

Finding: It is relatively easy to hijack a session which is not regenerated.

3.0 Technical Findings: Defence

3.1. Defending against a Brute Force Attack

To prevent brute-force attacks is just to lock accounts following a specified number of unsuccessful password attempts and changing port numbers. By configuring rules in the iptables which control network traffic based on the defined rules. Account lockout firewall rule can be for a specified time interval, for example, an hour, or the accounts could remain locked until manually released by an administrator. The rule is specified in the command line in figure16.

This firewall will detect and mitigate brute force attacks by blocking malicious IP addresses. Another simple way to prevent brute force attack is to change the default port to something else. This is not the greatest way to defend the attack as the attacker can discover the open port but it can prevent automated attacks.

```
[kali㉿kali)-[~/Downloads] $ time medusa -h 192.168.117.133 -u msfadmin -P mypasslist.txt -M ftp -t 5 -T 16 -v 6  
Medusa v2.3_rc1 [http://www.foofus.net] (C) JoMo-Kun / Foofus Networks <jmkm@foofus.net>  
-H [FILE] : File containing target hostnames or IP addresses  
GENERAL: Parallel Hosts: 16 Parallel Logins: 5 test  
GENERAL: Total Hosts: 1 FILE : File containing usernames to test  
GENERAL: Total Users: 1 TEXT : Password to test  
GENERAL: Total Passwords: 45 FILE : File containing passwords to test  
ERROR: [ftp.mod] Server sent unknown response code: 500 entries. See README for more info  
ERROR: [ftp.mod] Server sent unknown response code: 500 formation to  
ERROR: [ftp.mod] Server sent unknown response code: 500 hecks ([n] No Password, [s] Password  
ERROR: [ftp.mod] Server sent unknown response code: 500 execute (without the .mod extension)  
ERROR: [ftp.mod] Server sent unknown response code: 500 the module. This can be passed mult  
GENERAL: Medusa has finished.  
  
real    1.04s    i.e., different parameter each time and they will all be sent  
user    0.00s  
sys     0.00s    -m Param1 -m Param2, etc.)  
cpu     0%       -d : Dump all known modules  
                  -n [NUM] : Use for non-default TCP port number
```

Changing the default ftp port from 20 to 2121 refuses the automated brute force attack.

```
msfadmin@metasploitable:~$ sudo iptables -A INPUT -p tcp --dport 22 -m recent --name sshguard --update --seconds 60 --hitcount 4 -j DROP
msfadmin@metasploitable:~$ _
```

Figure 15 - Firewall configuration using IPTables

This firewall command protects from brute-force attacks by refusing connection when many login attempts are conducted in a short period of time. The firewall rule is explained as follows.

'*Iptables -A INPUT*': Adds a rule into the iptables rules.

'-p tcp -dport 22': Applicable to TCP on port 22.

'--seconds 60 --hitcount 4': If the IP connects 4 times in 60 seconds it triggers the rule

'-j DROP': Drops the connection

```
msfadmin@metasploitable:~$ sudo iptables -L -n -v
Chain INPUT (policy ACCEPT 160 packets, 29011 bytes)
pkts bytes target     prot opt in     out      source          destination
  0    0          tcp  --  *      *      0.0.0.0/0            0.0.0.0/0
      tcp dpt:22 recent: SET name: sshguard side: source
  0    0   DROP     tcp  --  *      *      0.0.0.0/0            0.0.0.0/0
      tcp dpt:22 recent: UPDATE seconds: 60 hit_count: 4 name: sshguard side:
source

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target     prot opt in     out      source          destination

Chain OUTPUT (policy ACCEPT 159 packets, 28685 bytes)
pkts bytes target     prot opt in     out      source          destination
msfadmin@metasploitable:~$ _
```

Figure 16 - Listing current IPTables configuration

Check existing Configuration by 'sudo iptables -L -n -v'.

