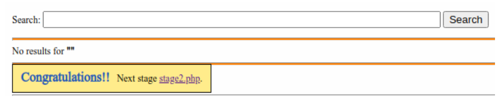
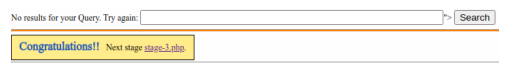
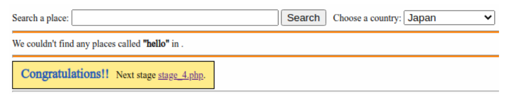
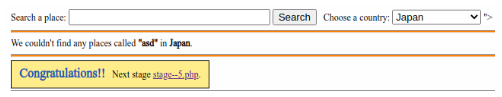
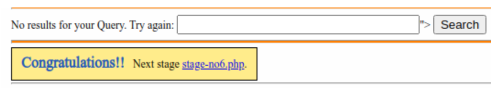
CSE345/545: Endsem

Mihir Chaturvedi (2019061)

Question 1

Part	XSS Code	HTML Element	Screenshot
a	`<script>alert(document.domain)</script>`	`<input type="text" name="p1" size="60" value="">`	 Search: Search No results for "" **Congratulations!!** Next stage stage2.php.
b	`"/><script>alert(document.domain)</script>`	`<input type="text" name="p1" size="50" value="">`	 No results for your Query. Try again: "> Search **Congratulations!!** Next stage stage-3.php.
c	`<script>alert(document.domain)</script>`	`<select name="p2"> <option>Japan</option> <option>Germany</option> <option>USA</option> <option>United Kingdom</option> </select>`	 Search a place: Search Choose a country: Japan We couldn't find any places called **"hello"** in . **Congratulations!!** Next stage stage_4.php.
d	`"/><script>alert(document.domain)</script>`	`<input type="hidden" name="p3" value="hackme">`	 Search a place: Search Choose a country: Japan "> We couldn't find any places called **"asd"** in **Japan**. **Congratulations!!** Next stage stage--5.php.
e	`"/><script>alert(document.domain)</script>`	`<input type="text" name="p1" maxlength="15" size="30" value="">`	 No results for your Query. Try again: "> Search **Congratulations!!** Next stage stage-no6.php.

Question 2

a.

1. Via this technique it is very simple to deny service to legitimate users.
    - If a malicious actor attempts to login into another user’s account and enters the incorrect password three consecutive times, the (victim) user’s account will be locked.
    - The victim user will now not be able to login into their own account even with a valid password.
2. Since both the attacker and the victim have a shared, common mechanism of accessing the account login, through the open internet, any form of access-control imposed on this shared mechanism will be applicable to all users of the shared mechanism.
    - These include all honest users and malicious attackers.
    - A more refined mechanism to align with the principle of least common mechanism would be to block access by IP address, user agent, device type, etc.

b. I agree with the statement that the principle of fail-safe defaults is a *justification* for the above technique.

- A fail-safe default state is defined as one wherein the system takes actions to bring the system down to a known, safe state.
    - This state might affect the availability of the system for the users, but does not jeopardize the security of the system.
- A possible attacker attempting to break into a user’s account can be considered as an intrusion into the system. If gone unchecked, the attacker has the ability to brute-force their way into the account and impact the security of the victim’s account.
- Temporarily blocking login attempts into the user’s account is an aggressive fail-safe default state.
    - The security of the system is not compromised - the attacker cannot attempt more logins.
    - The system however has become unavailable to the honest victim user.
        - At this step, they must contact the admins to restore their access.
- **However,**
    - This technique is poorly designed, as explained in part a., where the attacker and victim have a common access mechanism.
    - With better access mechanisms, it could have been possible to *only* disable the attacker's access to the account login, thereby making the account still available to the honest user.

