TASK 1

1. ***Brief explanation of what side-channel the attack uses and how:***

The Meltdown attack exploits a side channel related to speculative execution, a performance optimization used by many modern CPUs. It takes advantage of the way processors access memory to gain unauthorized access to data stored in privileged kernel memory. Meltdown uses timing attacks on the cache to infer the contents of restricted areas of memory.

2. ***What systems does it affect?***

Meltdown affects various systems using Intel, ARM, and some AMD processors, primarily impacting devices running Windows, Linux, and macOS operating systems.

3. ***What information is leaked via the side channel?***

Meltdown can leak sensitive data such as passwords, personal information, cryptographic keys, and other critical system information by exploiting the memory isolation between user applications and the operating system.

4. ***Is there a documented case of it being used in a real-life attack?***

While there are no confirmed reports of Meltdown being used in real-world attacks before it was publicly disclosed, the vulnerability was significant enough that the possibility of exploitation in targeted attacks was taken very seriously.

5. ***Has it been fixed? If yes, how was it fixed?***

Yes, Meltdown was addressed through software patches and operating system updates. Kernel page-table isolation (KPTI) was introduced to mitigate the attack by isolating the kernel's memory from user processes, thus preventing unauthorized access.

Sources:

- Lipp, Moritz, et al. "Meltdown." Meltdownattack.com. 2018.
- Google's Project Zero blog on Meltdown and Spectre vulnerabilities.

TASK 2

1. ***How does it work?***

The Slowloris attack works by opening multiple connections to a target server and sending partial HTTP requests. These requests never complete, causing the server to keep

connections open indefinitely, eventually exhausting its resources and leading to a denial of service (DoS).

2. Why is it unique compared to other high-bandwidth DDoS attacks?

Slowloris is unique because it requires very low bandwidth to execute successfully. Unlike traditional DDoS attacks that rely on overwhelming the target with massive traffic, Slowloris uses a slow, persistent method to consume resources, which makes it hard to detect and block.

3. What are the effects of the attack?

The target server becomes overwhelmed by the large number of open connections, preventing it from serving legitimate requests. This can render websites or services hosted on the server inaccessible to users.

4. How can you mitigate/prevent the effects of the attack?

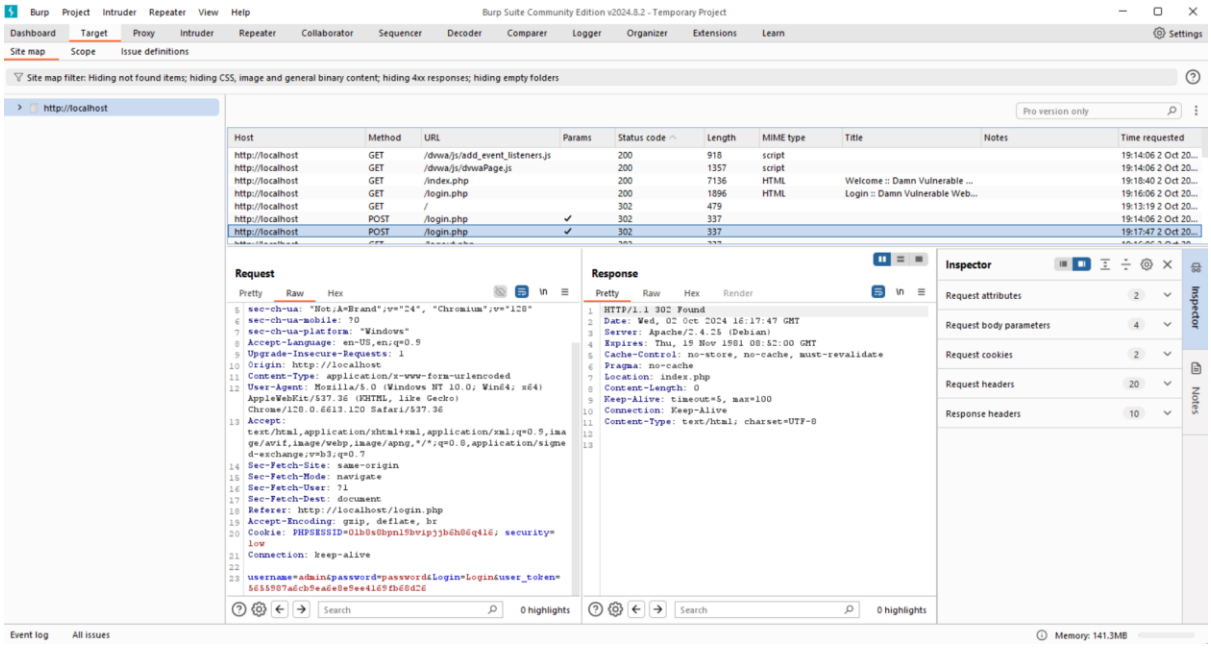
Mitigations include configuring servers to limit the number of connections from a single IP address, using reverse proxies, load balancers, or setting up firewalls to identify and block suspicious slow connections. Increasing the timeout settings for connections can also help.