Shows that the firewall has been configured.

```
(kali㉿kali)-[~/Downloads]
$ time medusa -h 192.168.117.133 -u msfadmin -P mypasslist.txt -M ssh -t 5 -T 16 -v 6
[kali㉿kali]-[~/Downloads] -> ./medusa -h 192.168.117.133 -u msfadmin -P mypasslist.txt -M ssh -t 5 -T 16 -v 6
Medusa v2.3_rc1 [http://www.foofus.net] (C) JoMo-Kun / Foofus Networks <jmk@foofus.net>

GENERAL: Parallel Hosts: 16 Parallel Logins: 5 Foofus Networks <jmk@foofus.net>
GENERAL: Total Hosts: 1
GENERAL: Total Users: 1 Parallel Logins: 5
GENERAL: Total Passwords: 45
ERROR: Thread 6FFFF6C0: Host: 192.168.117.133 Cannot connect [unreachable], retrying (1 of 3 retries)
ERROR: Thread 748576C0: Host: 192.168.117.133 Cannot connect [unreachable], retrying (1 of 3 retries)
ERROR: Thread 6FFFF6C0: Host: 192.168.117.133 Cannot connect [unreachable], retrying (2 of 3 retries)
ERROR: Thread 748576C0: Host: 192.168.117.133 Cannot connect [unreachable], retrying (2 of 3 retries)
ERROR: Thread 6FFFF6C0: Host: 192.168.117.133 Cannot connect [unreachable], retrying (3 of 3 retries)
ERROR: Thread 748576C0: Host: 192.168.117.133 Cannot connect [unreachable], retrying (3 of 3 retries)
NOTICE: ssh.mod: failed to connect, port 22 was not open on 192.168.117.133 User: msfadmin
NOTICE: ssh.mod: failed to connect, port 22 was not open on 192.168.117.133
[ 01:53:58 ACCOUNT CHECK: [ssh] Host: 192.168.117.133 (1 of 1, 0 complete) User: msfadmin
  0 complete) Password: sdfhsdfgh (4 of 45 complete)
[ 01:53:58 ACCOUNT CHECK: [ssh] Host: 192.168.117.133 (1 of 1, 0 complete) User: msfadmin
```

Figure 17 - Brute force attempt using Medusa, firewall set up

The ‘Cannot connect’, ‘retrying’ line means that the SSH service on the target machine is still running, however, the attack machine is being blocked from reaching it as it has failed to connect less than 4 attempts.

Firewall dropped packets, therefore showing as port 22 not opened.

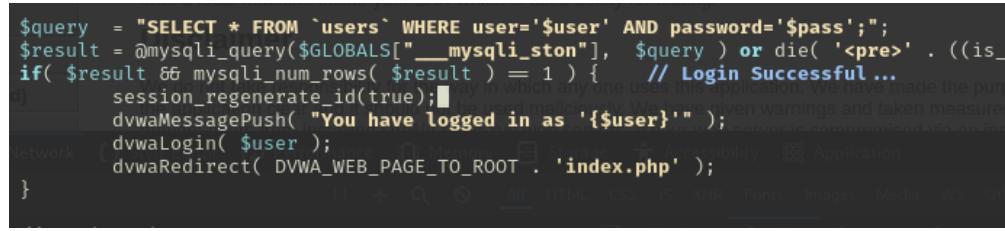
```
(kali㉿kali)-[~/Downloads]
$ ssh -oHostKeyAlgorithms=+ssh-rsa -oPubkeyAcceptedAlgorithms=+ssh-rsa msfadmin@192.168.117.133
```

Figure 18 - SSH connection to verify

To verify whether the IP address has successfully dropped, I attempted to connect to target SSH. The connection was left hanging and remained unresponsive, indicating that the attacker machine IP is blocked by the firewall.

3.2. Defending against Session Fixation

To prevent session fixation attacks is to regenerate the session ID after successful login. In figure 19, in the login.php file, session_regeneration_id(true) line was added to accomplish this. This line generates a completely new session ID and deletes the existing session ID which means that previously hijacked session ID becomes invalid. This technique defends against session fixation directly by not allowing an attacker to predict or reuse a session ID that has been issued prior to authentication.



