Web Application Exploits and Countermeasures

Project report in
TDA602/DIT101 Language Based Security

Arun Prakash Jothimani
Michel De Carvalho Folkemark

Group Number: 27

Chalmers University of Technology
Gothenburg, Sweden.
Version no.: 1.0

Table of Contents

1. Background

Everyday we are witnessing lot of cyberattacks happening all over the world. Particularly, we witness repetitively similar attacks happening even now, though these attacks are discovered decades back. Recent patch by authentication attacks at Microsoft for a vulnerability in Microsoft Windows 10 1803 and Windows Server 2019 and later systems that was allowing authenticated RDP-connected clients to gain access to user sessions without needing to interact with the Windows lock screen is a good example that even big corps are not exceptions to these common vulnerabilities. OWASP is keep listing, the same vulnerabilities as the Top 10 Vulnerabilities for the whole of the past decade because these vulnerabilities break out in one or other way.

OWASP Top 10 - 2013	→	OWASP Top 10 - 2017
A1 – Injection	→	A1:2017-Injection
A2 – Broken Authentication and Session Management	→	A2:2017-Broken Authentication
A3 – Cross-Site Scripting (XSS)	↘	A3:2017-Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged+A7]	U	A4:2017-XML External Entities (XXE) [NEW]
A5 – Security Misconfiguration	↘	A5:2017-Broken Access Control [Merged]
A6 – Sensitive Data Exposure	↗	A6:2017-Security Misconfiguration
A7 – Missing Function Level Access Contr [Merged+A4]	U	A7:2017-Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)	☒	A8:2017-Insecure Deserialization [NEW, Community]
A9 – Using Components with Known Vulnerabilities	→	A9:2017-Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	☒	A10:2017-Insufficient Logging&Monitoring [NEW,Comm

The Top 10 OWASP vulnerabilities in 2020 are:

- Injection
- Broken Authentication
- Sensitive Data Exposure
- XML External Entities (XXE)
- Broken Access control
- Security misconfigurations
- Cross Site Scripting (XSS)
- Insecure Deserialization
- Using Components with known vulnerabilities
- Insufficient logging and monitoring

As a cyber security student, we believe that, having a solid hand on experience with OWASP TOP attacks and counter measures will give a better foundation to our security career. As a part of course work, we explored XSS Attack and SQL Injection. In this report, we will be exploring other important OWASP TOP 10 vulnerabilities like XML External Entities (XXE) and Broken Authentication along with their counter measures.

2. Goal

The Goal of this project is the move a step ahead of course work exercises and investigate the significant OWASP TOP Vulnerabilities like XML External Entities and Broken Authentication and implementing the counter measures. To get hands on with Burp suite proxy to manipulate traffic, to automate Brute force attack and to gain the string foundation on OWASP Prevention principles. We have deployed the Web Goat which is a deliberately insecure web application to experiment our attacks and we patched the application. We also reverified the patch and presented the code changes as a part of this report. Following attacks and patches will be explored and performed:

- XML External Entities (XXE) Attack
- Authentication and Authorization
    - Authentication Bypass Exploit
    - Countermeasure
    - JSON Web Tokens - Authorization Exploit
    - Countermeasure

3. Theory

3.1 XML External Entities(XXE) Attack

An XML External Entity attack is a type of attack against an application that parses XML input. The possibility for this attack arises, when XML input containing a reference to an external entity is processed by a weakly configured XML parser. This attack may lead to the several serious problems like

- disclosure of confidential data,
- denial of service,
- server-side request forgery,
- port scanning from the perspective of the machine.

Even though this attack has been possible for years, this came to limelight when major web applications such as Facebook's third-party career service and PayPal's Ektron CMS become vulnerable to this attack

3.1.1 Vulnerability

Attackers can take advantage of the XML external entities to use this vulnerability to utilize its external functionality. Any Misconfiguration of the XML Parser may give a ticket to attacker to use its as a trap by turning it into a proxy server to execute Server Side Request Forgery Attacks to gain access to intranet network and external public servers from behind the firewall. The attacker is not confined to only accessing local files on the local exploited machine. They can recreate an RFI type of attack where they can access files remotely via http.

Any vulnerable XML processors can be exploited by uploading xml or by including hostile content in an XML Document. By default, some of the older XML Processors permit the specification of an external entity, a dereferenced URI which will be evaluated during XML processing. The application is exposed to XXE External Entities attack, When they accept XML directly from the untrusted

sources or XML Uploads Specifically from the untrusted random sources or it permits the insertion of untrusted data into XML Document, which can always be parsed later by an XML Processor.

With respect to language-based security, we can say Java applications using XML libraries are particularly vulnerable to XXE attacks because XXE is always enabled by default for most of the Java XML parsers. If the XML processors in the application or that exist in web services has Document Type Definitions (DTDs) enabled, then XXE Attack find its possibility there. In case of SAML usage, it may use XML for identity assertions which may bring a vulnerability.

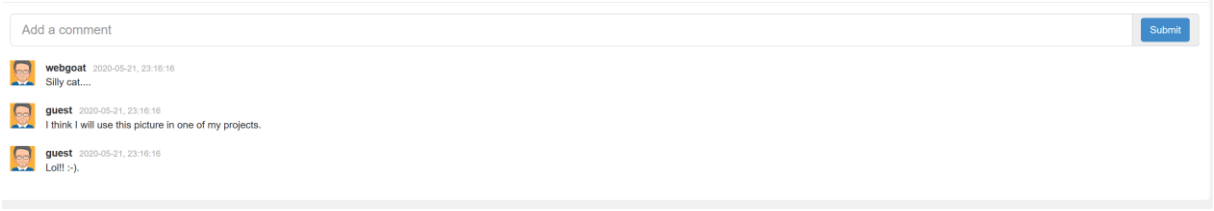


Fig: Threat Model

Attacks can include disclosing local files, which may contain sensitive data such as passwords or private user data, using file: schemes or relative paths in the system identifier. Since the attack occurs relative to the application processing the XML document, an attacker may use this trusted application to pivot to other internal systems, possibly disclosing other internal content via http(s) requests or launching a CSRF attack to any unprotected internal services. In some situations, an XML processor library that is vulnerable to client-side memory corruption issues may be exploited by dereferencing a malicious URI, possibly allowing arbitrary code execution under the application account. Other attacks can access local resources that may not stop returning data, possibly impacting application availability if too many threads or processes are not released.

Via this XXE attack, the attacker can disclose the local files of the user that may contain sensitive data like passwords, account numbers or any personal information. Further to this, the attacker may use this trusted application to disclose other internal contents via http request or by launching Cross Site Request Forgery [CSRF] attack to any insecure internal services. The attack has been executed and represented under Description section.

3.1.2 Countermeasures

The Developer should have a proper understanding before implementation so making awareness about common attacks to developers will stop them from repeating the mistakes. It is recommended to use simple data formats like JSON and avoid serialization of sensitive data whenever the possibility exists. It is important to use annotations carefully to enable only required privileges and to control the incoming data format. Disabling XML external entity and DTD processing in all XML parsers in the applications as recommended by OWAP Cheat sheet may help. It is always essential to upgrade all the XML Processers, libraries, vulnerable SOAP versions of the application. The server-side inputs should be validated, and they must be filtered and sanitized to protect hostile data within XML documents, headers or nodes. It is a necessary to verify if the XML file upload functionality validates the incoming XML using XSD Validation. Employing source code analyser tools Bandit, pmd,etc can help to easily detect XXE in the source code.

We have deployed Web Goat application to showcase the attacks. We used Burp Suite proxy to intercept Traffic from client to server. Yes, triggered a Man in the Middle attack to modify the traffic on its way back to server. We were able to modify the session ID of the user. We got control over what is posted, some configurations and what is being posted. We also able to read some sensitive data and was able to exploit the REST Framework as well. The Process, Results and countermeasures are under Discussion topic[1].

3.2 Authentication and Authorization

It is very important that, we should allow only the right user to modify the content. Only right content should be displayed to the user. The recent authentication bypass attack happened at PayPal easily convinced us to accept that, still this authentication bypass is commonly happening even in Big corporations and exploring it practically will really give a good expose to us. We evaluated the source code of Web Goat, and hunted the authentication vulnerabilities , implemented patches to mitigate the authentication bypass.

3.2.1 Authentication Bypass Exploit

We exploited Web Goat Application to try an attack similar to the recent exploit with PayPal two-factor authentication bypass. Consider a use case, when user is not getting his one-time password and he goes for the alternative method of answering security questions to login his account. If the attacker answers the security question, he can bypass the authentication. If the application is vulnerable , with the help of proxy, it's possible to intercept the traffic and remove the security questions from the post data that are sent to the end points. We exploited the vulnerability in WebGoat without removing the security question. For whole process, we relied on Burp Suite proxy. We added a new type of request under intercepts client requests of Burp proxy. We have provided match type as HTTP method and match relationship as "Matches", thus we configured Burp Proxy to intercept only when the post requests are made by the browser towards the server. When the user tried to log IN, we found that browser is waiting after sending the traffic to server. From Burp proxy we modified the password and thus user will get error as wrong username/password.

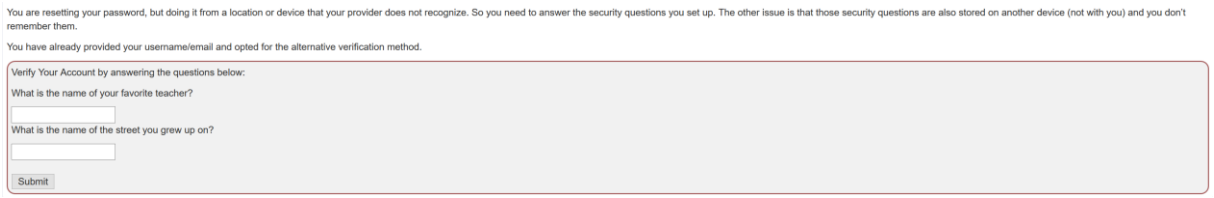


Fig: Threat Model

We tried exploiting the security questions field in variety of ways like, giving random false inputs to security question, changing the parameter names from secQuestion0 to secAnswer0 ,and secQuestion1 to secAnswer1 etc and finally we found the pitfall when we incremented the secQuestion0 to secAnswer2 ,and secQuestion1 to secAnswer3 and forwarded that from Intercept proxy. This worked and we were able to change the password to new password.

Other few hacks also worked for us. Like we passed empty parameter to security questions. But doing all these permutations manually and checking is a hectic job, so we tried making it automated using the built-in utilities in burp suite proxy. We sent the traffic to intruder, which helps us in picking parts of our payload and fuzz it to try a lot of scenarios. We used the attack type called Pitchfork to individually manipulate every required value. We fuzzed the numeric values and tried all possible values from 0 to 10000 and incorporated brute force inputs to get picked automatically. Finally, we were able to find the right combination. Execution part of this attack can be found at Description Section.

3.2.2 Counter Measure

Nowadays, most of the applications like banking applications, email, etc started using a second factor authentication to check the authorization of the user. But this authentication can be bypassed if the code implementation is poor.

It is recommendable that, Authorization should be performed and enforced server-side. We should not give any room to the attacker to influence the authorization by altering the data flow that happens between client and server by tampering with parameters that contain transaction data , by adding/removing which will disable authorization check or by causing some error. This can be achieved by adopting best programming practices like,

- Default Deny
- Avoid debugging functionality in production code

Transaction verification data should be generated server-side. When a significant transaction data are programmatically transmitted to an authorization component, its very important to enforce a default deny for any client modifications on the transaction data. This significant data that has to be verified by the user should be generated and stored on the server before it reaches authorization component without giving any chance for tampering of data by the client. This method shields us from attacks like collecting significant transaction data on client side to manipulate them to fake transaction data in an Application should prevent authorization credentials brute forcing. When transaction authorization credentials are sent to the server for verification, the application must prevent brute forcing. We made a simple authentication attack using Web Goat. The execution part and patch are under Description section

3.2.3 JSON Web Tokens - Authorization Exploit

JWT is a secure and trustworthy standard for token authentication. JWT makes the information digitally signed with a secret signature that can be verified later with the secret key. After the advent of JWT, most applications depending upon their needs, they started using JWTs rather than Session Cookies.

We had an understanding that JWTs are way secure and impossible to exploit but loopholes exist everywhere. To exploit JWT, we first explored what JWT is.

JWT is separated into three sections, the header, payload, and signature.

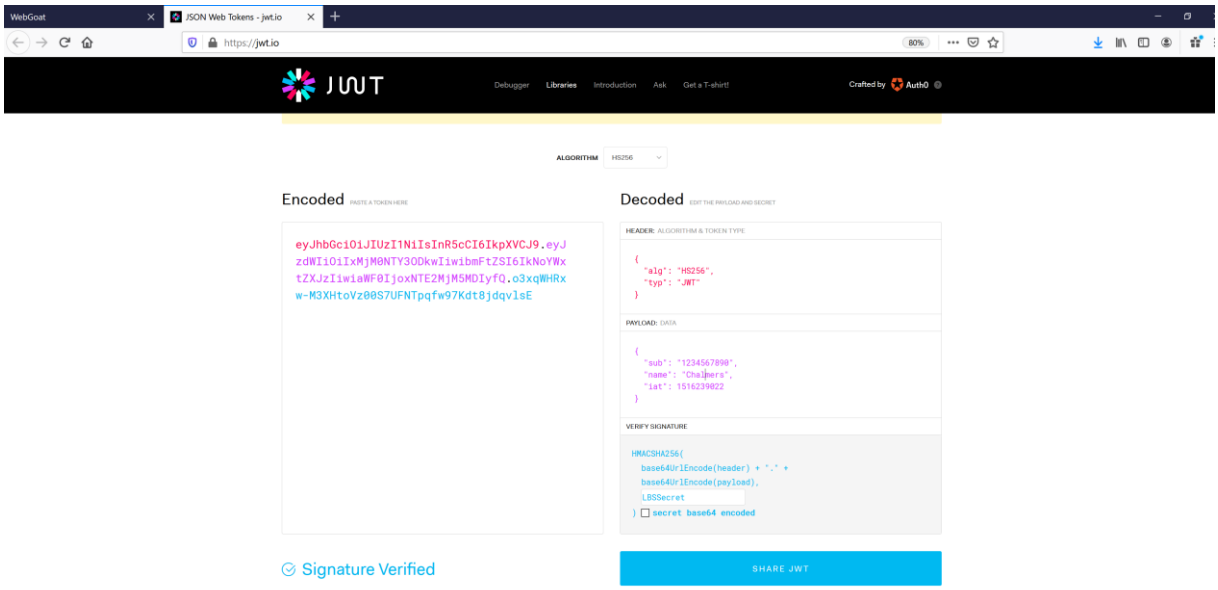


Fig 1: Structure of JWT

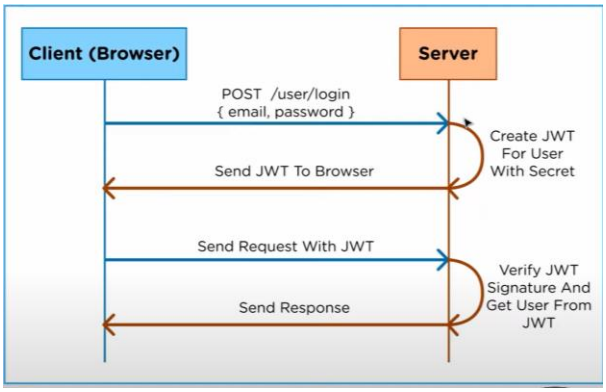


Fig 2: JWT Execution

JWT is used for authorization of the user in web application. Authorization is all about making sure that user that sends request to the server is the same user that Logged in during the authentication process. Thus, authorization is all about enabling the fact that this user has access to this system. Normally it is done using the session. For example , When we have session ID, it is send down in the cookies of the browser and every time the client makes your request, they send that session id to the server and the server evaluates the same against the ID in its memory to grant authorization but now, instead of using the cookies, systems started using JSON Web tokens to do the authorization. Whenever we make a client request to the server, instead of storing information on the server inside session memory, the server creates JWT and encodes ,serialize and signs it with its secret key so that the server knows if someone tamper it. Then sends the JSON Web token back to the browser so that browser can choose to store however it wants. Nothing stored in the server and JWT has all the information about the user. Now when browser sends a new request, it sends JWT as well thus the server can verify JWT Signature and authorize the user. Using tokens in place of session IDs can lower the server load, streamline permission management and enables a better cloud based infrastructure. The security of any token or sessions is mostly decided by the developer who codes the application and uses these stuffs. We explored about the flaw that can happen with JWT using Web Goat Application.