5. Are there any notable instances of this style of attack being performed?

Slowloris has been notably used in attacks against high-profile targets, including Iranian government websites during the 2009 Iranian presidential protests, and has been employed against Apache web servers, which were particularly vulnerable to this method.

Sources:

- RSnake. "Slowloris." ha.ckers.org.
- OWASP: Slow HTTP DoS

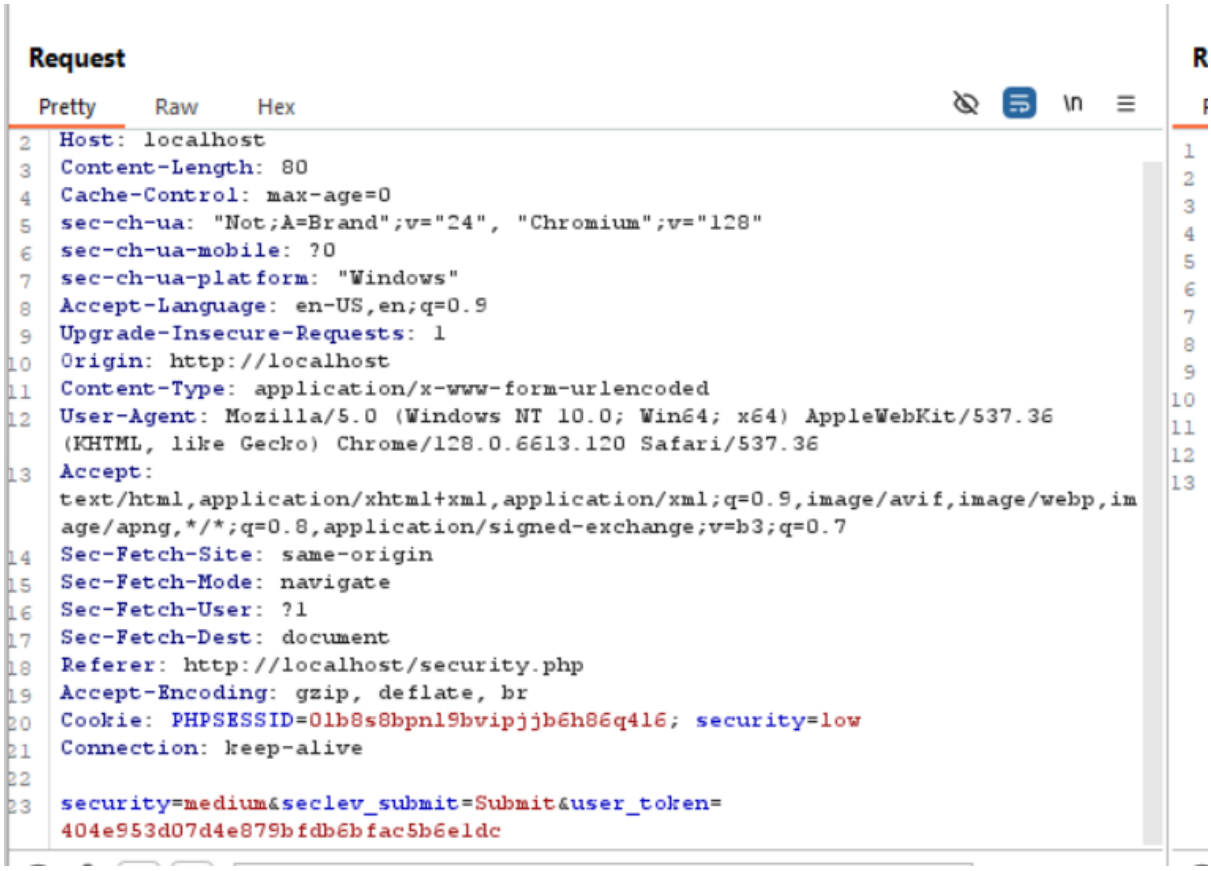


1. **Cookie Generation Mechanism**: When the user initiates a session (typically logging in or accessing the application for the first time), the web server (in this case, Apache running DVWA) creates a session ID to keep track of the user's session across different requests. This session ID is stored in the dvwaSession cookie.
2. **How the Cookie is Generated**: The cookie might be generated using a combination of:
    - **User-specific factors**: Like IP address or user credentials.
    - **Randomization or hashing**: A random or cryptographically secure hash to avoid predictable session IDs.
