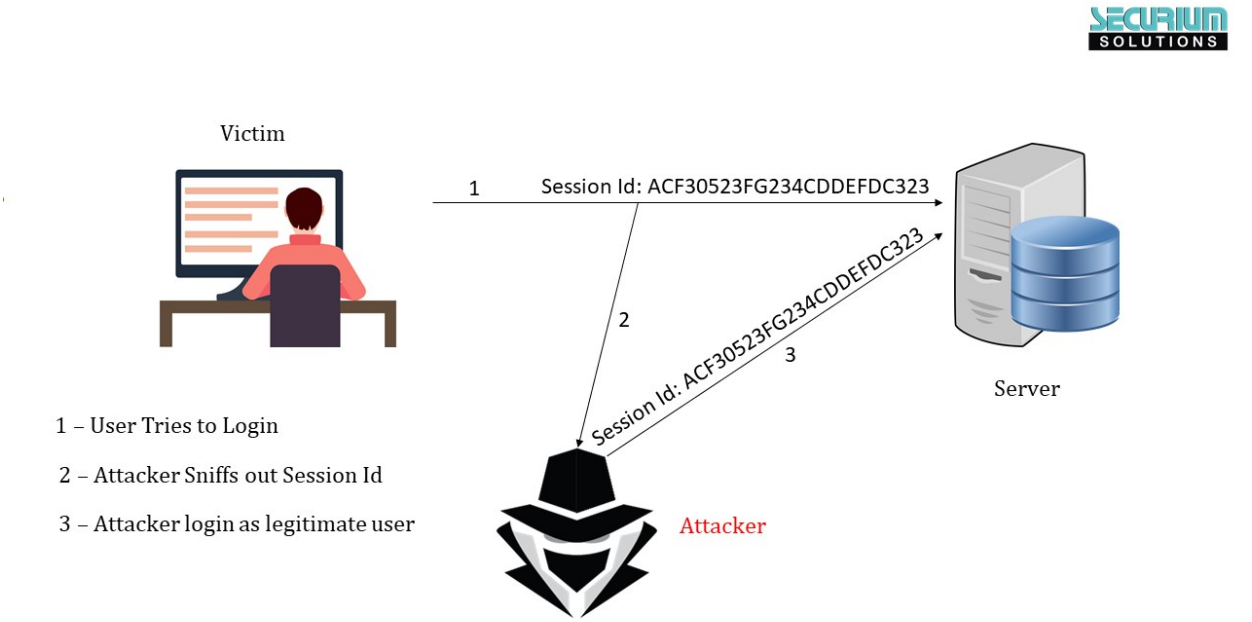
Browser Session Hijacking Procedure

- Description
  - **Browser Session Hijacking**, also known as simply **Session Hijacking** or **Cookie Hijacking**, is an attack technique in which an attacker gains access to a user's session cookie through **cross-site-scripting (XSS)**, **man-in-the-middle attacks**, **session sniffing**, etc. The attacker can then access user session information such as login credentials and anything the user has entered or done on their web page session.



