CSYE6225
Spring 2019

Web Application
Firewall Implementations
&
Penetration Testing

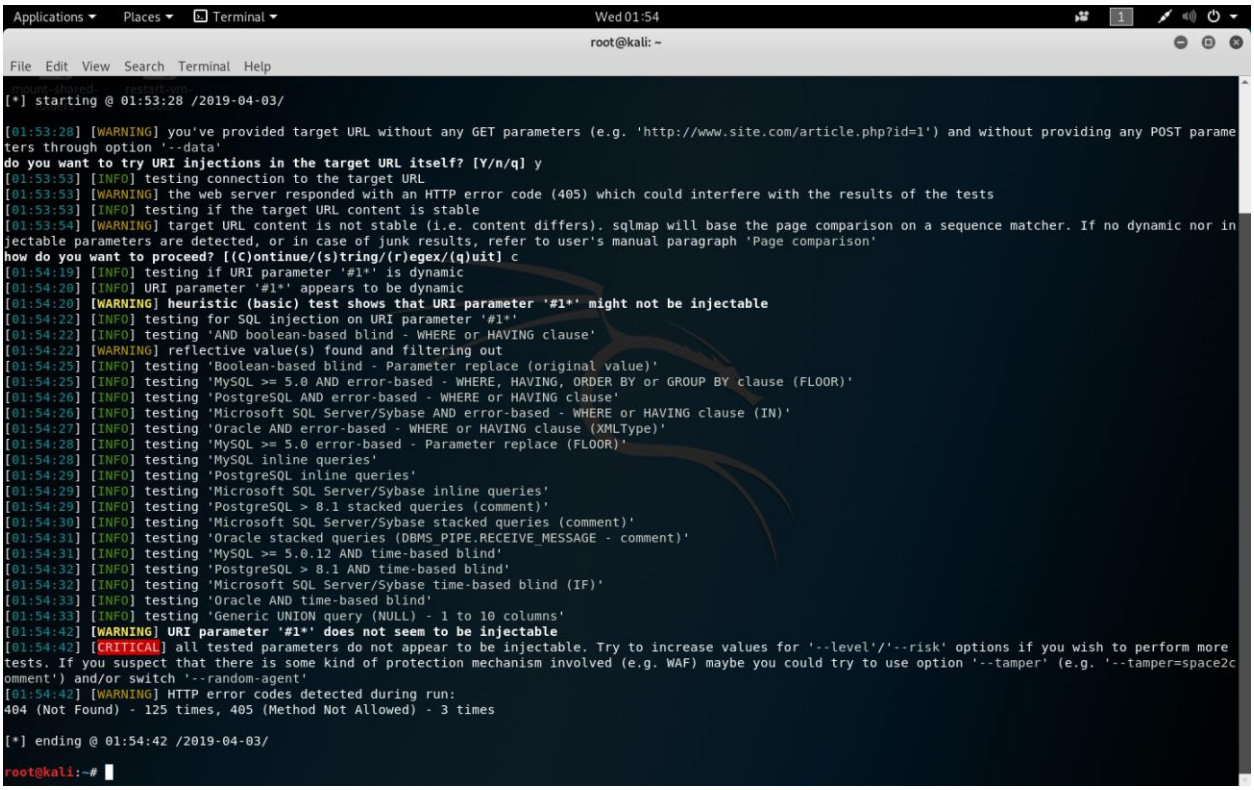
Yunan Shao (1818832)
Haowen Chang (1497263)
Zeyu Huang (1407537)
Andong Wang (1407537)

Selected Attack Vectors in Red

1. Sql Injection - Implemented

Implementation: Added ACL rules to block matched sql injection strings.

