EventID 115 SOC165 Possible SQL Injection Payload Detected

This alert is of type “Web Attack”, and the name of the SIEM rule which created it suggests that there could be the presence of a SQL Injection attack.

The alert metadata indicates that it has been triggered because “OR 1 = 1” payload has been identified on the request.

- Alert creation time: 2022, February (Friday) 25th at 11:34h
- Server hostname: WebServer1001
- IP Addresses:
  - Source: 167.99.169.17
    - Owned by “DigitalOcean, LLC”
    - Located in: Santa Clara, California, U.S.
  - Destination: 172.16.17.18
    - Windows Server 2019
    - Owned by “webadmin” user with last login at: 2022, Feb 10th at 23:12h
- HTTP Method: GET
- Requested resource:
  - “https://172.16.17.18/search/?q=%22%20OR%201%20%3D%201%20--%20-”

- User-Agent:
    - Mozilla/5.0 (Windows NT 6.1; WOW64; rv:40.0) Gecko/20100101 Firefox/40.1
        - WOW64 is a Windows subsystem that allows 32-bit applications to run on a 64-bit OS. Translates 32-bit API calls into 64-bit equivalents
        - “Windows NT 6.1” is a version number for Windows 7
        - “rv:40.0” stands for “Release Version 40.0”, indicating that the browser is Mozilla Firefox 40.0

About the Requested Resource

By looking at the request URL, I noticed special characters included in a query param:

- “?q=%22%20OR%201%20%3D%201%20--%20-”

The percentage characters followed by hexadecimal values are URL Encoded symbols:

- “%20” → Space
- “%22” → Double quote (”)
- “%3D” → Equal sign (=)

URLs can only contain certain characters. Reserved characters and other unsafe characters can’t directly be included in URLs, because it would break them.

URL Encoding converts special characters into a “%” followed by two hexadecimal digits.

By knowing what URL Encoding is, I just need to translate the special symbols to understand the request. The query param of the request translates to:

- `?q=" OR 1 = 1 -- -`

This is a SQL Injection payload, because:

- A single double quote breaks the SQL sentence (two double quotes would make more sense):
  - SELECT * FROM users WHERE user = <query_param>
  - If the SQL sentence is expecting an username from a login form, and the URL decoded string is expected to be placed there programmatically, the SQL sentence becomes:
    - SELECT * FROM users WHERE users.username = " OR 1 = 1 -- -
- The "OR 1 = 1" expression tricks the database to execute the "SELECT" operation, because it always resolves to true
  - Its objective is to leak sensitive data to the attacker
- The "--" is a SQL comment, followed by a dummy character '-' to ensure the comment is valid
  - The attacker includes this to trick the database to ignore the part of the SQL sentence that was originally set after the comment
  - If there are more SQL directives after the SQL injection payload they are ignored, and if the attack works, the attacker is able to modify the SQL sentence at will

The request is definitely suspicious. I'm going to prove if it's a malicious request and if it was successful or not.

Threat Intelligence

I've looked for source IP Address "167.99.169.17" in ipinfo.io, identifying its location in Santa Clara, California, U.S., owned by DigitalOcean LLC.

The same IP Address was reported 14.982 times on abuseipdb.com, categorized as: hacking, SQL injection, brute-force, ssh, etc. VirusTotal detects malicious activity on this IP aswell.

Both VirusTotal and abuseipdb.com analyses are recent.

EDR

I'm going to look for "WebServer1001" (172.16.17.18) in Endpoint Security of LetsDefend, in order to get more information of the web server.

The computer system is a Windows Server 2019 owned by "webadmin" user, with last login in 2022 February (Thursday) 10th at 23:12h.

There's a discrepancy between the last login time of "webadmin" and the alert creation time, which indicates that the owning user has not logged in for 15 days before the alert. This could indicate:

- The machine was unused (e.g. employee on leave, or abandoned system)
- The user account was compromised earlier, and the attacker disabled logs to hide activity
- The machine could have been compromised after the last legitimate login
- The attacker could have scanned the network for an inactive system, in order to have more chances of attacking without being noticed

Log Management

In Log Management section of LetsDefend, I've looked for the URL of the requested resource:

- "https://172.16.17.18/search/?q=%22%20OR%201%20%3D%201%20--%20-"

There's a firewall log that matches the alert creation time, which shows a connection from 167.99.169.17:48575 to 172.16.17.18:443.

This log shows that the request was allowed by the firewall and that the request had a response status 500 (internal server error), with a response size of 948B.

This indicates that the DB data was not leaked, but the attacker could obtain valuable information from the application behavior anyway, like error messages.

I've done another search to look for all logs of network connections targetting the "WebServer1001" host at the time the alert was created, and I've found the following:

- 4 requests exist, one day before the alert creation date
  - The 4 requests have the exact same timestamp:
    - 2022, February (Thursday) 24th at 22:30h
    - This indicates the requests were automated, probably for reconnaissance purposes
  - All sources have the exact same User-Agent as the one from the alert being investigated
    - This indicates a high probability of relationship with the requests made the next day

    - All come from IP addresses under “176.33.208.x” private IP range, starting from IP “176.33.208.182” to “176.33.208.185”
        - This suggests automated scanning
        - ipinfo.io shows that this IP range (176.33.192.0/19) is located on Istanbul, Turkey
        - Both abuseipdb.com and VirusTotal show no traces of malicious activity related to these IPs
        - The attacker may have used this sources to perform previous reconnaissance from inconspicuous IPs to determine possible targets
        - The attacker may spread traffic across IPs intentionally to evade rate-limiting
        - These IPs could belong to a proxy used by the attacker
    - All the requests are requesting the "https://172.16.17.18/” resource
        - This may have been done to verify availability of the server, prior to the attack
- 6 requests exist, the day the alert was created, including the one that triggered the alert
    - All requests come from the same IP Address
    - The first request was made against "https://172.16.17.18/” resource, very likely to confirm that the server is running
    - 5 out of 6 requests contain different SQL injection payloads
        - All requests with SQL injection payloads are abusing the same “q” query param
        - All requests with SQL injection payloads return status code 500 (internal server error) and the same response size of 948B
            - The same response size across requests proves that the attack didn’t leak information to the attacker and thus it was not successful

What if it’s a Planned Test

The User-Agent on the requests would display the name of attack simulation software if it was a penetration testing from known actors.

To make sure, I’ve looked on Email Security section of LetsDefend, for emails sent to the owner of the machine and emails sent on dates close to the attack. No emails related to the incident are present.

Artifacts

I’ve attached the following artifacts to this investigation:

- “%22%20OR%201%20%3D%201%20--%20-” SQL Injection Payload
- “167.99.169.17” IP Address of the attacker

Since the attack is proved to be unsuccessful, tier 2 escalation is not needed.

Conclusion

Evidence points that the attacker attempted a SQL Injection attack after performing reconnaissance from a proxy.

There are no emails that relate this event to scheduled tests, and the nature of the requests support this aswell.

This attack was not successful because the server returned status code 500 and the same response size in all tries.