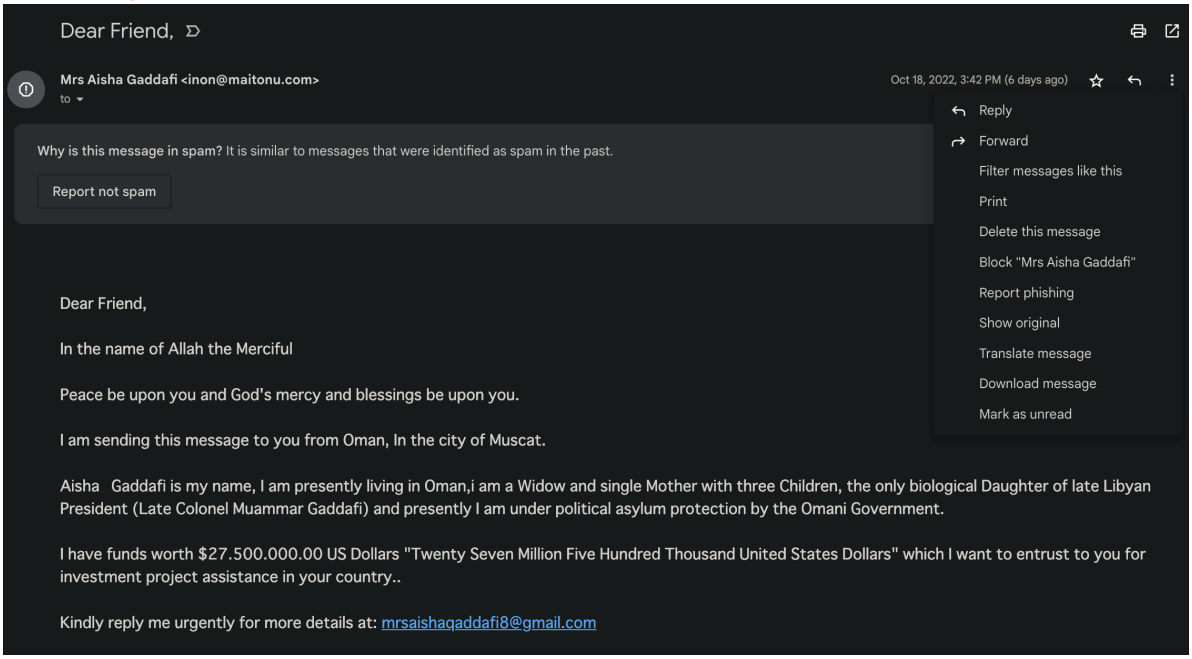
SESSION HIJACKING-HTTP ONLY USING XSS

- Incident Trigger
  - A user suddenly clicks on a spearphishing link and is prompted with a `javascript` pop-up displaying their session cookie. The user has no idea what this is but understands that it is not good by the looks of it.
- Triage
  - The browser used matters a lot. Here is a Privacy Test conducted on the most popular browsers, with few browsers such as **Brave**, **Firefox**, **Librewolf**, and **Safari** passing many of the tests. This is dangerous as **Chrome** is the ***MOST*** popular web browser. With that being said, which browser was being used?
  - Which system was the user using?
  - Determine the suspicious link(s) that caused the pop-up.
  - Copy down the email it was sent from, it may be useful later.
  - Keep a copy of the email.
  - Was the user on an insecure internet connection?
  - Was the user using `http` or `https`?
- Detection

- Attempt to safely look through the malicious links sent to determine the techniques used to grab the user's session cookie.
- Detection of XSS tactics is simple. There are *three* types of XSS.

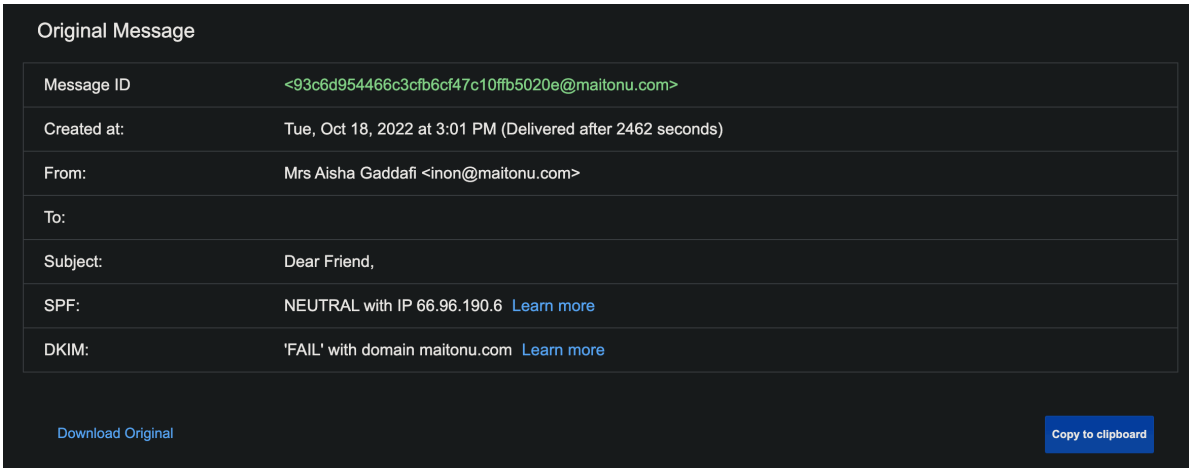
1. DOM-based XSS is a variant of XSS where the attacker uses the `Document Object Map (DOM)` to write user input out of `HTML` forms.
2. Reflected XSS, also known as Non-Persistent XSS is similar to DOM-based XSS and occurs when a malicious script is reflected off of a web application to the victim's browser. The script is activated through a link, such as `https://example.com/test/<script>alert('TEST');</script>`
3. Stored XSS occurs when user-created data is stored in a database that is loaded onto a page allows for HTML tags. For examle, imagine a platform like Twitter where users are able to tweet to massive audiences. Now imagine `HTML` tags were rendered in user-created data. This is a large security issue, because attackers can easily leave `<script>` tags with malicious code that will load onto every user's page.

- Locate the mail server that was used to send the phishing email.
  - This can be done by clicking on the three dots on the email header, and then clicking Show Original.