Attack Vector: By inserting sql code into requests to the interpreter, an attacker can alter the intent of the requests and cause unexpected actions



```
Applications ▾  Places ▾  Terminal ▾        Wed 01:54
root@kali: ~
File  Edit  View  Search  Terminal  Help
[*] starting @ 01:53:28 /2019-04-03/

[01:53:28] [WARNING] you've provided target URL without any GET parameters (e.g. 'http://www.site.com/article.php?id=1') and without providing any POST parame
ters through option '--data'
do you want to try URI injections in the target URL itself? [Y/n/q] y
[01:53:53] [INFO] testing connection to the target URL
[01:53:53] [WARNING] the web server responded with an HTTP error code (405) which could interfere with the results of the tests
[01:53:53] [INFO] testing if the target URL content is stable
[01:53:54] [WARNING] target URL content is not stable (i.e. content differs). sqlmap will base the page comparison on a sequence matcher. If no dynamic nor in
jectable parameters are detected, or in case of junk results, refer to user's manual paragraph 'Page comparison'
how do you want to proceed? [(C)ontinue/(s)tring/(r)egex/(q)uit] c
[01:54:19] [INFO] testing if URI parameter '#1*' is dynamic
[01:54:20] [INFO] URI parameter '#1*' appears to be dynamic
[01:54:20] [WARNING] heuristic (basic) test shows that URI parameter '#1*' might not be injectable
[01:54:22] [INFO] testing for SQL injection on URI parameter '#1*'
[01:54:22] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[01:54:22] [WARNING] reflective value(s) found and filtering out
[01:54:25] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[01:54:25] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)'
[01:54:26] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[01:54:26] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[01:54:27] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[01:54:28] [INFO] testing 'MySQL >= 5.0 error-based - Parameter replace (FLOOR)'
[01:54:28] [INFO] testing 'MySQL inline queries'
[01:54:29] [INFO] testing 'PostgreSQL inline queries'
[01:54:29] [INFO] testing 'Microsoft SQL Server/Sybase inline queries'
[01:54:29] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[01:54:30] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[01:54:31] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[01:54:31] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind'
[01:54:32] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[01:54:32] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[01:54:33] [INFO] testing 'Oracle AND time-based blind'
[01:54:33] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[01:54:42] [WARNING] URI parameter '#1*' does not seem to be injectable
[01:54:42] [CRITICAL] all tested parameters do not appear to be injectable. Try to increase values for '--level'/'--risk' options if you wish to perform more
tests. If you suspect that there is some kind of protection mechanism involved (e.g. WAF) maybe you could try to use option '--tamper' (e.g. '--tamper=space2c
omment') and/or switch '--random-agent'
[01:54:42] [WARNING] HTTP error codes detected during run:
404 (Not Found) - 125 times, 405 (Method Not Allowed) - 3 times

[*] ending @ 01:54:42 /2019-04-03/

root@kali:~#
```

Result: All tested parameters do not appear to be injectable at level 3 with WAF or without WAF.

Why choose this vector: Attackers can use any source of data as an injection vector, even without login and all types of users and these vulnerabilities are very often in SQL. Injection can result in huge data leak or even losing data in the database.

2. Broken Authentication and Session Management – Not Implemented

Reason why no implement: Our Restful API is not token or session based. If there are account credentials are exposed, we can use other methods such as asking for changing password to those exposed account.

3. XSS - Implemented

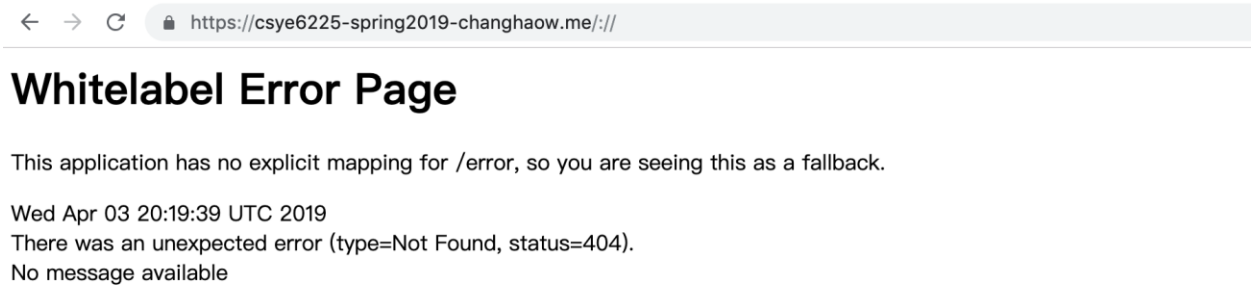
Implementation: ACL rules to block matched xss attack strings.