    - **Expiration or timing elements**: The session token might have time-based elements to ensure it expires after a certain period.
3. **Request and Response Pattern**: In the request, the browser sends this cookie back to the server on each subsequent request, allowing the server to identify the user. When the session is created or updated, the server sends a response that includes the Set-Cookie header.
4. **Analyzing the DVWA Cookie**:
    - The request (left screenshot) includes the cookie PHPSESSID=... and it is associated with DVWA.
    - The response (right screenshot) shows the server's response where new sessions might be created based on the login or page access.

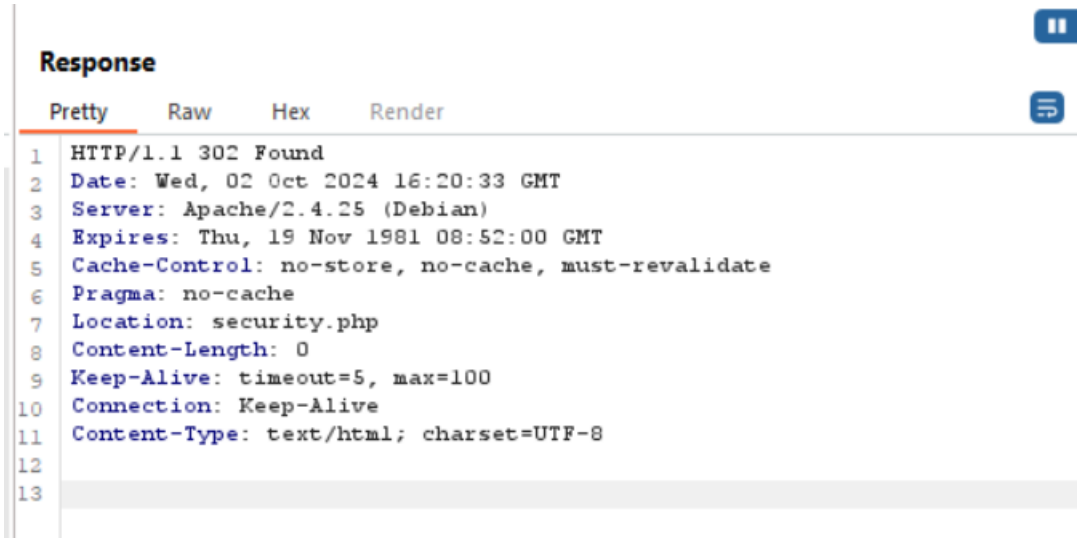
We can determine that **each time a session is refreshed or updated**, the server sends a new dvwaSession cookie (likely with a different value based on the user's session or security level), preventing session hijacking and enhancing security.