Question 3

Part	Commandment
a	7: Thou shalt not use other people's computer resources without authorization or proper compensation
b	6: Thou shalt not copy or use proprietary software for which you have not paid (without permission)
c	10: Thou shalt always use a computer in ways that ensure consideration and respect for other humans
d	7: Thou shalt not use other people's computer resources without authorization or proper compensation
e	3: Thou shalt not snoop around in other people's computer files

Question 4

Security Risk	Handled?	Details
Broken Access Control	Yes	• Client-side access control is done by blocking requests to admin-only or seller-only routes if the user is not an admin or seller respectively. • Every API route is protected by proper access control through middlewares that perform checks that determine whether this request is properly authenticated AND authorized to access the route.
Cryptographic Failures	Yes	• Passwords are hashed and salted, and then stored in the database. We use the bcrypt hashing algorithm, which is performed by the bcrypt npm library. • Session IDs are long, randomly generated, expirable and not bruteforce-able or guessable.
Injection	Yes	• SQL injection is not possible because we are not using raw SQL queries in our application on the server end ○ We use the Prisma ORM that queries the database for us • XSS injection is not possible since React escapes HTML characters before appending them to the DOM ○ No unexpected elements are created • Input validation is done on the server side using Joi library
Insecure Design	Yes	• All requests with data are sent via a POST request • Option to allow user input via a virtual keyboard is offered • Strong and expirable OTPs are present before sensitive or dangerous actions. • Resource identifiers are long and unguessable.
Security Misconfiguration	Yes	• Nginx server was configured properly ○ Enable strong rate-limiting on the API and client ○ Disallow any direct directory listing ○ Detailed errors or entire call-stack and traces are not visible to the user on the production machine ○ Upgrade HTTP connections to HTTPS • All ports except HTTP and HTTPS ports are closed • Default configurations for MySQL database are not used - root/admin user password is very strong and cannot be accessed through the network.
Vulnerable and Outdated Components	Yes	• All npm packages are up-to-date ○ No package was found to have any security vulnerabilities relevant to our system • Using the latest version of Node, Nginx and MySQL • Vulnerability checks were performed to ensure system packages are up-to-date

Identification and Authentication Failures	Yes	• Strong rate-limiting is enabled to prevent easy bruteforce • Passwords are strong and required to have a mix of special/numerical/mixed-case characters of minimum length 12 • No default passwords are kept for any services that are deployed on the machine, such as any user-login or the database login • Passwords are hashed and salted before being stored • Session IDs are long, randomly generated, expirable and not bruteforce-able or guessable.
Software and Data Integrity Failures	Yes	• All npm packages are installed from the official npm package repository. • All apt packages are installed from Ubuntu's official package repository. • The CI/CD operations are not externally accessible or controllable. Package builds and deployments can only be done post-logging into the machine.
Security Logging and Monitoring Failures	Yes	• Application-side event logs are maintained that are accessible to the admin users ○ Contains info of all state-change operations made by any user on the website ○ Event logs can only be read, not created by the user, deleted or modified. ○ Error handling is performed by try-catching all API requests; showing appropriate error pages on the client ○ RESTful API error codes are sent when things fail. • Application errors and other logs are recorded by PM2 (our process manager for the API and client servers) ○ Any standard output/error logs are recorded ○ When the application crashes for whatever reason, PM2 automatically restarts the application • All HTTP requests are recorded by Nginx in their access logs ○ These include info about referrer, user agent, IP address, endpoint, status code, etc. ○ Errors on Nginx are recorded in their error logs
Server-Side Request Forgery	Yes	• Proper HSTS headers are applied • HTTPS is used to send and receive responses • CORS is used to enforce origin policies • Input validation is done both client- and server-side

Question 5

- **Example *for* Security through Obscurity (STO)**
    - Changing default ports for services can significantly reduce chances of malicious programs attempting to brute force into the application/network.
        - For example, changing the default SSH port from 22 to some other random port number between 1 and 65,535.
        - A general malicious program would only attempt to connect with port 22 and determine whether an SSH connection can be established. Most programs won't waste their time and computational power in order to scan *all* the thousands of ports - they will move on to other networks.
        - However, *just* having this won't cut it - the network should also enable a firewall that enforces only allowed IP addresses to connect to the network.
    - Removing server or server version information from response headers can help in making it harder for attackers to determine whether you are running vulnerable software or not.

