101.1.21
Name:   paypal-dynamic.map.fastly.net
Address: 151.101.129.21
Name:   paypal-dynamic.map.fastly.net
Address: 151.101.193.21
Name:   paypal-dynamic.map.fastly.net
Address: 151.101.65.21
```

## 1.2 Nmap



The screenshot shows a Kali Linux terminal window titled "(kali@kali)-[~]". The terminal displays the output of the `nmap 192.168.133.193` command. The output shows the scan results for the host 192.168.133.193, indicating it is up and showing the open ports (53/tcp) and service (domain).

```
(kali@kali)-[~]
$ nmap 192.168.133.193
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-10-01 04:27 EDT
Nmap scan report for 192.168.133.193
Host is up (0.016s latency).
Not shown: 999 filtered tcp ports (no-response)
PORT      STATE SERVICE
53/tcp    open  domain

Nmap done: 1 IP address (1 host up) scanned in 6.62 seconds
```

## 1.3 Dimtry

```
KALI_LINUX [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
kali@kali: ~
$ dimtry demo.paypal.com
Command 'dimtry' not found, did you mean:
  command 'dmitry' from deb dmitry
Try: sudo apt install <deb name>

kali@kali:~$ dmitry demo.paypal.com
Deepmagic Information Gathering Tool
"There be some deep magic going on"

HostIP:151.101.65.21
HostName:demo.paypal.com

Gathered Inet-whois information for 151.101.65.21

inetnum:        151.101.0.0 - 151.105.255.255
ED-ADDRESS-BLOCK
descr:          IPv4 address block not managed by the RIPE NCC
remarks:
remarks:
remarks:        For registration information,
remarks:        you can consult the following sources:
remarks:
remarks:        IANA
remarks:        http://www.iana.org/assignments/ipv4-address-space
remarks:        http://www.iana.org/assignments/iana-ipv4-special-registry
ess-space
remarks:
remarks:        AFRINIC (Africa)
remarks:        http://www.afrinic.net/ whois.afrinic.net
remarks:
remarks:        APNIC (Asia Pacific)
remarks:        http://www.apnic.net/ whois.apnic.net
remarks:
remarks:        ARIN (Northern America)
remarks:        http://www.arin.net/ whois.arin.net
remarks:
remarks:        LACNIC (Latin America and the Caribbean)
remarks:        http://www.lacnic.net/ whois.lacnic.net
remarks:
country:        EU # Country is really world wide
admin-c:        IANA1-RIPE
tech-c:         IANA1-RIPE
status:         ALLOCATED UNSPECIFIED
mnt-by:         RIPE-NCC-HM-MNT
created:        2019-01-07T10:46:38Z
last-modified:  2019-01-07T10:46:38Z
source:         RIPE
```

```
KALI_LINUX [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
kali@kali: ~
File Actions Edit View Help
created:        2019-01-07T10:46:38Z
last-modified:  2019-01-07T10:46:38Z
source:         RIPE

role:           Internet Assigned Numbers Authority
address:        see http://www.iana.org.
admin-c:        IANA1-RIPE
tech-c:         IANA1-RIPE
nic-hdl:        IANA1-RIPE
A services
remarks:        go to IANA web site at http://www.iana.org.
mnt-by:         RIPE-NCC-MNT
created:        1970-01-01T00:00:00Z
last-modified:  2001-09-22T09:31:27Z
source:         RIPE # Filtered
% This query was served by the RIPE Database Query Service version 1.114 (BUSA)

Gathered Inic-whois information for demo.paypal.com
ERROR: Unable to locate Name Whois data on demo.paypal.com

Gathered Netcraft information for demo.paypal.com

Retrieving Netcraft.com information for demo.paypal.com
Netcraft.com Information gathered

Gathered Subdomain information for demo.paypal.com

Searching Google.com:80 ...
Searching Altavista.com:80 ...
Found 0 possible subdomain(s) for host demo.paypal.com, Searched 0 pages containing 0 results

Gathered E-Mail information for demo.paypal.com

Searching Google.com:80 ...
Searching Altavista.com:80 ...
Found 0 E-Mail(s) for host demo.paypal.com, Searched 0 pages containing 0 results

