CSE 4503

Attack Vectors

Definitions

In software security, attack vectors refer to the methods or pathways that malicious actors use to exploit vulnerabilities in a system or network. These vectors provide entry points for attackers to gain

- Denial Of Service
  - Could not access the server
- Information Leakage
  - Publicly access personal NID information
- Privilege Escalation
  - Users can do admin’s function.

DoS

The Denial of Service (DoS) attack is focused on **making a resource (site, application, server) unavailable** for the purpose it was designed. Denial-of-service attacks significantly degrade the service quality experienced by legitimate users. These attacks introduce :

- Large response delays.
- Service interruptions.

And both of the above causes excessive loss to a financial web application. Two risk factors of DOS are,

- Insufficient resources (bandwidth or scalability) might make a system vulnerable to DOS attack.
- Poorly written or unoptimized applications may not handle resource allocation efficiently, making them easier to exhaust with a small amount of traffic.

Before Poor Coding We Need to Know GET HTTP request

A GET request is one of the most commonly used HTTP methods. It is used to retrieve data from a server. The following is a GET request done by a client to onlineStorage website.

```
GET /addBucket?userId=1&numBuckets=10000000000 HTTP/1.1
Host: www.onlineStorage.com
```

Here, the client with userId 1 wants to **get** storage equal to 1e10 buckets where 1 bucket is equals to 1 megabyte. **(Bonus: what is the total storage client is asking for?)** That's why Server is running that addBucket script (because client asked him to run that through the GET HTTP request) which add buckets (of storage) to client's profile (so that client use can that online storage like we use google drive). Now the addBucket script takes two arguments, userId that takes id of the client asking for storage and numBuckets that takes the number of buckets our client need.

clientRequestingbucket.js (Client Side Code)

```
fetch('https://www.onlineStorage.com/addBucket?userId=1&numBuckets=1000000000')
.then(response => response.json())
.then(data => console.log(data))
.catch(error => console.error('Error:', error));
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "results": [
    {"status":"Memory is allocated","userId":1}
  ]
}
```

addBucket.java (Server Side Code using servlet)

```
public class SearchServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        String userIdString = request.getParameter("userId");
        int userId = Integer.parseInt(userIdString);

        String numBucketsString = request.getParameter("numBuckets");
        int numBuckets = Integer.parseInt(numBucketsString);

        Bucket[] storage = new Bucket[numBuckets];

        User client = UserList.getUser(userId);
        client.setStorage(storage);

        response.setContentType("application/json");
        PrintWriter out = response.getWriter();
        out.print("{\"results\": [{\"status\": \"Memory is allocated\", \"userId\": "+Integer.toString(userId)+"}]}");
        out.flush();
    }
}
```

Server Response

The left side server side code is poorly written, can you find the problem?

```java
@WebServlet("/addBucket")
public class SearchServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        String userIdString = request.getParameter("userId");
        int userId = Integer.parseInt(userIdString);

        String numBucketsString = request.getParameter("numBuckets");
        int numBuckets = Integer.parseInt(numBucketsString);

        if(numBuckets <= thresholdAllocation) {

            Bucket[] storage = new Bucket[numBuckets];

            User client = UserList.getUser(userId);
            client.setStorage(storage);

            response.setContentType("application/json");
            PrintWriter out = response.getWriter();
            out.print("{\"results\": [{\"status\": \"Memory is allocated\", \"userId\": "+Integer.toString(userId)+"}]}");
            out.flush();

        } else {

            response.setStatus(HttpServletResponse.SC_BAD_REQUEST);
            response.setContentType("application/json");
            PrintWriter out = response.getWriter();
            out.print("{\"results\": [{\"status\": \"Memory is not allocated due to crossing threshold\",
            \"userId\": "+Integer.toString(userId)+"}]}");
            out.flush();
        }
    }
}
```

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
  "results": [
    {"status":"Memory is not
allocated due to crossing threshold",
     "userId":1}
  ]
}
```

Hopefully the correct server side code implementation and the server response if threshold Allocation is lower than 1e10.

Bonus : user Input as a Loop Counter

```javascript
const values = [10, 20, 30, 40, 50];

fetch('https://ValueProcessor.com/processValues', {
    method: 'POST',
    headers: {
        'Content-Type': 'application/json',
    },
    body: JSON.stringify({
        numbers: values
    })
})
.then(response => response.json())
.then(data => console.log('Success:', data))
.catch(error => console.error('Error:', error));
```

Client Side Code

```java
@WebServlet("/processValues")
public class ProcessValuesServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        BufferedReader reader = request.getReader();
        StringBuilder jsonData = new StringBuilder();
        String line;
        while ((line = reader.readLine()) != null) {
            jsonData.append(line);
        }

        JSONObject jsonObject = new JSONObject(jsonData.toString());
        JSONArray numbersArray = jsonObject.getJSONArray("numbers");

        for (int i = 0; i < numbersArray.length(); i++) {
            int number = numbersArray.getInt(i);
            ProcessValue.process(number);
        }

        response.setContentType("application/json");
        response.setStatus(HttpServletResponse.SC_OK);
        response.getWriter().write("{\"message\": \"Array received successfully\"}");
    }
}
```

Server Side Code

DoS vs DDoS

Feature	DoS	DDoS
Number of Sources	Single	Multiple
Complexity	Less	More
Resources	Less	More
Effectiveness	Less	More
Example	A single computer flooding a server with HTTP requests.	A botnet of thousands of devices simultaneously sending traffic to a server.

Some DoS or DDoS attack :

- A DDoS attack ranges from the accidental – genuine users overwhelming the resources of popular sites, such as in a ‘Reddit hug of death’ – to sophisticated exploits of vulnerabilities.
- Simple attacks include the ‘Ping of Death’ – sending more data to the host than the Ping protocol allows, or Syn Flood, which manipulates TCP connection handshakes.
- More recent and sophisticated attacks, such as TCP SYN which requires three step hand shaking, might attack the network.

Information Leakage

```java
public class InformationLeakageExample {

    // Simulate sensitive information
    private static String username = "admin";
    private static String password = "SuperSecretPassword123";

    public static void main(String[] args) {
        try {
            // Simulate a login attempt
            login(username, password);
        } catch (Exception e) {
            // Catch and print stack trace to console/log file (includes sensitive information)
            e.printStackTrace();
        }
    }

    // Simulate login logic
    public static void login(String username, String password) throws Exception {
        if (username.equals("admin") && password.equals("SuperSecretPassword123")) {
            System.out.println("Login successful!");
        } else {
            throw new Exception("Login failed for username: " + username + " with password: " + password);
        }
    }
}
```

Information Leakage Solution

```java
public class SecureExample {

    private static String username = "admin";
    private static String password = "SuperSecretPassword123";

    public static void main(String[] args) {
        try {
            login(username, password);
        } catch (Exception e) {
            // Log a generic error message
            System.err.println("Login failed due to an internal error. Please try again.");
        }
    }

    public static void login(String username, String password) throws Exception {
        if (username.equals("admin") && password.equals("SuperSecretPassword123")) {
            System.out.println("Login successful!");
        } else {
            // Throw a more secure exception without sensitive data
            throw new Exception("Login failed.");
        }
    }
}
```

Information leakage In an Enterprise System

PaymentServlet parse credit card number and cvv and pass it to payment service to process the transaction.

```
public class PaymentServlet extends HttpServlet {

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        String creditCardNumber = request.getParameter("creditCardNumber");
        String cvv = request.getParameter("cvv");

        PaymentService paymentService = new PaymentService();
        try {
            paymentService.processPayment(creditCardNumber, cvv);
        } catch (Exception e) {
            System.err.println("Payment processing failed with card: " + creditCardNumber + ", CVV: " + cvv);
            e.printStackTrace();
        }
    }
}
```

Continued

PaymentService checking if credit card information valid or not. If valid sending payment through PaymentGateway.

```
public class PaymentService {

    public void processPayment(String creditCardNumber, String cvv) throws Exception {
        if (!isValidCard(creditCardNumber, cvv)) {
            throw new Exception("Invalid card information: " + creditCardNumber + " (CVV: " + cvv + ")");
        }

        PaymentGateway gateway = new PaymentGateway();
        gateway.sendPayment(creditCardNumber, cvv);
    }

    private boolean isValidCard(String cardNumber, String cvv) {
        return cardNumber.length() == 16 && cvv.length() == 3;
    }
}
```

Continued

PaymentGateway serializing the credit card information into bytes and sending it through Network class to some banking system to process the actual payment.

```
public class PaymentGateway {

    public void sendPayment(String creditCardNumber, String cvv) throws Exception {
        try {
            byte[] serializedData = serializePaymentData(creditCardNumber, cvv);

            Network portal = new Network();
            portal.send(serializedData);

        } catch (Exception e) {
            throw new Exception("Failed to send payment. Data: " + creditCardNumber + ", CVV: " + cvv);
        }
    }

    private byte[] serializePaymentData(String cardNumber, String cvv) throws IOException {

        ByteArrayOutputStream bos = new ByteArrayOutputStream();
        ObjectOutputStream out = new ObjectOutputStream(bos);
        out.writeObject(cardNumber);
        out.writeObject(cvv);
        out.flush();
        return bos.toByteArray();
    }
}
```

Privilege Escalation

Privilege escalation is an unintended increase of privileges. There are two types of privilege escalation :

1. Vertical Privilege Escalation : For example, if a non-administrative user can gain access to an admin page where they can delete user accounts, then this is vertical privilege escalation.
2. Horizontal Privilege Escalation : For example, if a user can gain access to another user’s account.

Vertical Privilege Escalation

- A user might be able to access the administrative functions by browsing to the relevant admin URL. Like, using the following URL and accessing the admin page “`https://insecure-website.com/admin`”
- In some cases, the administrative URL might be disclosed in other locations, such as the robots.txt for the ***web crawlers***. So the user can use the following URL “`https://insecure-website.com/robots.txt`” and access the robots.txt and from there can get the URL for the admin page.
- LinkForLab
- Even if the administrative URL is not disclosed in robots.txt, it can be common and intruders might find it using brute force attack.
    - Solution is to use random administrative URL like “`https://insecure-website.com/administrator-panel-yb556`”

Continued With New Problem

The application might still leak the URL to users. The URL might be disclosed in JavaScript that constructs the user interface based on the user's role:

```
<script>

var isAdmin = false;

if (isAdmin) {

...

var adminPanelTag = document.createElement('a');
adminPanelTag.setAttribute('https://insecure-website.com/administrator-panel-yb556');


adminPanelTag.innerText = 'Admin panel';

...

} </script>
```

LinkForLab

Horizontal Privilege Escalation

- Can access another user by incrementing one’s id,

  `https://insecure-website.com/myaccount?id=123`

- Solution GUID (Globally Unique Identifiers). This may prevent an attacker from guessing or predicting another user's identifier.
- But GUID must be kept secret without disclosing it anywhere.
- LinkForLab