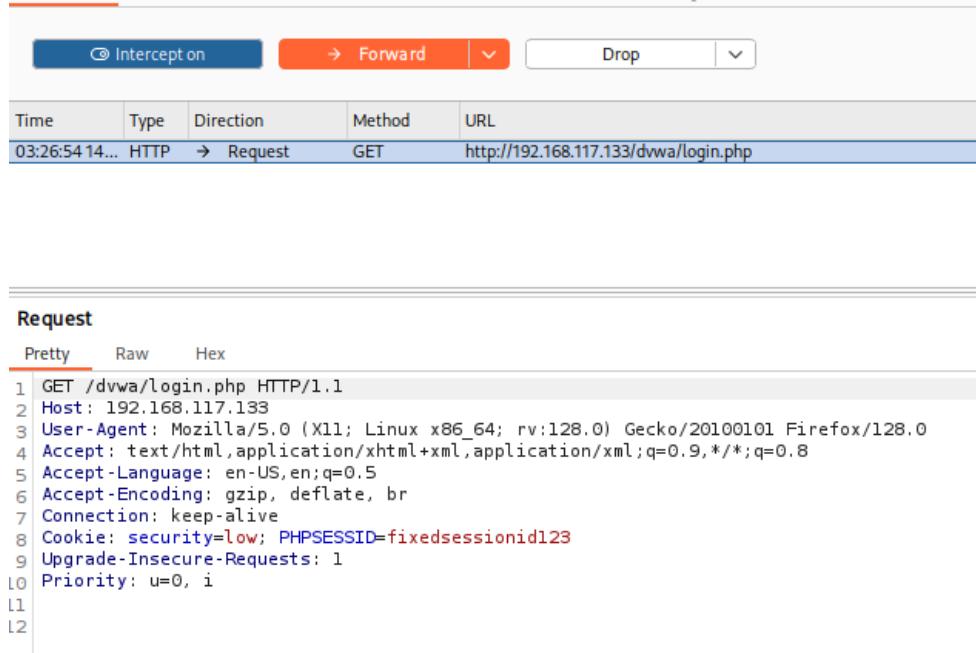
```

$query = "SELECT * FROM `users` WHERE user='$user' AND password='$pass';";
$result = @mysql_query($GLOBALS["__mysqli_ston"], $query) or die( '<pre>' . ((is_
if( $result && mysqli_num_rows( $result ) == 1 ) { // Login Successful ...
    session_regenerate_id(true);
    dwvaMessagePush( "You have logged in as '{$user}'" );
    dwvaLogin( $user );
    dwvaRedirect( DVWA_WEB_PAGE_TO_ROOT . 'index.php' );
}

```

Figure 19 - MySQL query

Adding a line in “session_regeneration_id(true) in login.php code, it prevents session fixation as it creates a new session and deletes the old session. The old hijacked session from the hacker becomes useless.



Time	Type	Direction	Method	URL
03:26:54 14...	HTTP	→ Request	GET	http://192.168.117.133/dvwa/login.php

Request

Pretty Raw Hex

```

1 GET /dvwa/login.php HTTP/1.1
2 Host: 192.168.117.133
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Connection: keep-alive
8 Cookie: security=low; PHPSESSID=fixedsessionid123
9 Upgrade-Insecure-Requests: 1
10 Priority: u=0, i
11
12

```

Figure 20 - Burpsuite packet interception

Burpsuite intercept the HTTP request and session ID is fixed to ‘fixedsessionid123’.