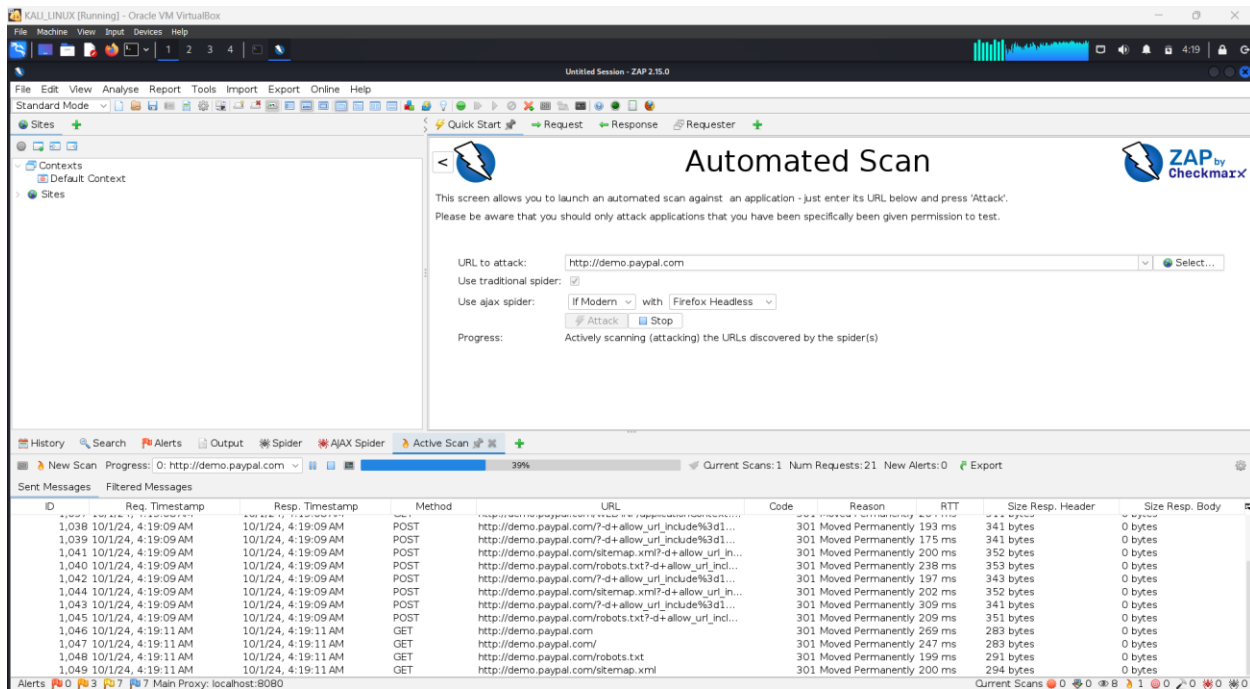
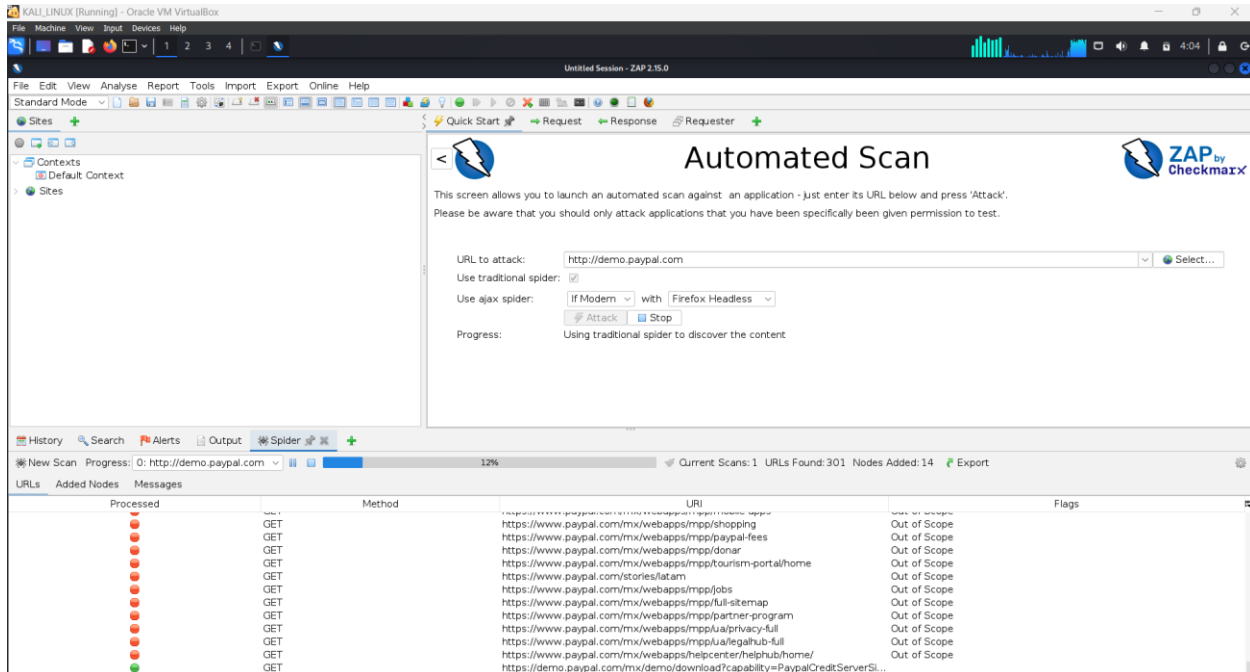
Gathered TCP Port information for 151.101.65.21