We deployed the Web Goat Application and exploited the threat model. Here the server creates the JWT Token with a secret and returns to the browser. The request with JWT are checked for signatures to verify the user information.

The threat model looks like the figure below

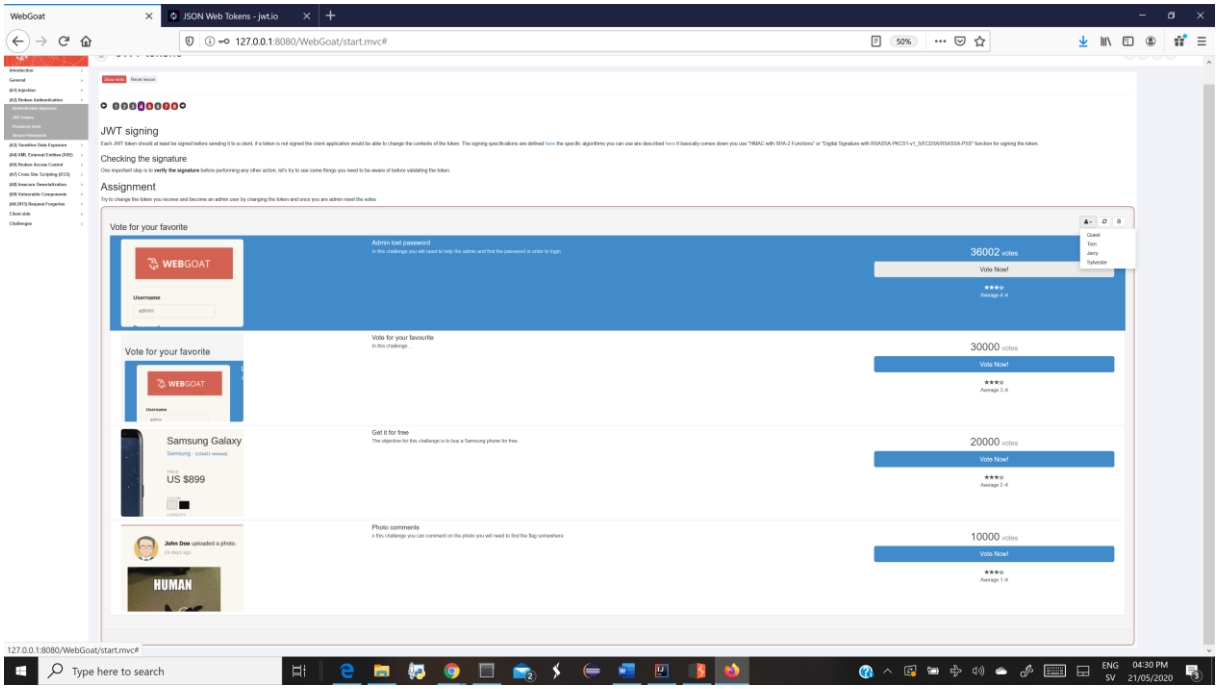


Fig: Threat Model

The drop-down box contains three individual users and the guest. The individual users all allowed to vote while guest cannot. It is designed in a way that only admin can delete the votes while others can only cast, and they cannot delete. Our intention was to exploit JWT to make delete possible without being the admin. We jumped in Burp suit proxy to explore the background conversations between the client and server. We understood that every time the reset post request is denied with the message, only the admin can reset the votes, but we have seen an associated JWT with it. We explored the structure of the token to look for the possibility to modify.

We copied the access token and we know that it contains three parts. Header Claim and signature. We pasted this header in the Decoder utility that is built inside the burp suit proxy. You can also use the tool cyberchef as well. After hitting the official sites, we studied about the Header parameter values for JWS. Based on the study, we changed the algorithm in the header to None. We reconstructed the access token with our new header and encoded modified claim where admin parameter is turned false.

Now when we passed this access token, we were able to break the system.

3.2.4 Counter Measure

Nowadays, most of the applications like banking applications, email, etc started using a second factor authentication to check the authorization of the user. But this authentication can be bypassed if the code implementation is poor. We understood that Its always not about how secure the token is. The real trap is, how securely we are using.

By studying about the specification of JWT, we found that to be a valid token, it is not mandatory for JWT tokens to have signatures. By debugging the Web Goat application, it's evident that the parse JWT function simply parse the access token into JWT or JWS based on the requirement and it didn't care to validate the signatures. At the same time, if it doesn't witness the algorithm in the header of the JWT, it will fail in that case as well.

As a solution, we ensured that, parse function is always going to validate our signature. Whenever the token is tampered, we made it to throw signature exception, thus the jwt-invalid-token message is called. The demo of our execution and patches can be found at the description section

4. Description of Work

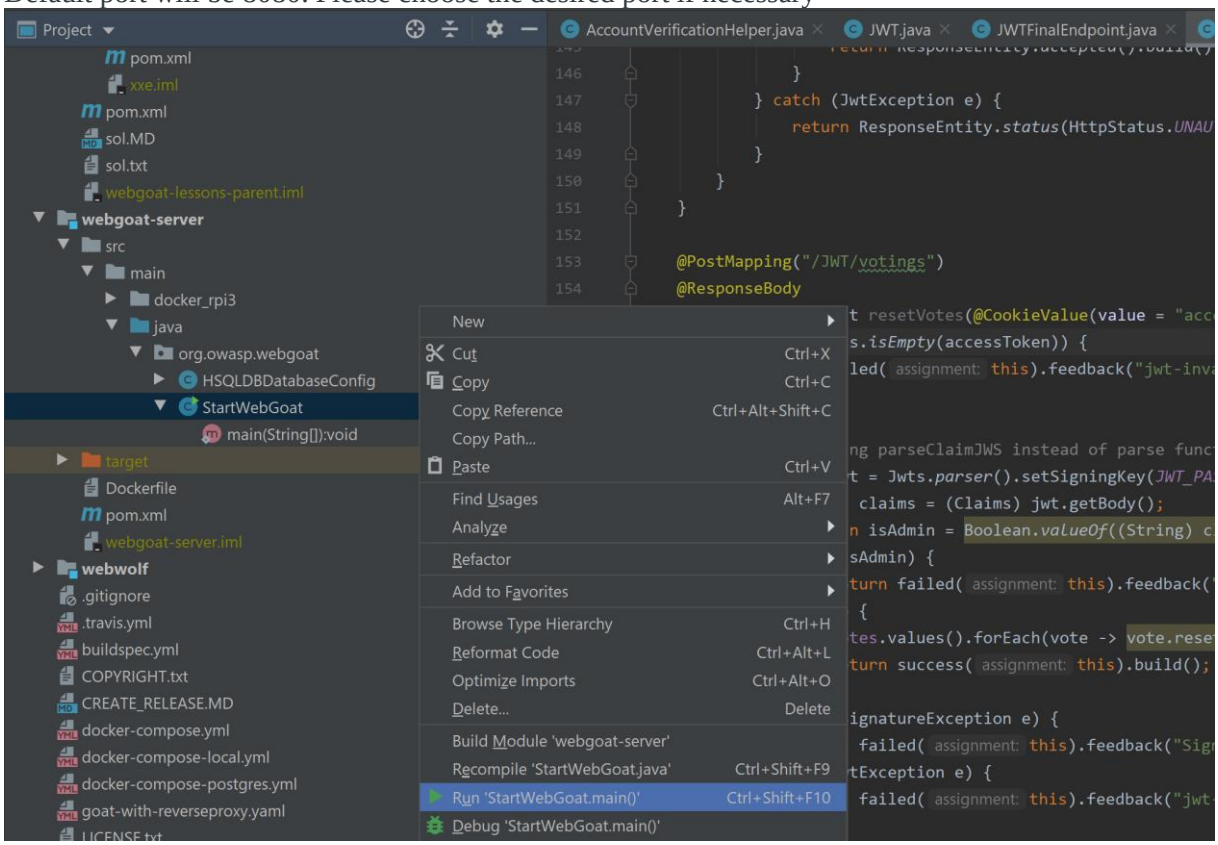
We have used a purposefully vulnerable application called WebGoat to explore OWASP TOP 10 API attacks.

4.1 Project Setup

Step 1:

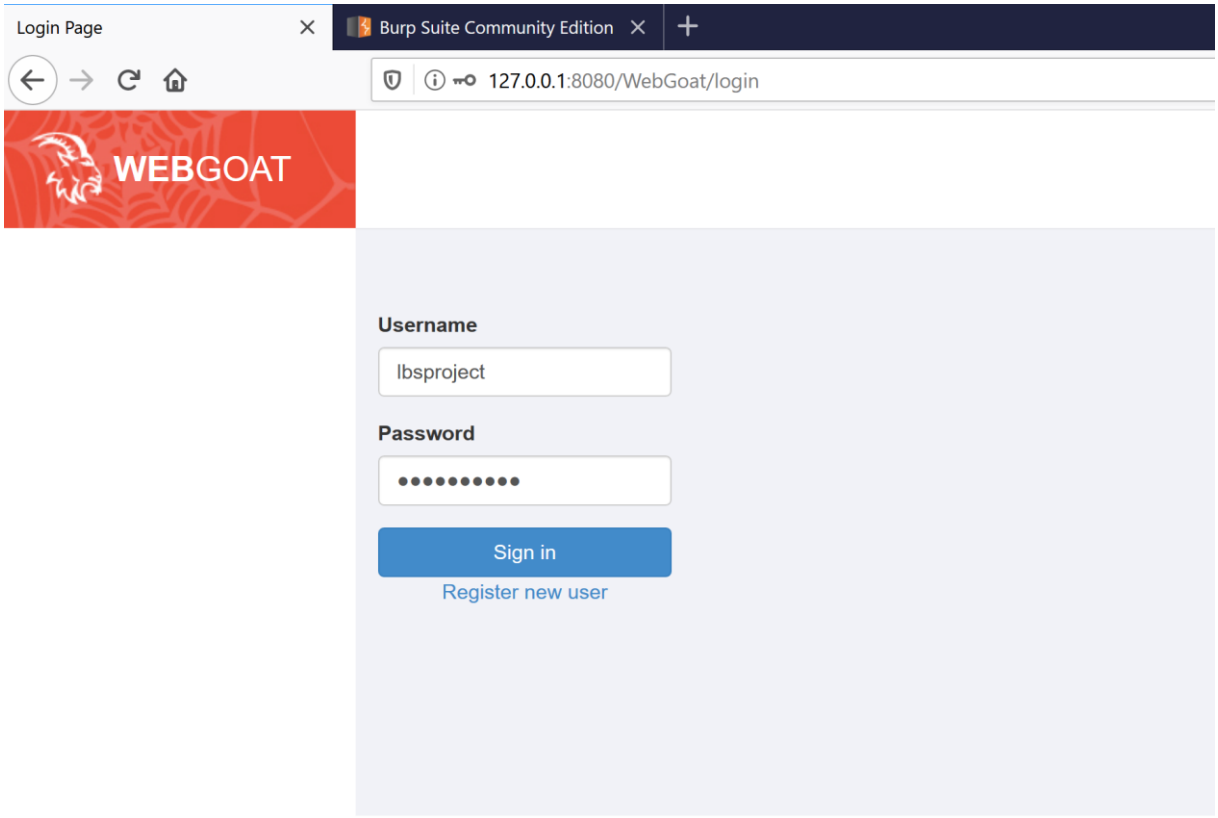
Download and install the latest version of Web Goat Server to a suitable location. Latest release can be found at https://github.com/WebGoat/WebGoat/releases

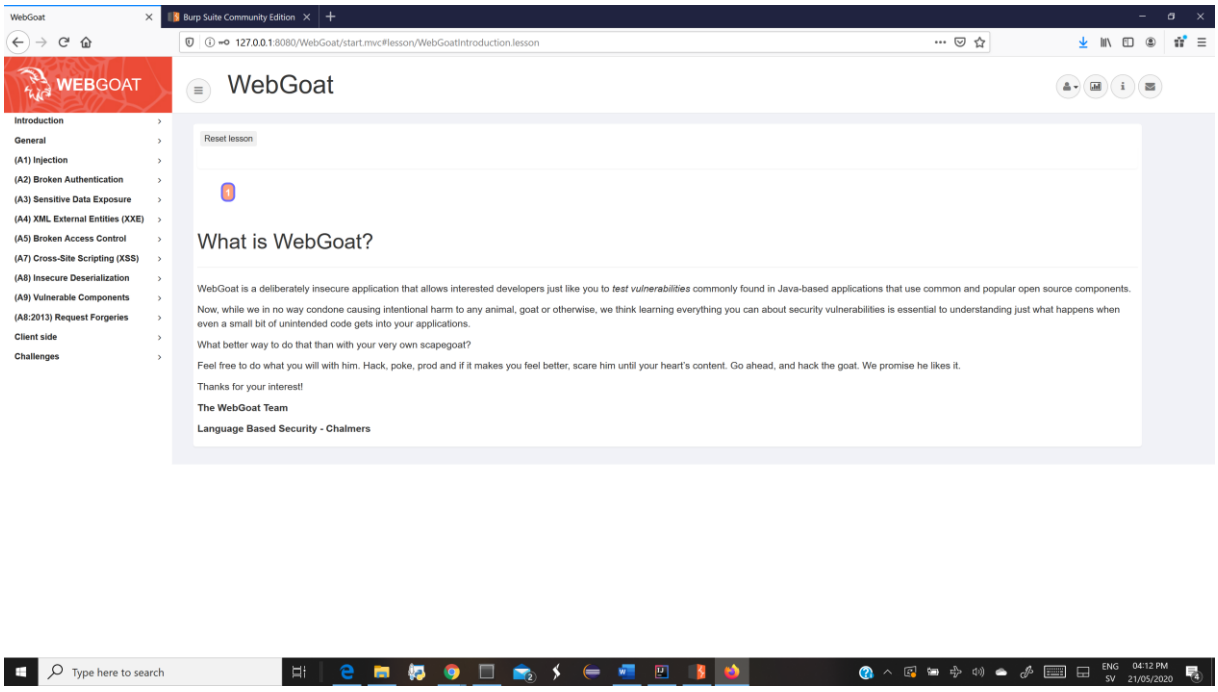
Step 2:
Default port will be 8080. Please choose the desired port if necessary



Step 3:

Load the URL and Register a new user to hack the WebGoat
http://localhost:8080/WebGoat


Login Page
Burp Suite Community Edition
127.0.0.1:8080/WebGoat/login
WEBGOAT
Username
lbsproject
Password
Sign in
Register new user


WebGoat
Burp Suite Community Edition
127.0.0.1:8080/WebGoat/start.mvc#lesson/WebGoatIntroduction.lesson
WEBGOAT
WebGoat
Introduction
General
(A1) Injection
(A2) Broken Authentication
(A3) Sensitive Data Exposure
(A4) XML External Entities (XXE)
(A5) Broken Access Control
(A7) Cross-Site Scripting (XSS)
(A8) Insecure Deserialization
(A9) Vulnerable Components
(A8:2013) Request Forgeries
Client side
Challenges
Reset lesson
What is WebGoat?
WebGoat is a deliberately insecure application that allows interested developers just like you to test vulnerabilities commonly found in Java-based applications that use common and popular open source components.
Now, while we in no way condone causing intentional harm to any animal, goat or otherwise, we think learning everything you can about security vulnerabilities is essential to understanding just what happens when even a small bit of unintended code gets into your applications.
What better way to do that than with your very own scapegoat?
Feel free to do what you will with him. Hack, poke, prod and if it makes you feel better, scare him until your heart's content. Go ahead, and hack the goat. We promise he likes it.
Thanks for your interest!
The WebGoat Team
Language Based Security - Chalmers
Type here to search

4.2 Burp Suit Proxy Setup

Step 1:
Install Burp Suit Proxy from https://portswigger.net/burp/communitydownload

Step2:
Launch the Burp Suit Proxy. Run it .

