**CSYE 6225 Summer 2019**

Penetration Testing Report

By:

Akshay Bhusare

Pratik Patil

Rajat Acharya

Application Firewall:

A web application firewall (WAF) is an application firewall for HTTP applications. It applies a set

of rules to an HTTP conversation.

While proxies generally protect clients, WAFs protect servers. A WAF is deployed to protect a

specific web application or set of web applications. A WAF can be considered a reverse proxy.

WAFs may come in the form of an appliance, server plugin, or filter, and may be customized to

an application. The effort to perform this customization can be significant and needs to be

maintained as the application is modified.

Assignment Objective:

1. Deploy AWS WAF to the Application Load Balancer (ALB) that fronts your web servers

running on EC2.

2. Use AWS WAF to Mitigate OWASP’s Top 10 Web Application Vulnerabilities.

3. All WAF resources and web security rules should be added to your application cloudformation stack.

Identify and test your application against at least 3 attack vectors that do not exploit UI vulnerabilities.

**1) Broken Authentication and Session Management**

Flaws in the implementation of authentication and session management mechanisms for web applications can lead to exposure of unwanted data, stolen credentials or sessions, and impersonation of legitimate users.9 These flaws are difficult to mitigate using a WAF. Broadly, attackers rely on vulnerabilities in the way client-server communication is implemented. Or they target how session or authorization tokens are generated, stored, transferred, reused, timed-out, or invalidated by your application to obtain these credentials. After they obtain

credentials, attackers impersonate legitimate users and make requests to your web applications using those tokens.

IP Blacklisting: Matches IP addresses that should not be allowed to access content. A hacker or

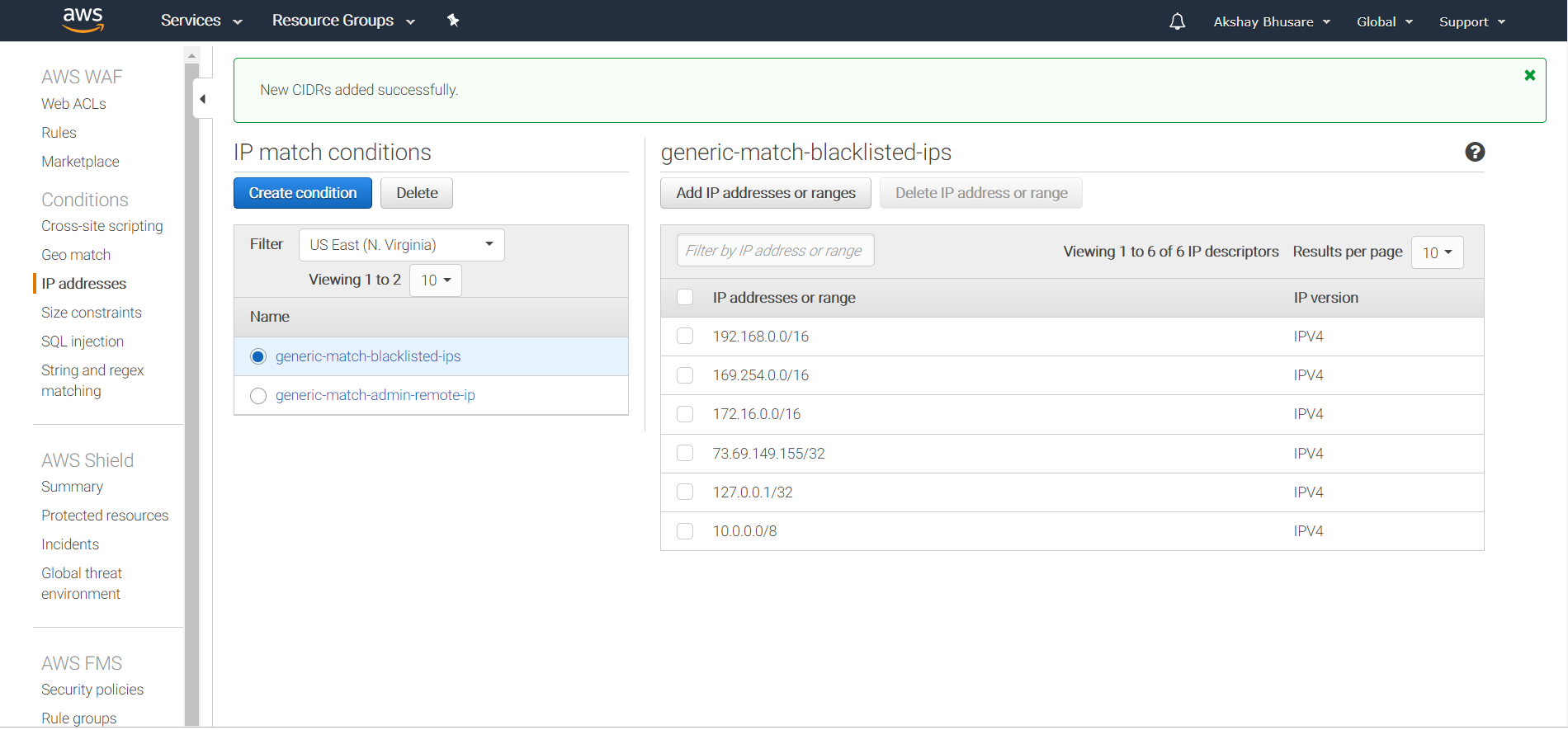
cracker can gain access to a computer and deliver payload or malicious code. Hacker can access personal data etc. and we can stop that by Blacklisting them for a while until the user provides his/her credentials.