  - Once you click Show Original, you will be brought to a popup that displays valuable information such as messasge ID, time created, `SPF` and `DKIM`, etc. Below is an

example of what this popup looks like.



Original Message

Message ID	<93c6d954466c3cfb6cf47c10ffb5020e@maitonu.com>
Created at:	Tue, Oct 18, 2022 at 3:01 PM (Delivered after 2462 seconds)
From:	Mrs Aisha Gaddafi <inon@maitonu.com>
To:	
Subject:	Dear Friend,
SPF:	NEUTRAL with IP 66.96.190.6 Learn more
DKIM:	'FAIL' with domain maitonu.com Learn more

Download Original

Copy to clipboard

- Location of evidence
  - The email that was sent.
  - Any networks the user was connected to. It is entirely possible that the user was connected to an insecure network.
  - The website the user was browsing can be used as evidence.
  - Web browser history on the day of the hijacking.
  - Checking DNS firewall logs for any malicious traffic.
  - If the user was connected to an insecure internet connection, it may be possible to see all users who were or are still connected.
- Tools & Commands
  - `nslookup` is useful. `nslookup` can query the DNS server the link uses for DNS records.

```
DevinnoMacBook-Pro:Downloads devinrankin$ nslookup www.google.com
Server:         66.253.214.16
Address:        66.253.214.16#53

Non-authoritative answer:
Name:   www.google.com
Address: 142.250.176.196
```

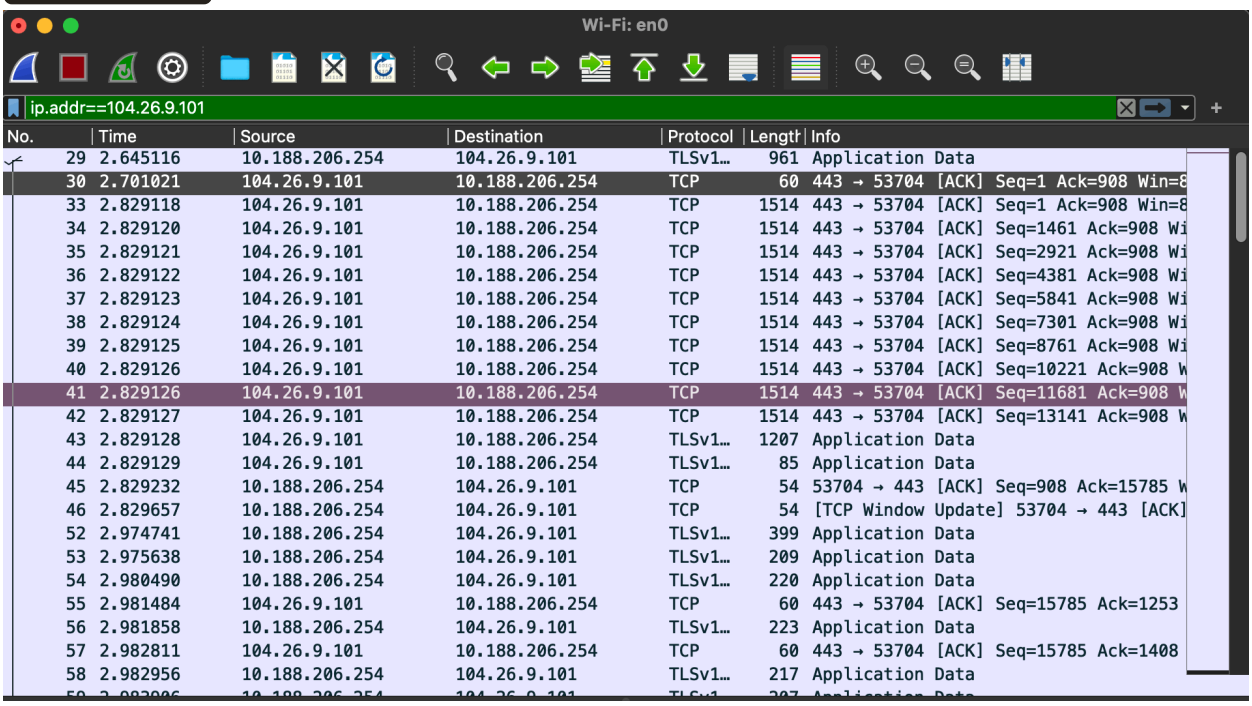
  - `nslookup type=soa` is a lookup for an SOA (State of Authority) record. This provides authorative information about the domain, the email address of the domain admin, the

domain serial number, etc.

```
DevinnoMacBook-Pro:Downloads devinrankin$ nslookup -type=soa google.com
Server:         66.253.214.16
Address:        66.253.214.16#53

Non-authoritative answer:
google.com
        origin = ns1.google.com
        mail addr = dns-admin.google.com
        serial = 483332882
        refresh = 900
        retry = 900
        expire = 1800
        minimum = 60
```