4. Broken Access Control - Implemented

Implementation: ACL rules to block urls that have strings related to path which can be potential path traversal attack. And our web application checks for authentication internally for reach request.

Attack Vector: By guess direction or file and insert the location into an address, attacker can get access to file that is invisible to the users and may change or insert virus. Or guess the parameter in the url to get access to resource.

With WAF



Without WAF

Result: Although both web app didn't show anything but web app without WAF shows that the request can get into the server and do something but with WAF all the direction string is filtered.

Why choose this vector: Poorly designed access control can lead to exposure of unauthorized data, manipulation of internal web application state, path traversal, and file inclusion.

5.Security Misconfiguration – Implemented

Implementation: The ports and ingress rules between security groups are designed to block potential access to our server.

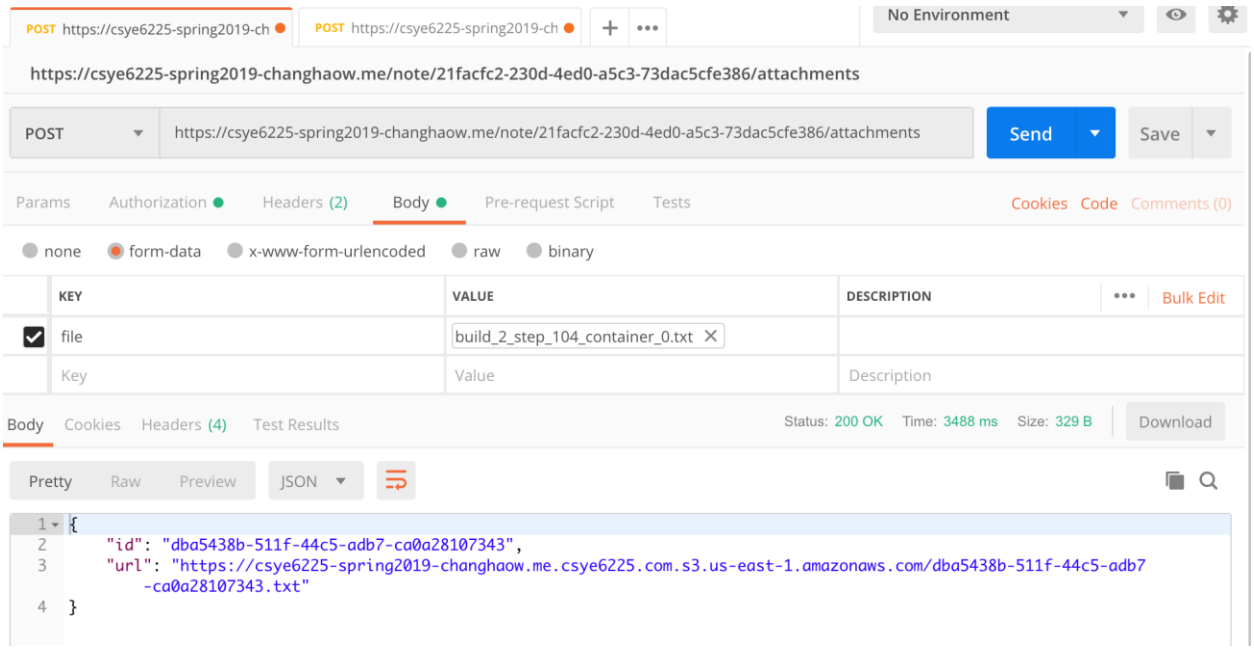
6. Sensitive Data Exposure - Implemented

Implementation: Only HTTPS requests are allowed with elastic load balancer. User credentials are encrypted using Bcrypt with unique salts.