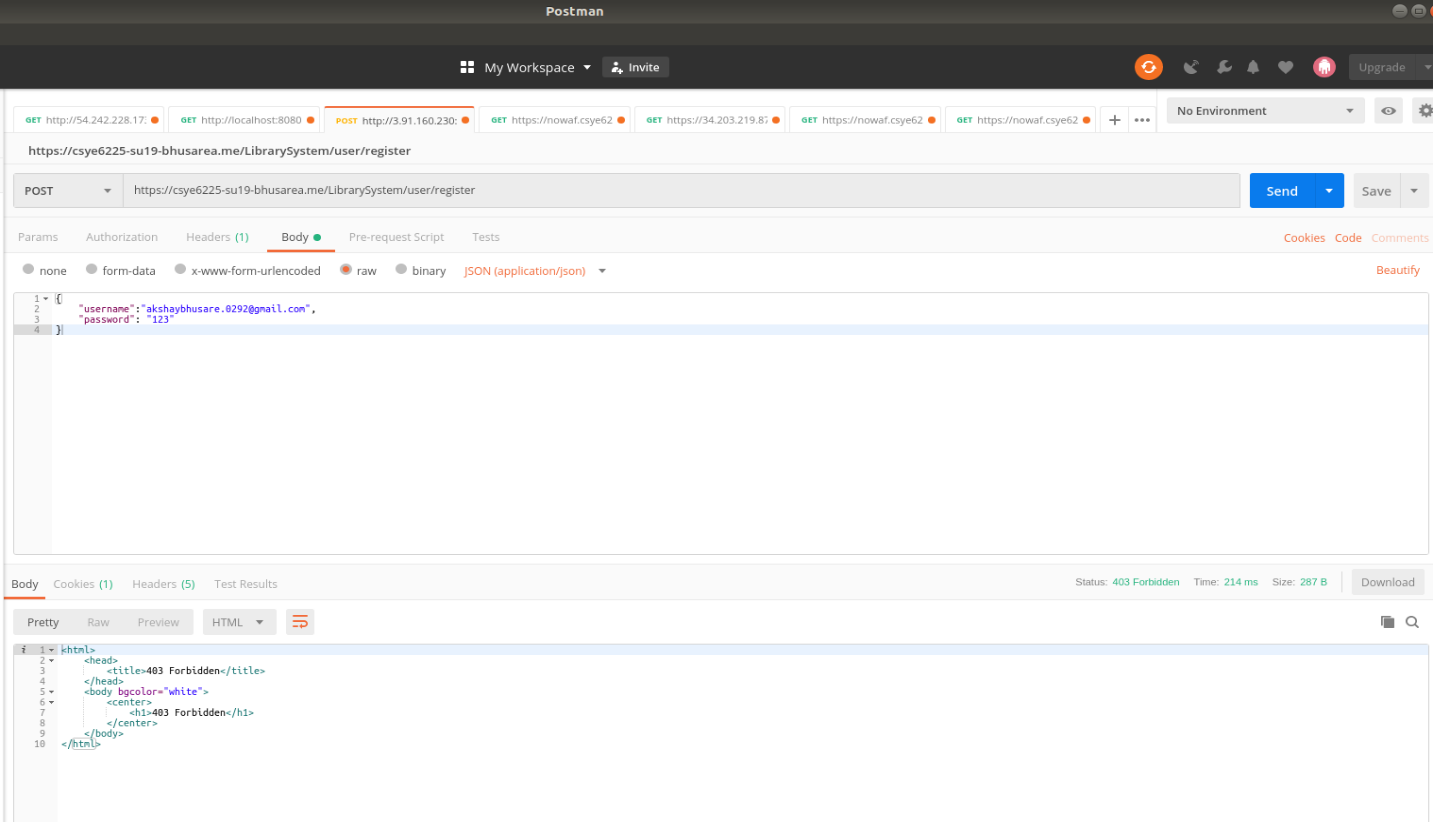
Why this attack vector was chosen ?