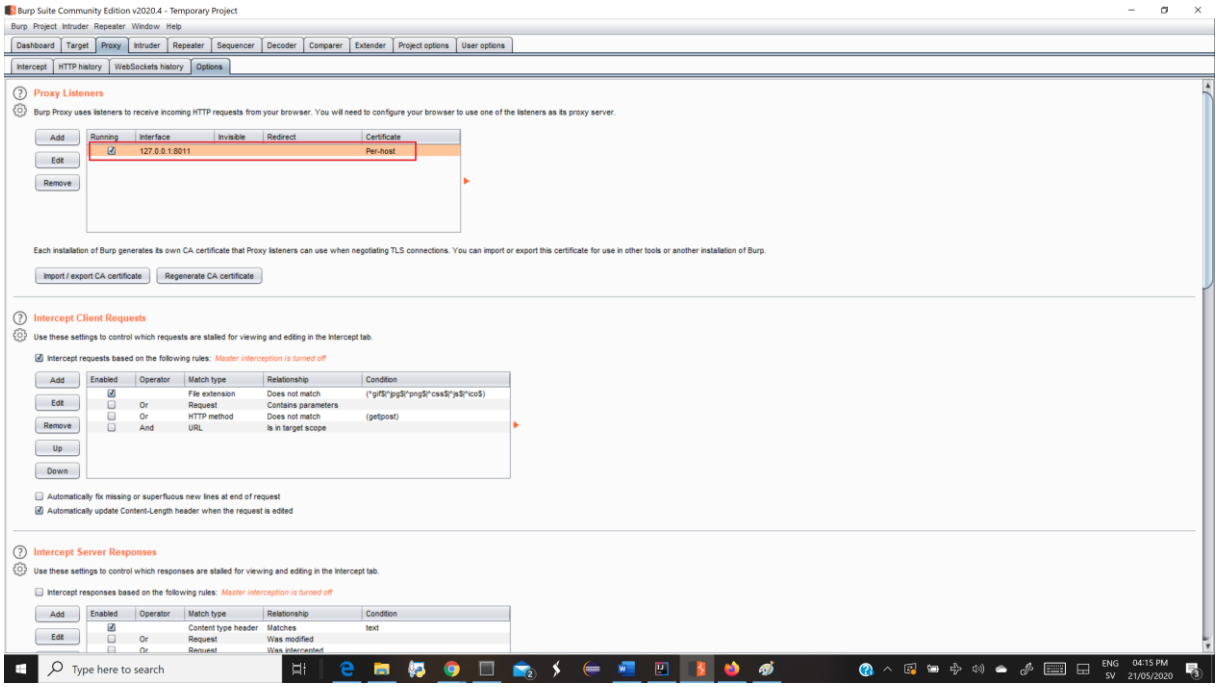


Fig: Burp Suit Proxy Running in Port 8011

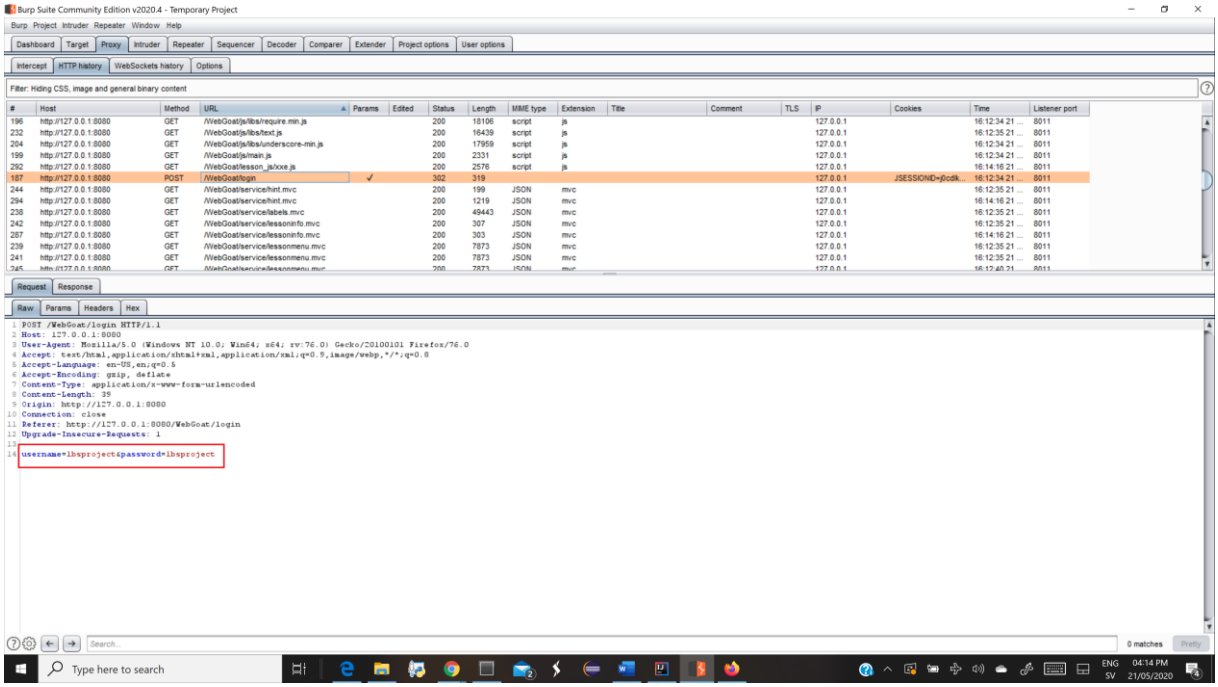


Fig: Tracing Username and Password via Burp Suit Proxy Traffic

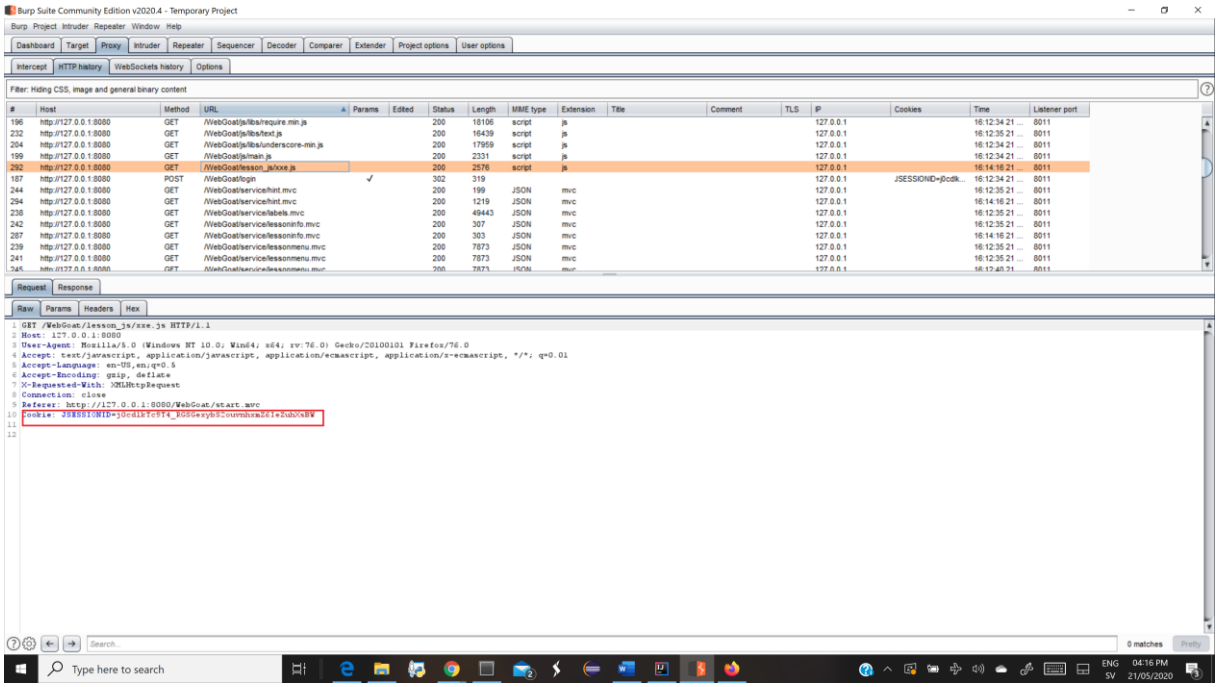


Fig: Tracing Session ID Burp Suit Proxy

4.3 Vulnerability and Countermeasures

4.3.1 XML External Entities(XXE) Attack

We used WebGoat application to do hands on with XML External Entities Attack. Access the XML External Entities menu in the left side of the landing page of WebGoat.The threat model is represented below.

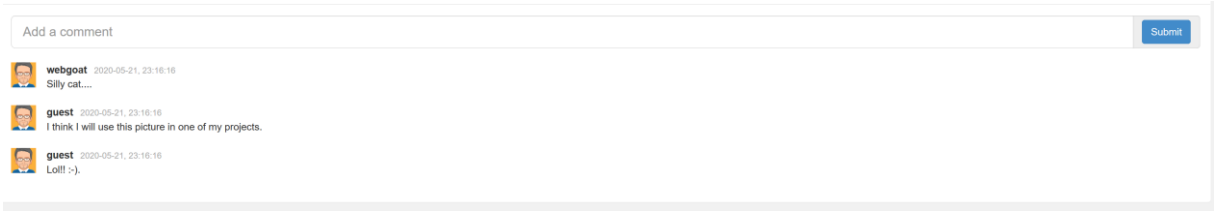


Fig: Threat Model

Step1: Add a test comment and hit submit

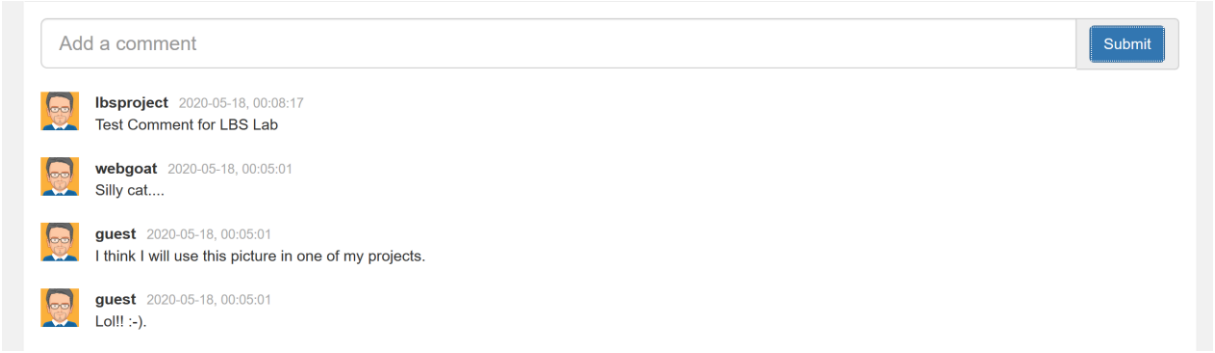


Fig: Making Test Comment for LBS Lab

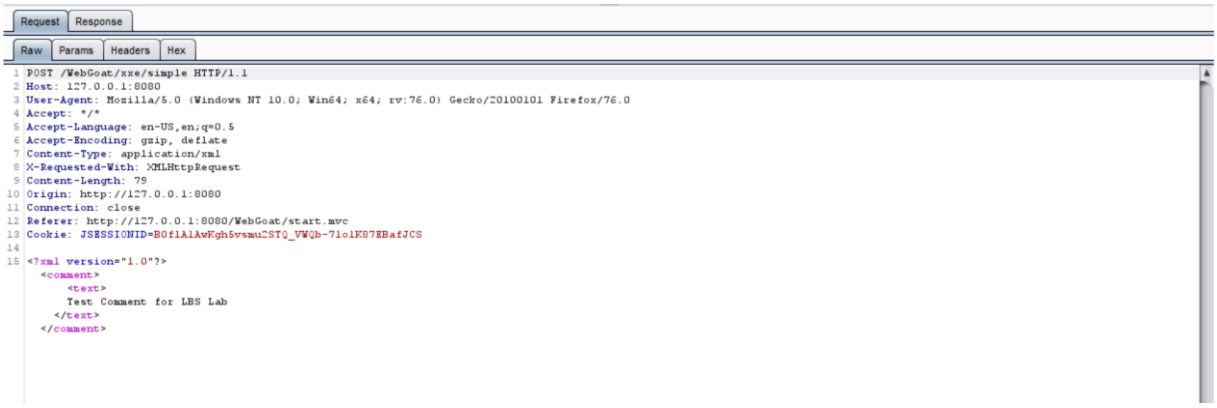


Fig: Traffic in Burp Suit Proxy

Step2:

We tried adding new comments from the Burp suite proxy.

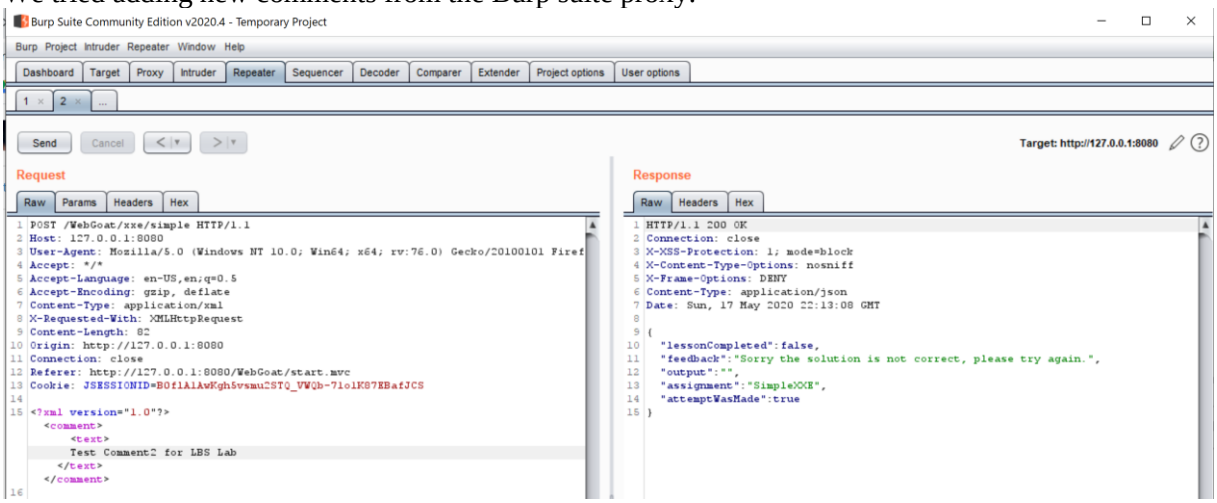


Fig: Adding Test Comment2 for LBS Lab

We have found that the comment is visible in the application. Thus, we were able to manipulate the traffic and insert comment in the application via Burp suit proxy

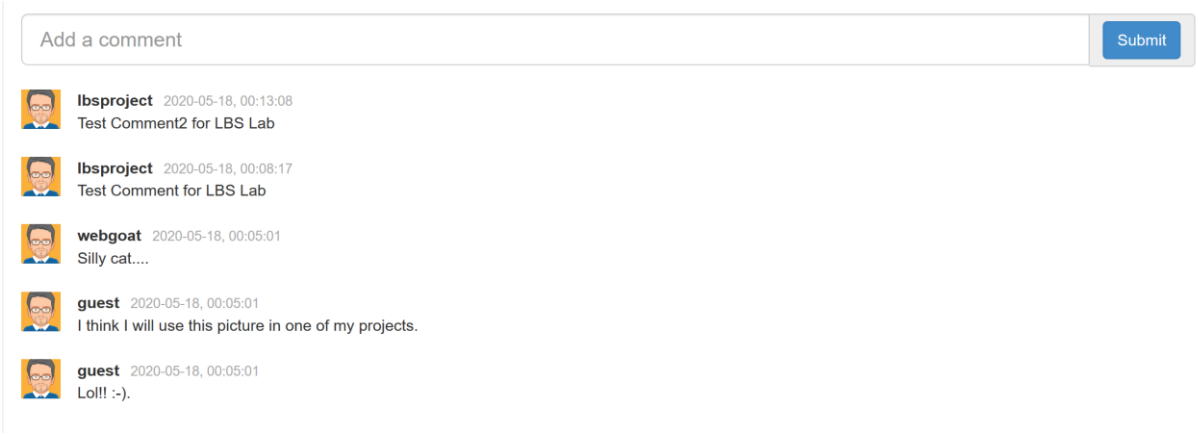


Fig: Test Comment2 for LBS Lab inserted from Burp Suit Proxy

Step3: We tried reading the files from the local systems via xml injection .