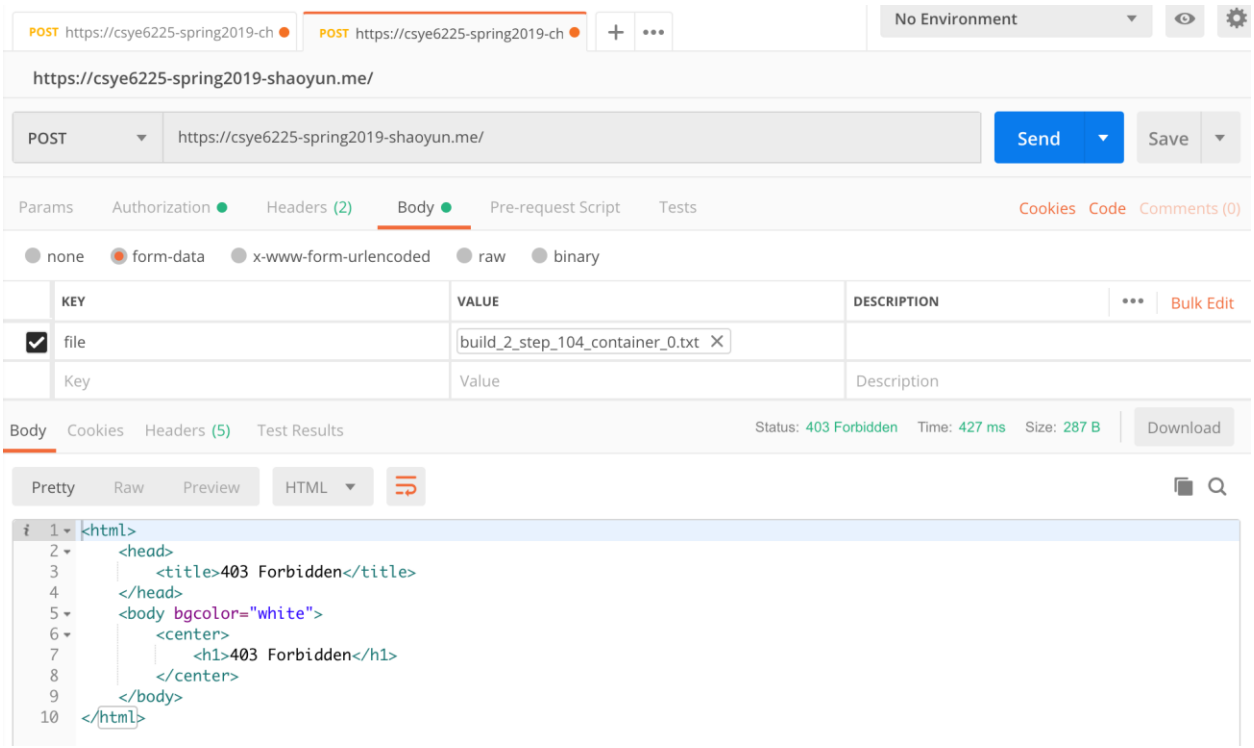
7. Insufficient Attack Protection - Implemented

Implementation: ACL rules to blocks requests that are larger than restricted file size.

Attack Vector: Attacker might send plenty of request that contains large file so that server need time to response then cause server slow or downtime like DDos. We added ACL rules that only accept 0.2mb size for uploading and tested with files that are larger than the restriction.



Without WAF



With WAF

8. Cross-site request forgery attack – Implemented

Implementation: ACL rule that blocks requests with tokens that are in certain size.

*Our application actually doesn't use token-based authentication. We still added one rule that is explained in the AWS WAF document for explanation purpose. Without the WAF check the head, requests with tokens in any sized token will be accepted. With WAF we can allow only requests with tokens in matched size.

9. Using Components with Known Vulnerabilities – Implemented

Implementation: ACL rules that block request with certain prefix of file extensions.

10. Under protected APIs - Implemented

Implementation: Many examples are in 1-9 and we can block requests with designed ACL rules such as string match or size restrictions to protect our application.

Citations:

https://d0.awsstatic.com/whitepapers/Security/aws-waf-owasp.pdf