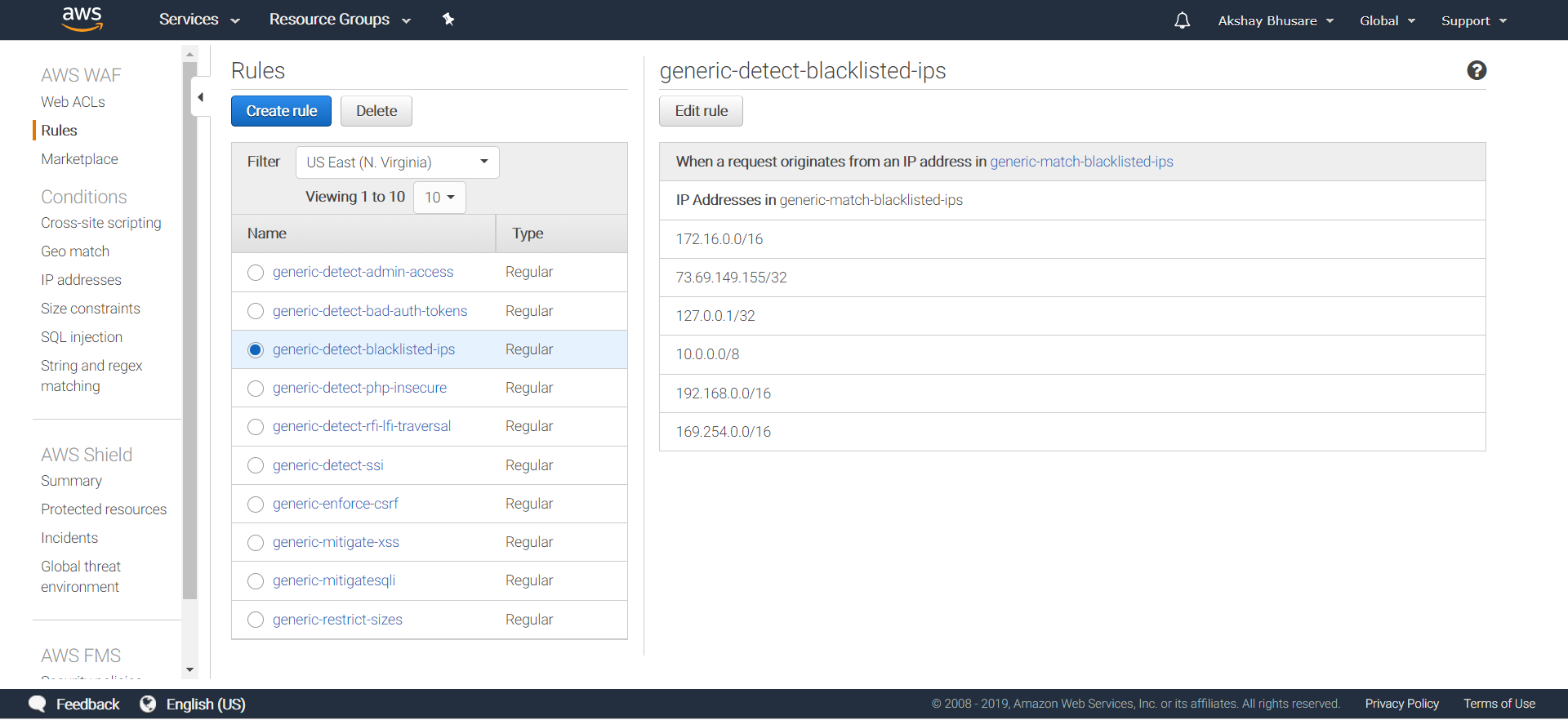
Attackers choose this attack vector as they access to millions of valid username and password