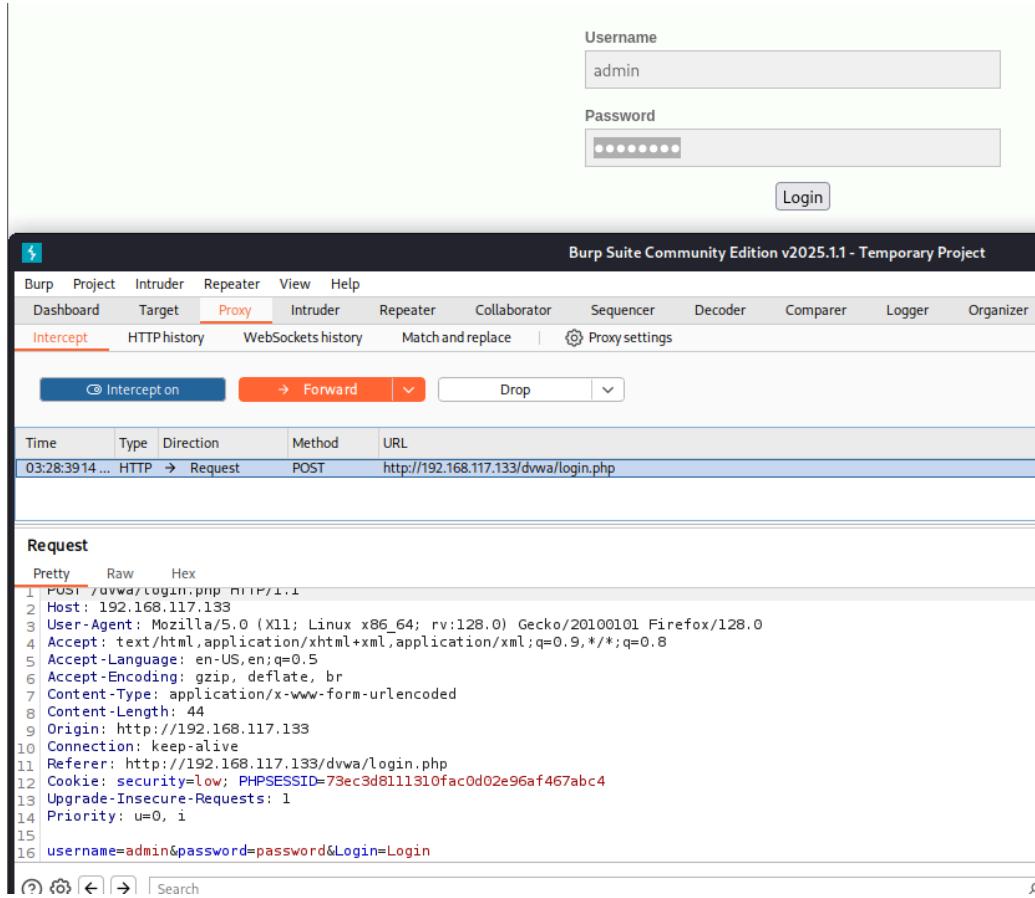


Figure 21 - Burpsuite packet interception w/Session regeneration

Burpsuite was used again to intercept HTTP request after the user successfully logged in. The session ID returned did not match the pre-fixed ID 'fixedsessionid123'. This indicates that the session ID had been re-generated after the login process. This confirms that the system is protected from session fixation attacks since previously fixed session ID becomes invalid after authentication.

4.0 Discussion and Analysis

4.1 Brute Force Attack

Brute force attacks are becoming increasingly effective with advancements in computational power. Jo and Yoon (2015) demonstrate how high-performance systems like GPUs and FPGAs can significantly reduce attack durations through parallel processing, achieving substantial throughput with CPU utilisation averaging 30%. In the first demonstration, no defensive mechanisms were in place, allowing the attacker to rapidly cycle through login attempts unimpeded. Mohan and Srivastava (2010) illustrate that precomputed lookup tables and rainbow tables can further expedite attacks by trading storage for time, though these methods were not utilised in the attack demonstration but could have drastically accelerated the process.

Machine learning also enhances brute force strategies by predicting common password patterns. Hamza and Al-Janabi (2024) demonstrate how classifiers such as Random Forest and LSTM can detect brute force attempts by analysing traffic patterns and identifying anomalous login behaviour. In the first demonstration, the attacker employed only a basic dictionary attack, bypassing more advanced predictive analytics. However, integrating machine learning could allow attackers to prioritise common passwords, effectively narrowing the keyspace and increasing the probability of success.