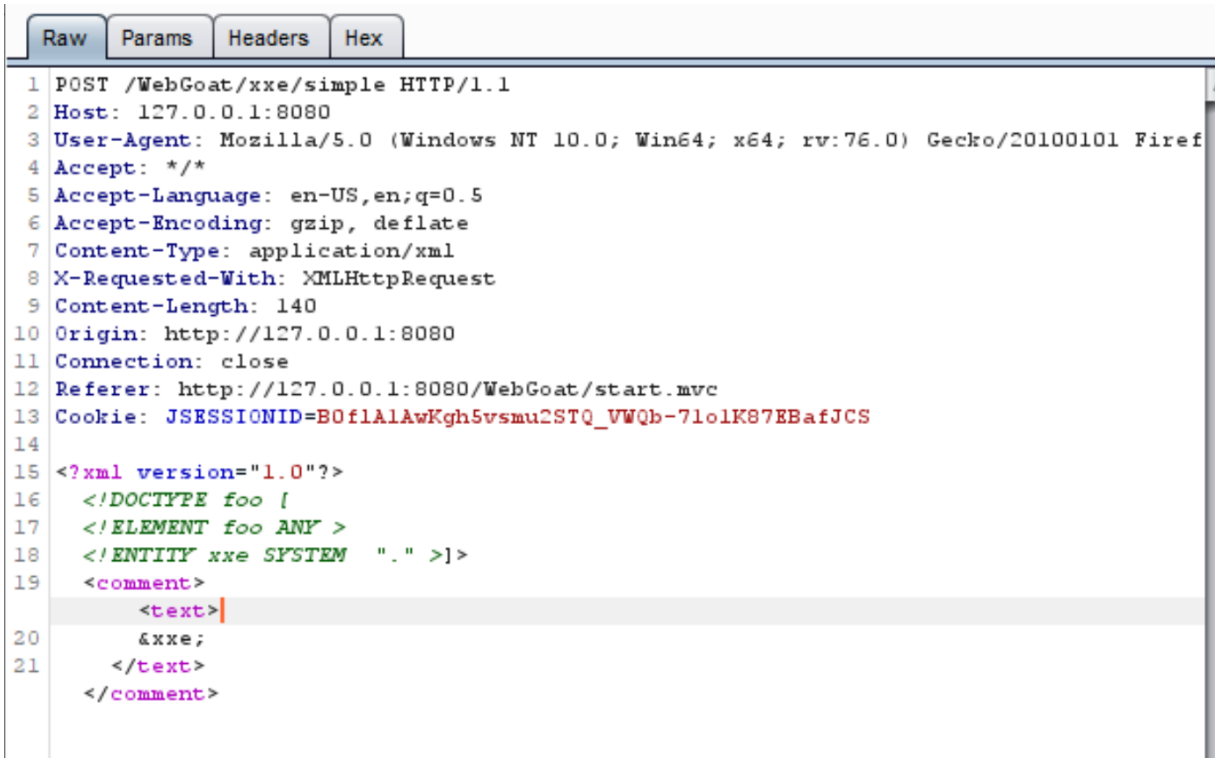


Fig: XML Attack from Burp Suit Proxy

The attack went successful. We were able to read the directory thus the data is in danger

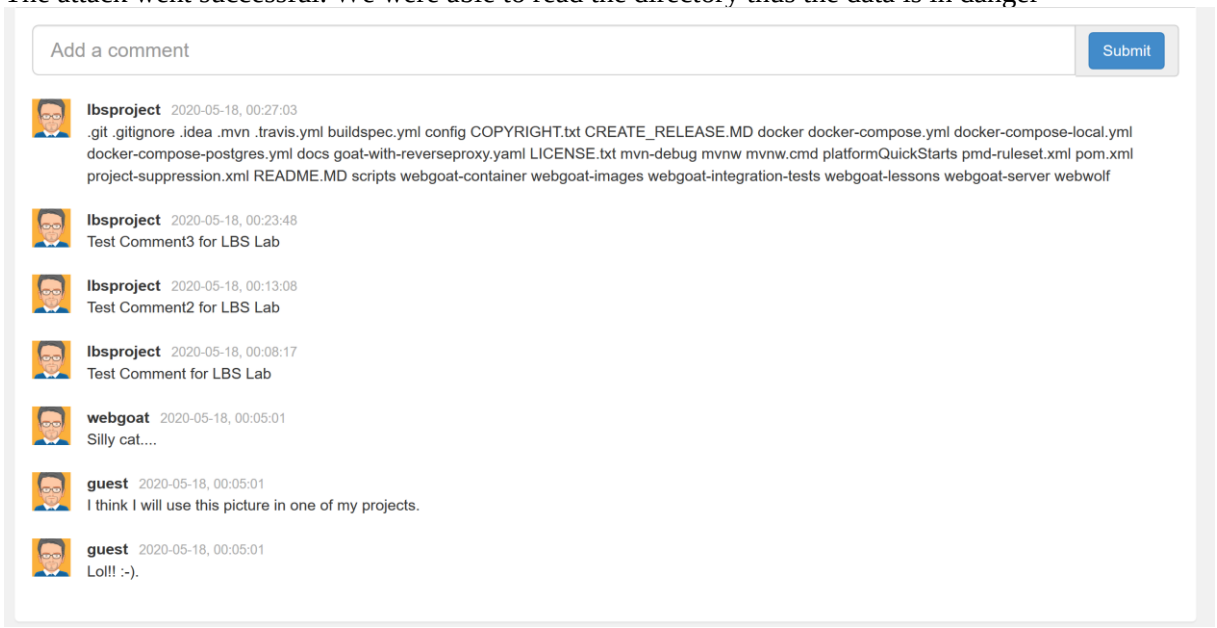


Fig: Printing the content of the directory in the local system

Step4: We tried reading any specific secret files in the system where the application deployed

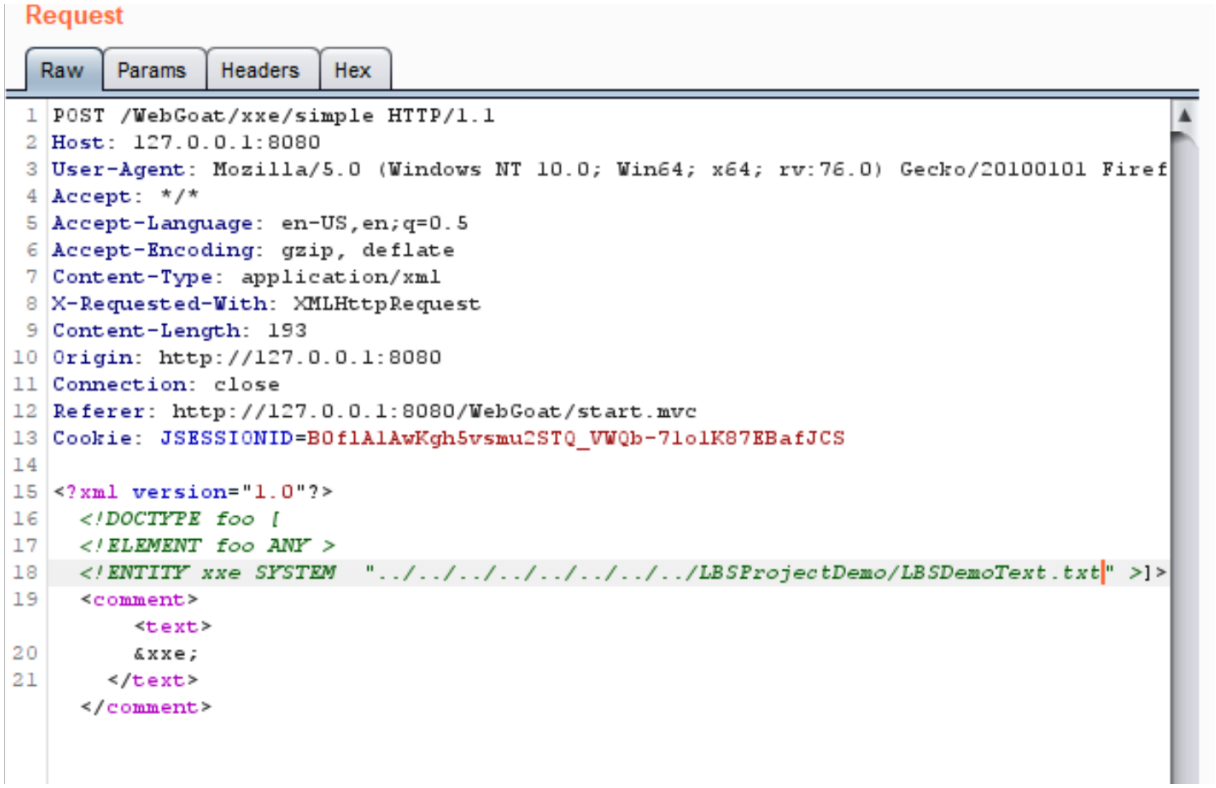


Fig: XML attack to read a secret text file

Attack went successful. We were able to read the secret text.

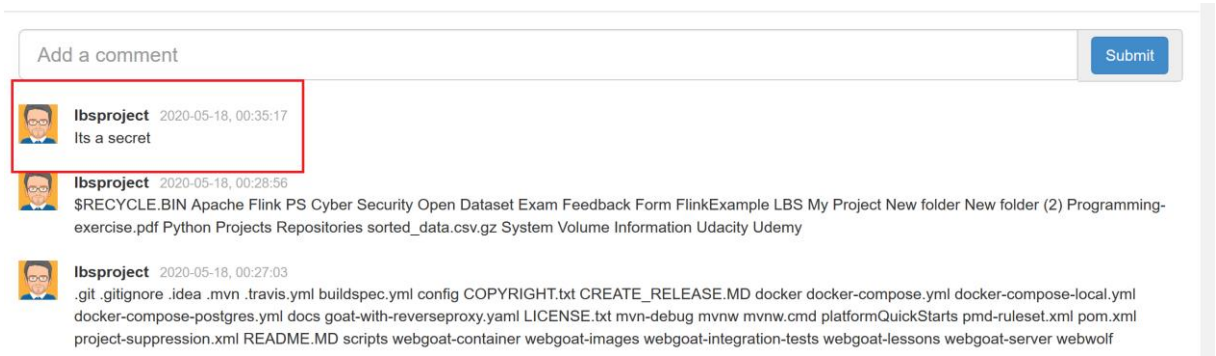


Fig: The content of the LBSDemoText.txt is added as the comment

XML Injection via Restful API

In Rest frameworks, the server will eb able to accept data formats. It may be better option to use , but still the vulnerability can exist in JSON Endpoints

Step 1: Pass the input via REST API.

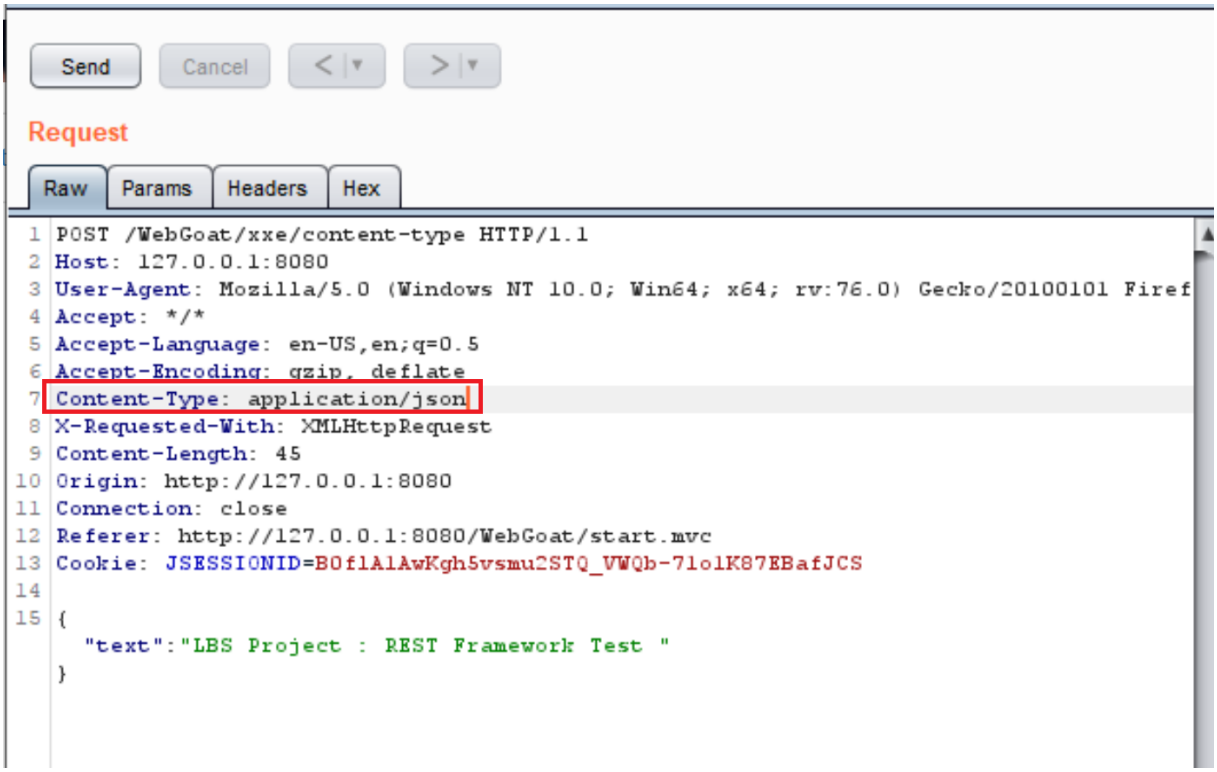


Fig: REST has the content type that accepts JSON

Step 2: We manually edited the content type to application/xml. Then hit Send.