combinations for credential stuffing, default administrative account lists,

automated brute force, and dictionary attack tools.

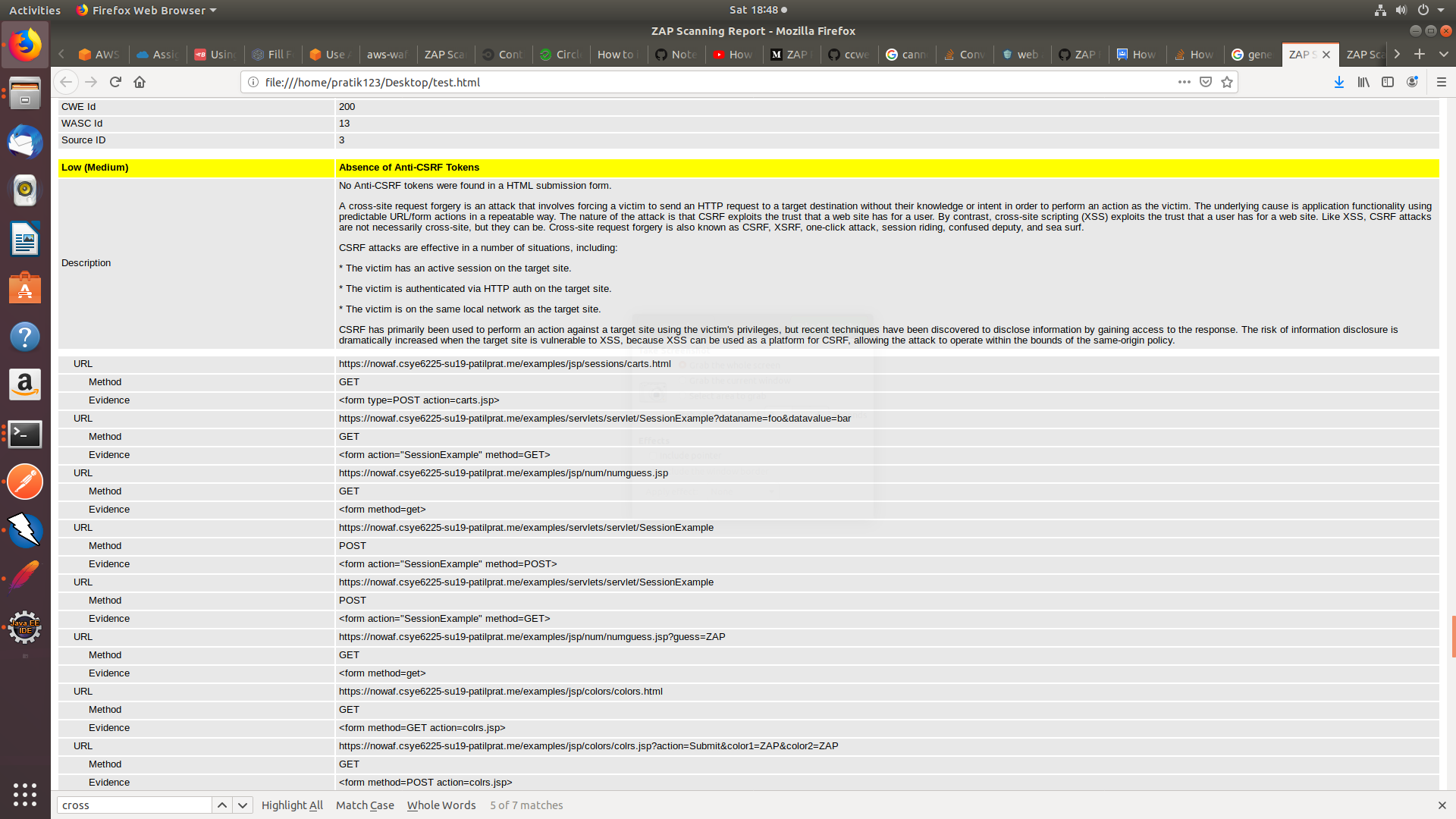






**2) Cross-Site Request Forgery (CSRF)**

Cross-site request forgery attacks predominantly target state-changing functions in your web applications.40 Consider any URL path and HTTP request that is intended to cause a state change (for example, form submission requests). Are there any mechanisms in place to ensure the user intended to take that action? Without such mechanisms, there isn’t an effective way to determine whether the request is legitimate and wasn’t forged by a malicious party. Depending solely on client-side attributes, such as session tokens or source IP addresses, isn’t an effective strategy because malicious actors can manipulate and replicate these values. CSRF attacks take advantage of the fact that all details of a particular action are predictable (form fields, query string parameters). Attacks are carried out in a way that takes advantage of other vulnerabilities, such as cross-site scripting or file inclusion—so users aren’t aware that the malicious action is triggered using their credentials and active session



Why this attack vector was chosen?

Because the web app uses JSON data as input and malicious scripts can be injected using the param in the URI.

The web application handles this attack by forming an object of the input parameter and thus disabling the execution of the script. But due to a flaw in the spring security module, this attack can’t be totally prevented.

**3)Unrestricted file size Vulnerability**

Uploaded files represent a significant risk to applications. The first step in many attacks is to get some code to the system to be attacked. Then the attack only needs to find a way to get the code executed. Using a file upload helps the attacker accomplish the first step.

The consequences of unrestricted file upload can vary, including complete system takeover, an overloaded file system or database, forwarding attacks to back-end systems, client-side attacks, or simple defacement. It depends on what the application does with the uploaded file and especially where it is stored.

There are really two classes of problems here. The first is with the file metadata, like the path and file name. These are generally provided by the transport, such as HTTP multi-part encoding. This data may trick the application into overwriting a critical file or storing the file in a bad location. You must validate the metadata extremely carefully before using it.

The other class of problem is with the file size or content. The range of problems here depends entirely on what the file is used for. See the examples below for some ideas about how files might be misused. To protect against this type of attack, you should analyse everything your application does with files and think carefully about what processing and interpreters are involved.

