# Identification or Authentication Failures

November 10, 2023

Emmanuel Benoist | BFH-TI



## Identification or Authentication Failures


- **Introduction**
- **Examples of Attacks**
  - Brute Force
  - Session Spotting
  - Session Fixation Attack
  - Session Hijacking
  - Session Expiration
- **Protection**
- **Conclusion**




# Introduction



## Identification or Authentication Failures

- **Account credentials and sessions tokens are often not properly protected**
  - A third party can access to one's account
  - Attacker compromise password, keys or authentication token
- **Risks**
  - Undermine authorization and accountability controls
  - cause privacy violation
  - Identity Theft
- **Method of attack: use weaknesses in authentication mechanism**
  - Logout
  - Password Management
  - Timeout
  - Remember me
  - …

# Examples of Attacks



## Brute Force Attack

- **Automated process of trial and error**
  - Guess a person username and password, credit-card number, cryptographic key, …
  - System sends a value and waits for the response, then tries another value, and so on.
  - Often done off-line with extracts of the DataBase
  - Can be done on-line on unprotected sites
- **Many systems allow the use of weak passwords**
  - An attacker will cycle through a dictionary (word by word)
  - Generates thousands (potentially millions) of incorrect guesses
  - When the guessed password is OK, attacker can access the account!
- **Same technic can be used to guess encryption keys**
  - When the size of the key is small,
  - An attacker will test all possible keys



## Brute Force Attack (Cont.)

- **Normal Brute Force**
  - For one username,
  - Attacker tests many passwords

```
Username = Emmanuel
Passwords = 1234567, qwertz, asdfgh, abcd, ....
[pet names], [birthdays], [car names], [dictionary]...
```

- **Lists of known passwords can be found**
  - Connection Username - Password (or hashed passwords) on the Darknet.
    Test all the pairs (user-pwd)
  - Lists of passwords (without usernames)
    for a user, test all the passwords
    `https://github.com/danielmiessler/SecLists/tree/master/Passwords`



## Permits automated attacks

- **Attackers can store lists of usernames and passwords**
  - Can be stored and sold on darknet markets
- **Can automate attacks**
  - Same credentials are reused over multiple web sites
  - Same username same password
  - Scripts for testing the validity of a pair
  - Lists are used to test access.

## Session Spotting

- **Attacker has the possibility to listen to the traffic of the victim**
  - Listens to the traffic at the IP level (sniffer)
  - Only the login page is secure, the rest of the application is not encrypted.
- **Client connects to the server** `http://www.mysite.com`
  - Visits a page containing a login form (url is HTTPS)
  - Receives a cookie containing his session ID
  - Sends his credentials encrypted (HTTPS)
- **Attacker receives following information**
  - Session ID
  - Sees that the user has sent his credentials (using an encrypted connection to the server)
- **Attacker can use the cookie to be recognized as the legitimate user!**



## Unsecure cookies

- **Attacker has the possibility to listen to the traffic of the victim**
  - Listens to the traffic at the IP level (sniffer).
- **Client connects to the HTTPS server** `https://www.mybank.com`
  - Client receives a cookie containing the session ID.
  - This cookie is resent each time the browser accesses this site.
  - The cookie is linked to an active session on the secure server.
- **Victim visits a page on the unsecure web site** `http://www.mybank.com`
  - For seeing some advertisement for instance.
  - The cookie (if not “secure”) will be sent unencrypted to the server.
- **Attacker can see the sessionID**
  - Attacker can impersonate the victim
- **Solution:**
  - Use only secure cookies (set the bit secure on)
  - Do not reuse existing cookies.



## Unsecure cookies

- **Attacker can run a XSS attack**
  - Victim will execute JavaScript inside the browser
  - JS will read the cookie
  - JS will send this cookie to another server
- **Solution**
  - Session cookies should be in http header only (no JavaScript)



## Session Fixation Attack

- **Attacker creates a session on a web site**
  - Sends a Request,
  - Get a Response containing a cookie (SESSION_ID=`1234`)
  - Attacker needs to maintain this session alive (send requests regularly)
- **Attacker sends this Session ID to the victim**
  - Can be included in a phishing.
    He sends an email containing the reference to the following URL :
    `https://www.ebanking.com/?page=...&SESSION_ID=1234`.
  - Can be a link:

```
<a href="https://www.ebanking.com/?SESSION_ID=1234">
```

## Session Fixation attack (Cont.)

- **Victim clicks on the given link**

  ```
  <a href="https://www.ebanking.com/?SESSION_ID=1234">
  ```

  - Browser sends automatically the SessionID within the request
- **Session** `1234` **is used by the victim**
  - Victim logs in,
  - The session is valid.
- **When the attacker checks the session he/she receives the rights of the victim!**



## Session Fixation Attack (Cont.)

- **Do not accept session identifiers in the URL**
  - It is the door for Session Fixation Attack
- **Reset the SessionID when a login occurs**



## Session Hijacking

- **Credential/Session Prediction**
  - Attackers deduce or guess the session id
  - Attackers can use the web site with victim's privileges
- **Rights are stored in a session, only the session id is used to link the browser and its session**
  - HTTP is session-less
  - Information is not resent in each request
- **Guessing the Session ID permits to be the user**



## Session Hijacking: Example

- **Many web sites generate session IDs with proprietary algorithms**
  - Increment static numbers
  - Can be more complicated (factoring in time and other computer specific variables)
  - Session ID is sent to the client
- **An attack can be:**
  - Attacker connects to the web site and gets a session ID
  - Attacker calculates or Brute Forces the next session ID
  - Attacker switches the value of the cookie and assumes the identity of the next user!