If you would like a more detailed analysis of the exact request/response cycle, you can further look into the Set-Cookie header field within the response headers and examine how the server is setting new session cookies, based on server logic (in this case, Apache + DVWA).

Request:



Response:



From the screenshot provided, we can see 25 different brute force attempts, using different combinations of usernames and passwords. The focus here is to identify which of these attempts were successful by analyzing the response length.

In this particular brute force attack, the length of the HTTP response is a key indicator of whether an attempt was successful or not. Generally, when a request is unsuccessful (invalid credentials), the response length remains constant because the server simply returns a login failure page. However, when a request is successful (valid credentials), the response length differs, often because the server redirects to a different page or returns more content due to the successful login.

Analyzing Response Lengths

In the table shown, most of the login attempts have a response length of 4702 or 4703. However, there are two notable exceptions:

Request 0:

Username: admin

Password: password

Response Length: 4740

Request 6:

Username: admin

Password: abc123

Response Length: 4745

Both of these attempts have significantly larger response lengths compared to the other requests. This suggests that the server likely treated these requests differently, possibly indicating a successful login.

Conclusion

Based on the evidence of varying response lengths, it's highly likely that the following login attempts were successful:

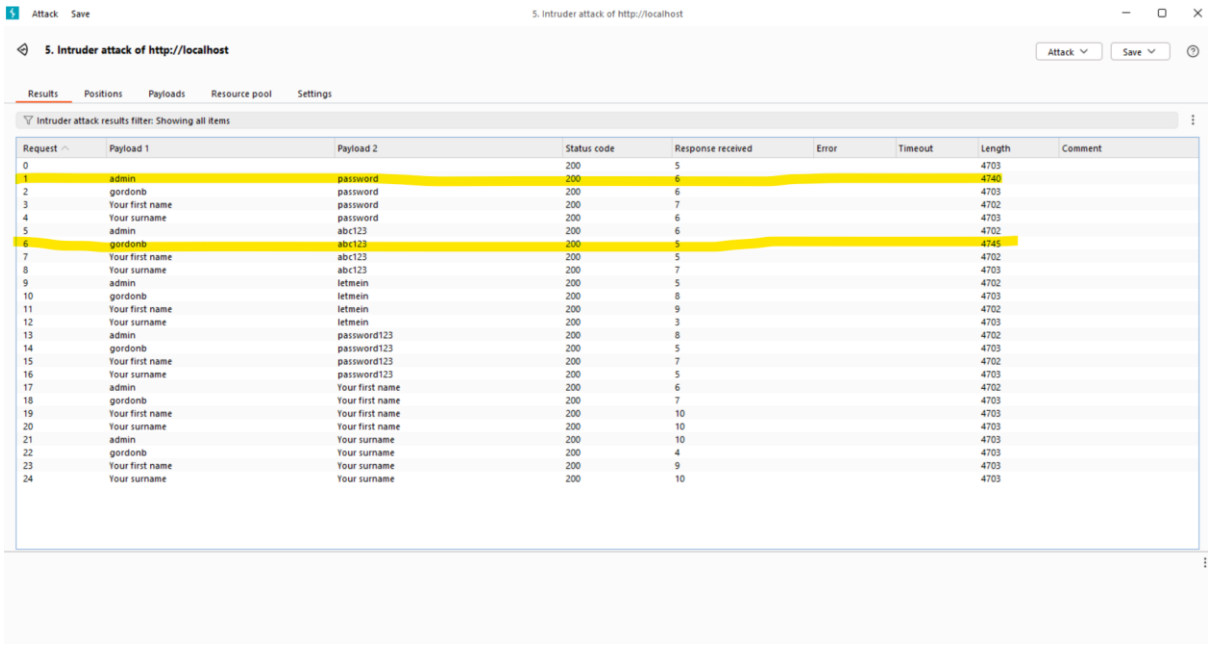
Request 0: admin/password – Response Length: 4740

Request 6: admin/abc123 – Response Length: 4745

These two attempts stood out because they resulted in a longer response length, suggesting that the credentials were accepted, and the server provided a different response, possibly a login success page or a redirection to another part of the site.