Port      State
21/tcp    open
80/tcp    open

Portscan Finished: Scanned 150 ports, 0 ports were in state closed

All scans completed, exiting
```

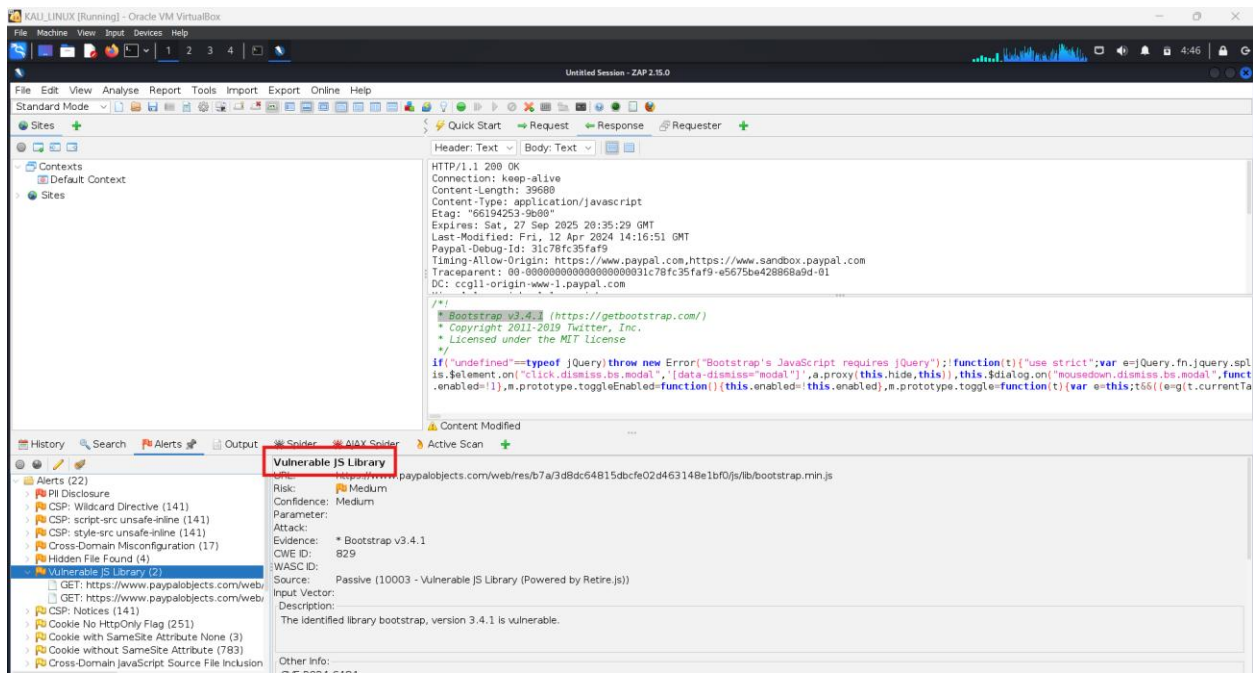
## 2 Using Zap Scanning tool Find the vulnerabilities in <http://demo.paypal.com>



## 2.1 Vulnerability

### 2.1.1 Vulnerability title

- Vulnerable JS Library.



### 2.1.2 Vulnerability description

- A Vulnerable JavaScript (JS) Library is a javascript library used in web applications that contains some security vulnerabilities posing threats to the application. Such defenses, for instance, can include threats such as Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), or Denial of Service (DoS) attacks. Usually such problems are caused by the third party libraries, which are too old and do not have updates with the security bug-fixes. Also because libraries are usually dependent on other libraries, one vulnerable library can compromise the entire application. To avoid such a scenario, the developers must not only keep on upgrading the libraries from time to time, but also employ the use of certain security measures such as Snyk or npm audit and reduce unnecessary libraries.

### 2.1.3 Affected components

- A compromise of a JavaScript (JS) library considered vulnerable usually impacts the following components:

**Client-Side Scripts:** The use of vulnerable libraries can create scenarios such as Cross-Site Scripting (XSS), where code in the form of a script is executed within the browser of the user with an intention of stealing sensitive information or a session of the user.

**Web application interface:** The enhancement of the functionality of the web interface has led to the development of various JS libraries. These may allow an attacker to compromise the web application by adjusting the functions of the web application or inserting malicious codes in the application's web forms, inputs or any other components that change dynamically.

**Data Integrity:** Discuss how, in limitation encompassing data, attacking exploiting vulnerabilities in these libraries provides access to sensitive information that is being or has been processed in the client and served.

**Session Management:** Some libraries, perhaps unintentionally, include features that leave stored session information more accessible to setters, which would otherwise require more advanced means of session-stealing such as cookie and user impersonation attacks.

**Third-party Dependencies:** Usually, any JS library has other dependent libraries or packages. If any of those components in the dependency chain are exploited, the application in question becomes at risk as an attacker can break any part of the dependency chain.

**Cross-Origin Resource Sharing (CORS):** A compromised JS library poses a threat in that users can cut across the CORS barriers put in place later resulting in the use of resources from other domains without authorization.



**API Endpoints:** When back-end API calls are carried out using a vulnerable library, a flaw in the system can be used to send a harmful request to the API causing unauthorized access or information leak or even an attack on the server.