## Real life Example

- **One web site has a “password lost” page**
  - Users having lost their password ask for a renewal
  - They receive an email containing a link:

```
<a href="https://site.com/reset?token=34349ab9938bc">
  Renew Password </a>
```

  - User clicks on this link,
  - he accesses a page where he can reset his password.
- **Token is generated by a PRNG**
  - Pseudo Random Number Generator are well known.
  - PRNG of PHP, Java, C, Phython, …are well documented
  - A number is used as seed for the next number
  - If one knows one number, one can generate the next one



## Real life Example (Cont.)

- **Exploit**
  - Ask for renewal of password of a real user
  - Get the token
  - Ask for the renewal of password of the administrator
  - Mail is sent to the admin (attacker can not read it)
  - Use the first token as seed to get the new token
  - Use the new token to reset the password of the admin.



## Insufficient Session Expiration

- **Can be exploited on a shared computing environment**
  - More than one person has physical access to a computer
- **Suppose logout function sends the victim to site’s home-page without deleting the session**
  - Or more likely, that the user just closed the window without logging-out
- **Another user could go through the browser’s history and view pages accessed by the victim**
  - Since the victim’s session ID has not been deleted,
  - The attacker would be able to get the privileges of the victim.



# Protection

## Protection

- **Prohibit brute force attack**
  - Add rules on your firewalls or application limiting the number of tests for a given user or IP address
- **Authentication relies on secure communication and credential storage**
- **SSL should be the only option for all authenticated parts of the application**
  - Otherwise, listening to credential is possible
- **All credentials should be stored in hashed or encrypted form**
  - Attack on the database or file system should not compromise credentials
  - password should systematically be hashed
  - Private keys should never be stored clear text



## No self-made session or SSO system

- **Only use inbuilt session management mechanism**
  - Do not write or use secondary session handlers!
- **Do not use “remember me” or home grown Single Sign On**
  - Does not apply to robust SSO or federated authentication solutions
- **Writing a robust and secure solution requires high knowledge in security**
  - Cryptography
  - Storage
  - …



## Protection (Cont.)

- **Use a single authentication mechanism**
  - With appropriate strength and number of factors
  - Ensure it is hard to spoofing and replay attacks
- **Do not make the mechanism overly complex**
  - it may become subject to an attack



## Start login process from an encrypted page

- **Do not allow the login process to start from an unencrypted page**
- **Always start login from a second page**
  - Encrypted
  - Using a fresh or new session token
- **Prevents credential or session stealing**
  - Phishing attacks
  - and Session Fixation attacks

## Take Care of Logout

- **Ensure that every page has a logout link**
  - Users should not have to go to the start page to logout
- **Logout should destroy the credentials**
  - All server side session state
  - Client cookies
- **Consider Human Factor**
  - Do not ask for confirmation
  - Users will end up closing the window rather than logging out successfully
  - Give the users information about closing sessions
- **Use a timeout period**
  - Automatically logs out an inactive session



## No spoofable credentials as authentication

- **Do not rely on credentials that can be spoofed**
- **TCP/IP spoofing**
  - IP Addresses
  - Address range masks
  - DNS
  - or reverse DNS lookups
  - …
- **HTTP spoofing**
  - Referrer Header



## Be careful with e-mails

- **Do not send e-mails containing passwords**
  - Can be read
- **Use limited-time-only random numbers to reset access**
  - And send a follow up e-mail as soon as the password has been reset
- **Be careful of allowing users to change e-mail**
  - Send a message to the previous e-mail address before enacting the change



## Use two Factor Authentication for specific actions

- **2FA?**
  - Send a one time password using a SMS
  - Send a one time password using a email
  - Show an encrypted challenge response
    Using the public key of the user (stored inside the system).
    Print an encrypted one time password in the web page. Ask the user to solve it
    Only the owner of the private key can read it
  - Can be done inside a app.
- **Specific actions**
  - Login of admins or privileged users
  - Transfer of money
  - Add new recipients for money
  - Change password
  - … (business specific)

## Specifications on Passwords[1]

- **Setting or Changing of Passwords**
  - Server shall verify that it is not in a previously leaked list
  - shall test if it is not too simple ('aaaaaa' or '1234abcd')
  - shall test also context-specific words (username or servername for instance).
- **When password is rejected**
  - Server shall advise the user of the rejection,
  - site must provide a reason for it.
  - and provide a metter to measure quality
- **SHOULD NOT impose composition rules**
  - Like requiring mixtures of different character types or prohibiting consecutively repeated characters
- **SHOULD NOT require memorized secrets to be changed arbitrarily (e.g., periodically)**
  - Should force the change if there is evidence of compromise of the authenticator.

[1] `https://pages.nist.gov/800-63-3/sp800-63b.html#memsecret`



# Conclusion



## Conclusion

- **Attacks on Credentials are numerous**
  - Session / Username and passwords / Keys
  - From Brute Force to Session Hijacking
- **Protection may be related with risk**
  - Risk for a guestbook $\neq$ e-banking
  - Security can not be maintained at the same level
  - Ratios Cost/Efficiency/Usability
- **New development**
  - Use Biometrics for providing the credentials
  - Axionics Cards used fingerprint
  - Keystroke biometrics may be used for password recovery.



## References


- **OWASP Top 10, A2:2017, A07:2021**
  `https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/`
- **OWASP - A Guide for Building Secure Web Applications and Web Services**
- **Web Application Security Consortium: Threat Classification**
  `http://www.webappsec.org`
- **NIST Special Publication 800-63B, DIgital Identity Guideline**
  `https://pages.nist.gov/800-63-3/sp800-63b.html`