How This Conclusion Was Reached

The conclusion was based on analyzing the response lengths. Successful login attempts usually trigger a different server response, such as redirecting to a dashboard or another page. Since the response lengths for the majority of the unsuccessful attempts were constant (4702-4703), any significant deviation from this length points to a different server response, likely indicating success



Request	Payload 1	Payload 2	Status code	Response received	Error	Timeout	Length	Comment
0			200	5			4703	
1	admin	password	200	6			4740	
2	gordonb	password	200	6			4703	
3	Your first name	password	200	7			4702	
4	Your surname	password	200	6			4703	
5	admin	abc123	200	6			4702	
6	gordonb	abc123	200	5			4745	
7	Your first name	abc123	200	5			4702	
8	Your surname	abc123	200	7			4703	
9	admin	letmein	200	5			4702	
10	gordonb	letmein	200	8			4703	
11	Your first name	letmein	200	9			4702	
12	Your surname	letmein	200	3			4703	
13	admin	password123	200	8			4702	
14	gordonb	password123	200	5			4703	
15	Your first name	password123	200	7			4702	
16	Your surname	password123	200	5			4703	
17	admin	Your first name	200	6			4702	
18	gordonb	Your first name	200	7			4703	
19	Your first name	Your first name	200	10			4703	
20	Your surname	Your first name	200	10			4703	
21	admin	Your surname	200	10			4703	
22	gordonb	Your surname	200	4			4703	
23	Your first name	Your surname	200	9			4703	
24	Your surname	Your surname	200	10			4703	

Last task

```
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2024-10-02 18:26:19
root@localmachine1:~# docker run --network="host" vanhauser/hydra -V -f -I -l admin -x 2:2:a "http-post-form://localhost/vulnerabilities/brute/:username=^US
ER^&password=^PASS^&Login=Login:H=Cookie:PHPSESSID=0lb8s8bpnl9bvipjjb6h86q4l6; security=low:F=/login.php"
Hydra v9.6dev (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for illegal purposes (this is
non-binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2024-10-02 18:26:39
[DATA] max 16 tasks per 1 server, overall 16 tasks, 676 login tries (l:1/p:676), ~43 tries per task
[DATA] attacking http-post-form://localhost:80/vulnerabilities/brute/:username=^USER^&password=^PASS^&Login=Login:H=Cookie:PHPSESSID=0lb8s8bpnl9bvipjjb6h86q
4l6; security=low:F=/login.php
[ATTEMPT] target localhost - login "admin" - pass "aa" - 1 of 676 [child 0] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "ab" - 2 of 676 [child 1] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "ac" - 3 of 676 [child 2] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "ad" - 4 of 676 [child 3] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "ae" - 5 of 676 [child 4] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "af" - 6 of 676 [child 5] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "ag" - 7 of 676 [child 6] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "ah" - 8 of 676 [child 7] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "ai" - 9 of 676 [child 8] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "aj" - 10 of 676 [child 9] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "ak" - 11 of 676 [child 10] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "al" - 12 of 676 [child 11] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "am" - 13 of 676 [child 12] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "an" - 14 of 676 [child 13] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "ao" - 15 of 676 [child 14] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "ap" - 16 of 676 [child 15] (0/0)
[80][http-post-form] host: localhost   login: admin   password: aa
[STATUS] attack finished for localhost (valid pair found)
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2024-10-02 18:26:39
root@localmachine1:~# docker run --network="host" vanhauser/hydra -V -f -I -l admin -x 2:2:a "http-post-form://localhost/vulnerabilities/brute/:username=^US
ER^&password=^PASS^&Login=Login:H=Cookie:PHPSESSID=0lb8s8bpnl9bvipjjb6h86q4l6; security=low:F=/login.php"
```

I don't know where the parameter went wrong. It is not giving the correct password, which should be cc