Despite the increasing effectiveness of brute force attacks, several defensive measures can mitigate their impact. Account lockout mechanisms are a commonly recommended defense, temporarily disabling user accounts after a predefined number of failed login attempts (as shown in the defence demonstration). While such mechanisms can effectively prevent repetitive login attempts, they are not without limitations. Sowmya et al. (2012) caution that account lockout mechanisms can be weaponized to launch denial-of-service attacks, as attackers can intentionally trigger lockouts for legitimate users, effectively locking them out of their accounts. This highlights a critical trade-off between security and usability, as overly restrictive lockout policies may inadvertently disrupt legitimate access while failing to prevent distributed brute force attacks conducted from multiple IP addresses.

Rate limiting presents another potential mitigation strategy, restricting the number of login attempts allowed within a specified timeframe. In the demonstration, the absence of rate limiting enabled the attacker to rapidly cycle through multiple login attempts in less than six seconds, significantly increasing the likelihood of success. However, as Hamza and Al-Janabi (2024) note, rate limiting alone may not be sufficient to thwart more sophisticated attackers employing distributed denial-of-service techniques. Attackers can distribute their login attempts across multiple IP addresses, effectively bypassing rate limits while maintaining a steady attack cadence. This underscores the importance of integrating rate limiting with other defensive measures, such as CAPTCHA systems or multi-factor authentication (MFA), to provide layered security.

The implementation of CAPTCHA systems can effectively thwart automated brute force attacks by requiring users to complete tasks that are difficult for bots to execute, such as identifying images or solving puzzles. However, as Hamza and Al-Janabi (2024) point out, advanced bots equipped with machine learning capabilities can bypass basic CAPTCHA systems, effectively rendering them ineffective against more sophisticated adversaries. This limitation suggests that CAPTCHA systems should be employed in conjunction with other security measures to provide comprehensive protection against automated attacks.

Jo and Yoon (2015) propose a countermeasure against brute force attacks through the use of structural code schemes and honey encryption. Unlike conventional encryption methods, which return either the correct plaintext or an error message upon decryption, honey encryption generates plausible but incorrect plaintexts when an invalid decryption key is used. This technique effectively conceals the true plaintext by producing syntactically and semantically valid but false plaintexts, thereby confusing the attacker and increasing the difficulty of identifying a successful decryption attempt. The structural code scheme further refines this concept by employing probabilistic language models to generate false plaintexts that adhere to natural language patterns, effectively masking the true content while maintaining plausible readability. While such countermeasures can significantly increase the

complexity of brute force attacks, they are computationally intensive and may not be feasible for systems with limited processing power.

4.2 Session Fixation

The session fixation attack demonstration reveals critical weaknesses in session management, revealing how attackers can manipulate session identifiers to gain unauthorised access. The attack involved three primary stages: session setup, session fixation, and session entrance (Kolsek, 2002). By setting a predetermined session ID and coercing the user to adopt it, the attacker bypassed authentication, as the session was established before login, eliminating the need to acquire the session ID post-login (Kolsek, 2002).

The attack demonstration exploited the system's permissive session management, which accepted arbitrary session IDs, aligning with Kolsek's (2002) findings that systems allowing user-specified session IDs are particularly vulnerable to fixation attacks. Nathani and Adi (2012) observed similar vulnerabilities, reporting that 48% of examined websites failed to regenerate session IDs after login, increasing susceptibility to attacks through static or predictable session IDs. Schrank et al. (2012) also noted that session fixation remains under-addressed in web security, often stemming from a mismatch between session management responsibilities across application layers.

Kolsek (2002) categorises session fixation techniques into three types based on how session IDs are conveyed: URL arguments, hidden form fields, and cookies. The attack demonstration employed the cookie method, embedding the attacker's session ID within a crafted cookie. Cookie-based methods remain more covert than its counterparts, particularly when attackers exploit cross-site scripting (XSS) to implant malicious cookies without user awareness.

The defence demonstration implemented session ID regeneration post-login to mitigate fixation attacks. However, Nathani and Adi (2012) emphasised that session regeneration is ineffective if the initial session ID persists or if the new ID follows predictable patterns, allowing attackers to maintain control despite regeneration. Kolsek (2002) further cautions that even systems using HTTPS to protect session IDs from interception can remain vulnerable if session IDs are not regenerated or invalidated upon login, leaving systems susceptible to fixation attacks despite encryption.