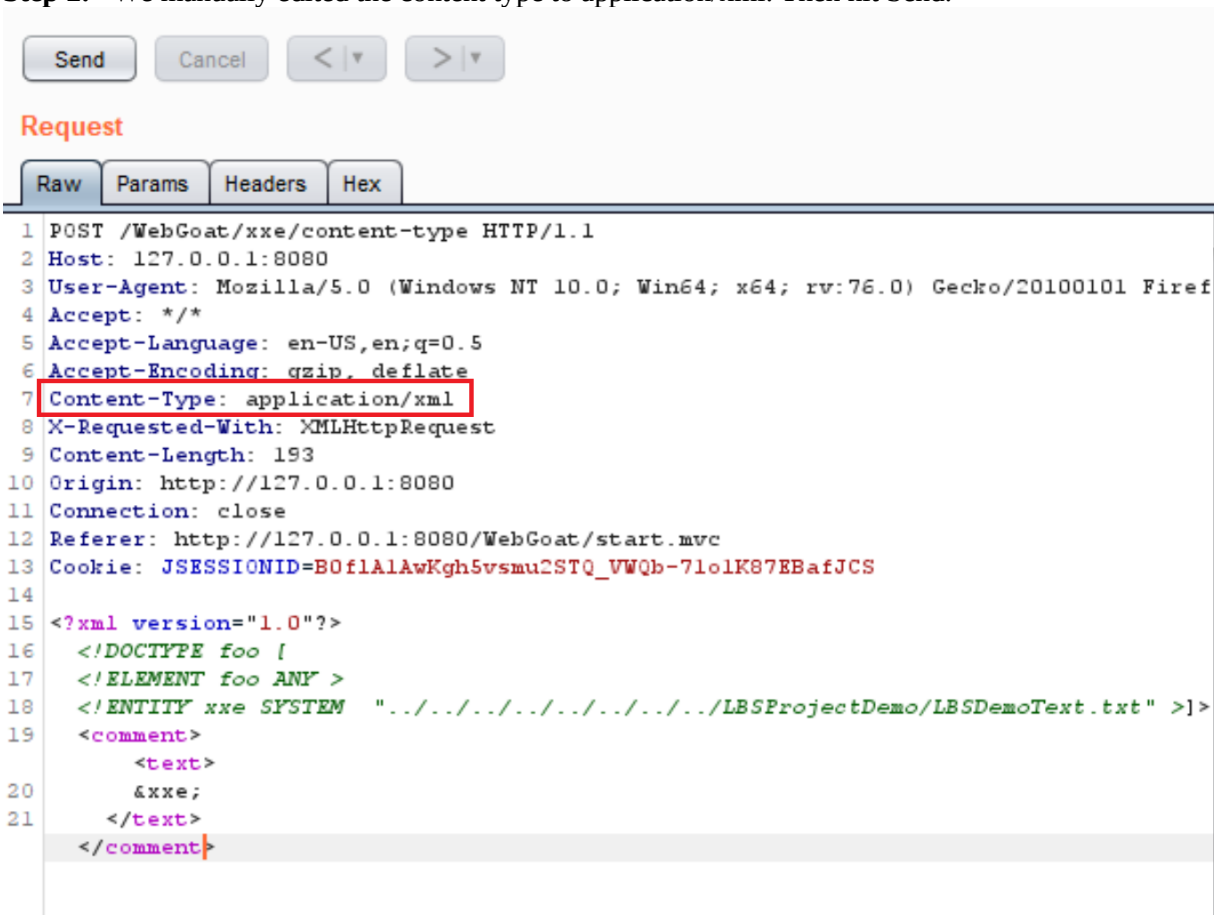
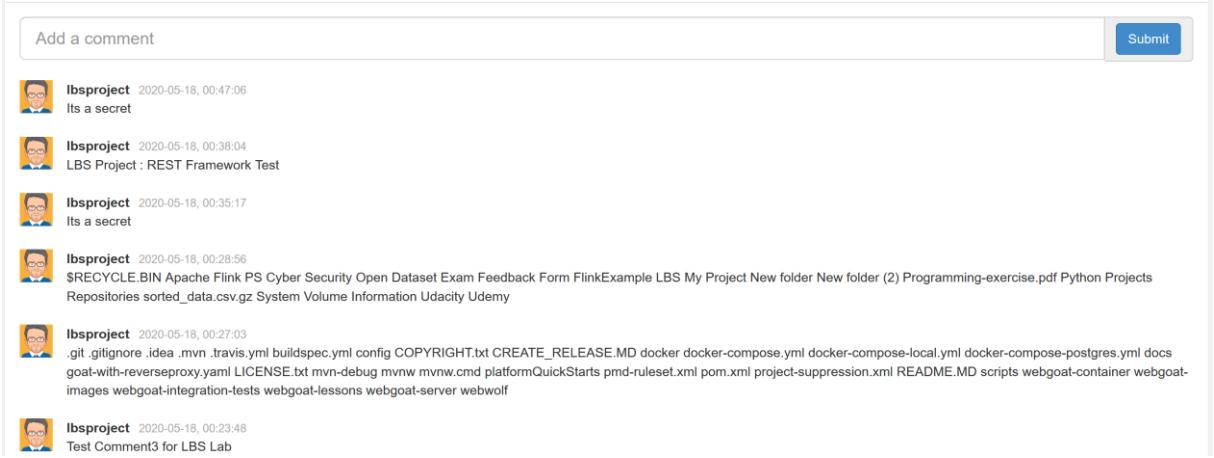


Fig: Content-Type is manually changed to application/xml

Step3: Attack Successful! The secret is printed from the path we modified via XML Injection.



4.3.2 XML External Entities(XXE) - Countermeasures

```java
@RestController
@AssignmentHints({"xxe.hints.content.type.xxe.1", "xxe.hints.content.type.xxe.2"})
public class ContentTypeAssignment extends AssignmentEndpoint {

    private static final String[] DEFAULT_LINUX_DIRECTORIES = {"usr", "etc", "var"};
    private static final String[] DEFAULT_WINDOWS_DIRECTORIES = {"Windows", "Program Files (x86)", "Program Files"};

    @Value("${webgoat.server.directory}")
    private String webGoatHomeDirectory;
    @Autowired
    private WebSession webSession;
    @Autowired
    private Comments comments;

    /*Restricting the application to consume only JSON Value type , as its the only expected type for us */
    @PostMapping(path = "xxe/content-type", consumes = MediaType.APPLICATION_JSON_VALUE, produces = MediaType.APPLICATION_JSON_VALUE)
    @ResponseBody
    public AttackResult createNewUser(@RequestBody String commentStr, @RequestHeader("Content-Type") String contentType) throws Exception {
        AttackResult attackResult = failed( assignment: this).build();
```

Fig: Restrict the application to consume only JSON Type Value in ContentTypeAssignment.java

```java
    protected Comment parseXml(String xml) throws Exception {
        JAXBContext jc = JAXBContext.newInstance(Comment.class);

        XMLInputFactory xif = XMLInputFactory.newFactory();

        /*Without DTD , There is no necessity for external entities*/
        //xif.setProperty(XMLInputFactory.IS_SUPPORTING_EXTERNAL_ENTITIES, true);

        /*As we are not doing any specific validation , Disabling the below line*/
        //xif.setProperty(XMLInputFactory.IS_VALIDATING, false);

        /*Disable the support for DTD if its not required */
        xif.setProperty(XMLInputFactory.SUPPORT_DTD, false);
        XMLStreamReader xsr = xif.createXMLStreamReader(new StringReader(xml));

        Unmarshaller unmarshaller = jc.createUnmarshaller();
        return (Comment) unmarshaller.unmarshal(xsr);
    }
```

Fig: Disable the support for DTD, Remove unnecessary external entities

We allowed only JSON data formats and avoided the serialization of sensitive data whenever the possibility exists. It is important to use annotations carefully to enable only required privileges and to control the incoming data format. We used annotations to accept only JSON values strictly. As recommended by OWAP Cheat sheet, we disabled the XML external entity and DTD processing in all XML parsers in the applications. Validated the server-side inputs, thus they must be filtered and sanitized to protect hostile data within XML documents, headers, or nodes.

4.3.3 Authentication Bypass Attack

Access the **Broken Authentication → Authentication Bypas**s option in the left side of the landing page of Web Goat.

Step 1:

Answer the questions randomly and click submit. Threat model is represented below.

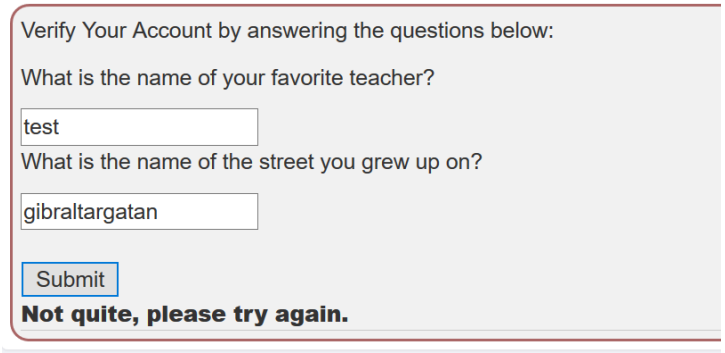


Fig: Threat Model

Step 2:
Intercept the traffic in Burp Suite proxy

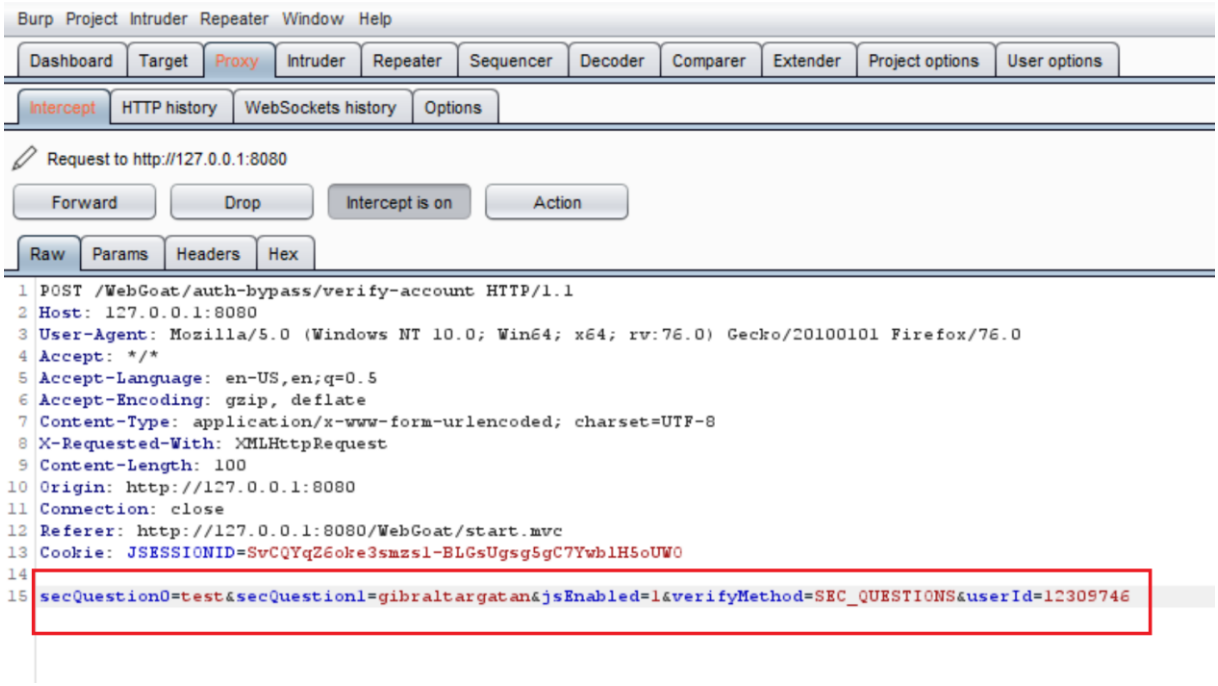


Fig: Traffic Analysis in Burp Suit Proxy

Step3: Lets automate a brute force attack using Intruder Option.

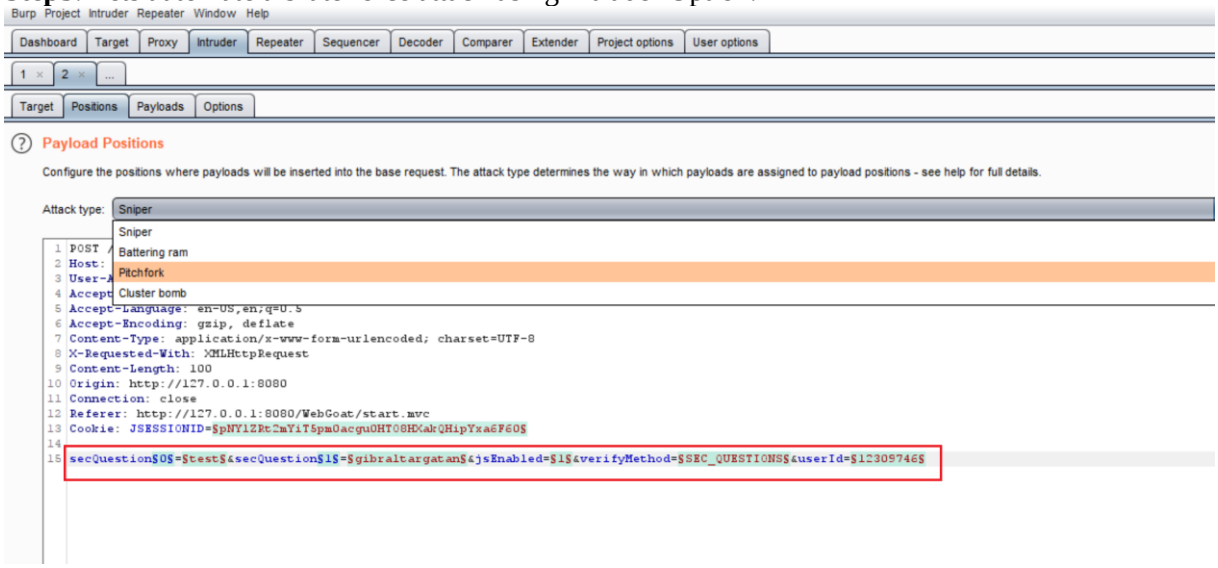


Fig: Using the Intruder Attack types to automate Bruteforce

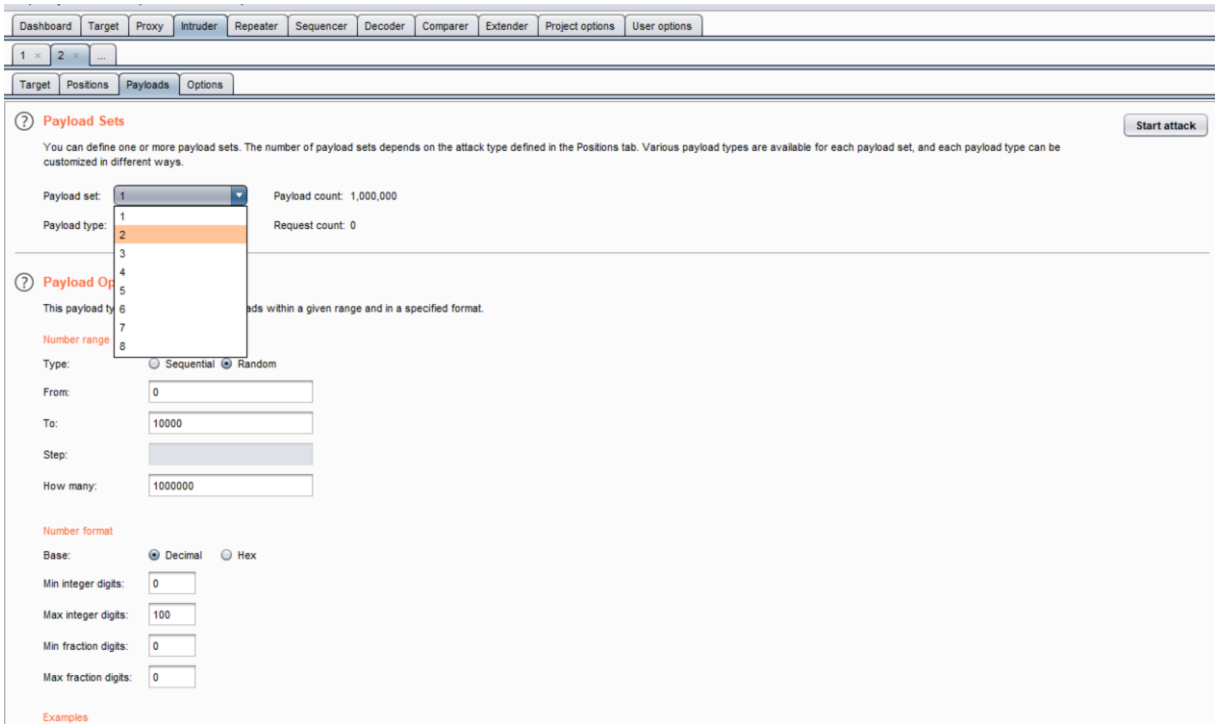


Fig: Intruder Attack

Thus, Brute force will fetch us the hack.