**User Information and its Safeguarding:** Such libraries can facilitate access to private data in a manner that contravenes several laws on data privacy such as those put in place in GDPR or HIPAA.

## 2.1.4 Impact assessment

### 1. Security Threats

**Data Compromise:** Existence of flaws within JavaScript libraries could enable cyber criminals to gain access to highly sensitive data, including but not limited to users' personal and banking information, logins and passwords and any other personal identifiable information(P.I.I).

**Cross-Site Scripting (XSS):** Through a malicious library, an attacker can compromise a website by including unwarranted scripts within its pages, leading to unlawful retrieval of data, masquerading as the original user or even distributing harmful programs.

**Cross-Site Request Forgery (CSRF):** This allows a hacker to make a request, which the server accepts as coming from the authenticated user. This may lead to unintended consequences, such as the transfer of funds, permission alterations, and so on.

**Denial of Service (DoS):** There are also weaknesses that would let the attacker crash the application or make it unavailable for normal interaction, thereby causing a disruption in service.

### 2. Operational Consequences

**Application Unavailability:** Usage of loopholes in applications can cause downtime of the application affecting normal service and risk exposing the entity to possible financial and reputational losses.

**Loss of Trust:** If a user is a victim of an attack due to a compromised third-party JS library than a familiar application, this may cause the user to lose trust in the application, which affects the customer base and the company image.

**Higher Upkeep Costs:** Majority of the time, when a security incident has occurred, growing the infrastructure, so that the attack does not recur will involve fixing the vulnerabilities, performing a security review, and adding more controls, which is costly.

### 3. Legal and Compliance Issues

**Regulatory Violations:** The use of weak JS libraries can lead to data breaches which may contravene some data protection laws for instance GDPR, HIPAA, CCPA among others resulting in fines, punitive measures, or legal charges.

**Liability:** Organizations are susceptible to liability for damages arising from a data breach that is caused by the use of a vulnerably known library especially if the breach was associated with the outdated status of the library without timely updating or patching.

#### 4. Business Reputation

**Customer Churn:** There is a segment of users that can be lost due to security incidents translating to reduced users and revenue in the long run.

**Negative Publicity:** Information breaches that are associated with vulnerabilities in common Javascript libraries can lead to extensive coverage in the media, which may adversely affect the reputation of the business.