Session fixation extends beyond initial login, enabling attackers to maintain control over session IDs throughout multiple interactions, facilitating ongoing monitoring or impersonation (Schrank et al., 2012). Thus, effective defences require comprehensive implementation of session ID regeneration, unpredictable session ID sequences, and layered security measures beyond initial authentication.

5.0 Remediation strategies

5.1 Brute force attack

5.1.1 Multi-Layered Authentication

Implementing multi-layered authentication using OTPs, CAPTCHA, and IP-based controls effectively mitigates brute-force attacks by adding complexity and unpredictability to the authentication process (Sowmya et al., 2012). Additionally, honey encryption and structural code schemes further enhance security by generating false plaintexts for incorrect decryption attempts, thereby complicating brute-force efforts (Jo & Yoon, 2015).

Implementation strategies:

- Implement OTP-based authentication using Okta or Azure AD for all critical access points.
- Integrate Google reCAPTCHA v3 to mitigate automated brute-force attacks.
- Enforce IP-based authentication using AWS Security Groups or Azure Network Security Groups.
- Deploy honey encryption for sensitive data using GenoGuard.

5.1.2 Network-Based Intrusion Detection and Monitoring Systems

Network-based monitoring through IPFIX effectively identifies brute-force attack patterns by analysing packet payload distributions (Hofstede et al., 2017). Furthermore, integrating GenoGuard provides additional protection by generating misleading plaintexts during brute-force decryption attempts, thereby confusing attackers and safeguarding sensitive data (Huang et al., 2015).

Implementation strategies:

- Deploy Cisco NetFlow or SolarWinds NPM to monitor and analyse flow data for anomalous patterns.
- Implement Splunk or Darktrace for AI-driven analysis of network traffic to detect potential brute-force attempts.
- Integrate GenoGuard to provide honey encryption for sensitive datasets.

5.1.3 Hash-Based Mutual Authentication and Secure Communication Protocols

Hash-based mutual authentication secures systems against brute-force and replay attacks, particularly for low-power devices like RFID tags (Cho, Yeo & Kim, 2011). Moreover, integrating real-time anomaly detection further strengthens this approach by monitoring for suspicious activity while maintaining secure communications (Verma, Dhanda & Nagar, 2021).

Implementation strategies:

- Apply SHA-256 hashing algorithms for password storage and transmission using AWS Cognito or Azure Key Vault.
- Implement RFID-based authentication using HID Global's RFID Access Control with hash-based protocols.

5.2 Session Fixation

5.2.1 Session ID regeneration:

According to Selenius (2021), generating a new session identifier after a successful login is a crucial way to prevent session fixation.

Implementation Strategy:

- Instead of authenticating the existing session ID, the web application needs to provide the user with a new authentication session ID.
- To do that: Implement the `session_regenerate_id(true)` function after every successful login to invalidate previous session IDs.
- Apply session regeneration at critical checkpoints, such as after password changes or account updates

5.2.2 Secure cookie functionality:

To secure the session cookies, utilising `HttpOnly` and `Secure` functionality ensures the cookie to be secured as it cannot be easily accessed through scripts or transmitted over unauthorised connections (Selenius, 2021).

Implementation Strategy:

- Set the `HttpOnly` flag to restrict JavaScript access to session cookies, preventing client-side script exploitation.
- Apply the `Secure` flag to ensure cookies are only transmitted over `HTTPS` connections, reducing the risk of interception.
- Implement the `SameSite` attribute to prevent cross-site scripting (XSS) attacks by limiting cookie transmission to the originating domain.

5.2.3 Session ID expiry:

Configuring the session ID timeout reduce the duration of active sessions, minimising the opportunity for attackers to hijack an existing session (Kolsek, 2002).