Step 3: We can redo it by manually as well, By Changing the parameters as follows

secQuestion**0**=&secQuestion**1**=&jsEnabled=1&verifyMethod=SEC_QUESTIONS&userId=yourid **to** secQuestion**2**=&secQuestion**3**=&jsEnabled=1&verifyMethod=SEC_QUESTIONS&userId=yourid.

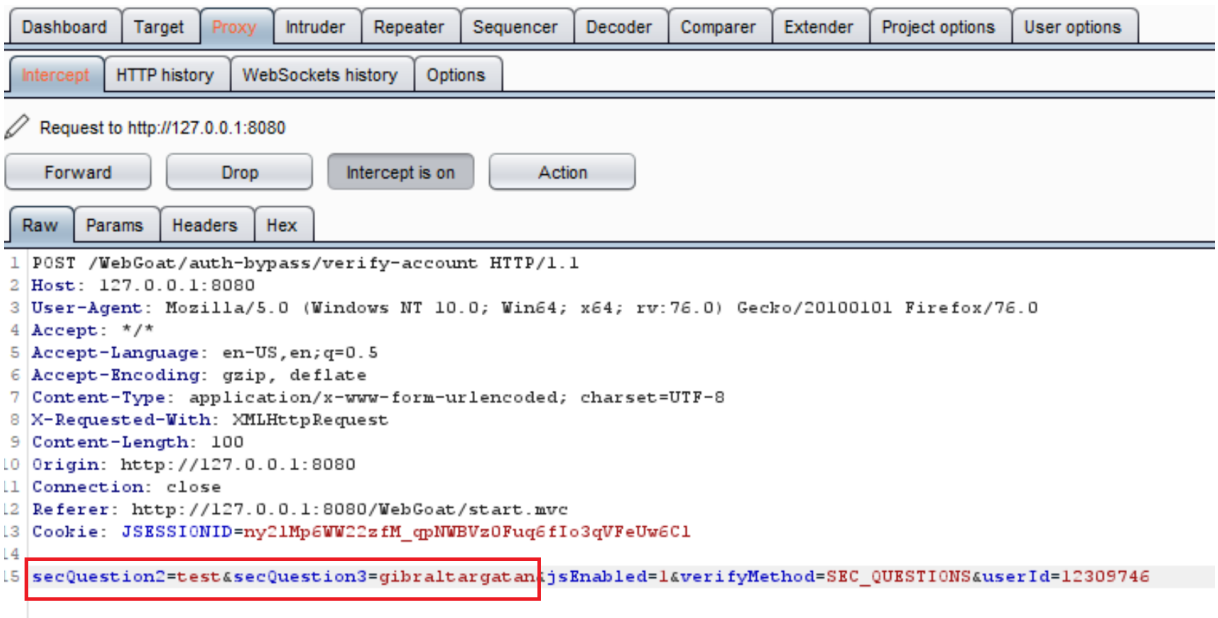


Fig: Forwarding the data from Burp Suite Proxy

Step 4:

When we hit forward button under Intercept option of Burp suite proxy, the page gets successfully exploited and prompts us to change the password. Just enter the new user password and submit it. The Application is **successfully** broken.

Fig: The Application Prompting the user to change the password

4.3.4 Authentication Attack – Countermeasures

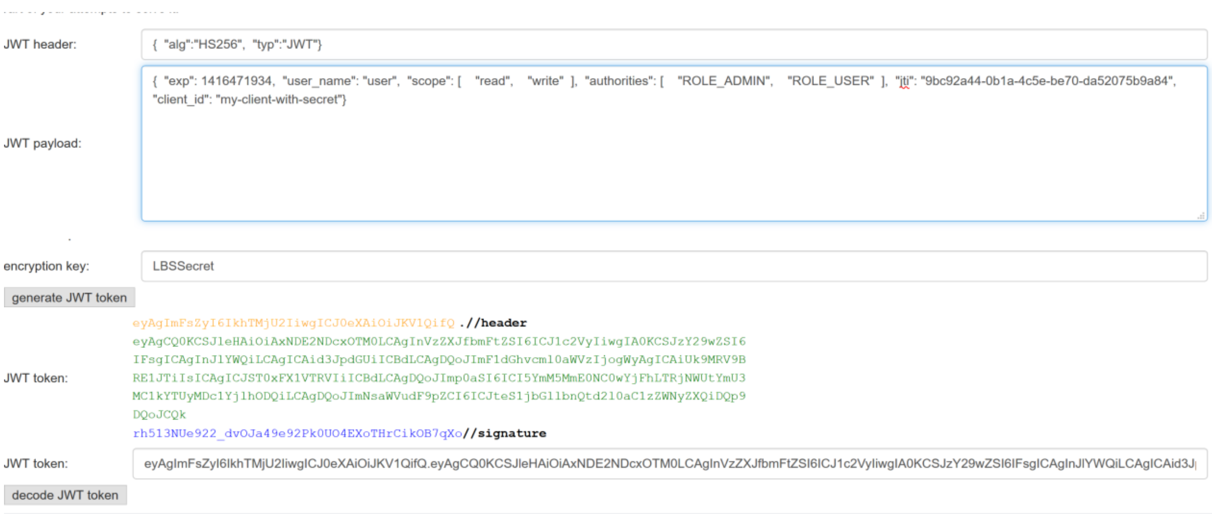
```
public boolean verifyAccount(Integer userId, HashMap<String, String> submittedQuestions) {
    //Avoiding the short circuit if no questions are submitted
    if (submittedQuestions.entrySet().size() != secQuestionStore.get(verifyUserId).size()) {
        return false;
    }
    if (submittedQuestions.containsKey("secQuestion0") && submittedQuestions.get("secQuestion0").equals(secQuestionStore.get(verifyUserId).get("secQuestion0"))) {
        return false;
    }
    if (submittedQuestions.containsKey("secQuestion1") && submittedQuestions.get("secQuestion1").equals(secQuestionStore.get(verifyUserId).get("secQuestion1"))) {
        return false;
    }
    // else
    return true;

}
}
```

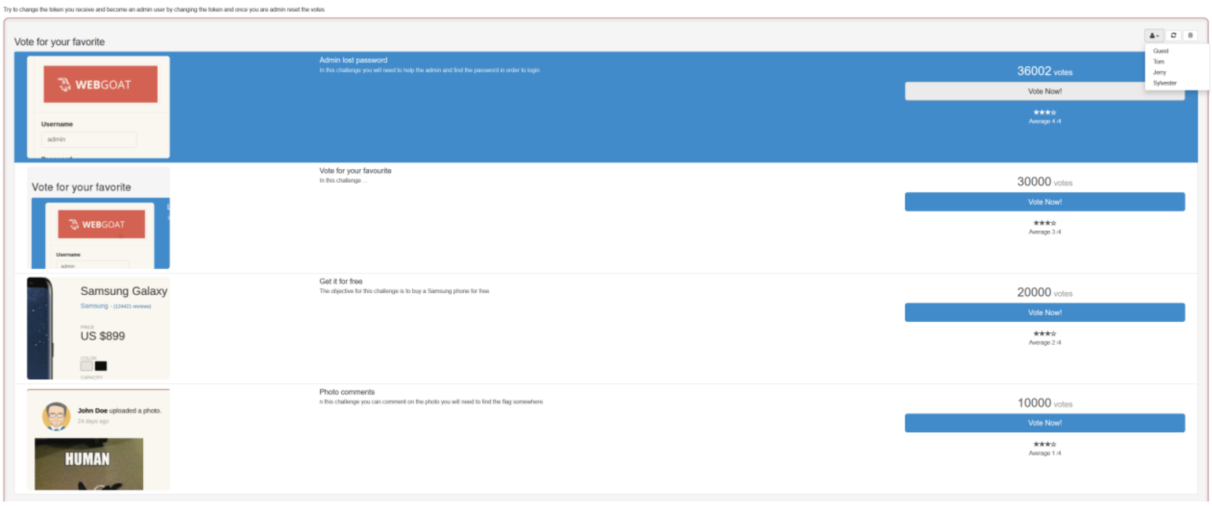
Fig: Always Fail Open

Transaction verification data should be generated server-side. When a significant transaction data is programmatically transmitted to an authorization component, it's very important to enforce a default deny for any client modifications on the transaction data. This significant data that must be verified by the user should be generated and stored on the server before it reaches authorization component without giving any chance for tampering of data by the client. This method shields us from attacks like collecting significant transaction data on client side to manipulate them to fake transaction data in an Application should prevent authorization credentials brute forcing. When transaction authorization credentials are sent to the server for verification, the application must prevent brute forcing.

4.3.5 JSON Web Tokens - Authorization Exploit

The JSON Web token is base64-url-encoded and consists of three parts header.claims.signature.

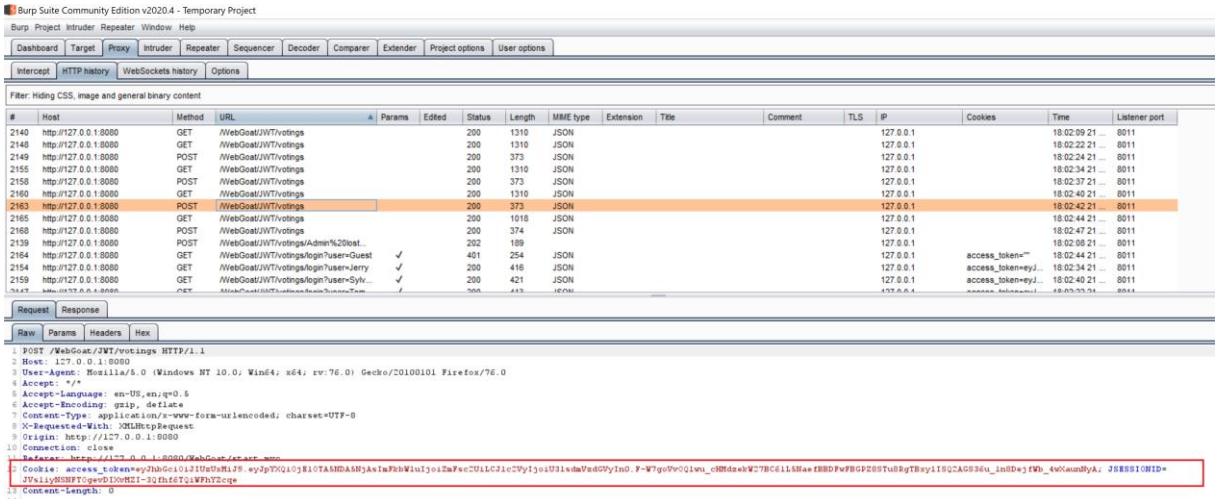


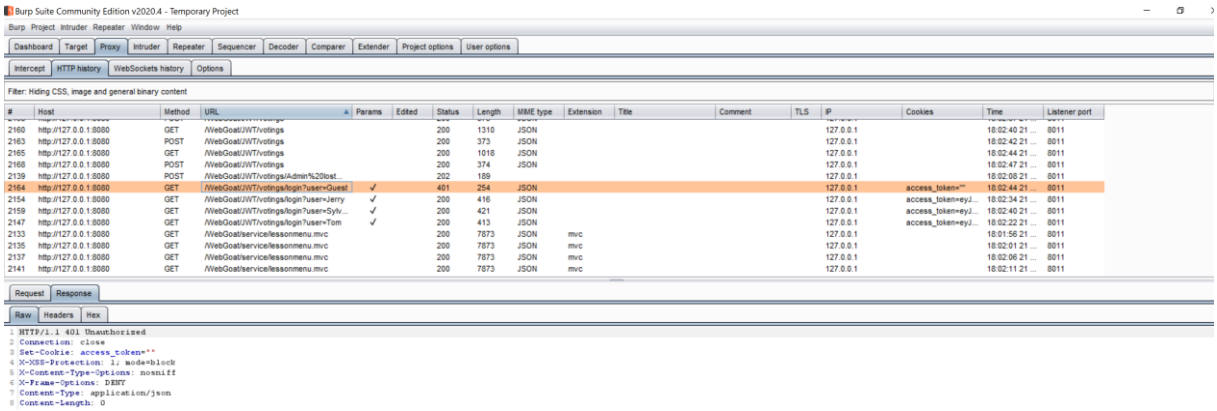


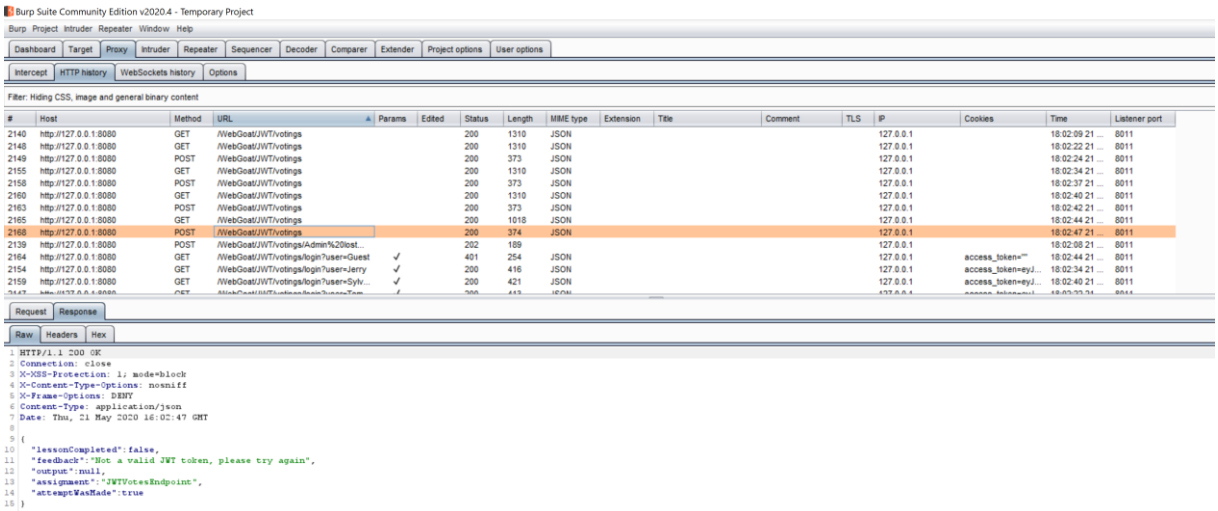
We have some features here to simulate a website that has many different roles. The drop-down has three individual users and a guest. As If we try to vote, I'm not allowed to vote as a guest user. Other three individual users can vote. The votes can be deleted only by admin and nobody else has the access to delete.