- **Example *against* STO**
    - Attempting to obfuscate and garble variable names in bundled/production files of web application's JS code helps in reducing file size but not in providing more security
        - The code can easily be reconstructed back into its original form, or very close to it.
        - The "obscurity" through obfuscation is more of an inconvenience/extra step rather than a security addition.

Question 6

a. Collection of raw network traffic data that goes to the Department of SSH will include electronic mails that are being sent and received by the department.
    - Let us assume that such traffic is unencrypted, or easily decryptable by third-party so that it can be usable for data analysis by the research group. Other than that, the network administrators already have read/write access to the mail servers of the Department.
    - As stated, since IIIT Delhi is a state university, it is partly bound by the Indian Constitution. There is reasonable expectation by users of the email service that their emails, even though not completely private or confidential, are to a large extent protected, and their right to privacy is not infringed.
        - The fact that the policy allows personal email usage further consolidates this expectation.
    - The policy states that their mail can be read and modified at will, but behind this policy there is a reasonable trust and belief that network administrators would not misuse such power.
    - Providing complete raw-data access of all email information of the Department to the research group breaks the implicit trust between the IIIT administration and the users of the email in that Department -- and by extension -- all IIIT email users.
    - Moreover, this is not only a matter of broken trust. The policy does not dictate whether it can *share* email data with outsiders.
        - The legal maxim "everything which is not forbidden is allowed" cannot apply here, since policies must explicitly dictate all their terms of use.
        - Anything that is not mentioned in the terms of use of the policy should require explicit permission from all users before-hand.

b. The policy change must reflect that electronic mail information **can be shared** with third party, external organizations, but in a way that **protects the individual's privacy**.
    - First of all, completely raw and untouched data access should not be allowed by any means.
    - IIITD must determine the type and amount of information that is required by the research group and the purpose for each.
    - IIITD must provide them only so much information that it sufficiently meets their requirements, but nothing more.
    - All information that's provided to external organizations, such as the research group, must be **properly anonymized** so as to prevent any personally identifying information from getting shared.

Question 7

- A public key cryptosystem comprises a private-public key pair. The public is sharable with everyone, but the private key must be kept a secret.
- A fundamental idea of the system is that one should not be able to methodically retrieve the private key from its public key counterpart.
  - This defeats the purpose of secrecy of the private key and collapses the entire cryptosystem.
- In the Caesar cipher, the private and public keys, albeit different, are related
  - One can easily retrieve the private (encipherment) key from the public (decipherment) key by simply subtracting it from 26.
- Thus, the Caesar cipher provides poor secrecy for the decipherment key, and is not considered to be a public-key cryptosystem.
- Moreover, since the size of the set of possible private keys is so small (26), the decipherment can easily be brute forced.

Question 8

c.

1. A checksum in the context of cryptography is a function that takes an arbitrary-sized input and produces a checksum or *hash*.
    - In almost all cases, the output of the checksum function (hash) is of a fixed size.
    - The checksum or hash functions are “collision-resistant”.
        - It is extremely difficult to find two inputs that produce the same hash.
        - No collisions have been found for newer and more sophisticated hashing algorithms, such as SHA-256.
    - Due to the property of collision-resistance, the hash can be said to be representative of the input data.
2. The identity function is NOT a good cryptographic function.
    - *Assumption:* In this identity function, the byte-stream of the input is identical to the byte-stream of the output.
    - Since the length of the input is the same as the output, the hash will be of arbitrary length.
    - Due to its arbitrary length and character set, it is difficult to replicate and compare with other hashes.
    - A minor change in the input does not produce a significantly changed hash.
        - Manual comparison is susceptible to mistakes.

d. *Assumption:*

1. We are doing bitwise XOR operations on each word.
2. We left-pad words with zeros such that all words have the same bit length.