#### 5. Technical Impact

**Code Integrity:** The codebase of the application can be intruded upon by attackers who can insert codes which will change the expected usage of the application due to the use of insecure libraries in the application.

**Dependency Risks:** Where a library is used within a complex dependency structure, should a vulnerability occur, this may lead to those other libraries being affected at risk, even in applications or services using that same library.

#### 6. Resource and Performance Degradation

**Excessive Resource Utilization:** Attackers may abuse any vulnerable JS libraries resulting in excessive resource utilization which causes slowdown of the application, performance interference, or system resource depletion.

**Malicious Software Proliferation:** Any loopholes and vulnerabilities can be abused and malwares injected into the application with the potential to disseminate destruction to the device of its handler.

#### 7. Financial Effect

**Loss of Revenue:** When a security lapse leads to a disruption of business activities, there is bound to be revenue losses especially in e-commerce and subscription based companies.

**Cost of Mitigation:** There are costs associated with dealing with vulnerability which include incident response, patching and even up to the recovery after an incident which may over strain resources.

#### 8. Long Range Effects

**Security Debt:** When vulnerabilities are left unattended for too long, then that will create security debt in the long term especially when many vulnerabilities have built over a period increasing the cost and complexity of securing the system.

**Exploitation in the Future:** As soon as one of the weaknesses in a JS library is common knowledge it is only a matter of time before hackers start attacking it.

### 2.1.5 Steps to reproduce

#### 1. Determine the Insecure JavaScript Library

Assess the Existing Vulnerabilities: Look for the following:

- npm audit (for Node.js based applications) for any vulnerabilities in the package dependencies.
- Snyk or OWASP Dependency-Check for existing vulnerabilities in the external libraries used.

Examine the Available Literature: Explore the National Vulnerability Database (NVD) or CVE Details in search of the mentioned vulnerabilities. Who can tell me the CVE for the inaccurate Lib and its associated version.

Validate Library Version: Check the package.json, yarn.lock, or other similar dependency files to ensure that the appropriate version of the vulnerable library was indeed used in the project.

#### 2. Set Up Vulnerable Environment

- Download Vulnerable Version: If you're using npm or another package manager, explicitly install the vulnerable version. For example:
  - Ensure that no automated tools (e.g., npm audit fix) automatically upgrade the library.
- Create a Test Application: Set up a basic web application that uses the vulnerable JS library. This can be a simple HTML or JavaScript application where the library is integrated.

#### 3. Simulate the Vulnerability

Depending on the type of vulnerability, the exploitation steps will differ. Here are some examples of how to simulate common vulnerabilities:

- Cross-Site Scripting (XSS):
  - Use a vulnerable input field (e.g., a form or search box) where user input is not properly sanitized.
  - Inject malicious JavaScript code (e.g., `<script>alert('XSS')</script>`) and observe if the code executes within the context of the page.

#### Cross-Site Request Forgery (CSRF):

- If the JS library improperly handles session tokens, simulate a CSRF attack by creating an HTML form that sends a request to the vulnerable endpoint when a user visits your test page.

#### Denial of Service (DoS):

- Identify if the library has inefficient algorithms (e.g., regular expression vulnerabilities or memory leaks).
- Simulate sending oversized payloads or malformed inputs to cause excessive resource usage.

### 4. Perform an Application Behavior Assessment

**Confirm Execution:** In the event that the attack method works and is validated (e.g., a designated script executes, certain requests are processed, or the whole application fails), you have demonstrated the attack vector.

**Make Use of Browsers' Developer Console:** Use the browser's control panel to observe traffic, output at the console and structure of the page especially where inserting of the undesired control or request is targeted to ensure its effectiveness.

### 5. Evaluate the repercussions

**Look for Warnings:** If there is an XSS attack, check if there are leaked session cookies or if there are any other dangers where sensitive data could be compromised. If CSRF is present, check whether any actions are carried out without consent.

**Screen Records And Other Sources:** Check both server and client-side activity logs and any error/warning messages regarding the activities triggered by the stated vulnerability already exploited in the service.

## 6. Fix It and Try Again (if Necessary)

With the successful reproduction of the flaw, replace the specific vulnerable JS library with its latest patched version.

Since the flaw was said to have been fixed, attempt to re-create the defect again.

Check the security of the defect corrected library by performing the same attack that was done before this time to ascertain that the problem has been fixed.

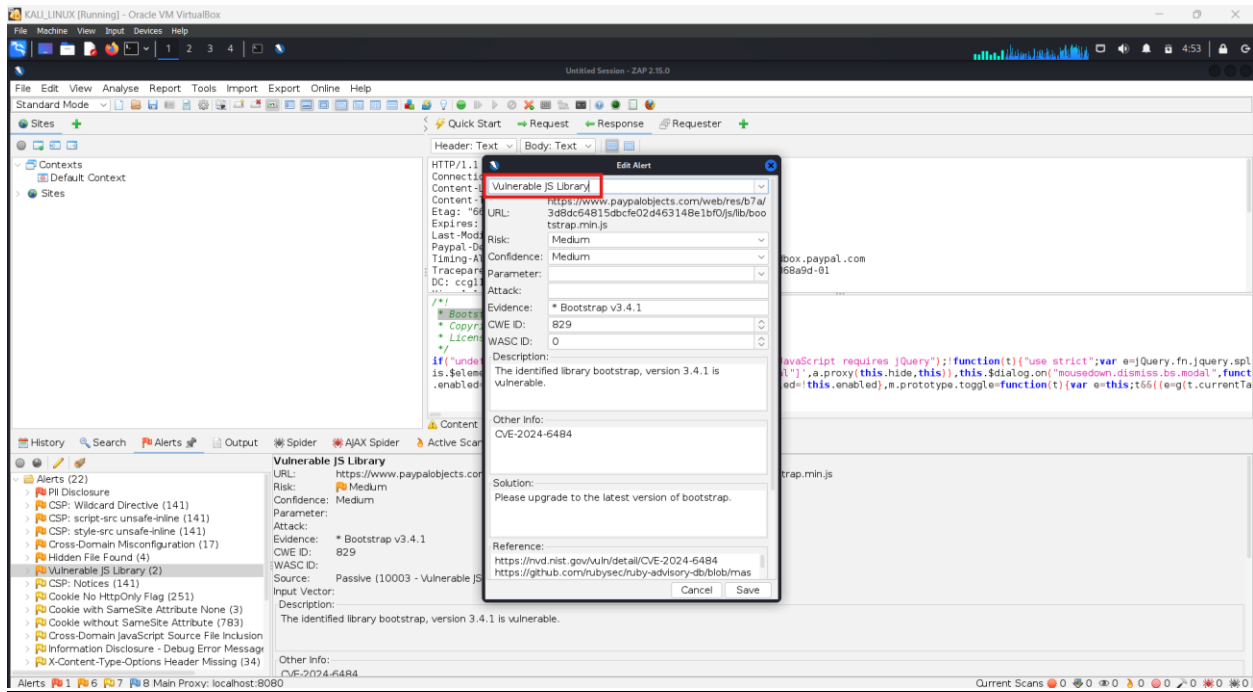
## 7. Provide the Processes Involved

Composed Written Report: Record the following information concerning the attempted reproduction of the vulnerability:

The library and the version that is vulnerable

- CVE or else vulnerability citation
- Installation of hardware and software used for testing
- Method(s) used to break into the system
- Promised Effects (e.g. XSS success, DoS activated, etc.)

## 2.1.6 Proof of concept



## 2.1.7 Proposed mitigation or fix

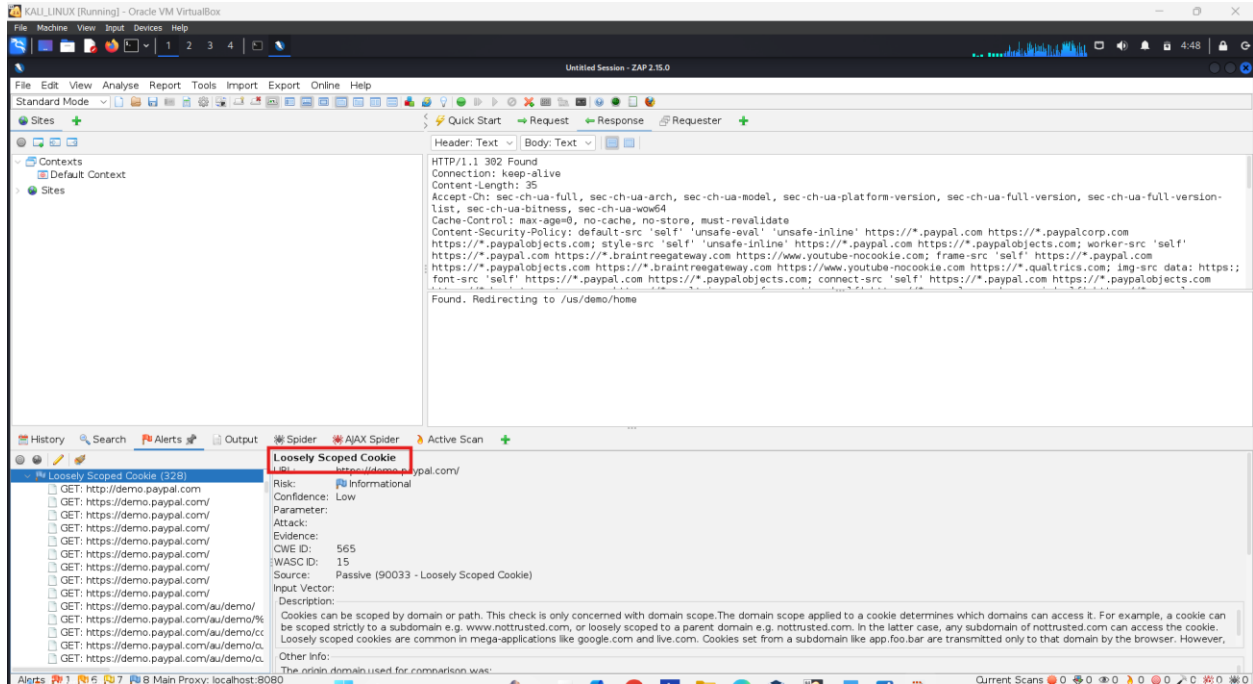
- Please upgrade to the latest version of bootstrap.
- Use Automated Vulnerable Scanning tools.
- Apply Tempary Paches.
- Security Configuration best practices
- Monitor and maintain Dependencies.



## 2.2 Vulnerability

### 2.2.1 Vulnerability title

- Loosely Scoped Cookies.