- We can use `Wireshark` to see what happens when the link is opened in a safe environment. In this instance, `TCP` is transferring `HTML` to the web browser. from `104.26.9.101`



No.	Time	Source	Destination	Protocol	Length	Info
29	2.645116	10.188.206.254	104.26.9.101	TLSv1…	961	Application Data
30	2.701021	104.26.9.101	10.188.206.254	TCP	60	443 → 53704 [ACK] Seq=1 Ack=908 Win=8
33	2.829118	104.26.9.101	10.188.206.254	TCP	1514	443 → 53704 [ACK] Seq=1 Ack=908 Win=8
34	2.829120	104.26.9.101	10.188.206.254	TCP	1514	443 → 53704 [ACK] Seq=1461 Ack=908 Wi
35	2.829121	104.26.9.101	10.188.206.254	TCP	1514	443 → 53704 [ACK] Seq=2921 Ack=908 Wi
36	2.829122	104.26.9.101	10.188.206.254	TCP	1514	443 → 53704 [ACK] Seq=4381 Ack=908 Wi
37	2.829123	104.26.9.101	10.188.206.254	TCP	1514	443 → 53704 [ACK] Seq=5841 Ack=908 Wi
38	2.829124	104.26.9.101	10.188.206.254	TCP	1514	443 → 53704 [ACK] Seq=7301 Ack=908 Wi
39	2.829125	104.26.9.101	10.188.206.254	TCP	1514	443 → 53704 [ACK] Seq=8761 Ack=908 Wi
40	2.829126	104.26.9.101	10.188.206.254	TCP	1514	443 → 53704 [ACK] Seq=10221 Ack=908 W
41	2.829126	104.26.9.101	10.188.206.254	TCP	1514	443 → 53704 [ACK] Seq=11681 Ack=908 W
42	2.829127	104.26.9.101	10.188.206.254	TCP	1514	443 → 53704 [ACK] Seq=13141 Ack=908 W
43	2.829128	104.26.9.101	10.188.206.254	TLSv1…	1207	Application Data
44	2.829129	104.26.9.101	10.188.206.254	TLSv1…	85	Application Data
45	2.829232	10.188.206.254	104.26.9.101	TCP	54	53704 → 443 [ACK] Seq=908 Ack=15785 W
46	2.829657	10.188.206.254	104.26.9.101	TCP	54	[TCP Window Update] 53704 → 443 [ACK]
52	2.974741	10.188.206.254	104.26.9.101	TLSv1…	399	Application Data
53	2.975638	10.188.206.254	104.26.9.101	TLSv1…	209	Application Data
54	2.980490	10.188.206.254	104.26.9.101	TLSv1…	220	Application Data
55	2.981484	104.26.9.101	10.188.206.254	TCP	60	443 → 53704 [ACK] Seq=15785 Ack=1253
56	2.981858	10.188.206.254	104.26.9.101	TLSv1…	223	Application Data
57	2.982811	104.26.9.101	10.188.206.254	TCP	60	443 → 53704 [ACK] Seq=15785 Ack=1408
58	2.982956	10.188.206.254	104.26.9.101	TLSv1…	217	Application Data

- Mitigation
  - Unfortunately, there aren't many things a user can do to prevent session hijacking, it is mainly a server side problem that web developers should keep in mind. There are however, a few mitigations to keep in mind.
  - A user should never click on links or download attachments from emails they do not know. Even if the emailer is someone from your network, it is never a good idea to click on the link sent by them as this can instantly give the attacker an entry point to your machine or browser sessions.
  - Always use HTTPS. This is important because HTTPS encrypts your data. A site that does not use HTTP will not encrypt your data, and may actually have your session cookie in plain

sight for attackers to grab. Installing browser plugins like `HTTPSEverywhere` can help mitigate usage of HTTP and even block certain HTTP websites from being accessed.
    - `HTTPOnly` is a cookie flag that should always be enabled. It essentially protects the session cookie from `XSS`.
    - Avoid using public WiFi or any insecure network. Most modern machines will notify you if a network is insecure. Insecure networks do not encrypt data transmitted.
- Eradication
    - To get rid of **session hijacking**, this is purely up to web developers and companies. They must use HTTPOnly flags for their session cookies, and they must install mitigation tactics such as Phishing alerts, VPN's, web frameworks, and malware detection/prevention software. It also helps if companies notify their employees on the dangers of **session hijacking**.