The XOR checksum function is a cryptographically poor checksum for the following reasons

- The output depends on the length of the input words (more specifically, the length of the *longest* input word). Thus, the hash can be of arbitrary size.
- The checksum is very susceptible to hash collisions.
    - It is very easy to construct input words such that their hashes collide.
    - Since we are doing bitwise XORs, and XOR is an associative function, we can simply reorder the bits of a column to produce the same output bit, but in-turn modify the inputs.
    - Due to this, hash-collision attacks can be very simply performed where a malicious file can be downloaded and run, which has the same hash as the original file.
    - *Strong data integrity is missing!*

Question 9

a. **Simple sanitization**
    i. Simple sanitization removes all information except commands.
    ii. This level of sanitization provides most anonymity as it retains minimal information.
    iii. It can be automated very easily, and requires next to no human assistance.

b. **Information-tracking sanitization**
    i. This form of sanitization maintains a map of sensitive information and its identifiers.
    ii. The information is pseudonymized, but not fully anonymous as the sensitive information can be retrieved through a lookup in the symbol table.
    iii. Automation of this form of sanitization requires more steps:
        1. Human assistance is required to identify which words or phrases are sensitive
        2. The sanitizer must build a dictionary, match such words, assign pseudonyms in the symbol table and save the data.

c. **Format sanitization**
    i. This format supports sanitization in files other than plain text.
    ii. Again, information-tracking sanitization is applied on the original form, which preserves pseudonymity but not anonymity.
        1. There is slightly less anonymity in this level of sanitization as the file/data is decoded first to its original form thereby revealing more information.
    iii. Automation for this sanitization is even harder:
        1. Human assistance is required to determine the level of encoding and the number of times the file/data has been encoded in order to properly decode it.
        2. As in (b), the sanitizer will build a dictionary of human-provided words and maintain the symbol table on the basis of that.

d. **Comprehensive sanitization**
    i. This format provides the least amount of anonymity amongst the four levels.
        1. *All* data is decoded, analysed, parsed and formatted.
    ii. Automation in this form of sanitization is the least, and by far requires the most human intervention and supervision.
        1. Humans must manually specify phrases that require sanitization (pseudonymization).
        2. Some phrases require all instances to be sanitized, while some phrases only require some instances to be sanitized.
        3. The symbol table and sanitizer's dictionary is large and complicated.
            a. It must be regularly updated whenever it encounters new phrases.

Question 10

a. This problem is, in essence, the Coupon Collector's Problem.
   i. Let M be the number of mails that the attacker needs to send to the remailer in order to flush all original $n-1$ mails from the pool.
   ii. Let m_i be the number of mails that the attacker needs to send to the remailer in order to send the i^{th} original mail after the first $i-1$ original mails have been sent.
   iii. Therefore, $M = m_1 + m_2 + m_3 + \dots + m_{n-1}$.
   iv. Probability of sending an original mail is $p_i = (n - i)/n$.
   v. We know from m_i's probability mass function (PMF) that m_i follows a geometric distribution. Thus, $E[m_i] = 1/p_i = n/(n - i)$.
   vi.
$$\begin{aligned} E[M] &= E[m_1 + m_2 + m_3 + \dots + m_{n-1}] \\ &= E[m_1] + E[m_2] + E[m_3] + \dots + E[m_{n-1}] \\ &= 1/p_1 + 1/p_2 + \dots + 1/p_{n-1} \\ &= n/(n-1) + n/(n-2) + \dots + n \end{aligned}$$

$$\boxed{\boldsymbol{E[M] = n(1 + \tfrac{1}{2} + \tfrac{1}{3} + \dots + 1/(n-1))}}$$

b. *Assumption:*
   i. All mails are successfully remailed by Cypherpunk, with no errors.
   ii. All mails take a fixed time to be finally received by the target inbox, without any delays.

- All mails sent by the attacker are sent to either his own inbox or an inbox he controls.
  - This is done so that the attacker can determine whether his email has been sent by the remailer (and thereby received in his inbox).
- Initially, the pool has $n-1$ mails.
  - The attacker sends 1 mail and the pool grows to size n.
- The remailer randomly chooses an email.
  - If the remailer chooses the attacker's email, the attacker will receive it in his inbox and get to know that his email was chosen.
  - If the remailer chooses one of the original emails, the attacker won't receive any email in his inbox after a fixed period of time, and will conclude that one of the original emails was sent.
- The attacker will continue sending emails until it concludes that the original $n-1$ emails have been sent.
  - The pool is now only filled with the attacker's emails.