### 2.2.2 Vulnerability description

The Loosely Scoped Cookies flaw occurs when cookies are set to an unnecessary wide domain, making the cookie available on many subdomains. Such a situation creates several security risks including access and data transfer exposures between subdomains of a single website. For instance in a real life scenario if a cookie is scoped to the top level domain say example.com, then all its subdomains (for instance sub.example.com vor api.example.com) will have access to this resource. This means that if there is a flaw on one subdomain or a malicious code is present, it will be able to use the cookies and data of other subdomains, thus allowing some malicious cross subdomain operations such as Cross Request Forgery (CRF) and Cross Site Scripting (XSS).

### 2.2.3 Affected components

#### **Web Applications:**

Any web application that needs cookies for either session management, authentication or user preference settings can be impacted. Such applications include, but are not limited to, online shopping sites, content management systems (CMS), and business related applications.

#### **Subdomains:**

Websites that use cookies that are set at a high level and shared among all their subdomains are at risk. For instance, if example.com has a parent cookie that is valid for all subdomains by using the wildcard \*.example.com, then vulnerabilities in one subdomain may be able to compromise the other subdomains.

#### **Web Browsers:**

Imprecise enforcement of scoping policies in web browsers might allow cross site subdomains to read cookies set by their peers, which can possibly be abused.

#### **Authentication Mechanisms:**

In the case where users are authenticated based on session cookies (e.g. SSO) and their scope is permissive, there is a risk that an attacker might gain access to the account of the user.

#### **API Endpoints:**

RESTful APIs that implement a session mechanism based on cookies tend to be more vulnerable if the cookies have a wider scope than the intended domain.

#### **Content Delivery Networks (CDNs):**

When web sites employ CDNs for content delivery, impatiently designed cookies can infamously cross the subdomain boundary causing loss of data or exposure to potential attacks.

**Third-Party Services:**

On the other hand, those that work with cookies for the purpose of tracking or authenticating already face risks, in more detail with regards to the cookies containing a wider scope.

## **2.2.4 Impact assessment**

### **Unauthorized Access:**

Description: If the cookies carrying authentication or session IDs are poorly scoped, malicious users could possibly access protected data or even user accounts.

Impact: Compromised accounts may lead to impersonation, fraud, or even data compromise.

### **Data Leakage:**

Description: Internal users or user's browser information like customizations or personal details can leak to the unintentional sub domains due to the cookies cache memory containing the certain information.

Impact: Data Leakage can result in breach of privacy policies and debarring them from use, such as in GDPR and CCPA making the organization liable to adhere to relevant laws and lose the customer to gain their trust back.