Implementation Strategy:

- Define session timeout durations based on user roles and application sensitivity (e.g., 15 minutes for standard users, 5 minutes for administrative access).
- Implement automatic session termination and prompt re-authentication after timeout to invalidate session IDs.

6.0 Conclusion

This report discussed the vulnerabilities associated with identification and authentication with attack and defend demonstrations of brute force attack and session fixation. The conducted scenarios offered insights into the exploitation of vulnerabilities and the technical measures necessary for their mitigation. The brute force attack illustrated the weakness caused by weak password strength and absence account lockout mechanisms. Whereas, the session fixation attack demonstrated how static session identifiers can be exploited for unauthorised access.

To reduce the risk of the vulnerabilities, defensive strategies such as iptables firewalls, altering default ports and regenerating session IDs. However, these measures should not be used alone as it does not guarantee complete protection as attacks can still discover alternative ways to bypass them. Therefore, these countermeasures should be layered and continuously monitored.

Ultimately, authentication security demands an active and multi-layered approach. Organisations have to constantly update and enhance their systems by integrating technical protections as well as behavioral monitoring. Through conducting such strategies, they can effectively minimise the possibility of exploitation and enhance their overall cybersecurity posture.

7.0 References

1. Cho, J. S., Yeo, S. S., & Kim, S. K. (2011). Securing against brute-force attack: A hash-based RFID mutual authentication protocol using a secret value. *Computer Communications*, 34(3), 391-397. doi: [10.1016/j.comcom.2010.02.029](https://doi.org/10.1016/j.comcom.2010.02.029)
2. Hamza, A. A., & Al-Janabi, R. J. (2024). Detecting Brute Force Attacks Using Machine Learning. *B/I/O Web of Conferences*, 97, 00045. <https://doi.org/10.1051/bioconf/20249700045>.
3. Hofstede, R., Jonker, M., Sperotto, A., & Pras, A. (2017). Flow-based web application brute-force attack and compromise detection. *Journal of Network and Systems Management*, 25(4), 735-758. doi: 10.1007/s10922-017-9421-4s10922-017-9421-4.
4. Huang, Z., Ayday, E., Fellay, J., Hubaux, J. P., & Juels, A. (2015). GenoGuard: Protecting genomic data against brute-force attacks. *IEEE Symposium on Security and Privacy*, 2015, 447-461. <https://doi.org/10.1109/SP.2015.34>
5. Jo, H.-J., & Yoon, J. W. (2015). A New Countermeasure against Brute-Force Attacks. *International Journal of Distributed Sensor Networks*, 2015. <https://doi.org/10.1155/2015/406915>.
6. Kolsek, M. (2002). Session Fixation Vulnerability in Web-Based Applications. ACROS Security. Retrieved from <https://www.chablop.eu/files/attaqueFixation.pdf>.
7. Mohan, R., & Srivastava, S. (2010). Brute Force Attack Prevention and Mitigation Techniques. *Computer Communications*, 34(4), 487–497.
8. Nathani, R., & Adi, J. (2012). An Analysis of Web Application Session Fixation Vulnerabilities. *International Journal of Computer Applications*, 47(21), 27-31. <https://doi.org/10.5120/7310-2299>
9. Schrank, P., Kamkar, S., & Martin, E. (2012). An Analysis of Session Management Vulnerabilities in Web Applications. *Journal of Web Security Research*, 6(2), 99-112. <https://doi.org/10.1016/j.websec.2012.04.005>
10. Selenius, T. (2021). Session Fixation Attacks and Prevention. [Www.appsecmonkey.com](http://www.appsecmonkey.com). <https://www.appsecmonkey.com/blog/session-fixation>
11. Sowmya, G., Jamuna, D., & Krishna Reddy, M. V. (2012). Blocking of Brute Force Attack. *International Journal of Engineering Research & Technology (IJERT)*, 1(6), 1-4.
12. Verma, R., Dhanda, N., & Nagar, V. (2021). Enhancing security with in-depth analysis of brute-force attack on secure hashing algorithms. *Lecture Notes in Networks and Systems*, 376, 513-522. https://doi.org/10.1007/978-981-16-8826-3_58