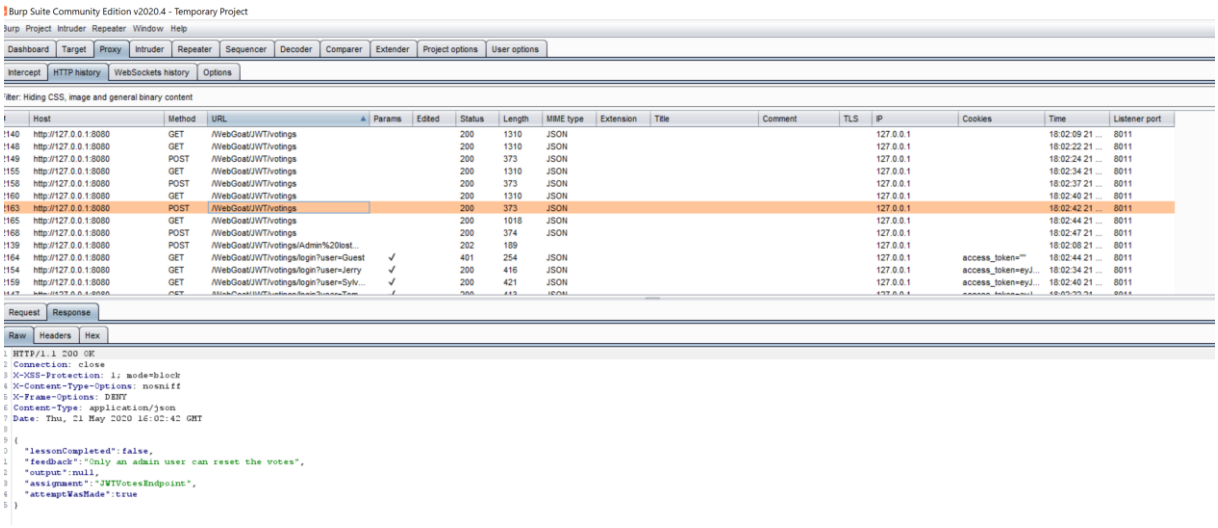
We were looking for a loophole to exploit this feature. We explored the traffic in Burp Suit proxy.

Step 1: Select a different user and look at the token you receive back, use the delete button to reset the votes count.


Burp Suite Community Edition v2020.4 - Temporary Project
Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Extender Project options User options
Intercept HTTP history WebSockets history Options
Filter: Hiding CSS, image and general binary content
Host Method URL Params Edited Status Length MIME type Extension Title Comment TLS IP Cookies Time Listener port
2140 http://127.0.0.1:8080 GET /WebGoat/JWT/votings 200 1310 JSON 127.0.0.1 18:02:09 21 ... 8011
2148 http://127.0.0.1:8080 GET /WebGoat/JWT/votings 200 1310 JSON 127.0.0.1 18:02:22 21 ... 8011
2149 http://127.0.0.1:8080 POST /WebGoat/JWT/votings 200 373 JSON 127.0.0.1 18:02:24 21 ... 8011
2155 http://127.0.0.1:8080 GET /WebGoat/JWT/votings 200 1310 JSON 127.0.0.1 18:02:34 21 ... 8011
2158 http://127.0.0.1:8080 POST /WebGoat/JWT/votings 200 373 JSON 127.0.0.1 18:02:37 21 ... 8011
2160 http://127.0.0.1:8080 GET /WebGoat/JWT/votings 200 1310 JSON 127.0.0.1 18:02:40 21 ... 8011
2163 http://127.0.0.1:8080 POST /WebGoat/JWT/votings 200 373 JSON 127.0.0.1 18:02:42 21 ... 8011
2165 http://127.0.0.1:8080 GET /WebGoat/JWT/votings 200 1018 JSON 127.0.0.1 18:02:44 21 ... 8011
2168 http://127.0.0.1:8080 POST /WebGoat/JWT/votings 200 374 JSON 127.0.0.1 18:02:47 21 ... 8011
2139 http://127.0.0.1:8080 POST /WebGoat/JWT/votings/Admin%20lost... 202 189 127.0.0.1 18:02:08 21 ... 8011
2164 http://127.0.0.1:8080 GET /WebGoat/JWT/votings/login?user=Guest ✓ 401 254 JSON 127.0.0.1 access_token="" 18:02:44 21 ... 8011
2154 http://127.0.0.1:8080 GET /WebGoat/JWT/votings/login?user=Jerry ✓ 200 416 JSON 127.0.0.1 access_token=eyJ... 18:02:34 21 ... 8011
2159 http://127.0.0.1:8080 GET /WebGoat/JWT/votings/login?user=Sylv... ✓ 200 421 JSON 127.0.0.1 access_token=eyJ... 18:02:40 21 ... 8011
Request Response
Raw Params Headers Hex
1 POST /WebGoat/JWT/votings HTTP/1.1
2 Host: 127.0.0.1:8080
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:76.0) Gecko/20100101 Firefox/76.0
4 Accept: */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded; charset=UTF-8
8 X-Requested-With: XMLHttpRequest
9 Origin: http://127.0.0.1:8080
10 Connection: close
11 Referer: http://127.0.0.1:8080/WebGoat/start.mvc
12 Cookie: access_token=eyJhbGciOiJIUzUxMiJ9.eyJpYXQiOjE1OTA5NDA5NjAsImFkbWluIjoiZmFsc2UiLCJleCJpYyIjoiU3lsdmVzdGVyIn0.F-W7goVvOQ1wu_cNMdzekW27BC6iLkNaefBBDFwFBGPZ8STu8gTBxylI8Q2AG83du_in8DejfWb_4wXaunNyA; JSESSIONID=
JVkliyNSNFTOgevDIOnMZI-IQjhf6TQiWFhYZcqe
13 Content-Length: 0
14


Burp Suite Community Edition v2020.4 - Temporary Project
Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Extender Project options User options
Intercept HTTP history WebSockets history Options
Filter: Hiding CSS, image and general binary content
Host Method URL Params Edited Status Length MIME type Extension Title Comment TLS IP Cookies Time Listener port
2160 http://127.0.0.1:8080 GET /WebGoat/JWT/votings 200 1310 JSON 127.0.0.1 18:02:40 21 ... 8011
2163 http://127.0.0.1:8080 POST /WebGoat/JWT/votings 200 373 JSON 127.0.0.1 18:02:42 21 ... 8011
2165 http://127.0.0.1:8080 GET /WebGoat/JWT/votings 200 1018 JSON 127.0.0.1 18:02:44 21 ... 8011
2168 http://127.0.0.1:8080 POST /WebGoat/JWT/votings 200 374 JSON 127.0.0.1 18:02:47 21 ... 8011
2139 http://127.0.0.1:8080 POST /WebGoat/JWT/votings/Admin%20lost... 202 189 127.0.0.1 18:02:08 21 ... 8011
2164 http://127.0.0.1:8080 GET /WebGoat/JWT/votings/login?user=Guest ✓ 401 254 JSON 127.0.0.1 access_token="" 18:02:44 21 ... 8011
2154 http://127.0.0.1:8080 GET /WebGoat/JWT/votings/login?user=Jerry ✓ 200 416 JSON 127.0.0.1 access_token=eyJ... 18:02:34 21 ... 8011
2159 http://127.0.0.1:8080 GET /WebGoat/JWT/votings/login?user=Sylv... ✓ 200 421 JSON 127.0.0.1 access_token=eyJ... 18:02:40 21 ... 8011
2147 http://127.0.0.1:8080 GET /WebGoat/JWT/votings/login?user=Tom ✓ 200 413 JSON 127.0.0.1 access_token=eyJ... 18:02:22 21 ... 8011
2133 http://127.0.0.1:8080 GET /WebGoat/service/lessonmenu.mvc 200 7873 JSON mvc 127.0.0.1 18:01:56 21 ... 8011
2135 http://127.0.0.1:8080 GET /WebGoat/service/lessonmenu.mvc 200 7873 JSON mvc 127.0.0.1 18:02:01 21 ... 8011
2137 http://127.0.0.1:8080 GET /WebGoat/service/lessonmenu.mvc 200 7873 JSON mvc 127.0.0.1 18:02:06 21 ... 8011
2141 http://127.0.0.1:8080 GET /WebGoat/service/lessonmenu.mvc 200 7873 JSON mvc 127.0.0.1 18:02:11 21 ... 8011
Request Response
Raw Headers Hex
1 HTTP/1.1 401 Unauthorized
2 Connection: close
3 Set-Cookie: access_token=""
4 X-XSS-Protection: 1; mode=block
5 X-Content-Type-Options: nosniff
6 X-Frame-Options: DENY
7 Content-Type: application/json
8 Content-Length: 0


Burp Suite Community Edition v2020.4 - Temporary Project
Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Extender Project options User options
Intercept HTTP history WebSockets history Options
Filter: Hiding CSS, image and general binary content
Host Method URL Params Edited Status Length MIME type Extension Title Comment TLS IP Cookies Time Listener port
2140 http://127.0.0.1:8080 GET /WebGoat/JWT/votings 200 1310 JSON 127.0.0.1 18:02:09 21 ... 8011
2148 http://127.0.0.1:8080 GET /WebGoat/JWT/votings 200 1310 JSON 127.0.0.1 18:02:22 21 ... 8011
2149 http://127.0.0.1:8080 POST /WebGoat/JWT/votings 200 373 JSON 127.0.0.1 18:02:24 21 ... 8011
2155 http://127.0.0.1:8080 GET /WebGoat/JWT/votings 200 1310 JSON 127.0.0.1 18:02:34 21 ... 8011
2158 http://127.0.0.1:8080 POST /WebGoat/JWT/votings 200 373 JSON 127.0.0.1 18:02:37 21 ... 8011
2160 http://127.0.0.1:8080 GET /WebGoat/JWT/votings 200 1310 JSON 127.0.0.1 18:02:40 21 ... 8011
2163 http://127.0.0.1:8080 POST /WebGoat/JWT/votings 200 373 JSON 127.0.0.1 18:02:42 21 ... 8011
2165 http://127.0.0.1:8080 GET /WebGoat/JWT/votings 200 1018 JSON 127.0.0.1 18:02:44 21 ... 8011
2168 http://127.0.0.1:8080 POST /WebGoat/JWT/votings 200 374 JSON 127.0.0.1 18:02:47 21 ... 8011
2139 http://127.0.0.1:8080 POST /WebGoat/JWT/votings/Admin%20lost... 202 189 127.0.0.1 18:02:08 21 ... 8011
2164 http://127.0.0.1:8080 GET /WebGoat/JWT/votings/login?user=Guest ✓ 401 254 JSON 127.0.0.1 access_token="" 18:02:44 21 ... 8011
2154 http://127.0.0.1:8080 GET /WebGoat/JWT/votings/login?user=Jerry ✓ 200 416 JSON 127.0.0.1 access_token=eyJ... 18:02:34 21 ... 8011
2159 http://127.0.0.1:8080 GET /WebGoat/JWT/votings/login?user=Sylv... ✓ 200 421 JSON 127.0.0.1 access_token=eyJ... 18:02:40 21 ... 8011
Request Response
Raw Headers Hex
1 HTTP/1.1 200 OK
2 Connection: close
3 X-XSS-Protection: 1; mode=block
4 X-Content-Type-Options: nosniff
5 X-Frame-Options: DENY
6 Content-Type: application/json
7 Date: Thu, 21 May 2020 16:02:47 GMT
8
9 {
10 "lessonCompleted":false,
11 "feedback":"Not a valid JWT token, please try again",
12 "output":null,
13 "assignment":"JWTVotesEndpoint",
14 "attemptWasMade":true
15 }



When we examined the post requests in Burp suite proxy, we found few informative feedbacks. When the individual user gave a delete request, we received a error message that, you need to be an admin for this one. Then we tried deleting the votes as guest user. In the next post request, we found a different error message "you need a JWT token". In the request we can see that no token has been sent.

Step 2 : Decode the token and look at the contents

We copied the access token and explored ways to tamper it safe. Just we copied the token to https://gchq.github.io/

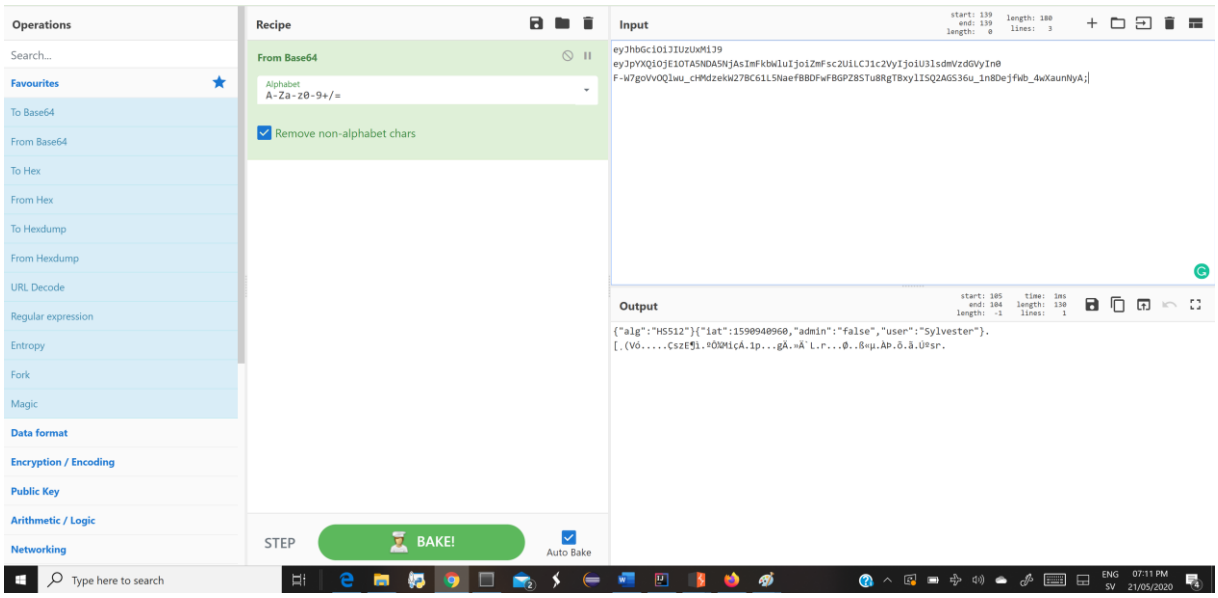


Fig: Decoding JWT

We found that the JWT token header contains the Algorithm HS512. Its iat is some number and username is Sylvester. He is not admin, so admin is given with a false flag. We were looking for a way to make ourselves admin without breaking the signature by making some modification in the JWT.

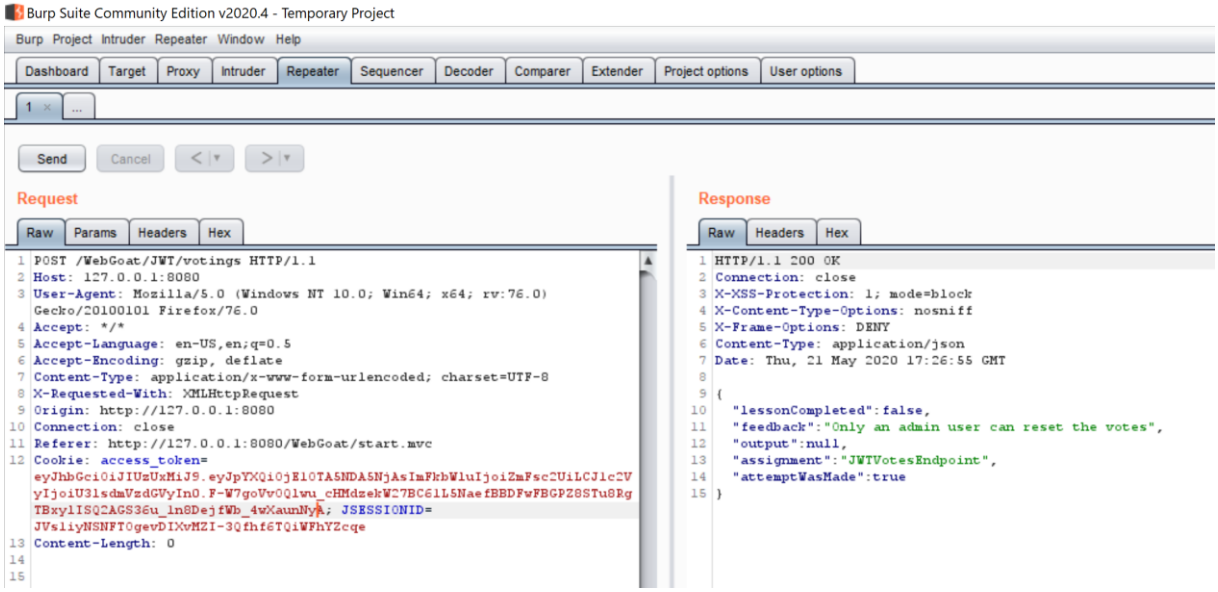


Fig: Repeater

Step 3: Modify the token contents and replace the cookie before sending the request to get the votes. Change the admin field to true in the token.