### **Cross Site Scripting (XSS):**

Description: In case of XSS vulnerability in any sub domain, then ETD will use cookies that are not tightly bound to a domain and will be able to read/write data on other active sub domains.

Impact: This can lead to further attacks such as theft of private data or installation of harmful software on targeted systems.

### **Cross-Site Request Forgery (CSRF):**

Description: Using these cookies, attackers can act as a user even from a different subdomain and perform actions that the person did not intend to do.

Impact: This may lead to unwanted action such as purchase of items or change of user profile details without the approval of the user.

**Loss of Trust:**

Description: Users tend to distrust a web app that is unable to protect their information effectively.

Impact: Loss of goodwill may lead to reduced active users, decline in revenue made and impact on the company brands management negatively for a long period of time.

**Non-Compliance Issues:**

Description: Organizations may experience compliance-related issues with respect to the handling of user information, more so, when sensitive data is compromised in any way.

Impact: This may result in fines, sanctions and heightened oversight from regulatory bodies.

**Wider Attack Surface:**

Description: Improperly scoped cookies expand the attack surface by allowing a number of sub-domains to access a given set of cookies.

Impact: This makes the overall security of the application very difficult as it can lead to sophisticated attack vectors.

### **2.2.5 Steps to reproduce**

#### **Set Up Subdomains:**

- Create two subdomains, sub1.example.com and sub2.example.com, and ensure both are accessible.

#### **Create a Test Page on Each Subdomain:**

- On sub1.example.com, create a simple HTML page (e.g., index.html) with the following code to set a cookie:

#### **Access the Test Page:**

- Open your browser and navigate to <http://sub1.example.com/index.html>.
- Use the browser's developer tools (usually accessible with F12) to inspect the cookies and confirm that testCookie has been set with the domain .example.com.

#### **Create a Test Page on the Other Subdomain:**

- On sub2.example.com, create another HTML page (e.g., index.html) with the following code to read the cookie:

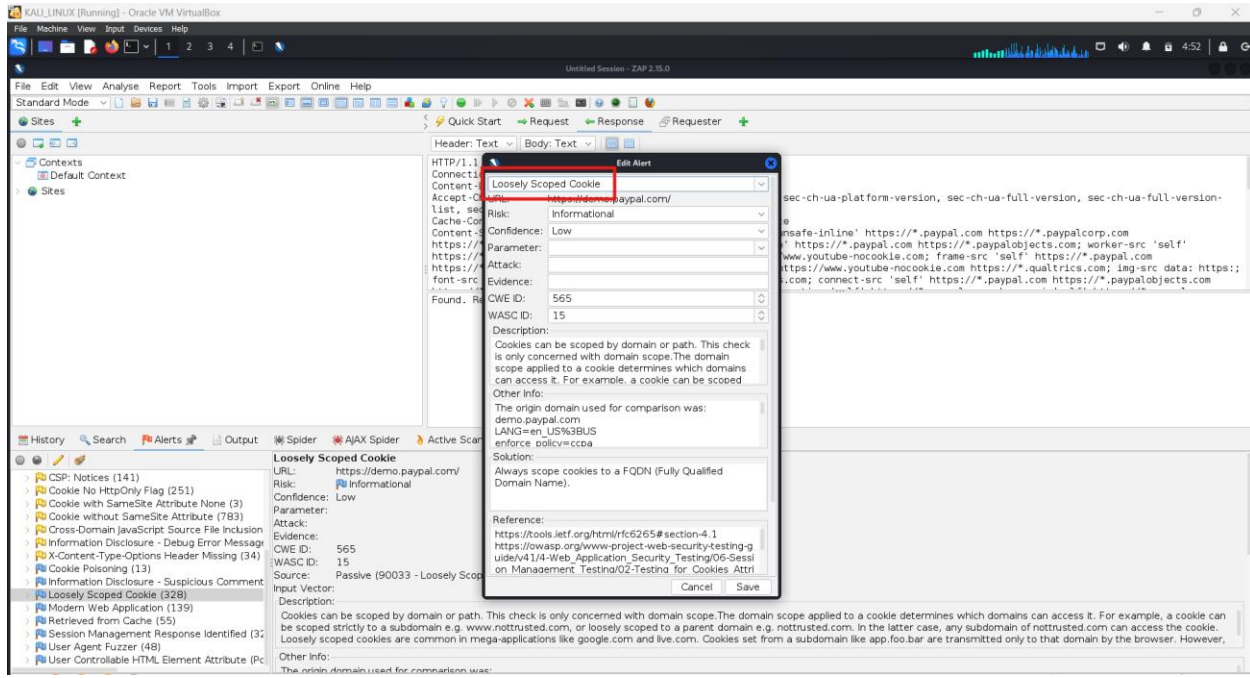
#### **Access the Second Test Page:**

- Open your browser and navigate to <http://sub2.example.com/index.html>.
- The alert should display the value of testCookie (testValue), demonstrating that the cookie is accessible across subdomains due to the loose scoping.

#### **Verification:**

- Confirm that the cookie from sub1.example.com can be accessed on sub2.example.com, illustrating the vulnerability.

## 2.2.6 Proof of concept



## 2.2.7 Proposed mitigation or fix

- Always scope cookies to a FQDN (Fully Qualified Domain Name).
- Limit Cookie lifespan.
- Regular Security Audits.
- Implement Content Security Policy.
- Monitor and log cookie usage.
- Implement Secure attribute.