We sent the post request to the repeater. We decoded this via decoder utility in Burp suit proxy as follows:

Original:

{"alg":"HS512"}{"iat":1590940960,"admin":"false","user":"Sylvester"}.

Modified:

{"alg":"HS512"}{"iat":1590940960,"admin":"true","user":"Sylvester"}.

Fig: Modified JWT

We found that the algorithm is HS512 is optimal, so possibility of changing is bleak. But we found an algorithm called “none” and we tried using it in the header.

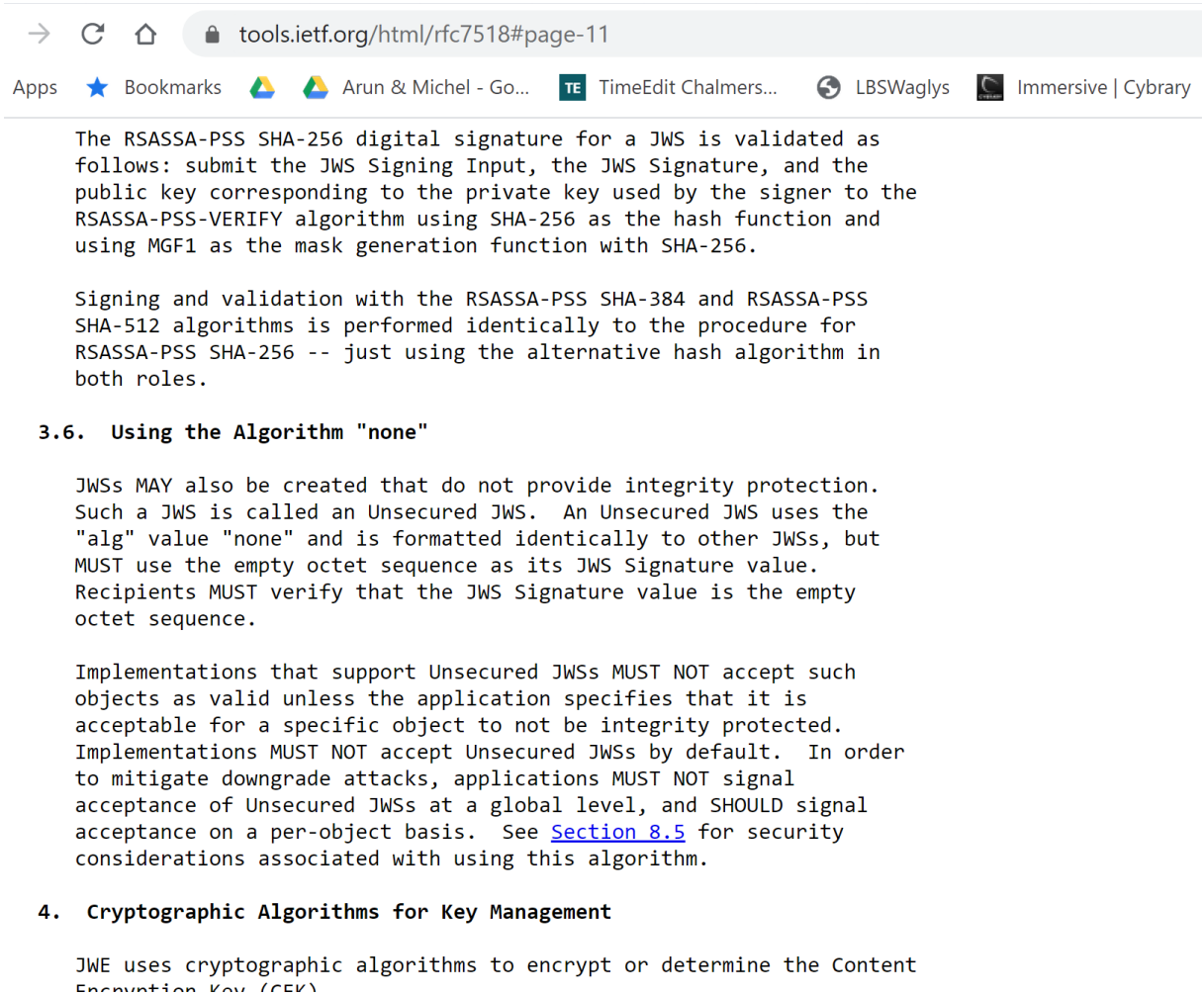


Fig: Algorithm Standards for JWT Header

Re Modified header and claim

{"alg":"none"}{"iat":1590940960,"admin":"true","user":"Sylvester"}.

Encoded Header :

Encoded header: eyJhbGciOiJub25lIn0

Encoded Claim: eyJpYXQiOjE1OTA5NDA5NjAsImFkbWluIjoidHJ1ZSIsInVzZXIiOiJTeWx2ZXN0ZXIifQ

Encoded Signature: Left Empty

This Header.claim.signature will be like

eyJhbGciOiJub25lIn0.
eyJpYXQiOjE1OTA5NDA5NjAsImFkbWluIjoidHJ1ZSIsInVzZXIiOiJTeWx2ZXN0ZXIifQ.

Step 4: Submit the token by changing the algorithm to None and remove the signature

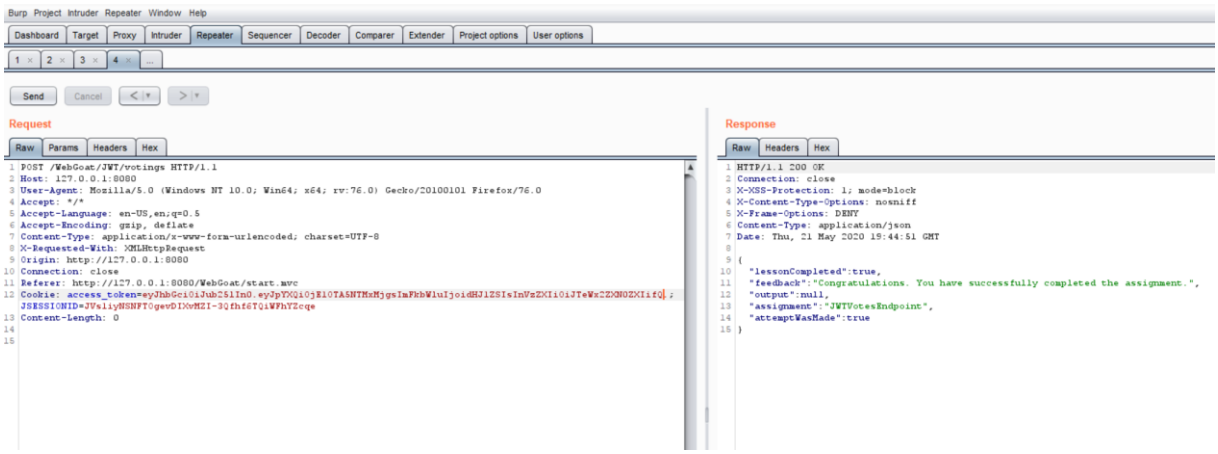


Fig: Result

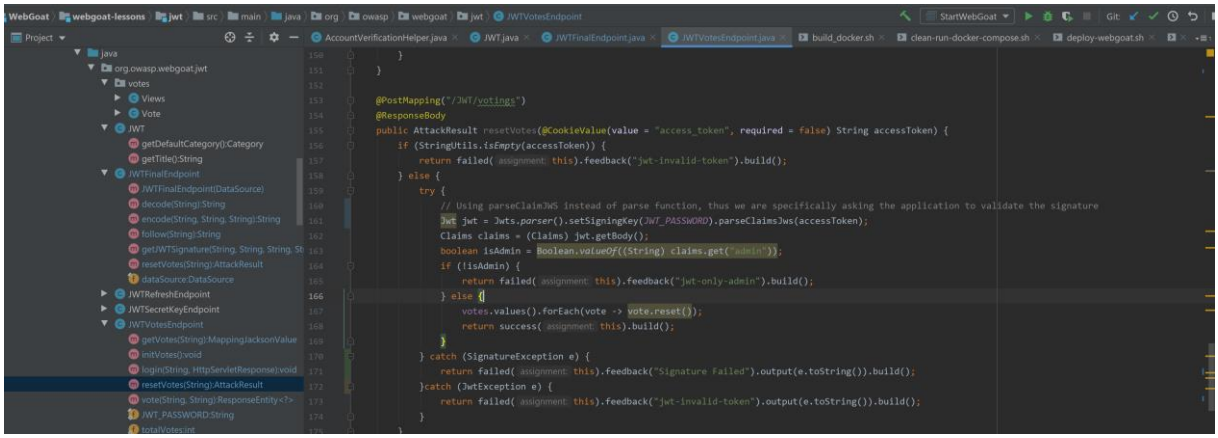
Step 5: Attack successful!

4.3.6 JSON Web Tokens – Authorization Countermeasure

By studying about the specification of JWT, we found that to be a valid token, it is not mandatory for JWT tokens to have signatures. By debugging the Web Goat application, it is evident that the parse JWT function simply parse the access token into JWT or JWS based on the requirement and it didn't care to validate the signatures. At the same time, if it does not witness the algorithm in the header of the JWT, it will fail in that case as well.

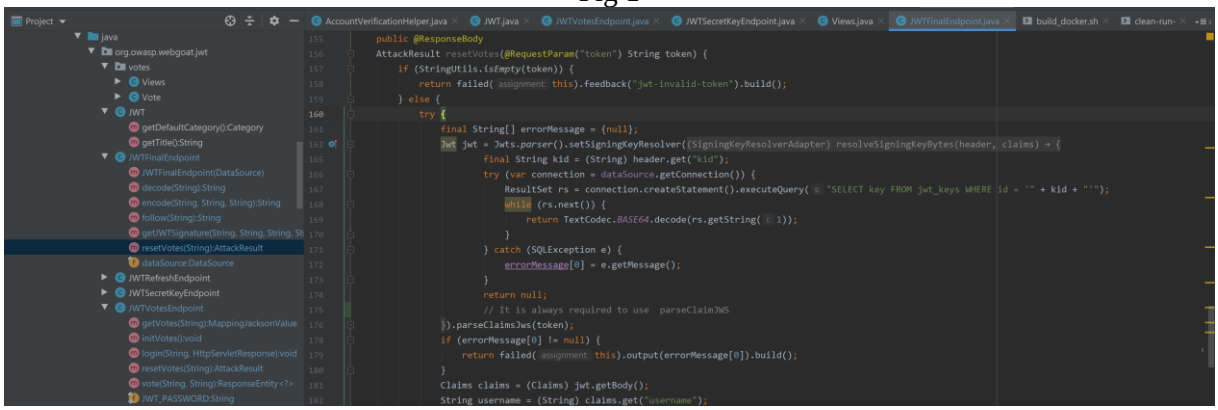
```
try {
   // Using parseClaimJWS instead of parse function, thus we are specifically asking the application to validate
the signature
   Jwt jwt = Jwts.parser().setSigningKey(JWT_PASSWORD).parseClaimsJws(accessToken);
   Claims claims = (Claims) jwt.getBody();
   boolean isAdmin = Boolean.valueOf((String) claims.get("admin"));
   if (!isAdmin) {
      return failed(this).feedback("jwt-only-admin").build();
   } else {
      votes.values().forEach(vote -> vote.reset());
      return success(this).build();
   }
} catch (SignatureException e) {
   return failed(this).feedback("Signature Failed").output(e.toString()).build();
}catch (JwtException e) {
   return failed(this).feedback("jwt-invalid-token").output(e.toString()).build();
```

We made patch in the files JWTVotesEndpoint.java and tested the changes and committed to final Git patch Project. Thus, a new Signature Failed exception will be thrown if someone tampers the JWT.



```
@PostMapping("/JWT/votings")
@ResponseBody
public AttackResult resetVotes(@CookieValue(value = "access_token", required = false) String accessToken) {
    if (StringUtils.isEmpty(accessToken)) {
        return failed( assignment: this).feedback("jwt-invalid-token").build();
    } else {
        try {
            // Using parseClaimJWS instead of parse function, thus we are specifically asking the application to validate the signature
            Jwt jwt = Jwts.parser().setSigningKey(JWT_PASSWORD).parseClaimsJws(accessToken);
            Claims claims = (Claims) jwt.getBody();
            boolean isAdmin = Boolean.valueOf((String) claims.get("admin"));
            if (!isAdmin) {
                return failed( assignment: this).feedback("jwt-only-admin").build();
            } else {
                votes.values().forEach(vote -> vote.reset());
                return success( assignment: this).build();
            }
        } catch (SignatureException e) {
            return failed( assignment: this).feedback("Signature Failed").output(e.toString()).build();
        }catch (JwtException e) {
            return failed( assignment: this).feedback("jwt-invalid-token").output(e.toString()).build();
        }
    }
```

Fig 1



```
public @ResponseBody
AttackResult resetVotes(@RequestParam("token") String token) {
    if (StringUtils.isEmpty(token)) {
        return failed( assignment: this).feedback("jwt-invalid-token").build();
    } else {
        try {
            final String[] errorMessage = {null};
            Jwt jwt = Jwts.parser().setSigningKeyResolver((SigningKeyResolverAdapter) resolveSigningKeyBytes(header, claims) -> {
                    final String kid = (String) header.get("kid");
                    try (var connection = dataSource.getConnection()) {
                        ResultSet rs = connection.createStatement().executeQuery( s: "SELECT key FROM jwt_keys WHERE id = '" + kid + "'");
                        while (rs.next()) {
                            return TextCodec.BASE64.decode(rs.getString( i: 1));
                        }
                    } catch (SQLException e) {
                        errorMessage[0] = e.getMessage();
                    }
                    return null;
                    // It is always required to use  parseClaimJWS
            }).parseClaimsJws(token);
            if (errorMessage[0] != null) {
                return failed( assignment: this).output(errorMessage[0]).build();
            }
            Claims claims = (Claims) jwt.getBody();
            String username = (String) claims.get("username");
```

Fig 2

5. Conclusions

Thus, we explored the XML External Entities (XXE) and Authentication & Authorization pitfalls and countermeasures in detail. We believe, this foundation will give us a foundation to begin our career in security. This project gave us a chance to hands on Burp Suite proxy which is a powerful tool in the field of security. From automating brute force to exploiting the JWT Tokens, it was a great learning experience. Though JWT is a powerful and secure technique, it can be exploited when it is implemented with a bad design. We had a great time understanding the fact that, however powerful the technique may be, implementation is the key , Language Matters. Language Based Security Matters.

References

[1] OWASP Top 10 API Attacks. URL: https://owasp.org/www-project-api-security/

Appendix: