Vulnerability Assessment and Penetration Testing (VAPT) report

Application Name: Damn Vulnerable Web Application (DVWA)

Assessment Type: web application VAPT

Prepared By: Anil kumar chaudhary

Date: 28/02/2026

Confidentiality Notice:

This document contains confidential security information and is intended for authorized stakeholders only.

Disclaimer

This assessment was conducted strictly in authorized lab environment for educational purpose only.

1. Executive Summary

A Vulnerability Assessment and Penetration Testing (VAPT) exercise was conducted on the Damn Vulnerable Web Application (DVWA) to identify security weakness that could be exploited by malicious attackers. This assessment revealed 2 Critical, 15 High, 13 Medium and 4 low that may allow unauthorized access to sensitive application data.

If exploited, these vulnerabilities could lead to data leakage, account compromise and reputational damage. Immediate remediation of High and Medium risk finding is strongly recommended, followed by retesting to validate fixes.

2. Scope of Assessment

In scope:

Target url: http://127.0.0.1/DVWA/ (OWASP vulnerabilities testing)

Testing is limited to the vulnerabilities modules available under:

http://127.0.0.1/DVWA/vulnerabilites/*

http://127.0.0.1/DVWA/index.php

Out of scope:

- Denial of service(dos) attacks
- Social engineering attacks
- Network level attacks
- Host operating system testing
- Any module outside the /DVWA/ directory
- Physical security testing
- Testing of any system application than DVWA

3. Methodology

The assessment was conducted following industry recognized standards:

- Open Web Application Security Project (OWASP) web security testing guide.
- Penetration Testing Execution Standard (PTES)

Testing phase:

1. Information Gathering
2. Vulnerabilities Identification
3. Exploitation and Validation
4. Risk Analysis
5. Reporting

4.Risk Rating Methodology

Risk severity was determined based on:

- Common Vulnerability Scoring Standard (CVSS v3.1) methodology
- Risk=likelihood * impact
- Likelihood Factors:

  Attack Complexity(AC)

  Privileged Required(PR)

  Attack Vector(AV)

  User Interaction(UI)

  Scope (S)

- impact :

  confidentiality

  Integrity

  Availability

- Severity Rating Table:

CVSS Score	Severity Level
0.0	None
0.1 - 3.9	Low
4.0 - 6.9	Medium
7.0 - 8.9	High
9.0 – 10.0	Critical

Based on calculated CVSS score, vulnerabilities were classified as None, Low, Medium, High and Critical.

CVSS score calculated using website: https://www.first.org/cvss/calculator/3.1

5. Vulnerability Summary

Id	Vulnerability	DVWA Security Level	Severity	CVSS
A01	Sql Injection	Low	High	8.1
A02	Sql Injection	Medium	High	8.1
A03	Sql Injection	High	High	7.1
A04	Reflected XSS	Low	Medium	4.6
A05	Reflected XSS	Medium	Medium	4.6
A06	Reflected XSS	High	Low	2.6
A07	Stored XSS	Low	High	7.3
A08	Stored XSS	Medium	medium	6.3
A09	Stored XSS	High	Low	3.7
A10	DOM XSS	Low	Medium	4.6
A11	DOM XSS	Medium	Medium	4.6
A12	DOM XSS	High	Low	3.7
A13	Command injection	Low	High	8.8
A14	Command injection	Medium	High	8.8
A15	Command injection	High	High	8.8
A16	Brute Force	Low	Critical	9.8
A17	Brute Force	medium	Critical	9.8
A18	Brute Force	High	High	8.1
A19	Insecure Captcha	Low	Medium	5.4
A20	Insecure Captcha	Medium	Medium	5.4
A21	Insecure Captcha	High	Medium	4.2
A22	Weak Session Id	Low	High	7.3
A23	Weak Session Id	Medium	High	7.3
A24	Weak Session Id	High	Medium	5.6
A25	Authorization Bypass	Low	High	8.8
A26	Authorization Bypass	Medium	High	8.8
A27	Authorization Bypass	High	High	7.5
A28	Open HTTP Redirect	Low	Medium	4.3
A29	Open HTTP Redirect	Medium	Medium	4.3
A30	Open HTTP Redirect	High	Low	3.0
A31	Csrf	Low	Medium	4.3

A32	Content Security Policy Bypass	Medium	Medium	4.6
A33	Content Security Policy Bypass	High	High	7.2
A34	Content Security Policy Bypass	Low	High	7.3

6. vulnerability Details - SQL Injection (Low security level)

Severity: High

CVSS: 8.1

Affected url: http://127.0.0.1/DVWA/vulnerabilities/sqli/?id=payload

Description: The application directly concatenates user input into the backend SQL query without implementing input validation and sanitization, parameterized queries or prepared statements. This allow the attacker to manipulate the structure of the SQL query by injecting malicious SQL code.

Proof of Concept:

payload:

Step 1. Column Enumeration : 1' order by 2#

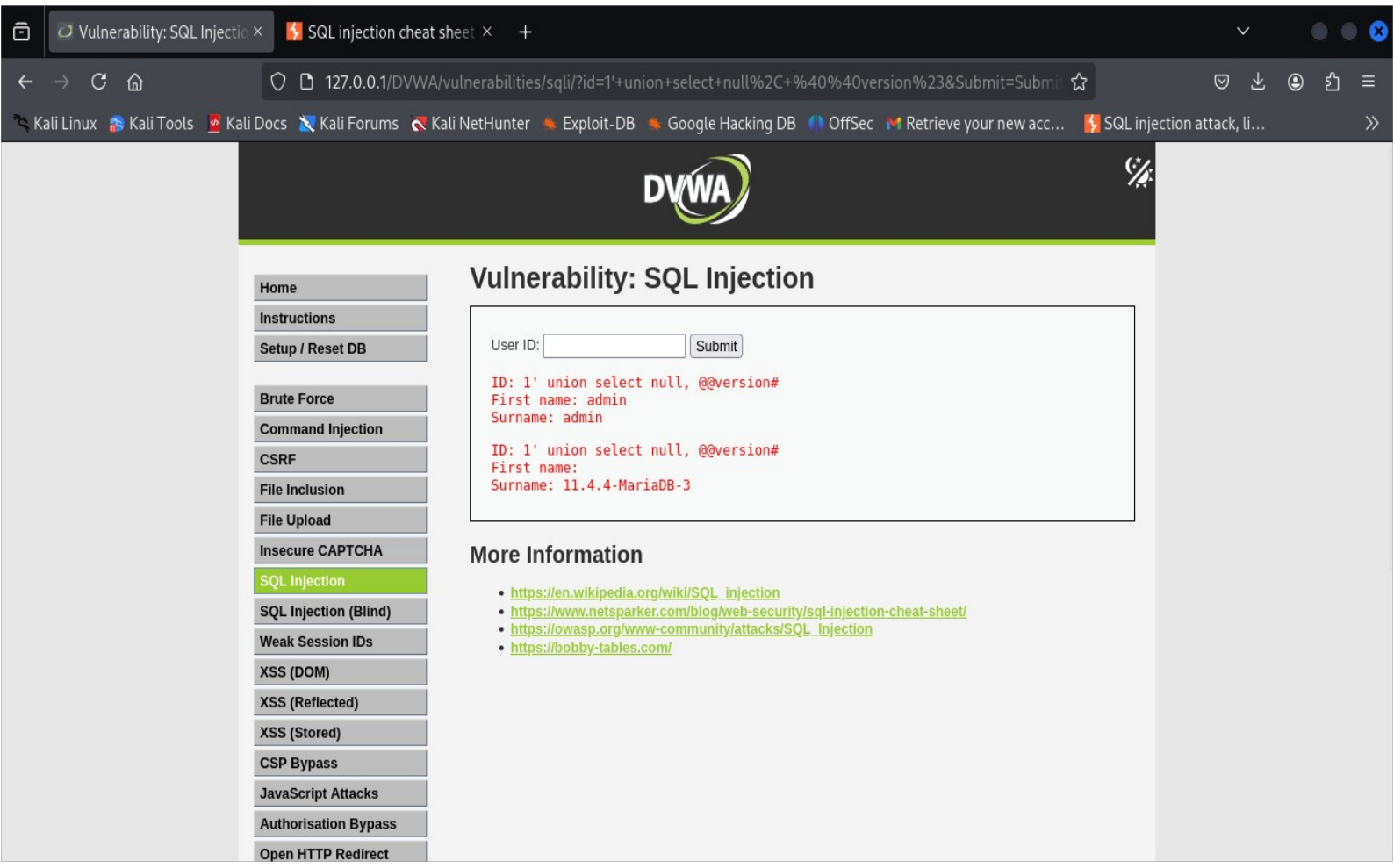
Step 2 . Column Reflection Testing: 1' union select '1', '1'#

Step 3. Data Extraction: 1' union select null, @@version#

Impact: It allow attacker to confidential data exposure, account compromise, data manipulation and complete database dump and system takeover.

Remediation: The application must implement proper input handling and secure database interaction mechanisms to prevent sql injection. Replace dynamic SQL query concatenation with parameterized queries, implement server-side input validation and apply the principle of least privilege.

Proof:



7. Vulnerability Detail – SQL Injection (Medium Security Level)

Severity: High

CVSS: 8.1

Affected url: http://127.0.0.1/DVWA/vulnerabilities/sqli/#

Affected Module: id=payload

Description: Basic filtering mechanisms are implemented such as escaping special character and basic encoding. However, these controls are incomplete and can be bypassed using crafted input techniques. The application still constructs SQL queries dynamically by concatenating user input into the query string.

Proof of Concept:

Payload: **(**use all steps using url encoding**)**

Step 1: Column Enumeration - 1 order by 2#

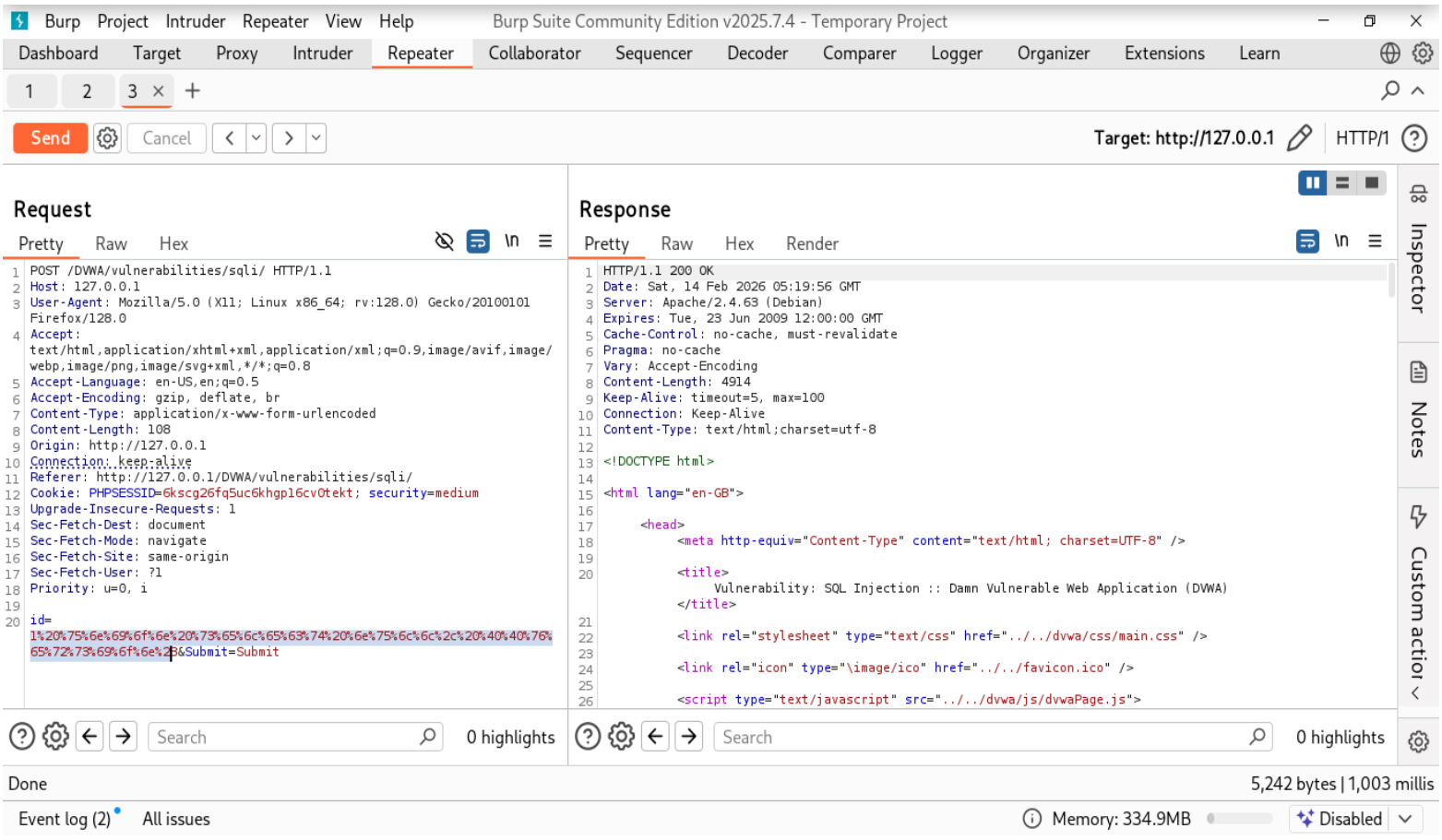
Step 2: Column Reflection Testing - 1 union select 'a', 'a'#

Step 3: Data Exploitation - 1 union select null, @@version#

Impact: It leads to Extraction of sensitive data, Database enumeration, credential disclosure, Authentication bypass and Unauthorized data access.

Remediation: Dynamic SQL query concatenation must be replaced with prepared statements, do not relay on Escaping alone, Enforce strict server-side input Validation implement principle of least privilege and Harden error Handling.

Proof:



8. Vulnerability Detail – SQL Injection (High Security Level)

Severity: High

CVSS: 7.1

Affected url: http://127.0.0.1/DVWA/vulnerabilities/sqli/session-input.php

Affected Module: id = payload

Description: The application constructs SQL queries using dynamic string concatenation instead of parameterized queries. Using input can be bypassed using crafted payload because user-supplied input are not validate properly.

Proof of Concept:

Payload: (use url encoding)

Step 1. Column Enumeration - ' order by 2#

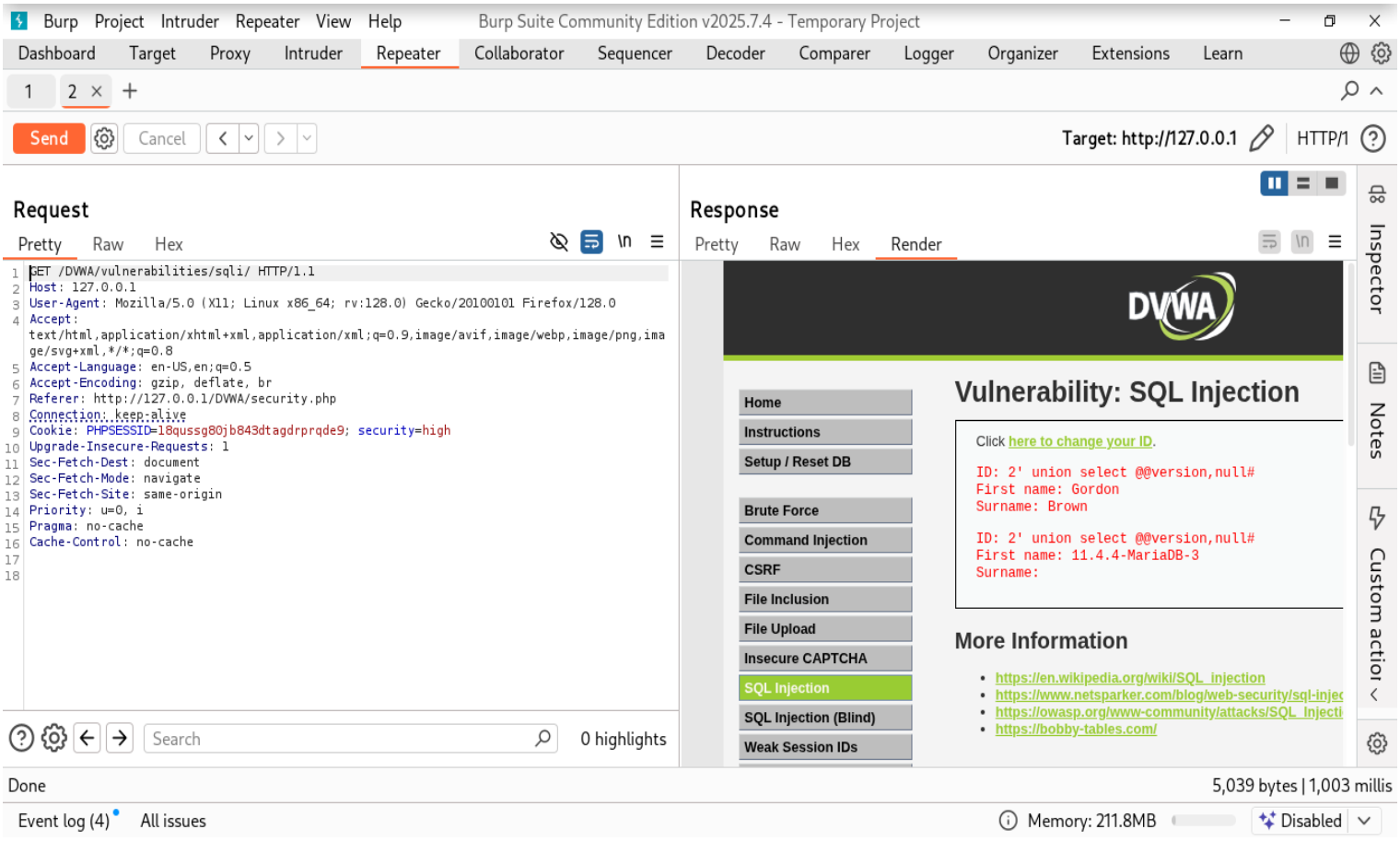
Step 2. Column Reflection Testing - ' union select 'a','a'#

Step 3. Data Exploitation - ' union select @@version,null#

Impact: It leads to extraction of sensitive database information, credential disclosure, Database enumeration, authentication bypass, unauthorized data manipulation and potential database takeover.

Remediation: Replace Dynamic Queries with prepared statements, implement strict Allow-list Validation, apply principle of least privilege and Harden Error Handling and enforce strict input validation.

Proof:



9. Vulnerability Detail – Reflected XSS (Security Level Low)

Severity: Medium

CVSS: 4.6

Affected Url: http://127.0.0.1/DVWA/vulnerabilities/xss_r/?name= payload

Description: The application directly reflects user input in the HTTP response without performing any sanitization, encoding or filtering. Because input is embedded into the HTML page without validation, attacker can inject malicious JavaScript code that executes in victim's browser.

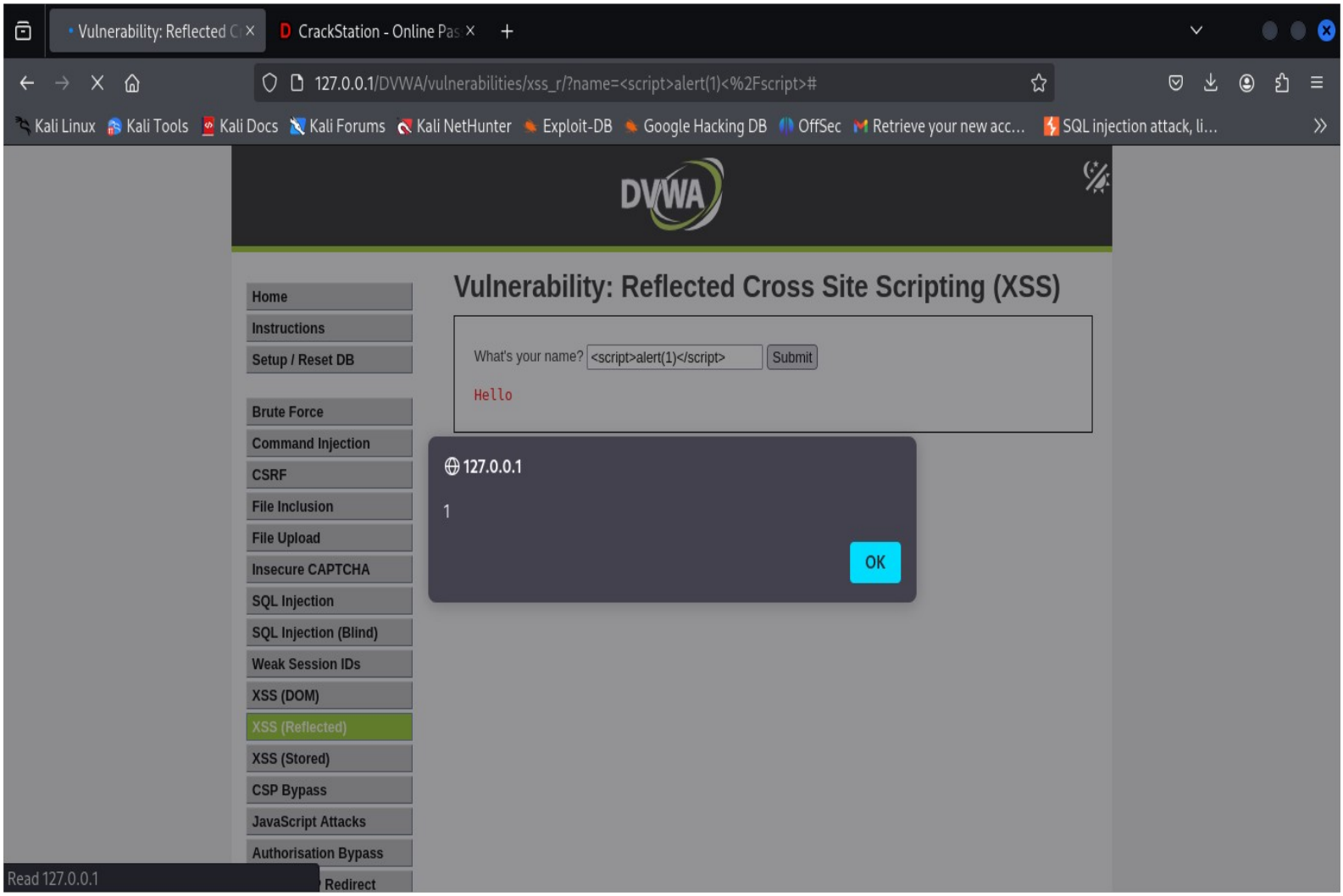
Proof of Concept:

Payload: <script>alert(1)</script>

Impact: It leads to the session cookie theft, account hijacking, phishing attacks, Redirection to malicious websites, executions of arbitrary JavaScript in victim's browser and defacement of the web page content.

Remediation: Apply proper output encoding for all reflected user input, implement strict input validation , use content security policy (csp) , protect cookies with secure flags and avoid unsafe JavaScript functions.

Proof:



10. Vulnerability Detail – Reflected XSS (Security Level Medium)

Severity: Medium

CVSS: 4.6

Affected url: http://127.0.0.1/DVWA/vulnerabilities/xss_r/?name= payload

Description: The application attempts to sanitize user input by escaping certain special character. Attacker can bypass using alternative character encoding (url encoding , HTML entities) , logical payload and special sequences that are not filtered, as a result JavaScript code can still inject into application's response and will execute in the victim's browser when target click a crafted malicious link.

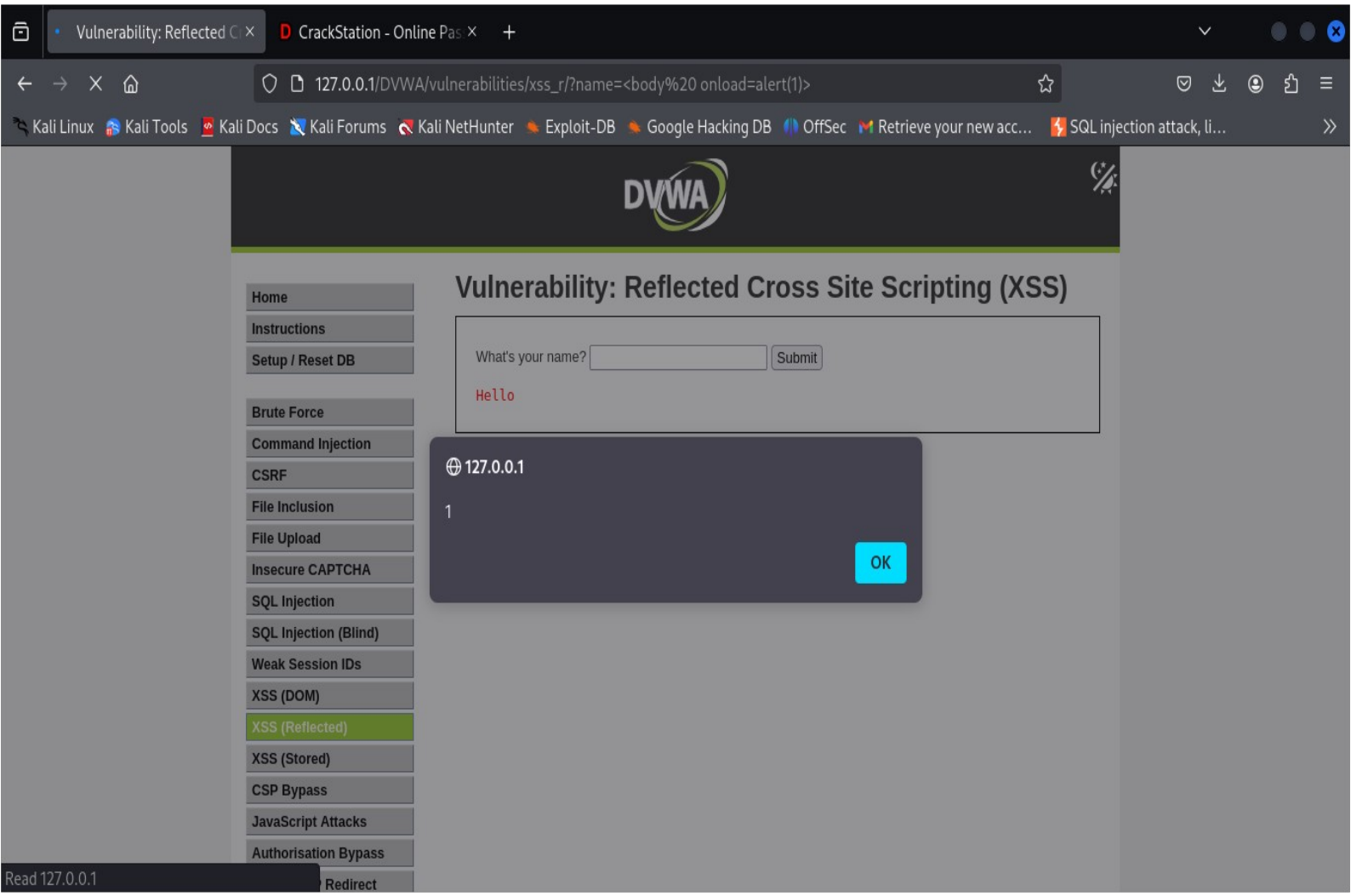
Proof of Concept:

Payload: <body onload=alert(1)>

Impact: It leads to session cookie theft, account hijacking, phishing, Defacement of application content and unauthorized actions in the context of the victim's session.

Remediation: All user-supplied input must be encoded before being rendered in the browser, don't relay on blacklist filtering use allow-list validation and proper encoding, apply server-side strict input validations and implement content security policy (csp).

Proof:



11. Vulnerability Detail – Reflected XSS (Security Level High)

Severity: Low

CVSS: 2.6

Affected url: http://127.0.0.1/DVWA/vulnerabilities/xss_r/?name= payload

Description: Blacklist certain keyword like <script> or filtering special character are insufficient because it relies on pattern-based filtering rather than proper contextual encoding. Bypasses are possible using case manipulation, event handlers, HTML element injection , encoded payload. The application reflects the malicious payload in the response, the victim's browser executes the injected JavaScript in the context of trusted domain.

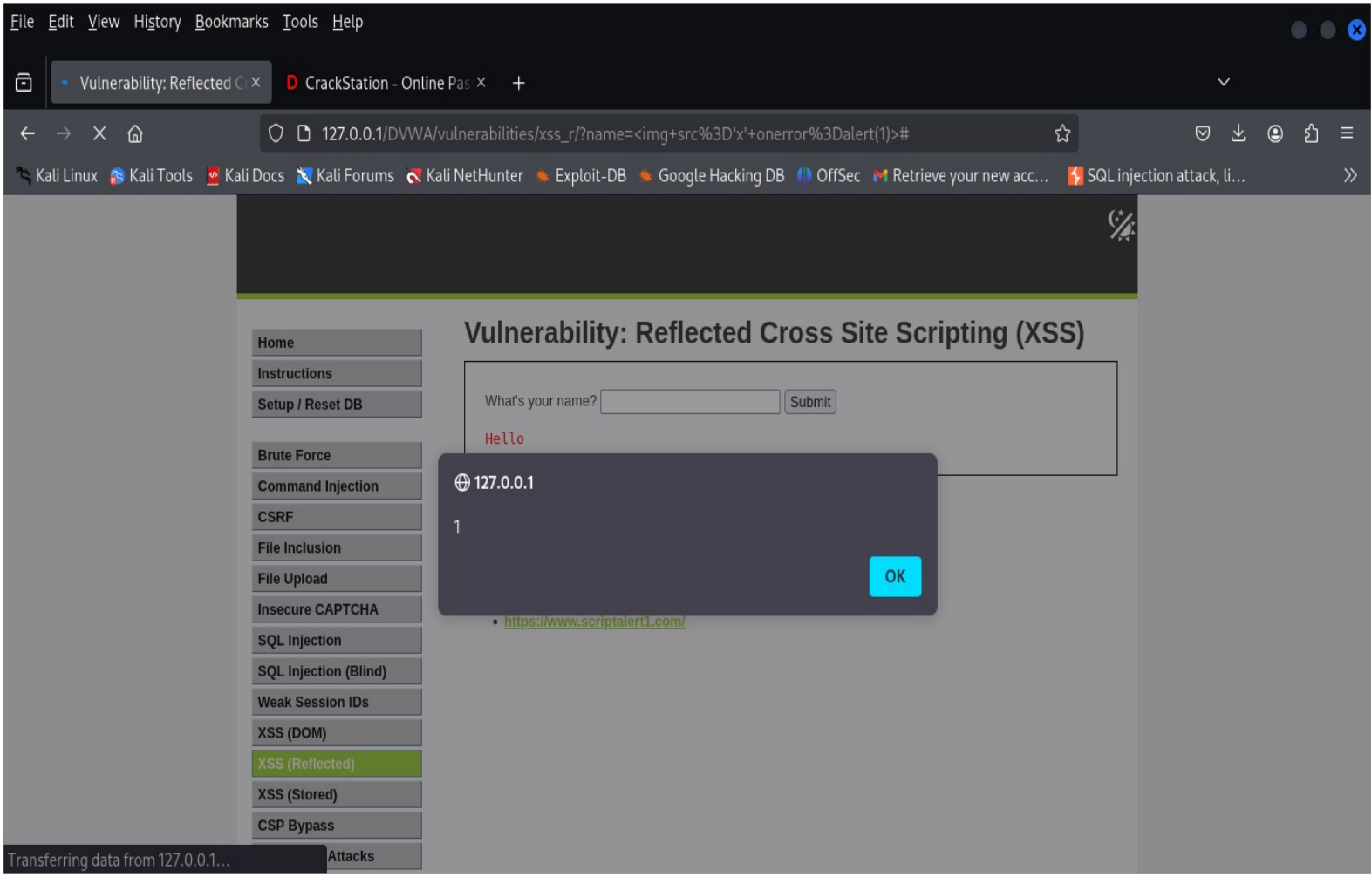
Proof of Concept:

Payload: <img src='x' onerror=alert(1)> or <body onload=alert(1)>

Impact: It leads to session hijacking, account takeover, credential harvesting via phishing forms, defacement of web contents, execution of arbitrary JavaScript in victim's browser and redirection to malicious websites.

Remediation: Encode user-supplied data before reflecting it in the HTML, use proper encoding functions depending on context, use whitelist validation with strict input validation rules, deploy strong csp header to prevent inline JavaScript execution.

Proof:



12. Vulnerability Detail – Stored XSS (Security Level Low)

Severity: High

CVSS: 7.5

Affected url: http://127.0.0.1/DVWA/vulnerabilities/xss_s/

Affected Module: message=payload

Description: The application does not perform adequate input validation or output encoding on user-submitted data. Attacker can inject arbitrary JavaScript code into input fields. Absence of server-side input validation , lack of contextual output encoding, failure to implement secure coding practices.

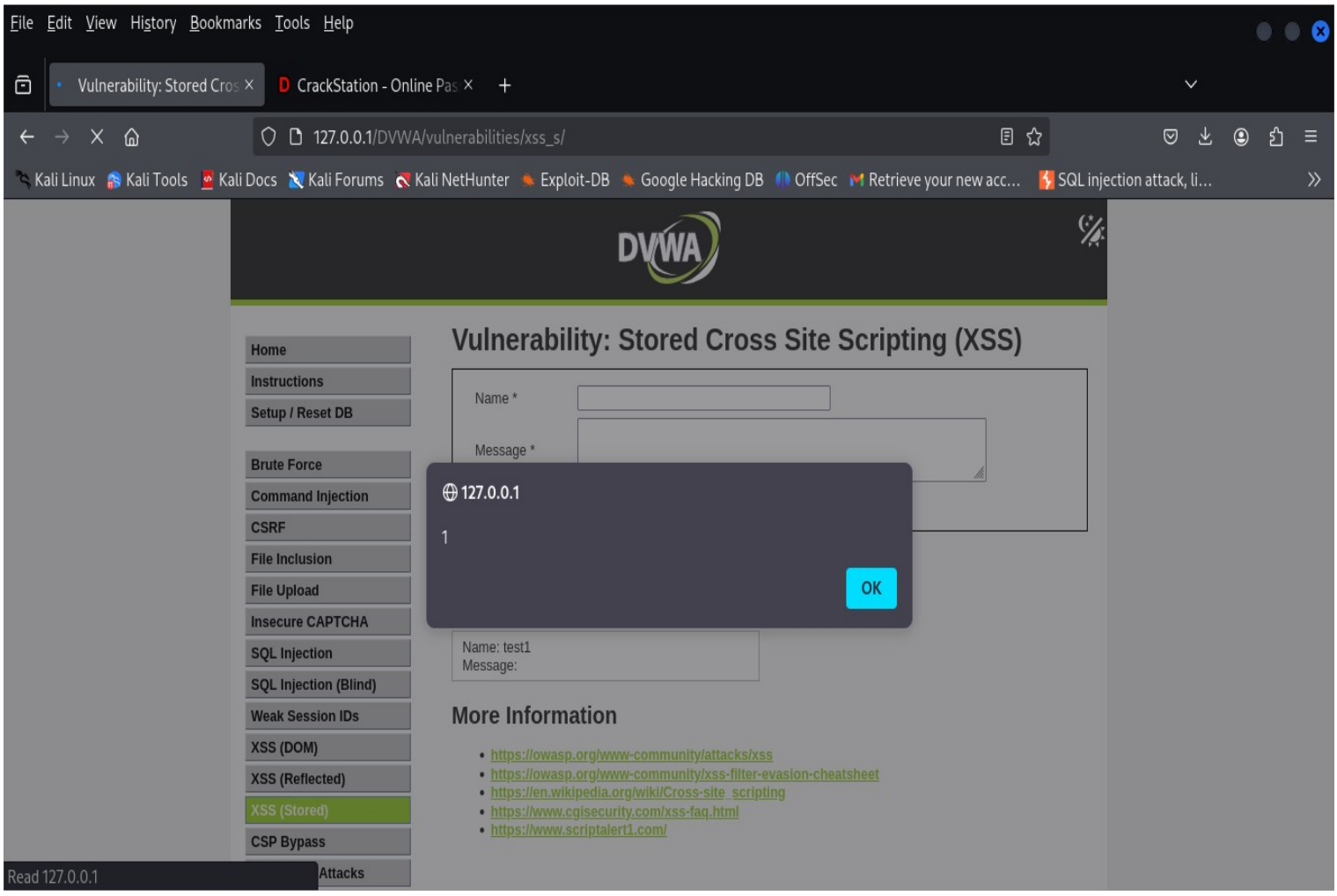
Proof of Concept:

Payload: <body onload=alert(1)>

Impact: Session hijacking, account takeover, privilege escalation if admin user is targeted, credential harvesting, Malicious redirection, websites defacement, Distribution of malware to users.

Remediation: All user-supplied data must be encoded before rendering in HTML, server-side input validations, use whitelist instead of blacklist filtering, Enforce maximum input length.

Proof:



13.Vulnerability Detail – Stored XSS (Security Level Medium)

Severity: Medium

CVSS: 6.3

Affected url: http://127.0.0.1/DVWA/vulnerabilities/xss_s/

Affected Module: name=payload

Description: The application attempts to filter certain characters and <script> tags from user input. However, the input validation is insufficient and can be bypassed using alternative payloads or encoding techniques. Malicious JavaScript code can still be stored in the application database through the message field.

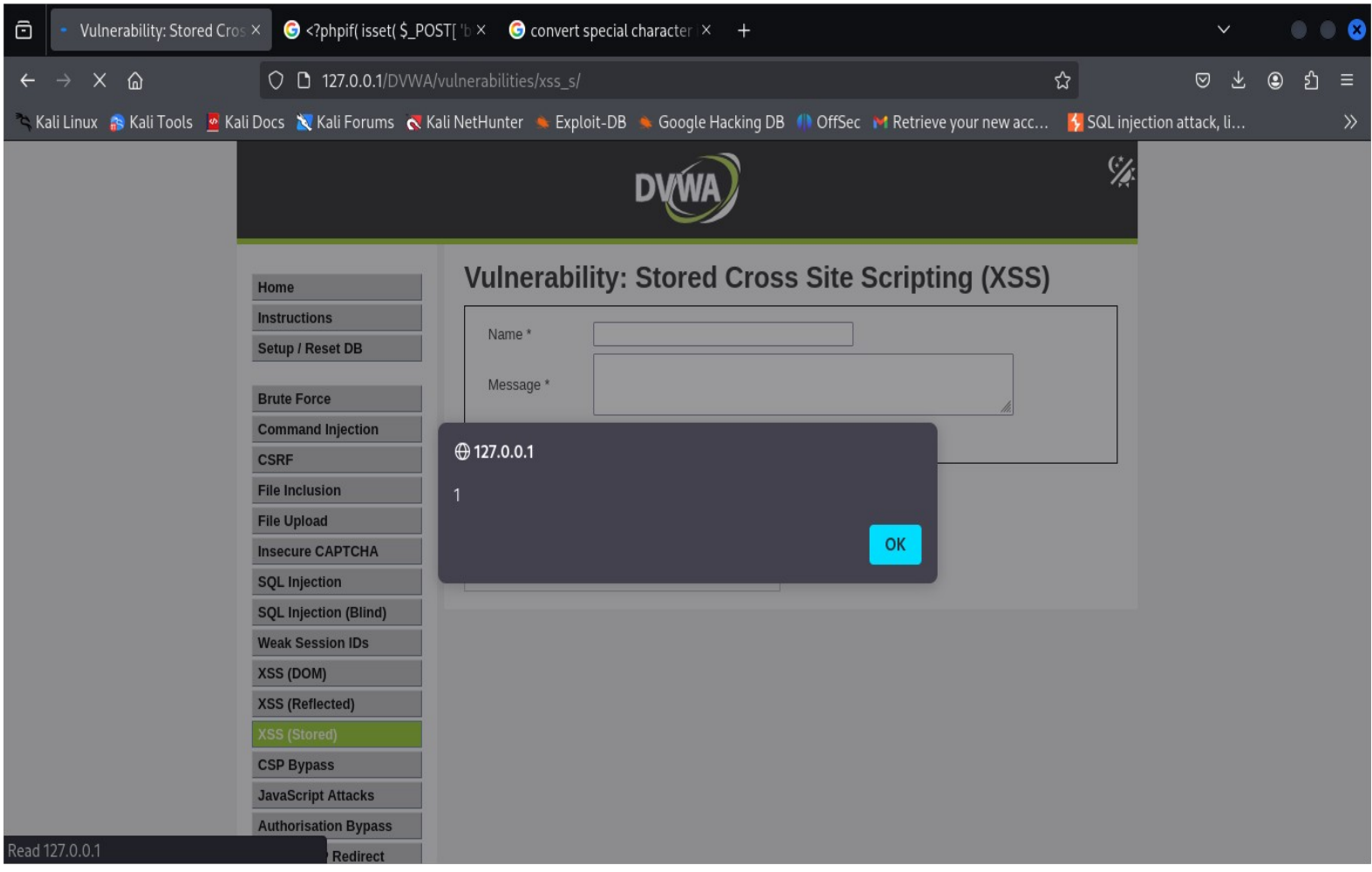
Proof of Concept:

Payload: <sc<script>ript>alert(1)</scr<script>ipt> or <ScRipT>alert(1)</ScrIPT>

Impact: It leads to session Hijacking, account takeover, malicious redirection, data theft, defacement and privilege escalatin.

Remediation: To mitigate vulnerabilities user inputs are validate, sanitize, and potentially danger characters and scripts on both client and server-side. Encode user-supplied data before displaying it in the browser and ensure scripts are rendered as text instead of executable code. Use security libararies , csp , HTTPOnly and secure cookies flags and regular security testing.

Proof:



14. Vulnerability Detail – Stored XSS (Security Level High)

Severity: Low

CVSS: 3.7

Affected url: http://127.0.0.1/DVWA/vulnerabilities/xss_s/

Affected Module: name = payload

Description: The application attempts to sanitize input tags and limiting certain characters or input length before storing the data. It relies on blacklist-filtering instead of proper contextual output encoding. Still restriction can bypass using alternative HTML elements, event Handlers or encoding techniques.

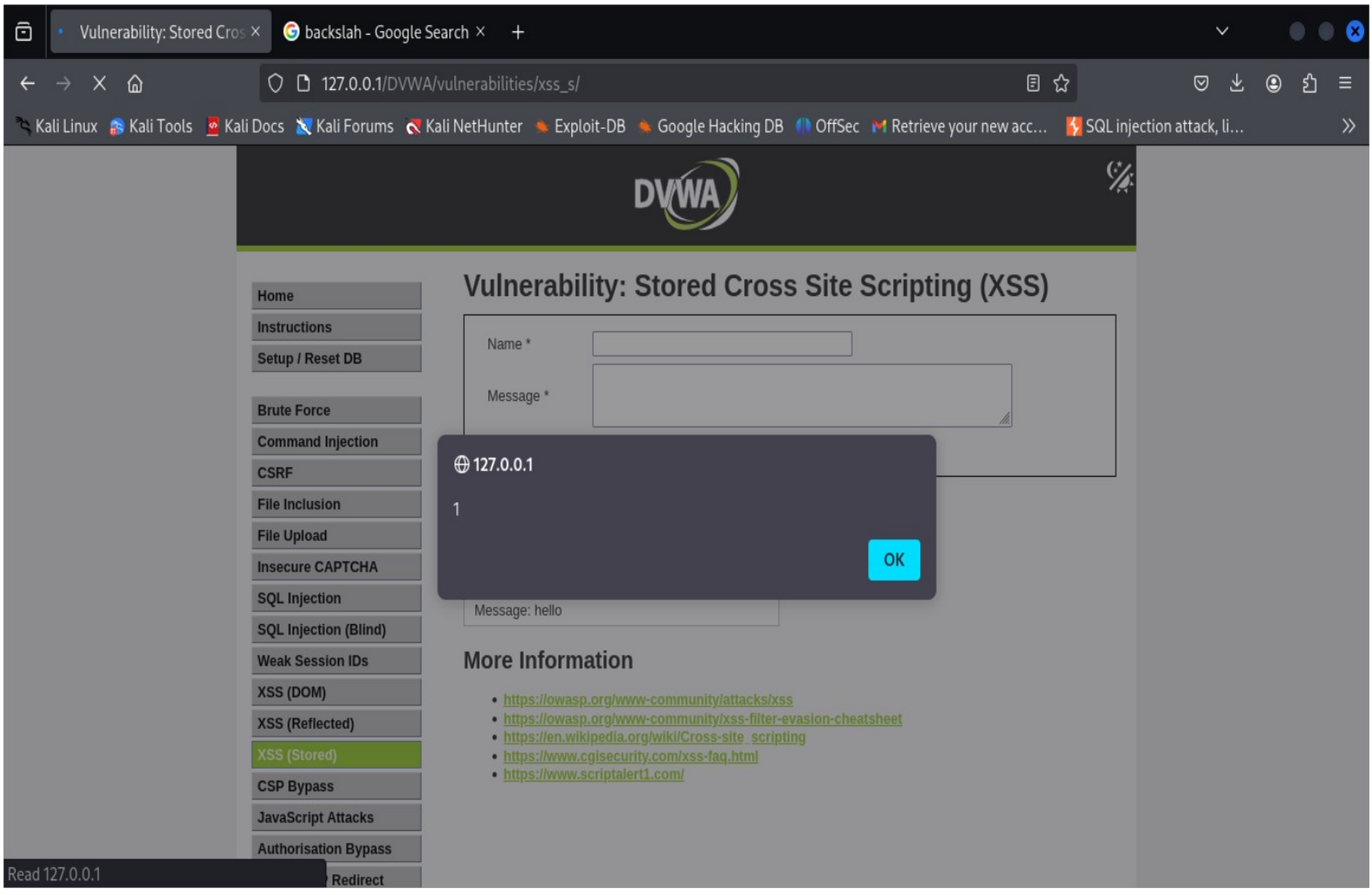
Proof of Concept:

Payload: <img src='x' onerror=alert(1)> or <body onload=alert(1)>

Impact: It leads to Session hijacking, account compromise, privilege escalation, sensitive data exposure, malicious actions on behalf of users and persistent attack surface.

Remediation: To mitigate effectively, Encode user-supplied data before rendering it in html, JavaScript or other contexts. Implement a whitelist approach allowing only safe and expected input instead of blacklist filtering specific tags or characters. CSP, secure cookie configuration with regular security testing.

Proof:



15. Vulnerability Detail – DOM Based XSS (Security Level Low)

Severity: Medium

CVSS: 4.6

Affected url: http://127.0.0.1/DVWA/vulnerabilities/xss_d/?default= payload

Description: The application reads user-controlled data from the url parameters and dynamically inserts it into the web page using JavaScript. Since the application does not perform any validation or sanitization on the input before updating the dom, attacker can manipulate the url to inject malicious JavaScript code. This indicates the improper handling of user input within client-side scripts and lack of secure dom manipulation practices.

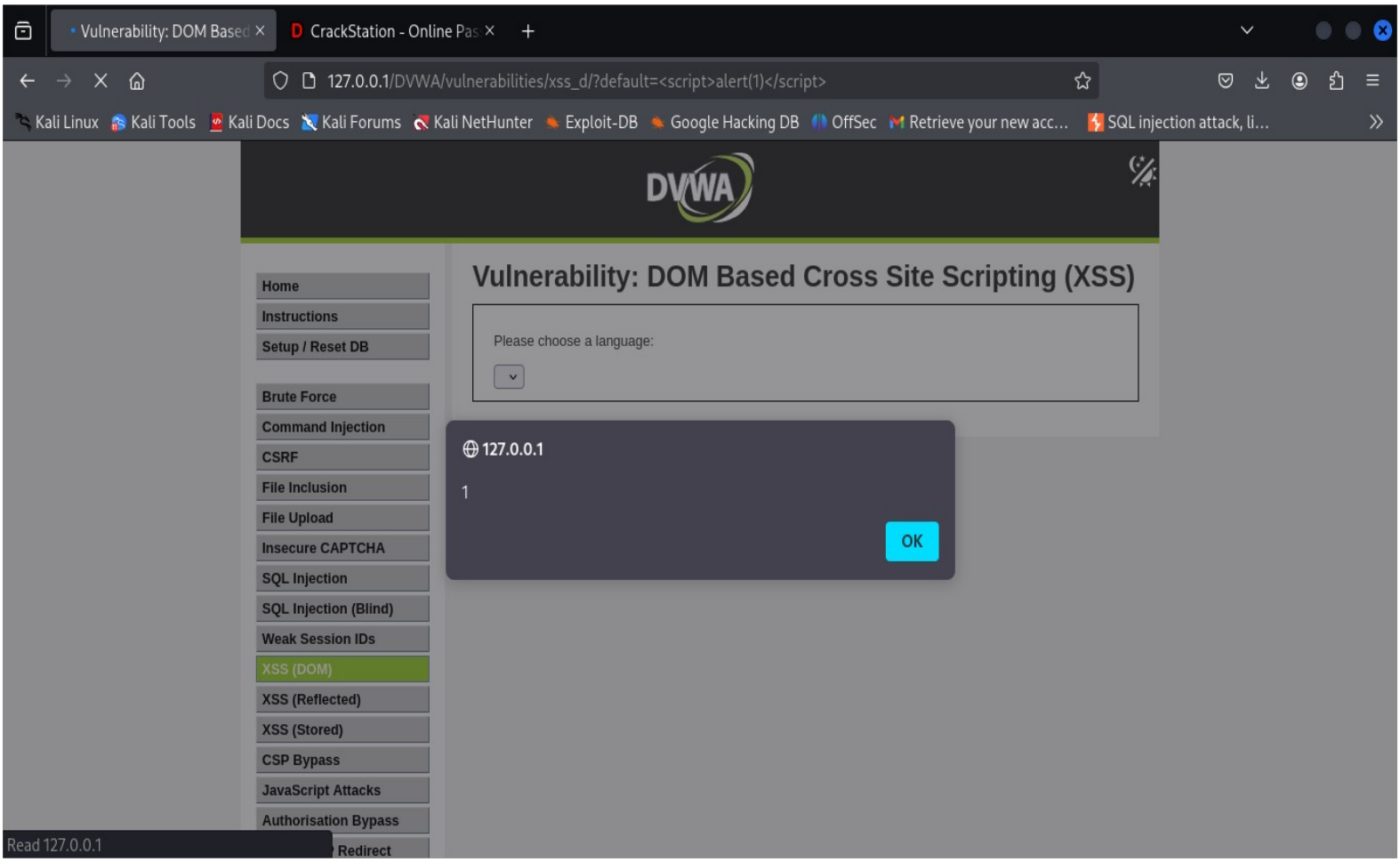
Proof of Concept:

Payload: <script>alert(1)</script>

Impact: IT leads to session Hijacking, account compromise, malicious redirection, sensitive data theft unauthorized actions.

Remediation: To prevent vulnerabilities, Don't insert untrusted user input directly into the dom using functions innerHTML, document.write() or similar methods instead use safer alternatives textContent, innerText or others. Validate and sanitize all users inputs before processing them within client-side scripts and ensure proper encoding of dynamic content before inserting it into the HTML document. Implement strong csp header and secure development practices.

proof:



16. Vulnerability Detail – DOM Based XSS (Security Level medium)

Severity: Medium
CVSS: 4.6

Affected url: http://127.0.0.1/DVWA/vulnerabilities/xss_d/?default = payload

Description: The application attempts to block specific script patterns or characters from the user-controlled input received through url parameters. Which can be bypassed using alternative payload structures or encoding methods. Because the application still inserts user-supplied data into the dom without proper contextual encoding or dom handling methods, attacker may still inject malicious JavaScript code. When victim accesses a specially crafted url containing the malicious payload, the script executes within the browser.

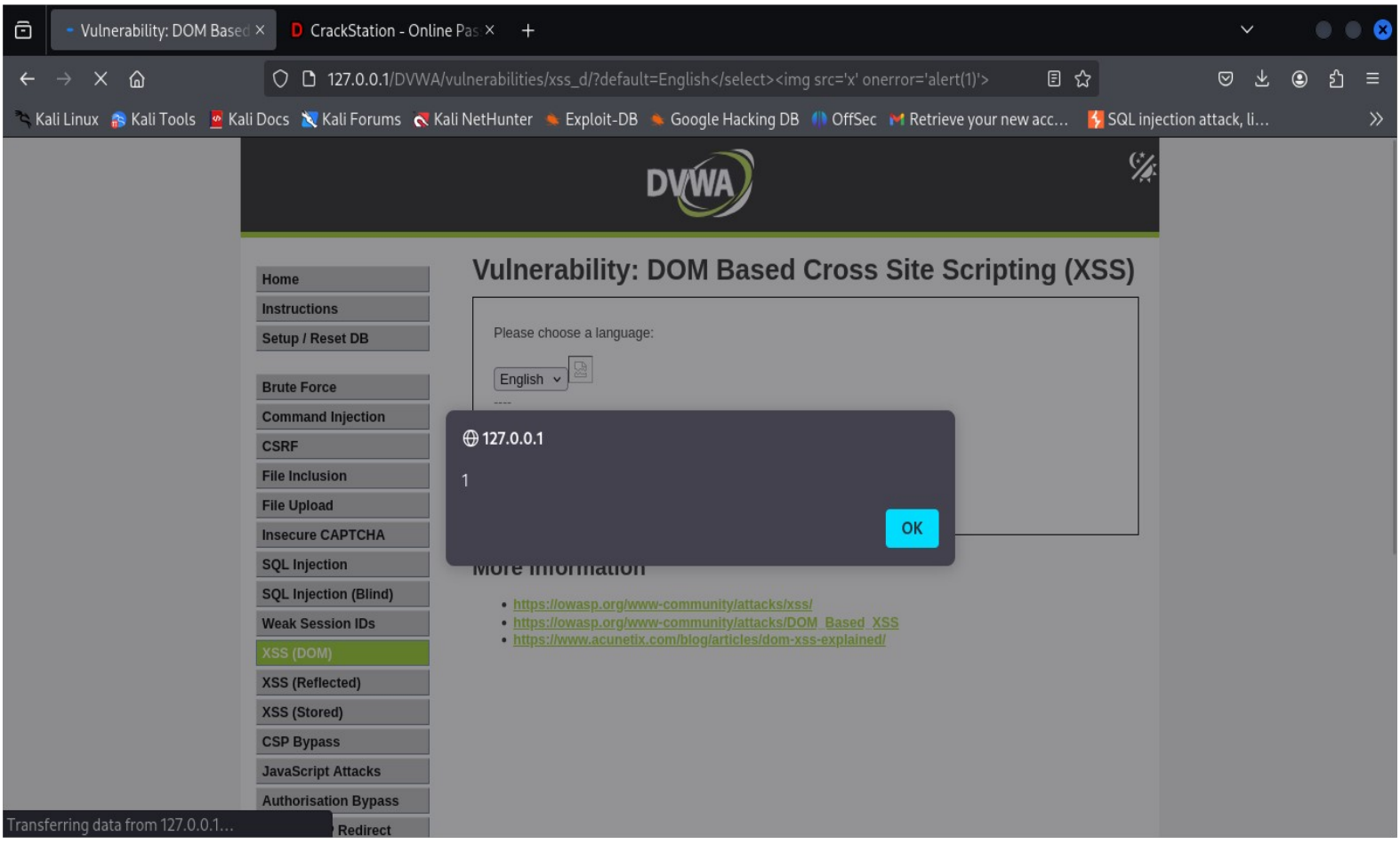
Proof of Concept:

Payload: </select><img src='x' onerror='alert(1)'>

Impact: It leads to session hijacking, account compromise, malicious redirection, sensitive data exposure, unauthorized actions.

Remediation: To mitigate avoid inserting user input into the dom using functions like innerHTML, document.write(). Use safer like textcontent or innerText and whitelist instead of blacklist. Validate user input to ensure conforms to expected formats before processing within client- side scripts. Apply output encoding as plain text rather than executable code and csp to restrict unauthorized scripts to execute.

Proof:



17. Vulnerability Detail – DOM Based XSS (Security Level High)

Severity: Low

CVSS: 3.7

Affected url: http://127.0.0.1/DVWA/vulnerabilities/xss_d/?default= payload

Description: The application attempts to restrict certain character or patterns that could lead to script injection. These protections rely mainly on client-side filtering and blacklist-based restrictions, which may still be bypassed using alternative payload techniques or encoded inputs. Because the application continues to use user-controlled data when dynamically modifying the dom without applying secure encoding practices and remain the potential risk of scripts execution if filtering is bypassed.

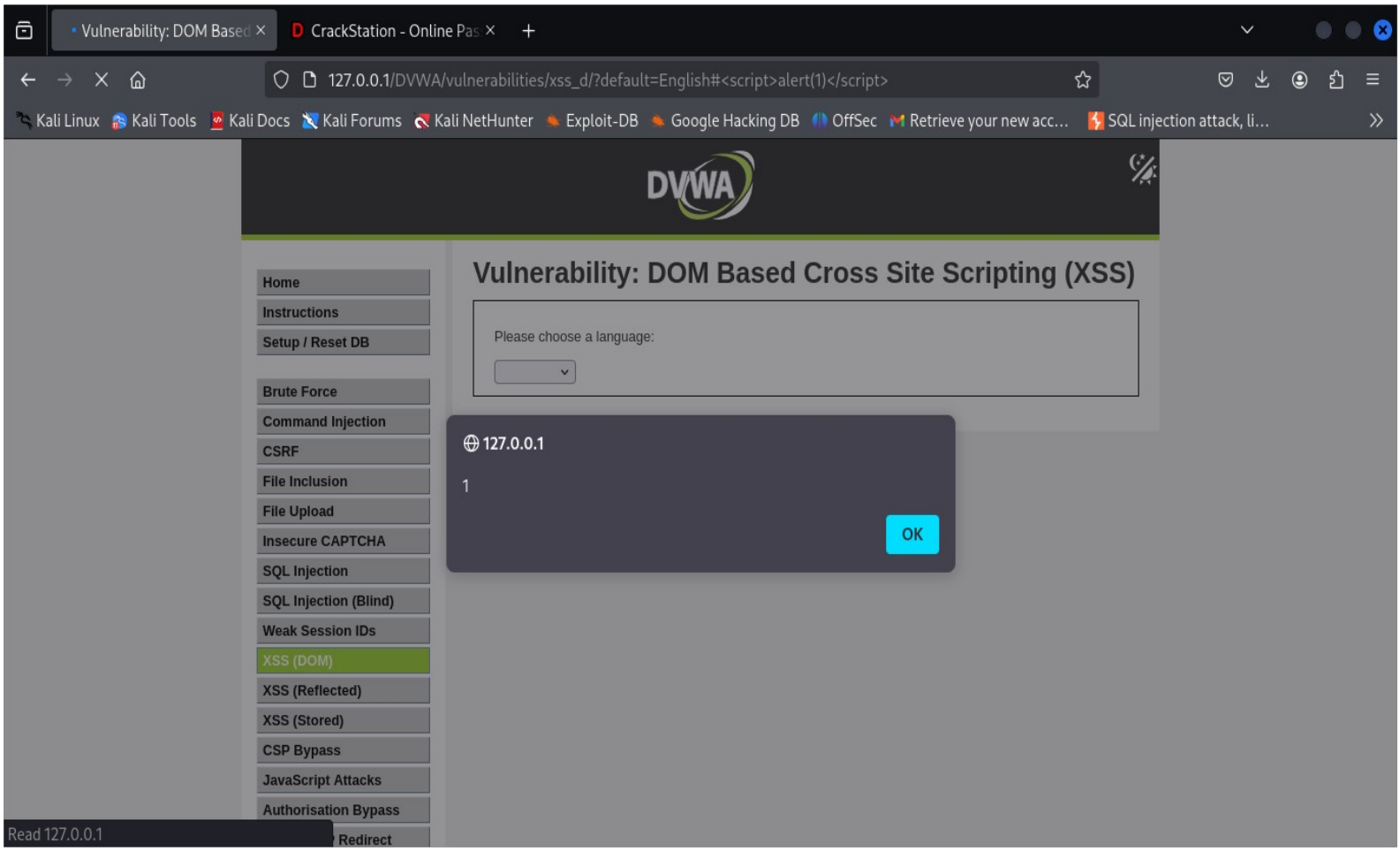
Proof of Concept:

Payload: #<script>alert(1)</script>

Impact: It leads to session Hijacking, account compromise, sensitive data exposure, malicious redirection and execution of unauthorized actions.

Remediation: To mitigate avoid using unsafe functions such as innerHTML, document.write() or similar instead use safer methods such as textcontent, innertext or other. Implement proper input validations before processed by client-side scripts and encode user input data before inserting it into the dom to prevent execution as code. Implementa strict csp header to restrict execution of unauthorized scripts.

Proof:



18. Vulnerability Detail – Command Injection (Security Level Low)

Severity: High

CVSS: 8.8

Affected url: http://127.0.0.1/DVWA/vulnerabilities/exec/#

Description: The application does not properly validate or sanitize the user input before appending it to the system command executed on the server. Because of this lack of input validation, an attacker can inject additional shell commands using commands separator (;,&&,..). attacker can execute unintended system commands and unsafe interaction with system level commands.

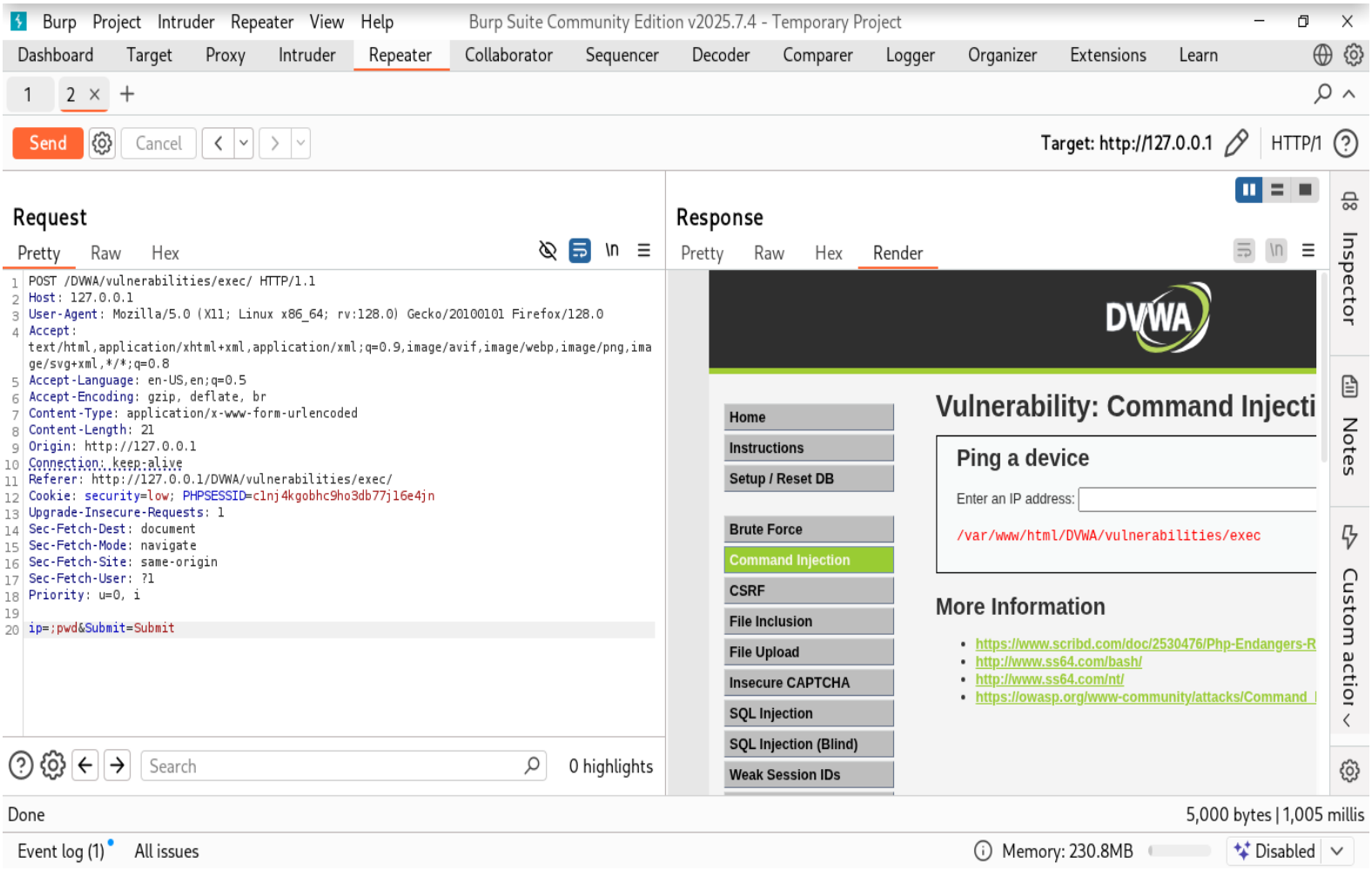
Proof of Concept:

Payload: ;pwd

Impact: It leads to remote command executions, unauthorized access, system information disclosure, privilege escalation, system compromise and lateral movement.

Remediation: To mitigate, validate user input against a whitelist of allowed characters or formats, avoid passing user input directly into system shell commands. If system commands must be executed, use secure api that separate commands parameters from user input. Input sanitize by removing special character that could be interpreted as command separator. Use principle of least privilege.

Proof:



19. Vulnerability Detail – Command Injection (Security Level medium)

Severity: High

CVSS: 8.8

Affected url: http://127.0.0.1/DVWA/vulnerabilities/exec/#

Description: The application attempts to block specific special character commonly used in command injection attack. However, the implemented filtering relies on simple blacklist based validation, which only block limited set of character that can still bypass using alternative command separators or encoding techniques. Because the application continues to pass user input directly into system commands without proper sanitization or secure handling that indicates to vulnerable to commands injection.

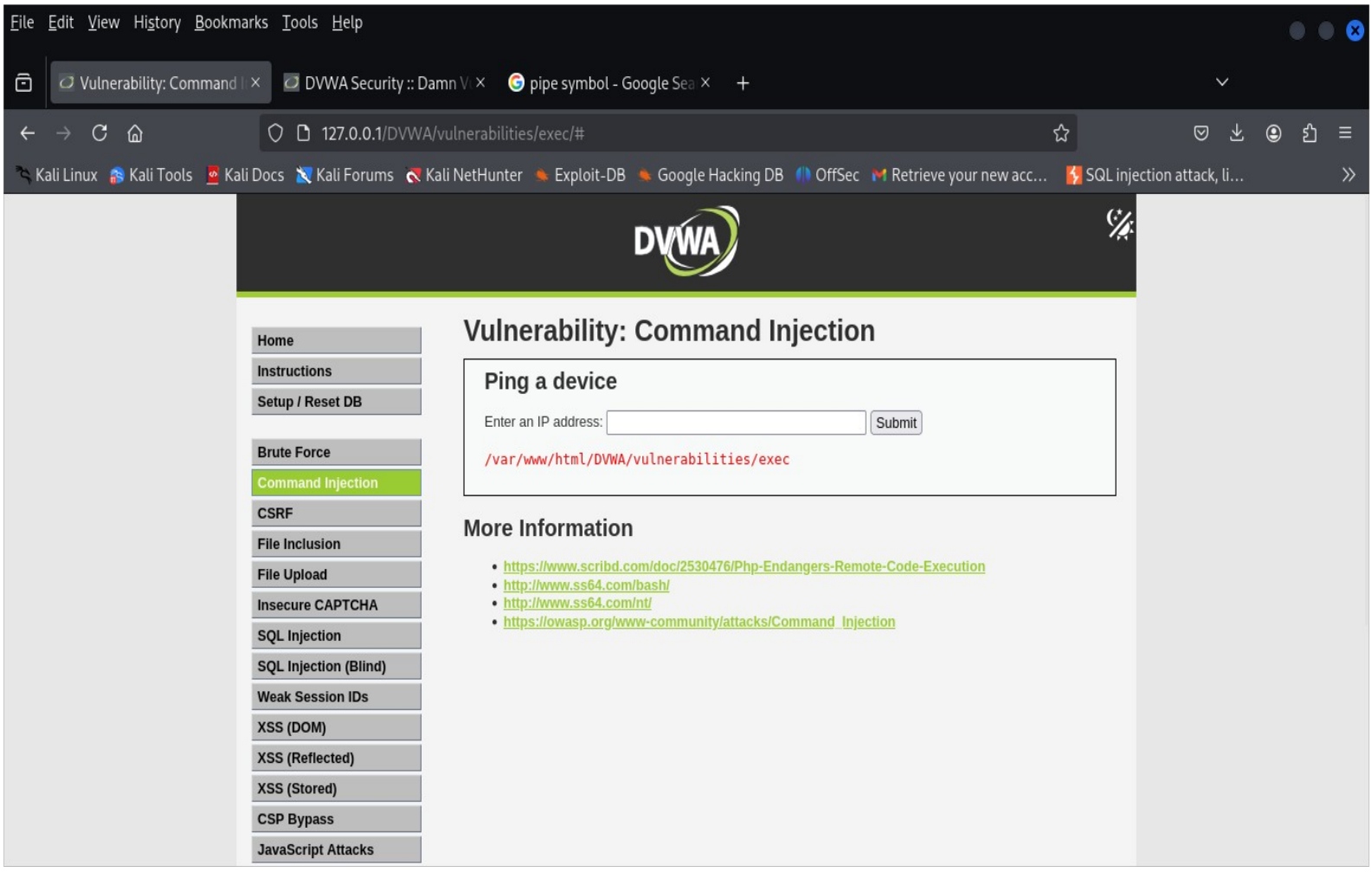
Proof of Concept:

Payload: x||ls

Impact: It leads to remote code execution, sensitive information disclosure, server compromise, privilege escalation and lateral movement.

Remediation: To mitigate, validate user inputs using whitelist approach that allows only expected values and don't only rely on blacklist based filtering. Use secure api instead of shell commands, properly sanitize and escape before including them in system commands. Apply the principle of least privilege.

Proof:



20. Vulnerability Detail – Command Injection (Security Level High)

Severity: High

CVSS: 8.8

Affected url: http://127.0.0.1/DVWA/vulnerabilities/exec/#

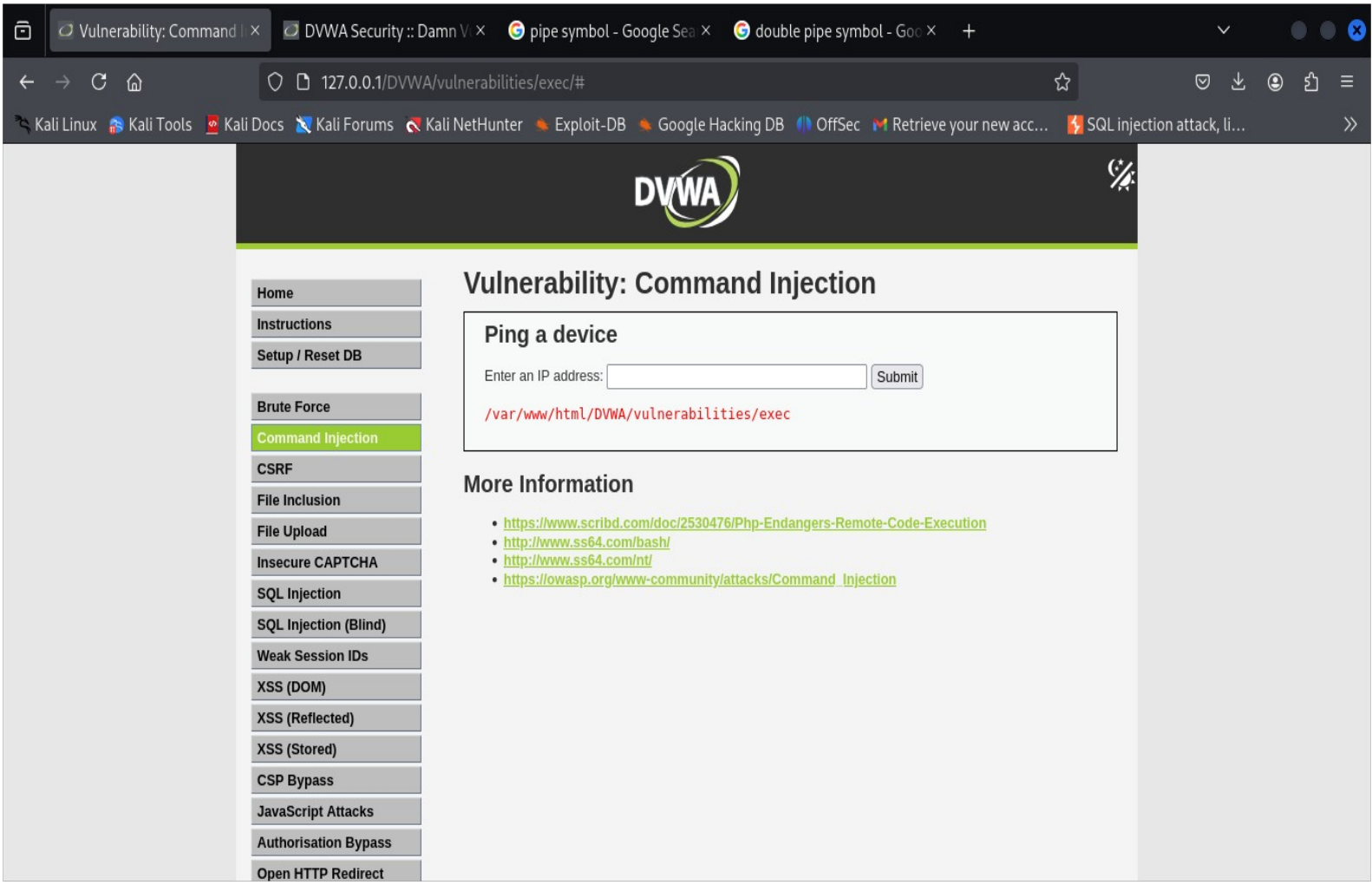
Description: The application attempts to restrict malicious input by applying stricter validation and filtering of special character primarily rely on input filtering mechanisms which is not fully prevent commands injection. Validation logic is bypassed and still includes user-controlled input in system commands without securely handling and it can bypass filtering control through craft payload.

Proof of Concept:

Payload: 127.0.0.1|pwd

Impact: It leads to remote commands execution, sensitive information disclosure, system manipulate, privilege escalation and network compromise.

Remediation: To mitigate avoid executing operating system commands with user input, use built-in programming language functions or libraries that perform task safely. Validate input using whitelist approach allowing only expected formats and avoid blacklist filtering with properly snaitize input before processing it in system-level operations. Apply the principle of least privilege to reduce potential impact and perform regular securely testing.
Proof:



21. Vulnerability Detail – Brute Force (Security Level Low)

Severity: Critical

CVSS:9.8

Affected url: http://127.0.0.1/DVWA/vulnerabilities/brute/?username=payload1&password=payload2&Login=Login&user_token=955449c0c8887df9a75a082c49719f52#

Description: The login mechanism does not implement protections such as account lock, login attempts limitation, captcha verification or multi-factor authentication. Because of no rate limiting or detection or multiple failed login attempts, automated tools or scripts can be used to systematically guess credential. So, attacker can repeatedly submit username and password combination until the correct credential are discovered.

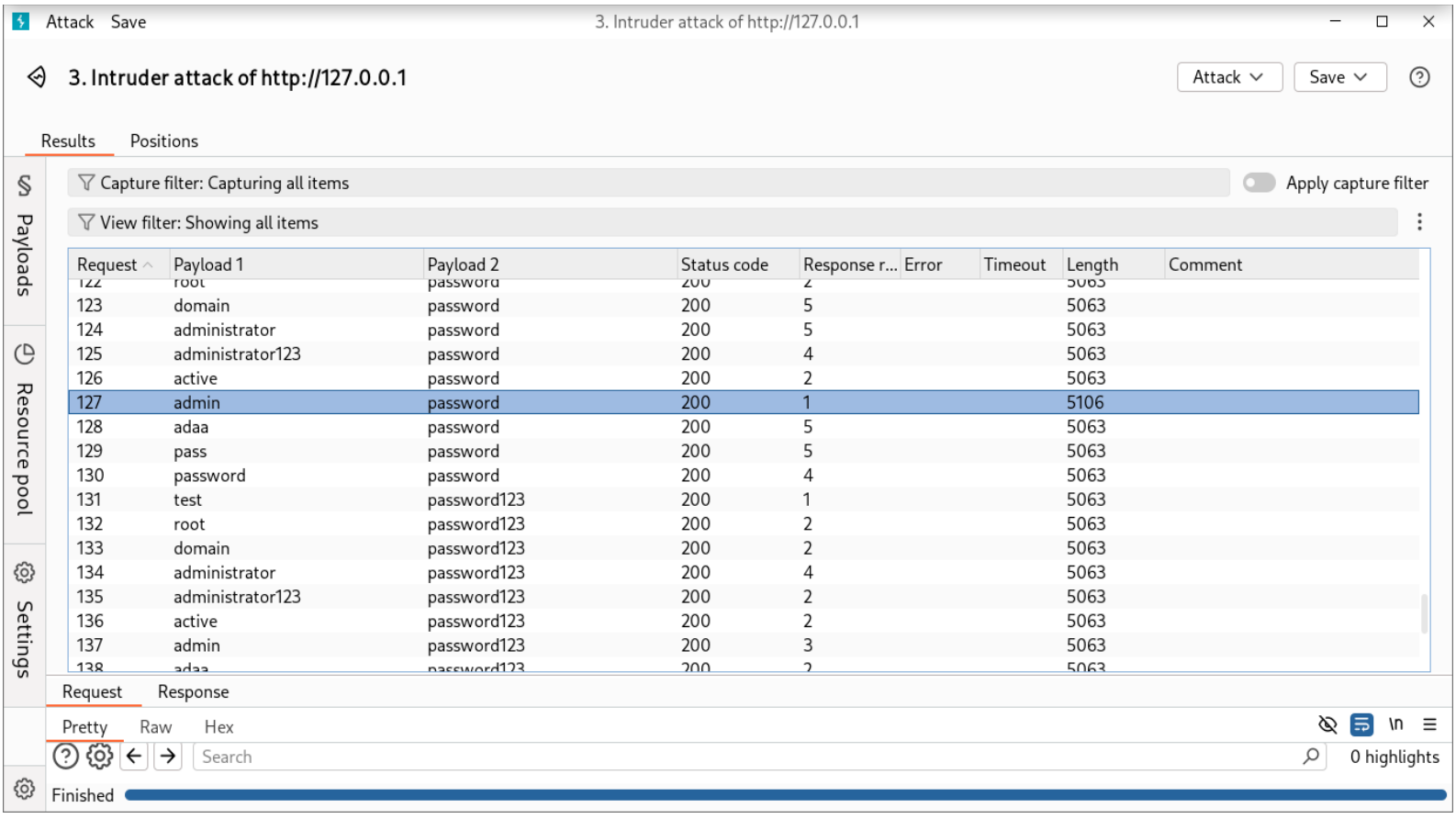
Proof of Concept:

Payload1: admin

Payload2: password

Impact: It leads to unauthorized accesses to user account, privilege escalation if admin credential are compromised, exposure of sensitive information, unauthorized modification of application data, potential full compromise of the application if admin account accessed.
Remediation: To mitigate, implement temporarily account lockout after consecutive failed login, limit the number of login request from a single ip within a specific time, use captcha or bot protections, enable multi-factor authentication, enforce strong password policy, monitor and log authentication attempts.

Proof:



22. Vulnerability Detail – Brute Force (Security Level Medium)

Severity: Critical

CVSS:9.8

Affected url: http://127.0.0.1/DVWA/vulnerabilities/brute/?username=payload1 &password =payload2&Login=Loginuser_token=955449c0c8887df9a75a082c49719f52#

Description: The application introduces a basic defense by adding a delay between login attempts. While this can slow down repeated login request but can't prevent from brute force by using automated tool and distributing login attempts across multiple requests. The application does not implement critical protections like account lockout, captcha validation, rate limiting. So, attacker can continue attempting multiple username and password combination until valid credential are discovered.

Proof of Concept:

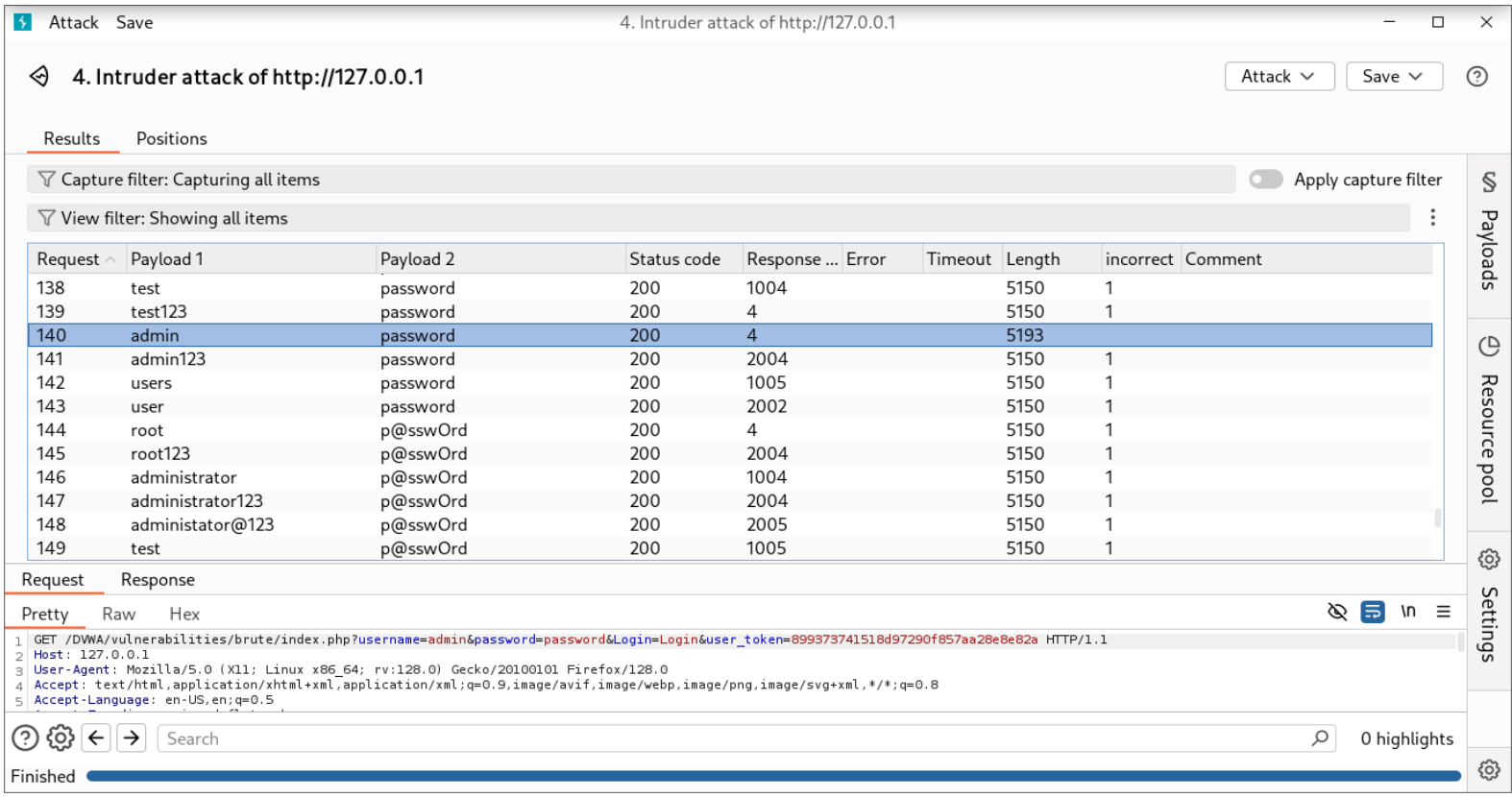
Payload1: admin

Payload2: password

Impact: It leads to unauthorized access to user accounts, compromise of sensitive information stored in the application, privilege escalation if administrator credentials are discovered, unauthorized modification of application data, potential full compromise of the application environment.

Remediation: To prevent, implement account lockout policy after certain failed attempts consecutive for specific time, limit the number of login attempts within a specific time frame from a single IP address. Use CAPTCHA verification, multi-factor authentication and implement strong password policies.

Proof:



23. Vulnerability Detail – Brute Force (Security Level High)

Severity: High

CVSS: 8.1

Affected url: http://127.0.0.1/DVWA/vulnerabilities/brute/?username=payload1 &password =payload2&Login=Loginuser_token=955449c0c8887df9a75a082c49719f52#

Description: The application introduces anti-csrf token protection during authentication process. However these protections are not fully preventing brute force attack. The application still allows repeated consecutive authentication attempts without enforcing account lockout or rate limiting. Attacker can bypass by capturing valid requests and automated login attempts while maintaining valid session parameter and token and continue brute force attack.

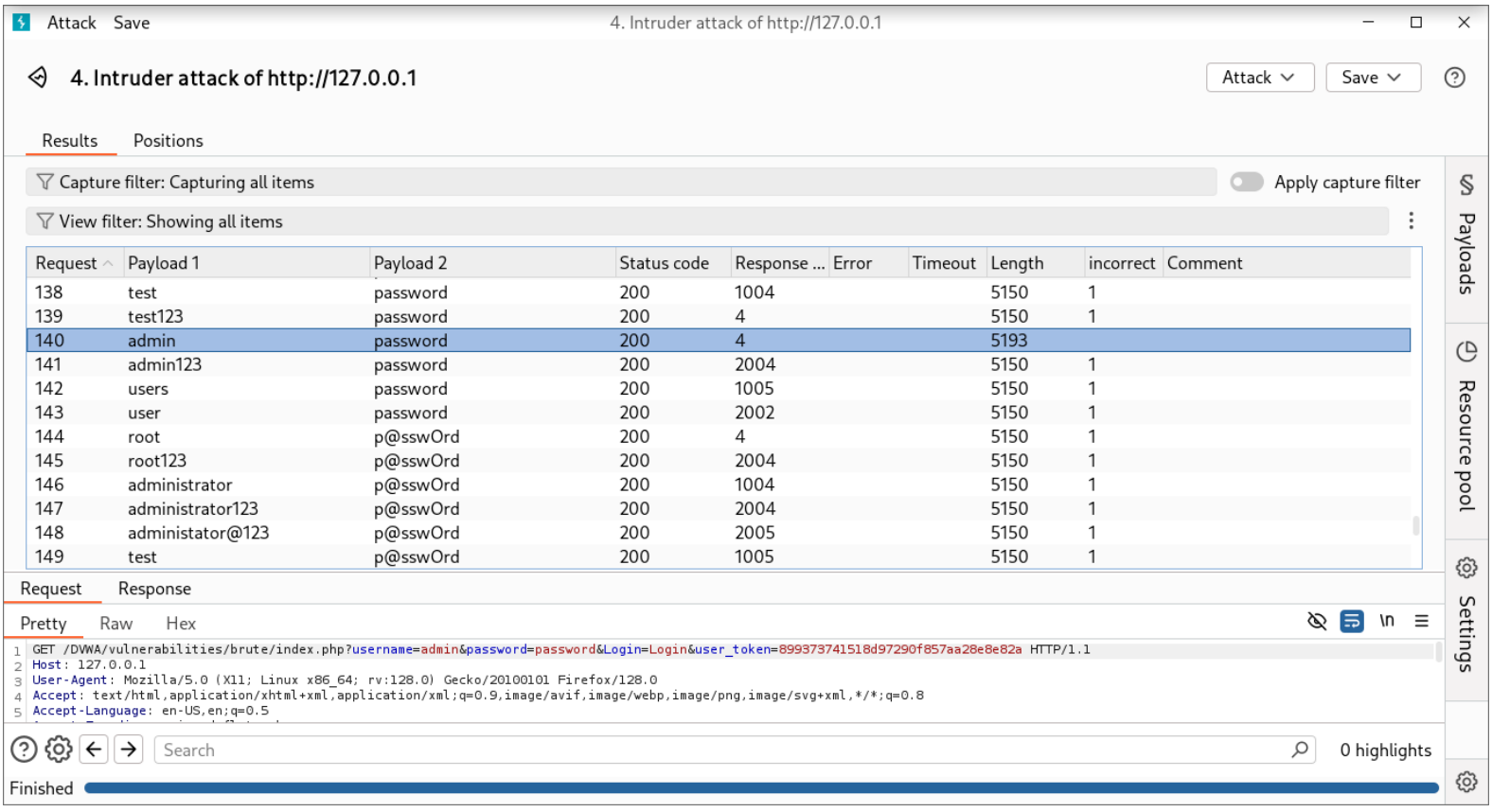
Proof of Concept:

Payload1: admin

Payload2: password

Impact: Successful exploit leads to unauthorized access to accounts, compromise of sensitive application data, privilege escalation if administrative credential are discovered, unauthorized modification or manipulation of application data, potential compromise of the entire application environment and exposure fo sensitive user information. **Remediation:** To mitigate, Lockout user account after a specific number of failed login attempts for certain time, Limit login attempts based on ip address, device fingerprinting. Implement captcha for bot detection, enable multi-factor authentication (otp etc), monitor and alert on suspicious login activity to legitimate user gmail or id.

Proof:



24. Vulnerability Detail – Insecure Captcha (Security Level Low)

Severity: Medium

CVSS: 5.4

Affected url: http://127.0.0.1/DVWA/vulnerabilities/captcha/

Description: The captcha verification process does not properly validate whether the correct captcha value is submitted before processing the authentication request. As a result, attacker can manipulate the HTTP request and directly send authentication parameter without successfully solving the captcha challenge Because server does not enforce strict captcha validation, automated tools or scripts can bypass the captcha mechanism and perform task repeatedly. This effectively renders the captcha protection ineffective and allow attacker to automated attack.

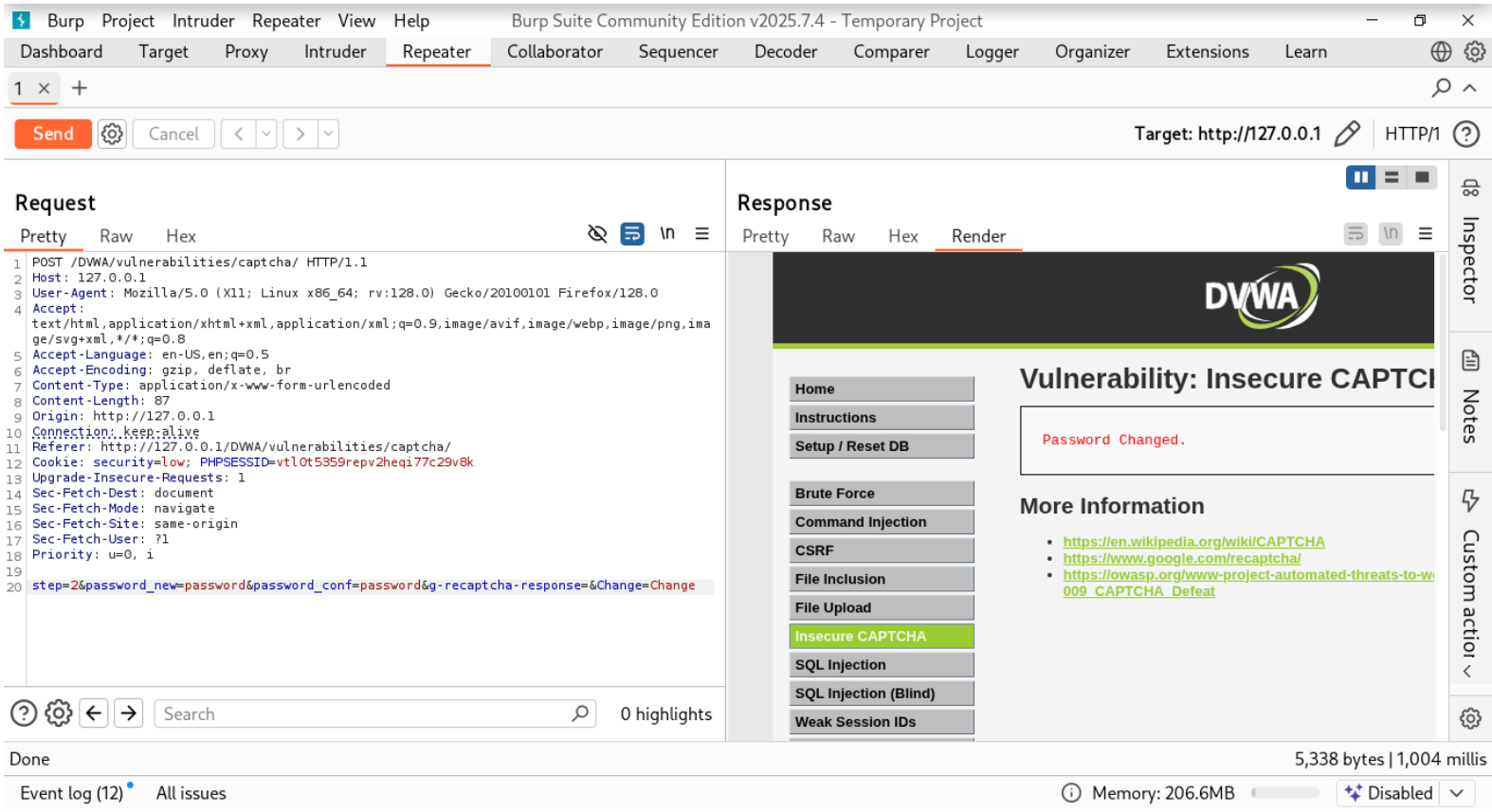
Proof of Concept:

Payload: Step=2

Impact: It leads to automated tasks, unauthorized access to application accounts, increase risk of credential stuffing attacks, Exposure of sensitive data if account is compromised, increased load on the server due to automated attack traffic.

Remediation: To mitigate, ensure that captcha values are validated on the server before processing authentication request if not then reject the request, Bind the captcha validation to session, Limit the number of authentication attempts from a single ip address within a defined time to reduce automated attacks. Use modern captcha framework that include bot detection mechanism, Monitor and log suspicious activity do detect automated attacks early.

Proof:



25. Vulnerability Detail – Insecure Captcha (Security Level Medium)

Severity: Medium

CVSS: 5.4

Affected url: http://127.0.0.1/DVWA/vulnerabilities/captcha/

Description: The application performs captcha verification but does not adequately protect the authentication workflow from manipulation. Attacker intercept and modify HTTP requests or manipulate parameter to bypass captcha verification. Because captcha verification is not tightly enforced through the authentication process, it is possible to submit change attempts without properly solving captcha challenge. Application lacks other verification such as rate limiting, account lockout, and advanced bot detection. As a result automated tool or script still perform repeated attempts with minimal restriction.

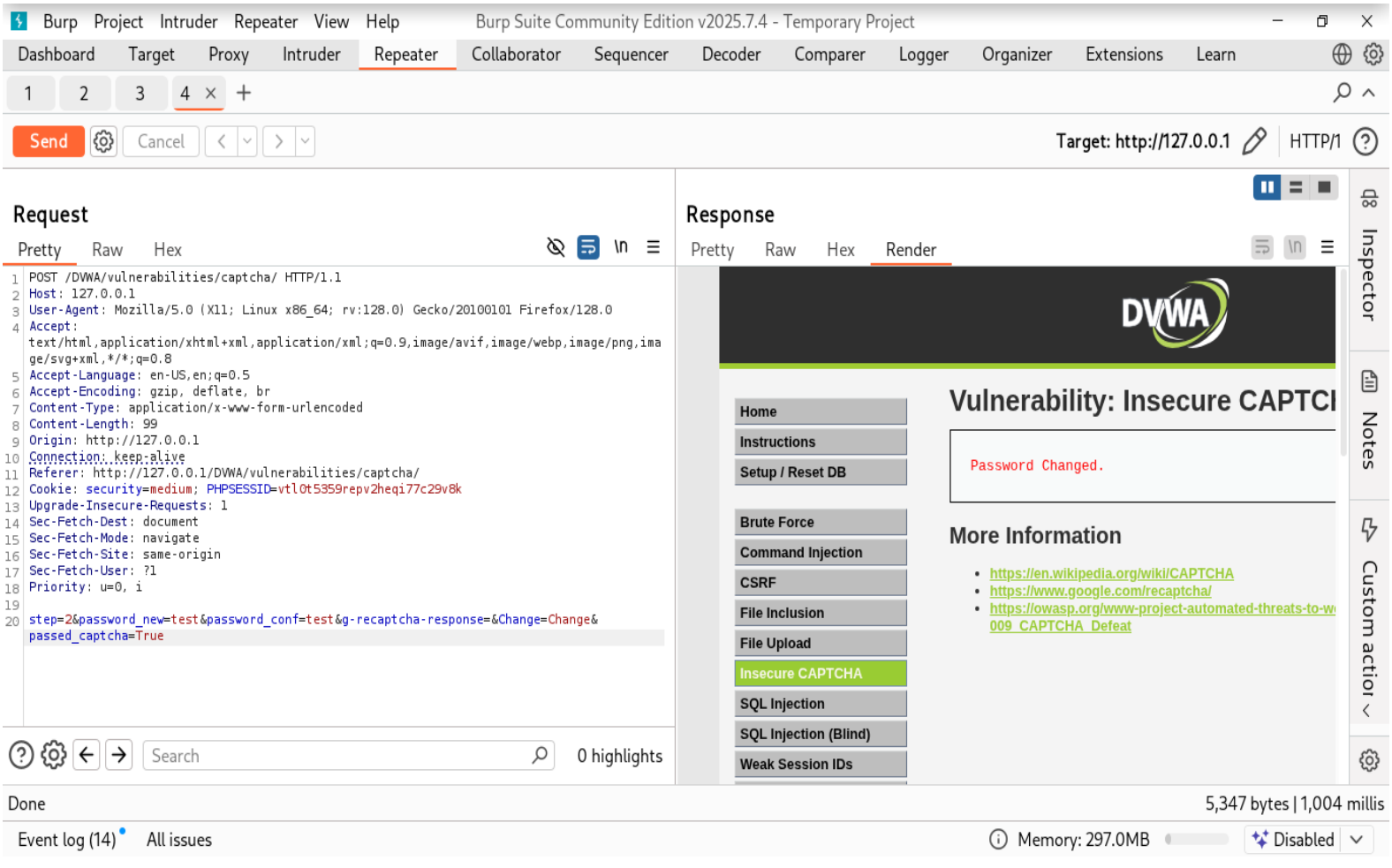
Proof of Concept:

Payload: Step=2 and passed_captcha = True

Impact: It leads to automated task against password change, increased risk of credential change and account compromise, unauthorized change of password.

Remediation: To prevent, Ensure that authentication requests are processed only after successful captcha verification if not then reject requests, Bind captcha validation to session state, implement rate limit of attempt from single IP address with temporarily lock function after multiple failed attempts and deploy advanced bot detection mechanism.

Proof:



26. Vulnerability Detail – Insecure Captcha (Security Level High)

Severity: Medium

CVSS: 4.2

Affected url: http://127.0.0.1/DVWA/vulnerabilities/captcha/

Description: The captcha implement can still be bypassed under the certain conditions due to insufficient enforcement of captcha validation within the authentication workflow. Attacker manipulate HTTP requests reuse valid session parameters, or automate interaction to bypass the captcha verification process.

Proof of Concept:

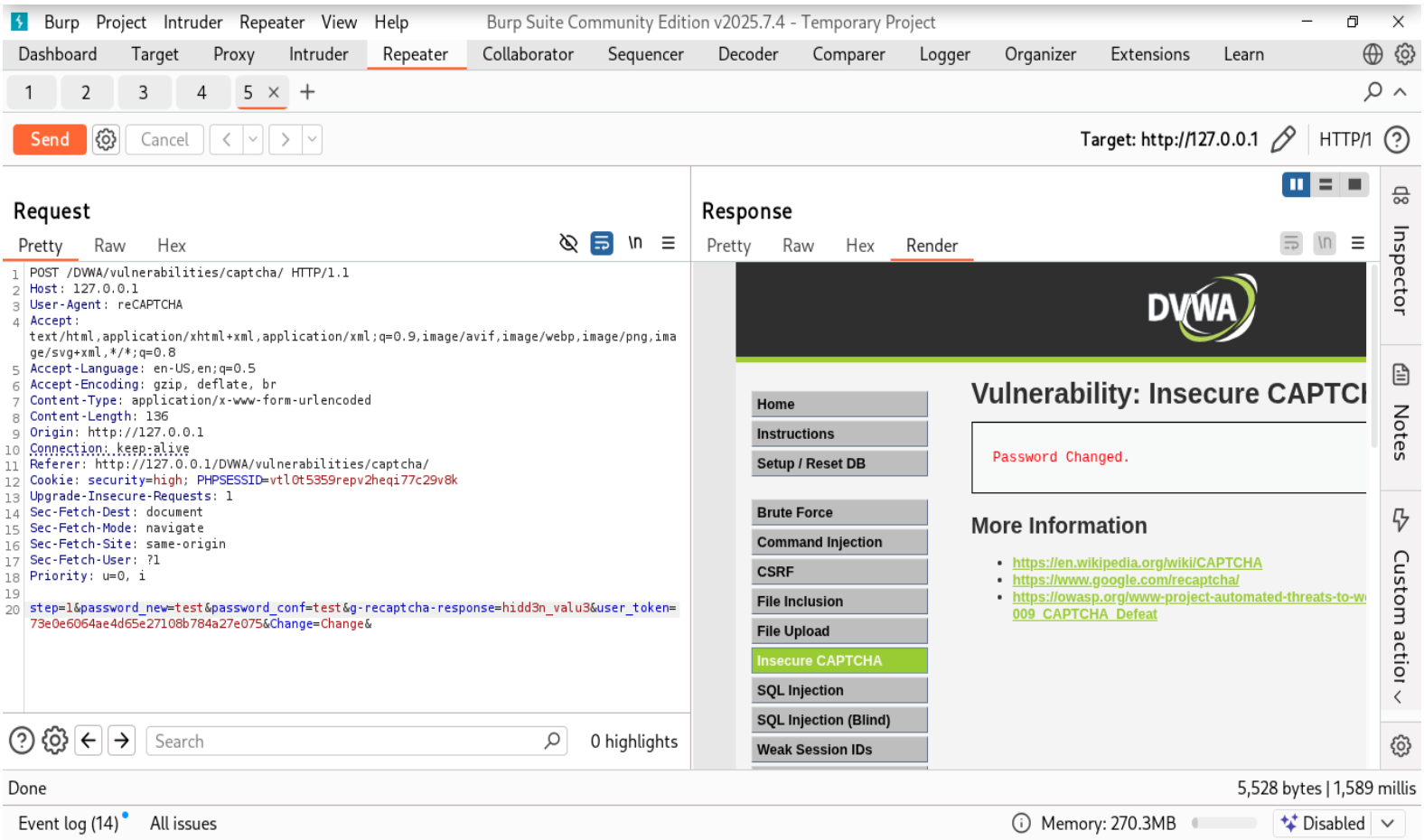
Payload:1 user-agent: reCAPTCHA (in http header value)

payload2: g-recaptcha-response=hidd3n_valu3 (in post data)

Impact: The potential impacts are automate password change, unauthorized change of password, potential privilege escalation if administrative account are targeted.

Remediation: Ensure that captcha verification is performed and validated on the server before any authentication request is processed. Restrict the number of attempts allowed within a specific time from a single ip address. Temporarily lock function after multiple failed attempts and use modern captcha framework that incorporate bot detection technique, enable multi-factor authentication.

Proof:



27. Vulnerability Detail – Weak Session Id (Security Level Low)

Severity: High
CVSS: 7.3

Affected url: http://127.0.0.1/DVWA/vulnerabilities/weak_id/

Description: The application generate session id using insufficient random values, which is easy to guess and predict. Because the session IDs lack strong randomness and entropy, attacker able to enumerate and perdict valid session token belong to other users. The session ID generation mechanism is weak and predictable.

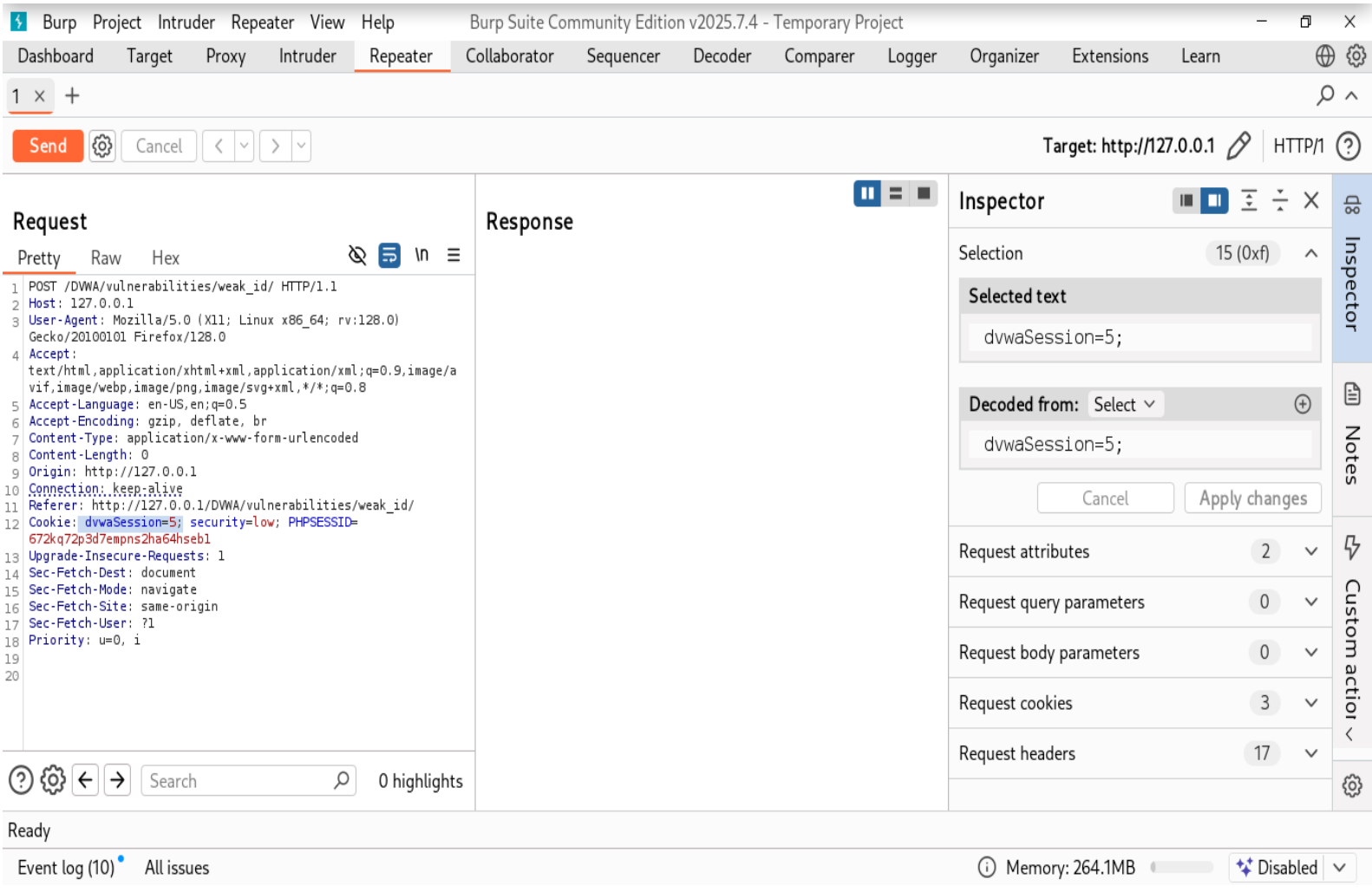
Proof of Concept:

Payload: dvwa_session= 5

Impact: Successful exploitation leads to unauthorized access to legitimate user accounts, session hijacking attacks, Exposure of sensitive user data, privilege escalation if session are compromised, unauthorized modification of application data, loss of confidentiality of user.

Remediation: To mitigate, Session identifiers should be generated using secure random number generators with high entropy, use long unpredictable session tokens that cannot be easily guessed or enumerated, implement secure cookie attributes like httponly, secure,SameSite, regenerate session ID after authentication, implement session expiration policies.

Proof:



28. Vulnerability Detail – Weak Session Id (Security Level Medium)

Severity: High
CVSS:7.3

Affected url: http://127.0.0.1/DVWA/vulnerabilities/weak_id/

Description: The session ID generation process lack of sufficient entropy and cryptographic strength. Because the session identifiers are not generated using a fully secure randomization mechanism, it is still possible to predict or enumerate valid session token through analysis or repeated requests. It demonstrate insufficient session management practice and inadequate protection against session IDs prediction.

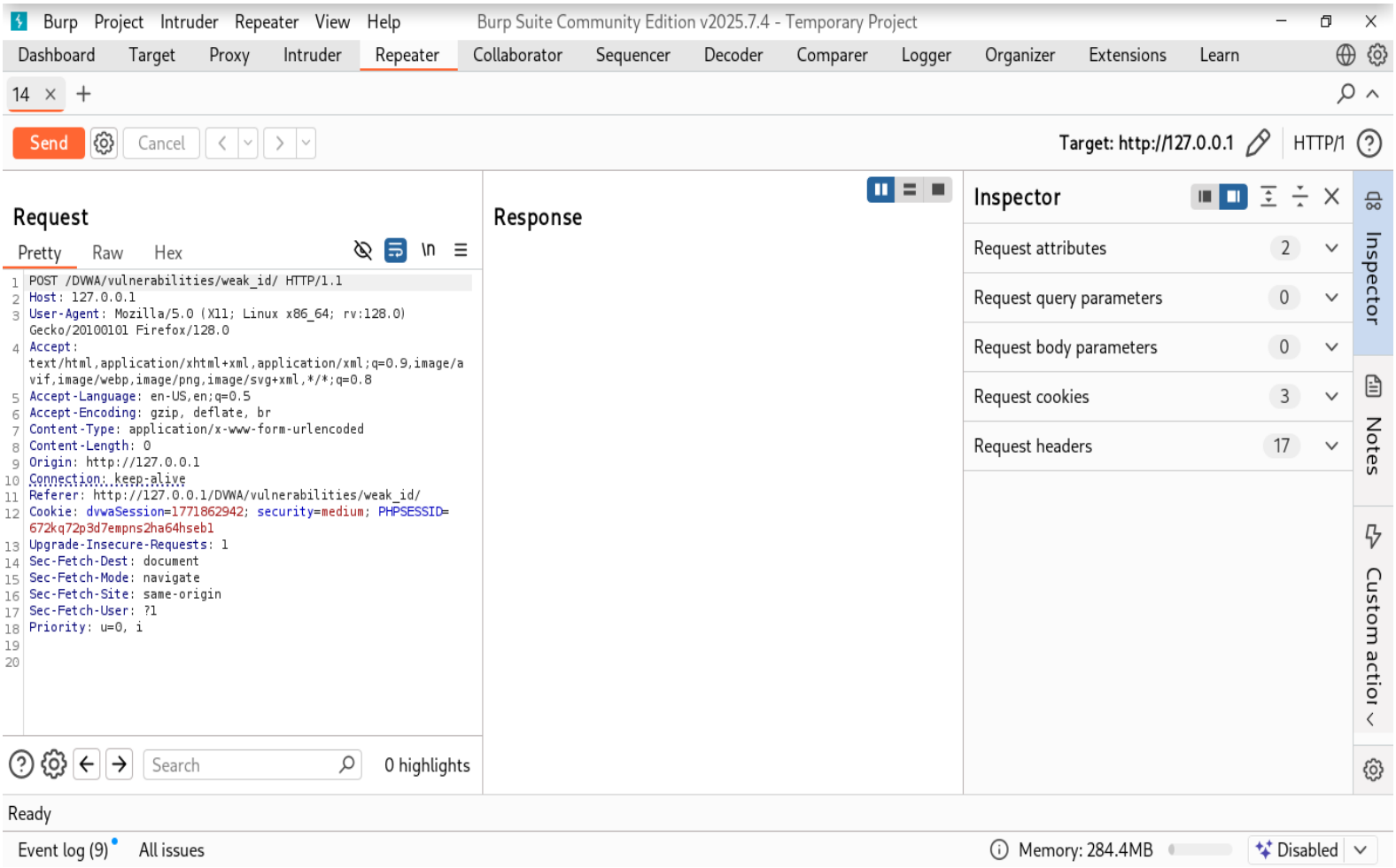
Proof of Concept:

Payload: dvwa_session= 1771862942 (increase with time last digit)

Impact: Successful exploit leads to unauthorized access to legitimate user accounts, session hijacking attacks, exposure escalation if administrative session are compromised, unauthorized modification of application information, compromise of user confidentiality.

Remediation: To mitigate, session identifiers should be generated using strong cryptographic random generators with sufficient entropy, increase session id length and randomness to prevent guessing or enumeration, implement secure cookie attributes like httponly, samesite, immediately regenerate new session id after authentication and implement session timeout mechanism.

Proof:



29. Vulnerability Detail – Weak Session Id (Security Level High)

Severity: Medium
CVSS:5.6

Affected url: http://127.0.0.1/DVWA/vulnerabilities/weak_id/

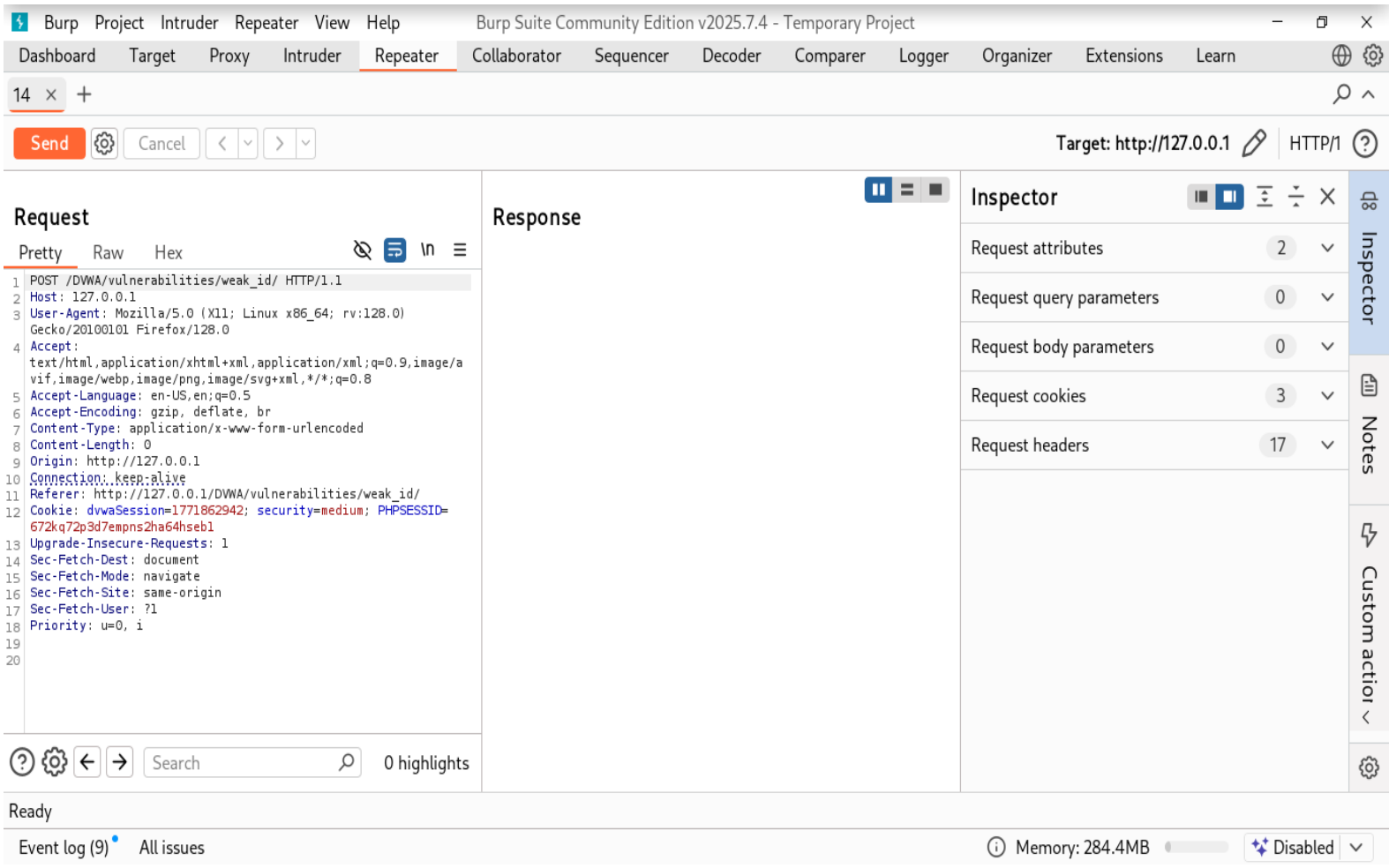
Description: The session IDs are generated using a more secure randomization method, which significantly reduces likelihood of predicting or enumerating valid session tokens. Despite these improvements, improper session management practice such as insufficient session expiration policies, lack of additional verification, improper session handling in certain scenarios still expose the application session hijacking if tokens are leaked.
Proof of concept:

Payload: dvwasession= guessalble

Impact: It leads to unauthorized accesses to legitimate user account, session hijacking attacks, exposure of sensitive user information, privilege escalation if administrative session are compromised, unauthorized modification of application data.

Remediation: To mitigate, Ensure that session identifiers are generated using secure cryptographic random number generator with sufficient entropy, generate new session id after successful authentication to prevent session fixation attacks, implement secure cookie attributes like httponly, secure, samesite. Implement session expiration after certain time.

Proof:



30. Vulnerability Detail – Authorization Bypass (Security Level Low)

Severity: High
CVSS: 8.8

Affected url: http://127.0.0.1/DVWA/index.php

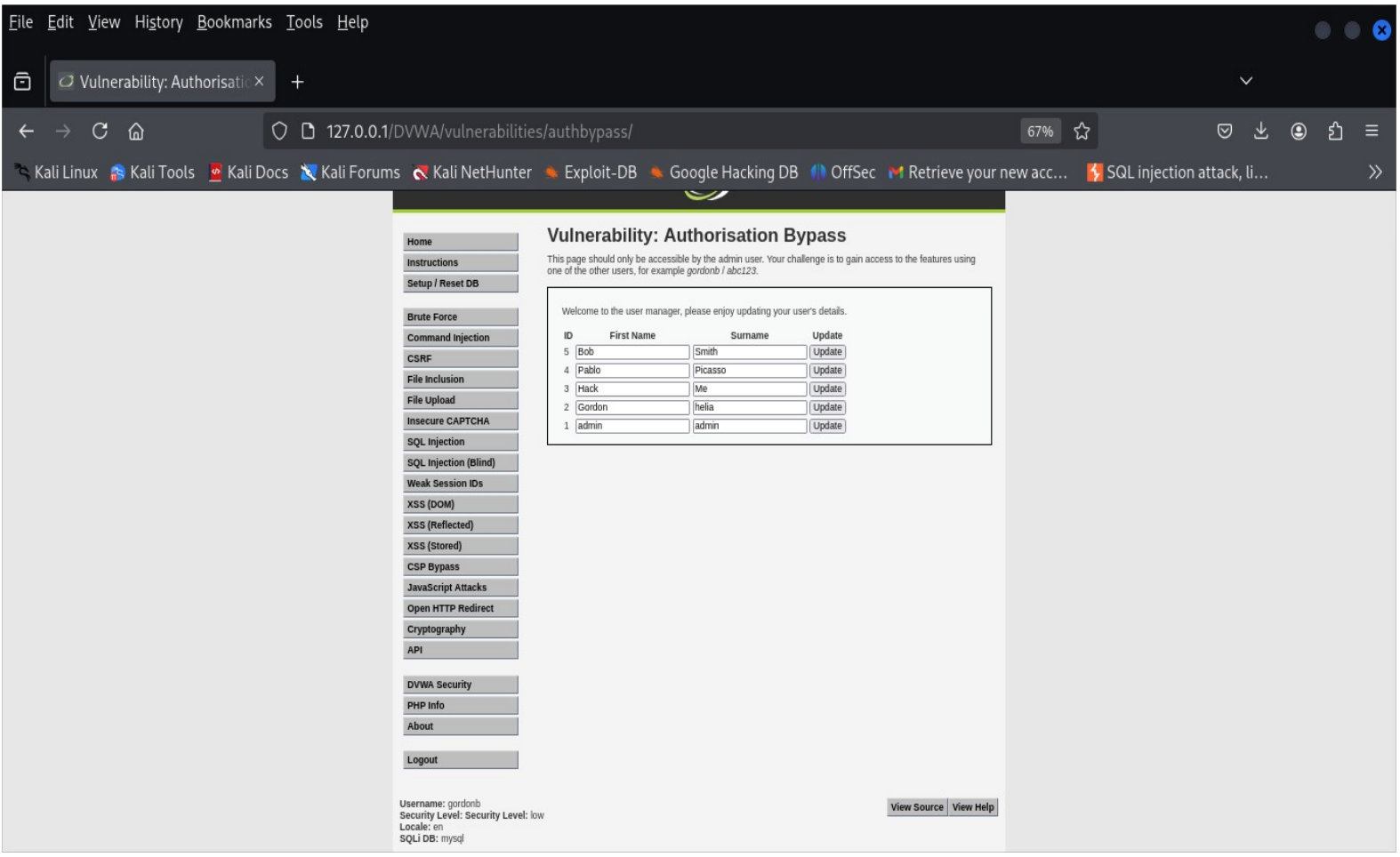
Description: The application relies on weak or missing server-side authorization controls to verify whether a user is permitted to access certain functionality or data because proper authorization validation is not implemented, directly manipulate request url or identifier to access resources belonging to other privilege or user. The application process these requests without verifying whether the requesting user has sufficient permissions. It demonstrate improper implemention of access mechanism and lack of server-side validation.
Proof of concept:

Payload: vulnerabilities/authbypass (http://127.0.0.1/DVWA/vulnerabilities/authbypass)

Impact: It leads to unauthorized access to sensitive data, access to restricted application functionality, privilege escalation to higher-privileged user roles, unauthorized modification or deletion of application data and exposure of confidential information belonging to other users.

Remediation: To mitigate, Implement proper server-side authorization to access the requested resources, define clear access control rules based on user roles, validate user identity for resource access, avoid trusting client-side parameter.

proof:



31.Vulnerability Detail– Authorization Bypass(Security Level Medium)

Severity: High
CVSS: 8.8

Affected url: http://127.0.0.1/DVWA/index.php

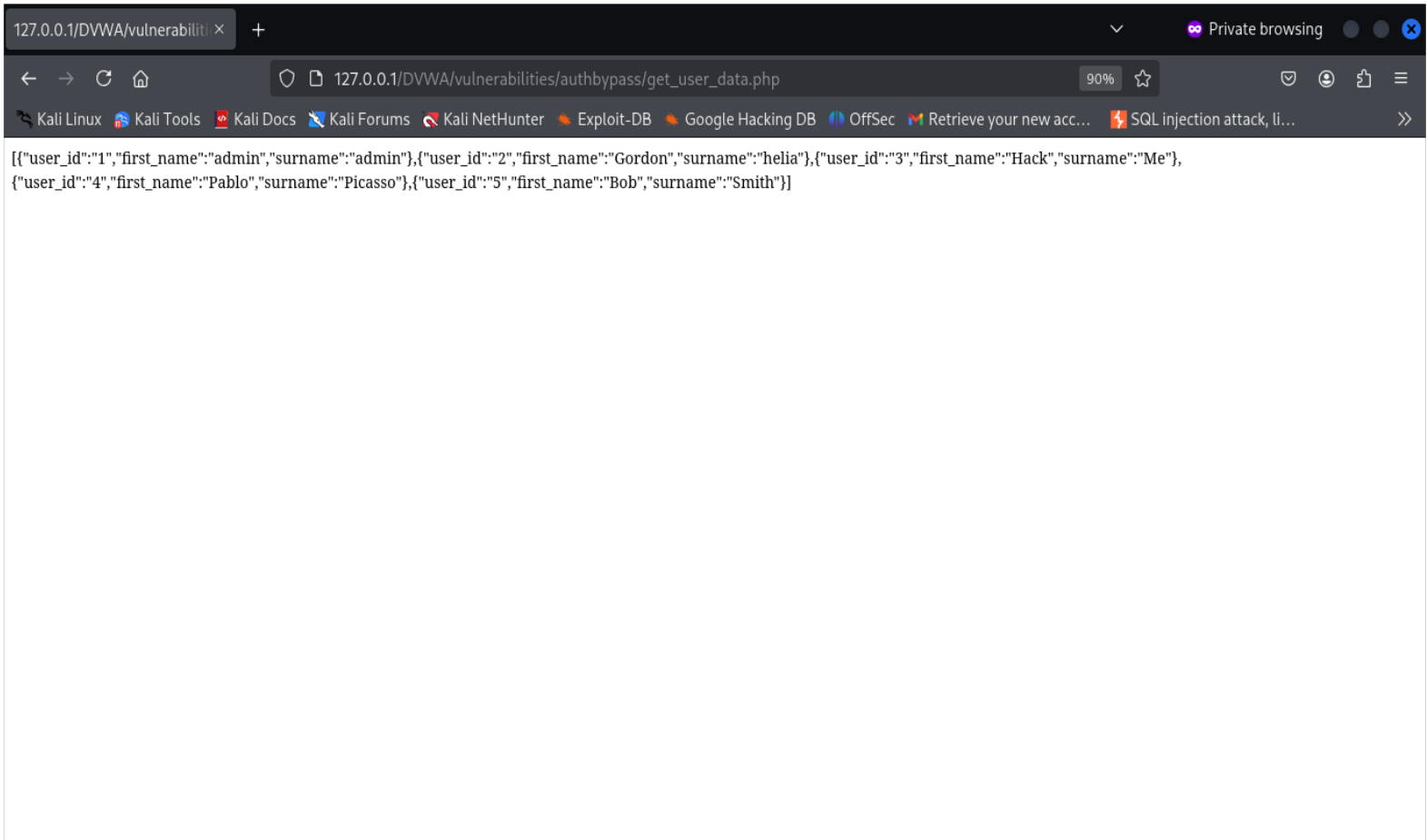
Description: The application implement basic authorization controls to restrict access to certain resource and functionalities, (/vulnerabliites/authbypass). However the authorization mechanism is still bypassed by manipulating url endpoint. The application rely on insufficient validation of request parameter or incomplete server-side authorization checks. Attacker manipulate url directly and can access restricted resource because authorization checks are not consistently enforced across all application components.
Proof of concept:

Payload: vulnerabilities/authbypass/get_users_data.php (http://127.0.0.1/DVWA/payload)

Impact: It leads to unauthorized access to sensitive information, access to restricted application functionalities, privilege escalation to higher user roles, unauthorized modification of application data, exposure of confidential user or system information.

Remediation: To mitigate, Ensure every request to protected resources is validated on the server side to confirm that the authenticated user has sufficient permissions, implement structured access control mechanism based on clearly defined user roles and permissions, validate resource ownership, avoid relying on client-side data for authorization.

proof:



32.Vulnerability Detail– Authorization Bypass(Security Level High)

Severity: High
CVSS: 7.5

Affected url: http://127.0.0.1/DVWA/index.php

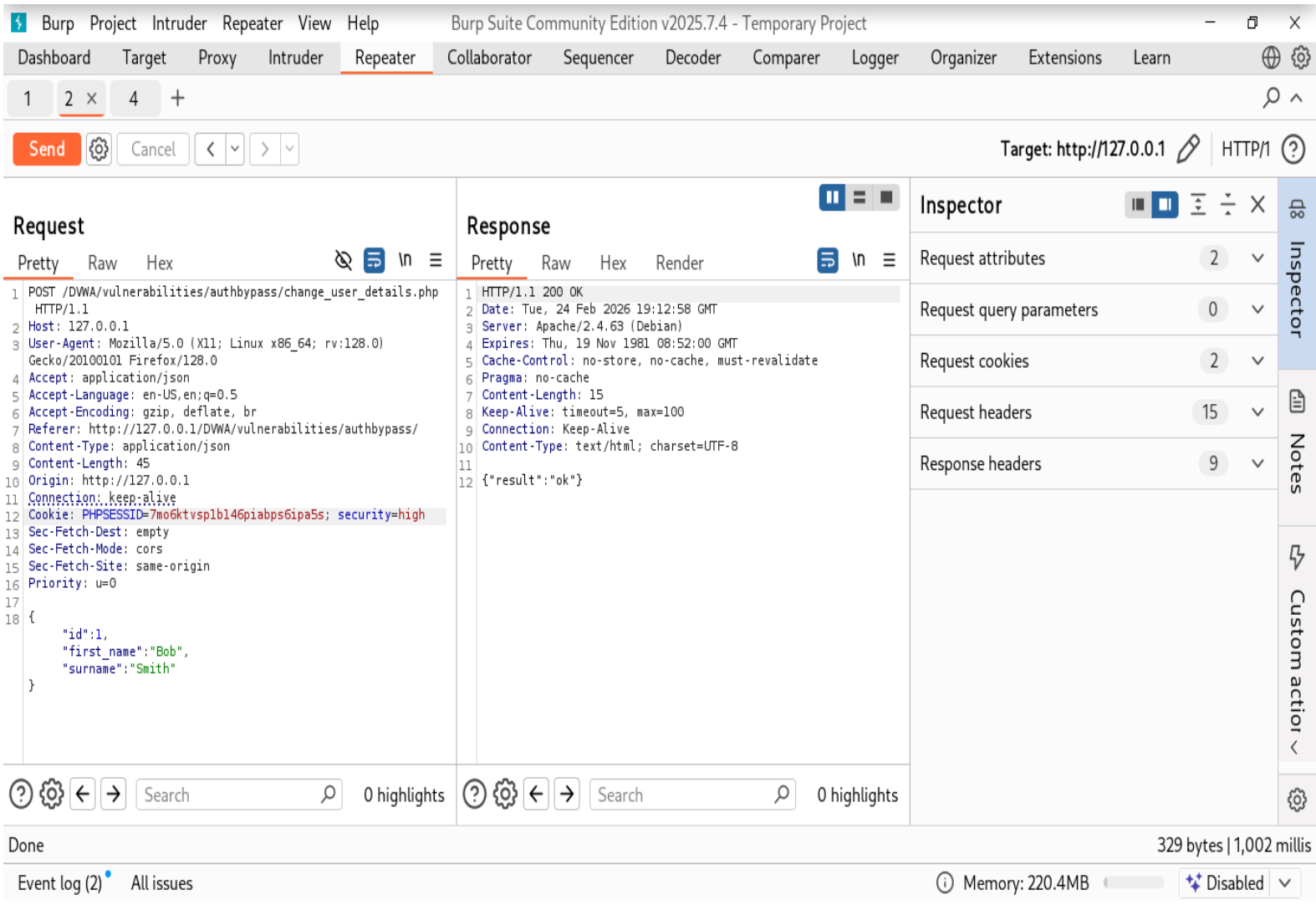
Description: The application perform server-side verification of user role and resource access right, ensuring that users can only perform action permitted by their assigned privilege. However the application allow to modify the data of higher privileged through lower privilege accounts by manipulation the http request, there is cookies logic flaw. By interchanging the cookies of normal user to the administrative cookies it allows to modify the data and vulnerable to the cookies logic and improper implement of validation.
Proof of concept:

Payload: PHPSESSID=normal user cookie

Impact: It leads to unauthorized modification of application data, potential compromise of sensitive information, violation of update control policies, loss of user trust.

Remediation: To mitigate, Ensure that all the request to protected resource are validated on the server side and that update permissions are properly enforced, apply principle of least privilege, Avoid trusting client-side parameters.

Proof:



33.Vulnerability Detail– Open HTTP Redirect(Security Level Low)

Severity: Medium
CVSS: 4.3

Affected url: http://127.0.0.1/DVWA/vulnerabilities/open_redirect/source/low.php?redirect=paylaod&id=1

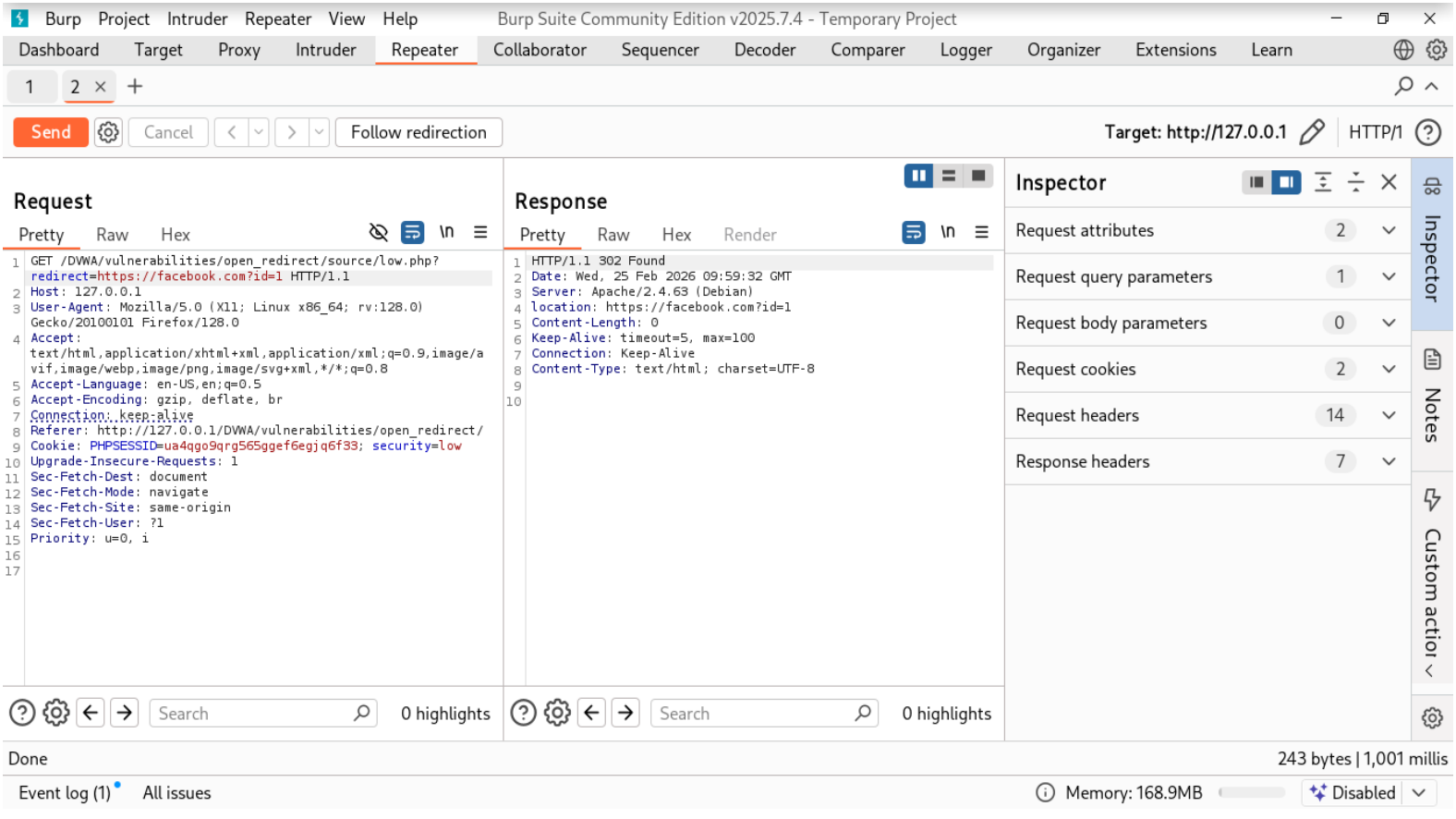
Description: The application accepts a url parameter that determines where the user will be redirected after performing a specific action. Because the application does not validate or restrict the redirection, an attacker can manipulate the parameter value to redirect users to an external malicious websites. The server processes the request and redirect the user without verifying whether the destination url is trusted or not. This weakness allows attackers to craft specially crafted urls that appears to originate from the legitimate application but redirect users to attacker-controlled websites.
Proof of concept:

Payload: https://facebook.com

Impact: It leads to phishing attacks targeting legitimate users, theft of login credential through fake authentication pages, distribution of malware through malicious websites, loss of user trust in the application, potential exploitation of other vulnerabilities through chained attacks.

Remediation: To mitigate, redirect only to validate urls and restricted to untrusted domains within the application. Maintain a list of approved redirect destination and only allow redirect to url in allowlist, avoid using user-controlled input for redirection, use internal application path instead of full external url.

proof:



34.Vulnerability Detail– Open HTTP Redirect (Security Level Medium)

Severity: Medium
CVSS: 4.3

Affected url: http://127.0.0.1/DVWA/vulnerabilities/open_redirect/medium.php?redirect=payload&id=1

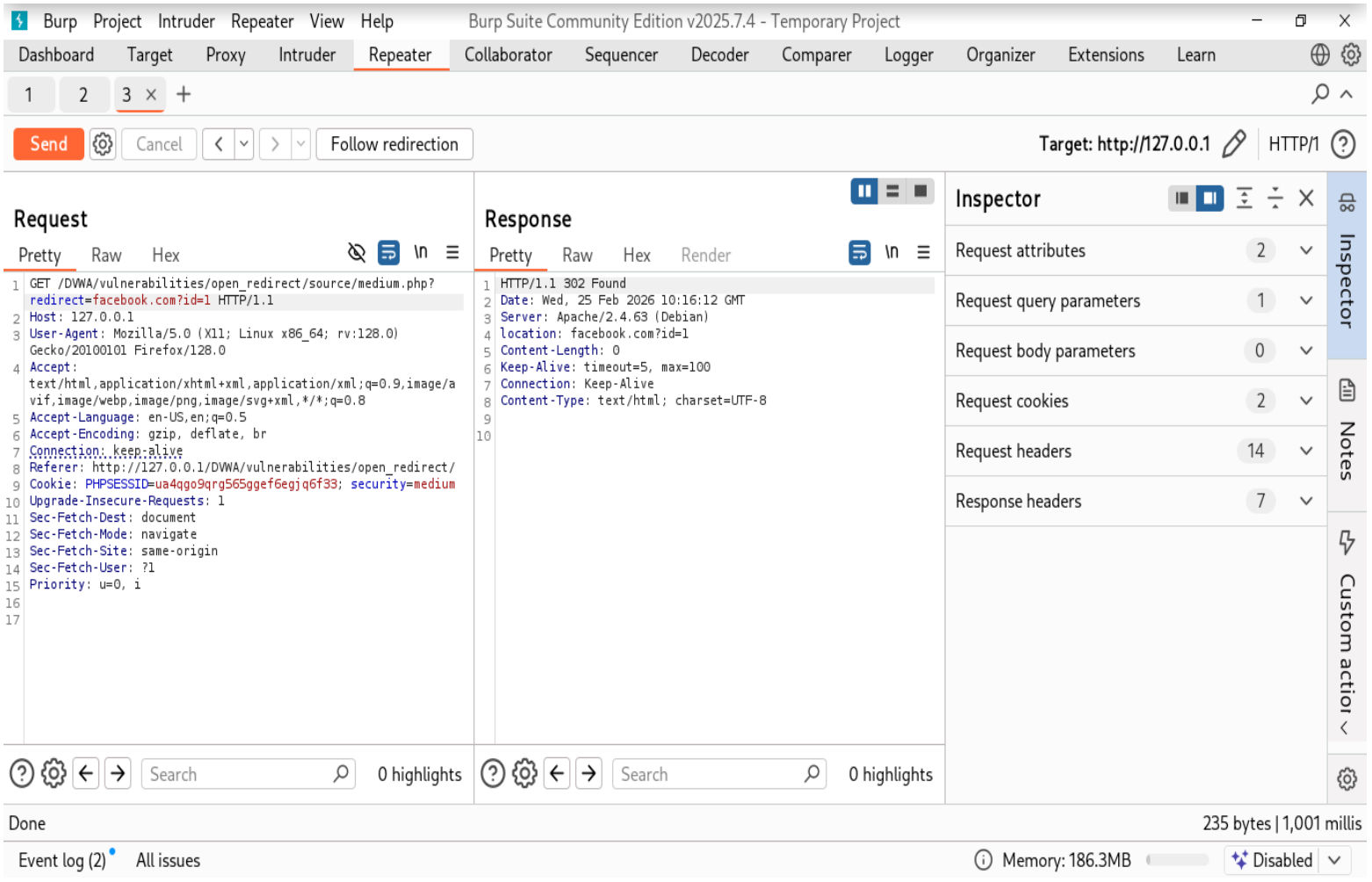
Description: The application introduce basic input validation to restrict malicious redirection attempts. However the validation mechanism is not comprehensive and only check for specific patterns or partially restrict external urls. Still bypass these checks by manipulating the redirect parameter using encoded urls, alternative formats or modified input values because redirection destination is influenced by user-supplied input. Proof of concept:

Payload: facebook.com

Impact: It leads to phishing attacks, credential theft through fake login pages, distribution of malware through malicious websites, loss of user trust and potential chaining with other social engineering attacks.

Remediation: To prevent, Validate redirect destination to ensure belong only to trusted domain within the application, maintain predefined list of approved redirect url only allow to these destination, avoid using external urls in redirect parameter, apply proper input validation and sanitization to prevent manipulation of redirect parameter.

proof:



35.Vulnerability Detail– Open HTTP Redirect (Security Level High)

Severity: Low
CVSS: 3.0

Affected url: http://127.0.0.1/DVWA/vulnerabilities/open_redirect/source/high.php?redirect= payload&id=1

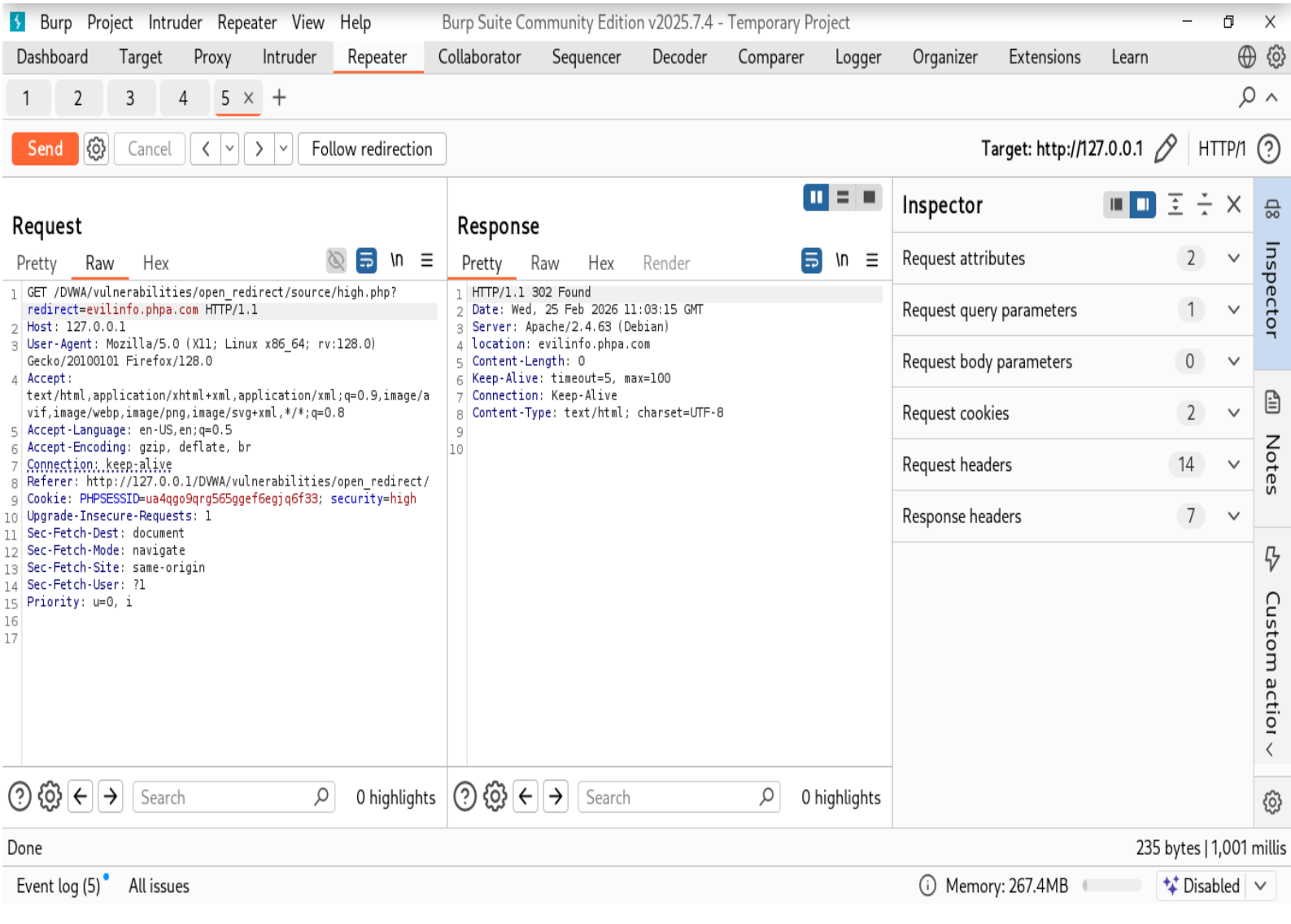
Description: The application validates the redirect destination and allow only predefined internal paths or trusted domain and check the pattern info.php. Because of this validation unable to manipulate the redirect parameter to send users to attacker-controlled external websites. However attacker can redirect to malicious websites pattern related to info.php like evilinfo.php.com. So attacker can abuse the parameter and redirect to fake website.
Proof of concept:

Payload: evilinfo.php.com

Impact: It leads to redirection of legitimate users to malicious websites, phishing attacks targeting application users, credential theft, malware distribution through phishing page.

Remediation: Ensure that the application only allows redirect to predefined internal application routes, implement allow-list based validation, avoid using user-supplied urls.

proof:



36.Vulnerability Detail– csrf (Security Level Low)

Severity: Medium
CVSS: 4.3

Affectedurl:http://127.0.0.1/DVWA/vulnerabilities/csrf/?password_new=test123& password_conf =test123&Change=Change

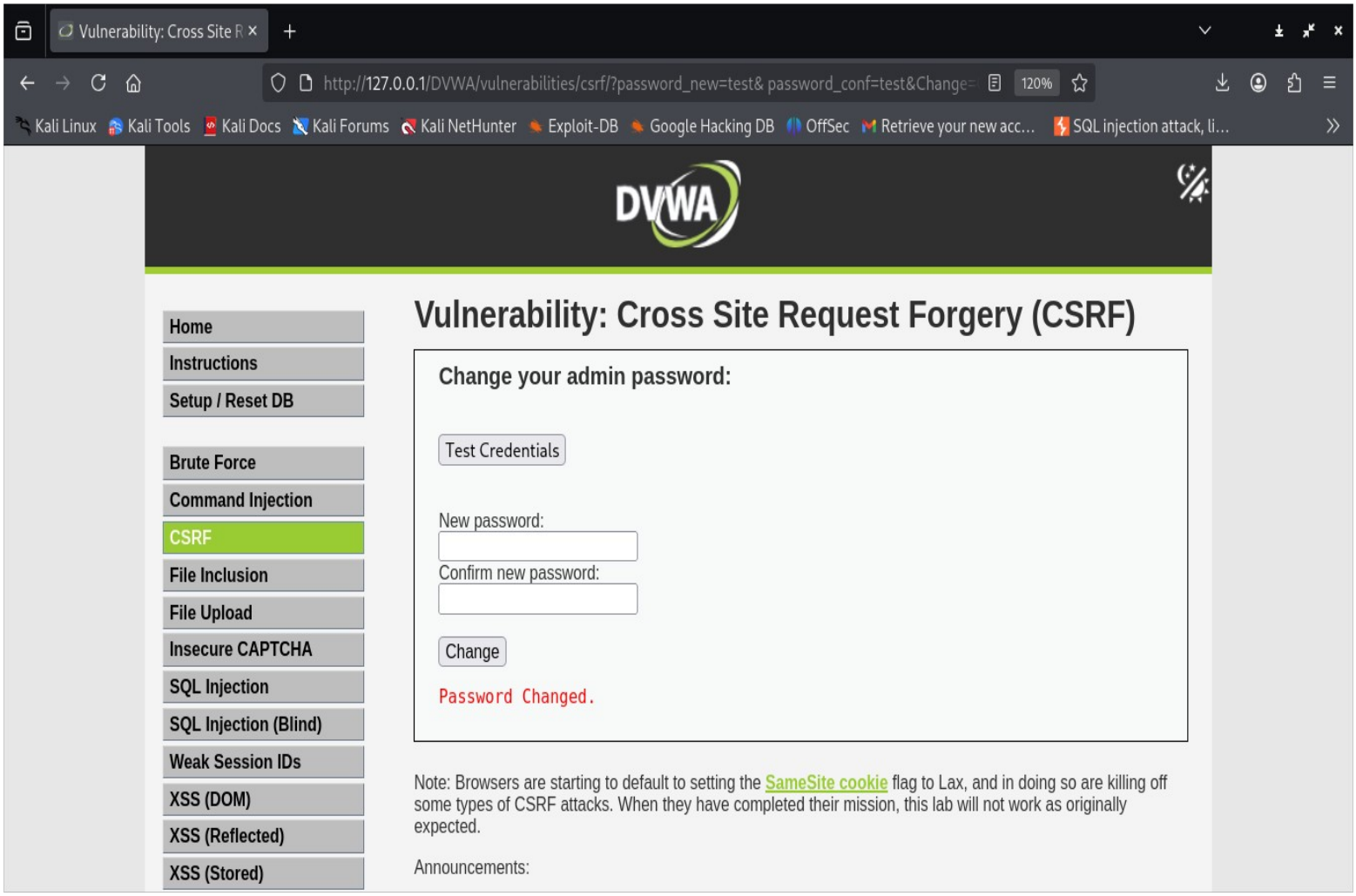
Description: The application processes state-changing request without verifying whether the request was intentionally initiated by the authenticated user. The application does not implement anti-csrf token or any other protection mechanism to validate the authenticity of requests. As a result attacker can craft and force to perform intended action in websites.
Proof of concept:

Pyaload:http://127.0.0.1/DVWA/vulnerabilities/csrf/?password_newPayload:ayload:=test123& password_conf =test123&Change=Change

Impact: It leads to unauthorized modification of user accounts information, changing user credential such as password, compromise of user account and application integrity.

Remediation: To mitigate, includes a unique and unpredictable csrf token in all state-changing requests. The server must verify the token before processing the request. Verify the origin or refer header to ensure that requests originate from trusted application domain, use samesite cookie attributes and require reauthentication for sensitive actions.

proof:



37.Vulnerability Detail– Content Security Policy (CSP) Bypass (Security Level Low)

Severity: Medium
CVSS: 4.6

Affected url: http://127.0.0.1/DVWA/vulnerabilities/csp/

Affected parameter: include= https://digi.ninja/dvwa/alert.js

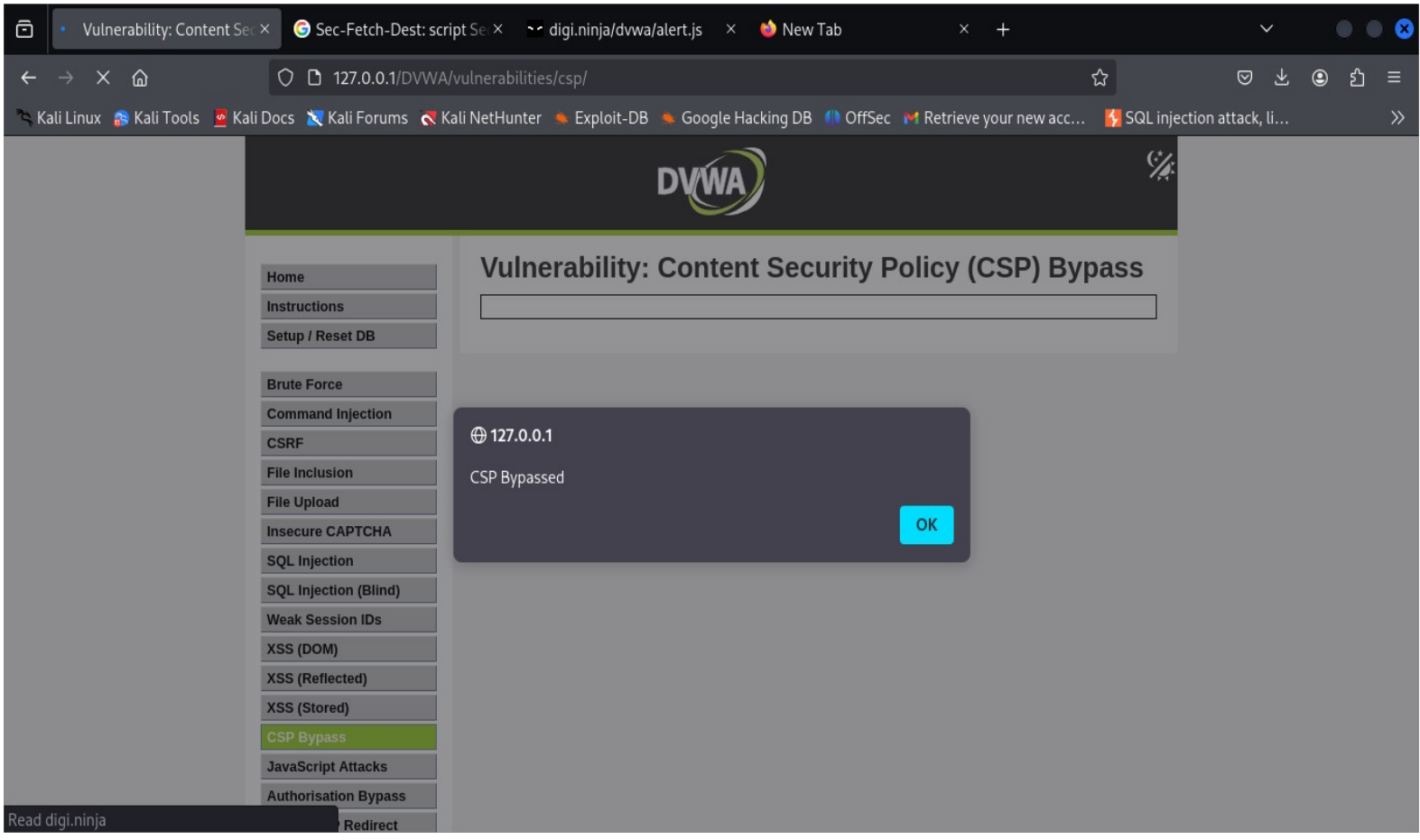
Description: The server response lacks the Content-Security-Policy (csp) header entirely. Without csp, the browser allows content from any source, including inline scripts and third party domains. An attacker exploits this weakness by injecting malicious scripts through vulnerabilities like xss. Since the browser has no policy restricting script execution, the injected paylaod execute without restriction. It executes all scripts from various sources. Proof of concept:

Payload: alert('CSP Bypassed');

Impact: It leads to execution malicious JavaScript in the victim's browser, steal session cookies or authentication tokens, perform session hijacking or unauthorized access, inject malicious external scripts from attacker-controlled domains, increase the severity and exploitability of xss vulnerabilities.

Remediation: To mitigate, Add a content-Security-Policy header in server response, Restrict script execution to trusted domain only, Disable inline JavaScript and unsafe evaluation, allow resource only from trusted origins.

proof:



38.Vulnerability Detail– Content Security Policy (CSP) Bypass (Security Level Medium)

Severity: High
CVSS:7.2

Affected url: http://127.0.0.1/DVWA/vulnerabilities/csp/

Affected parameter: include=payload

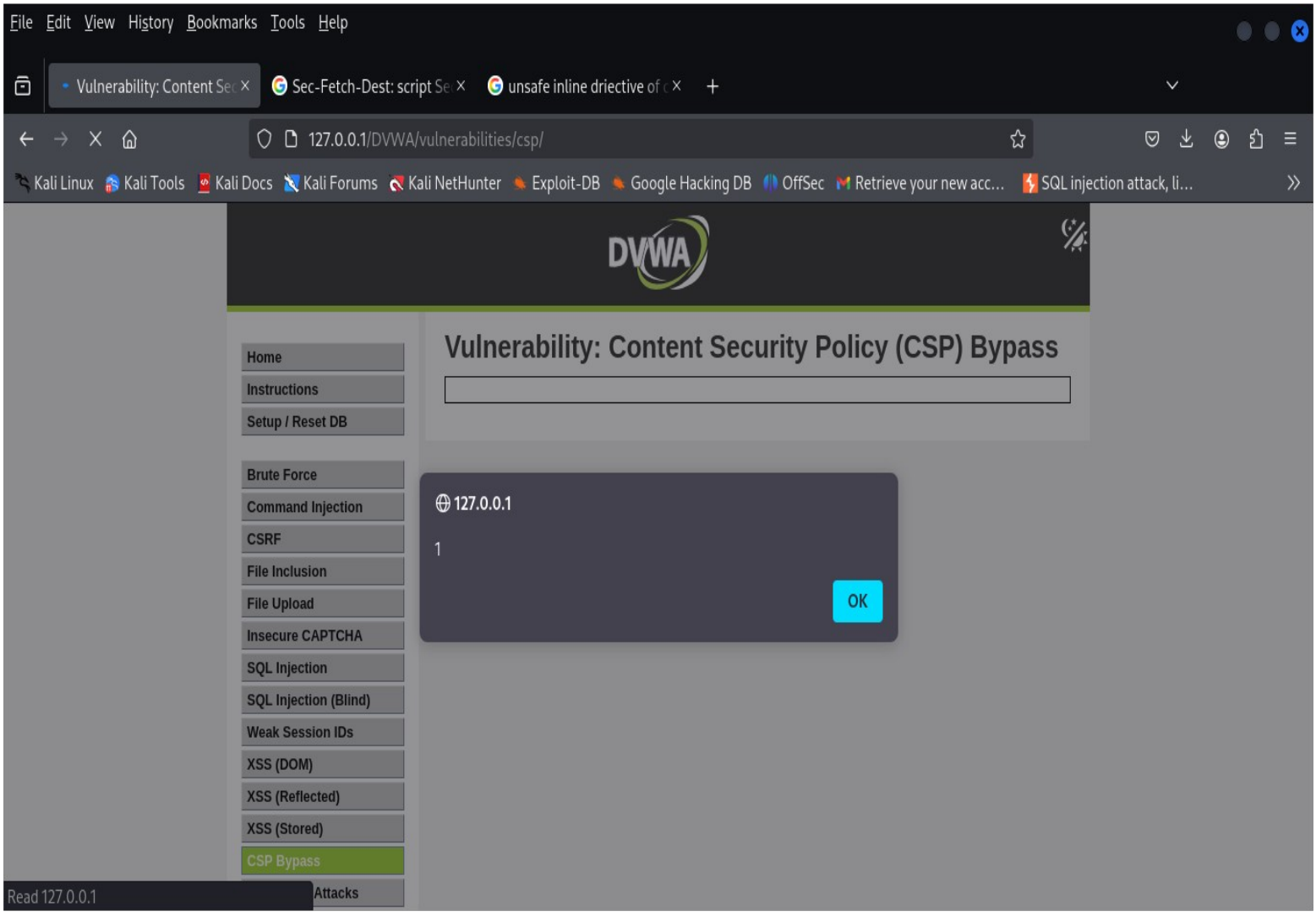
Description: The csp header allow directives such as unsafe-inline or unsafe-eval, which permit execution of inline JavaScript or dynamically generated code. The application only check only nonce key in the script to execute the JavaScript. So attacker can manipulate it.
Proof of concept:

Payload:<script nonce="TmV2ZXIgZ29pbmcgdG8gZ2l2ZSB5b3UgdXA=">alert(1)</script>

Impact: It leads to execute inline malicious JavaScript more easily, steal session cookies or authentication tokens, perform actions on behalf of the authenticated user and load malicious scripts from allowed or trusted domain.

Remediation: To prevent, Remove unsafe directive such as unsafe-inline, unsafe-eval. Allow scripts only from trusted domains, restrict other resource.

Proof:



39.Vulnerability Detail– Content Security Policy (CSP) Bypass (Security Level High)

Severity: High
CVSS:7.3

Affected url: http://127.0.0.1/DVWA/vulnerabilities/csp/

Affected parameter: include=payload

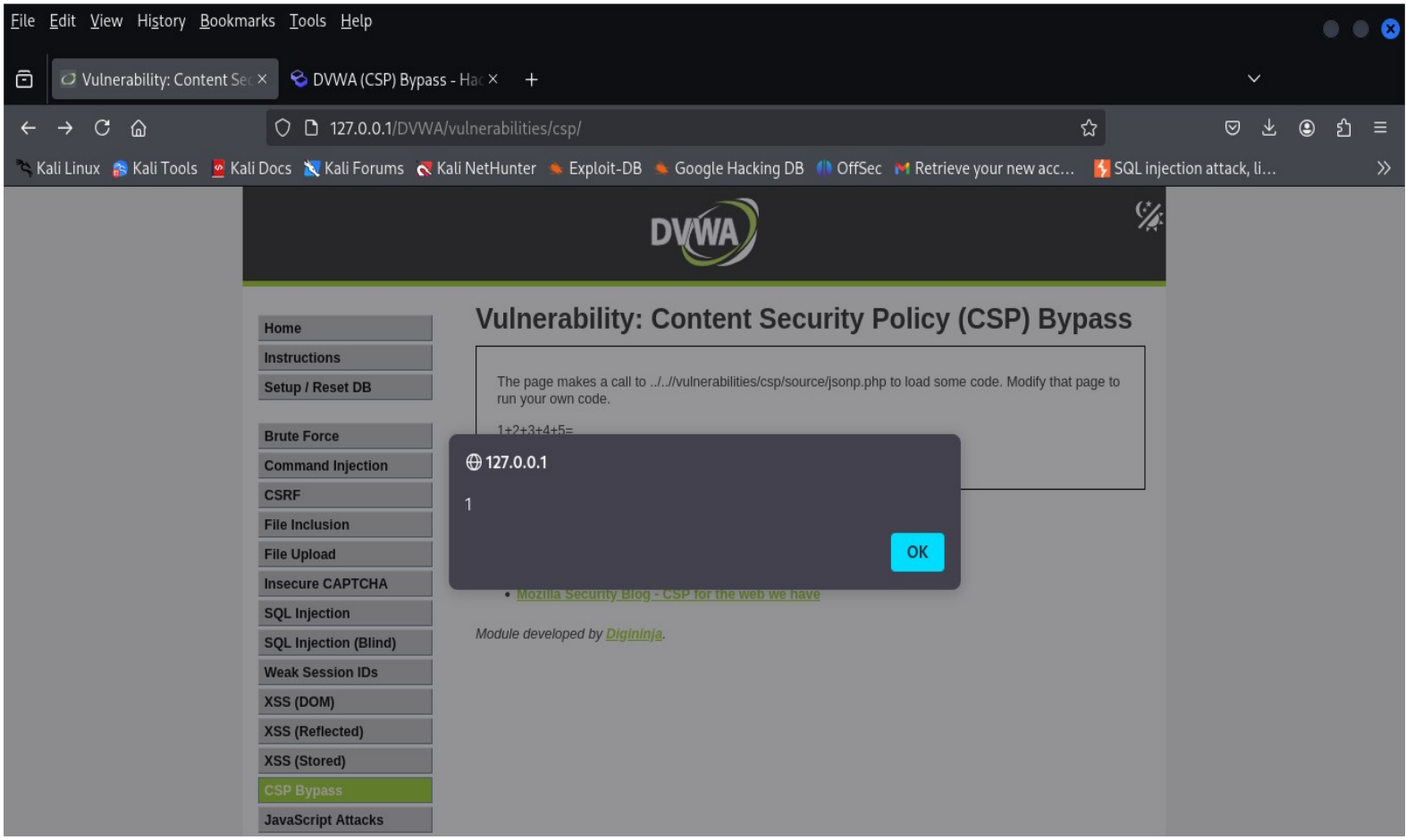
Description: The application implement a csp header intended to restrict the execution of nntrusted scripts and protect against client-side attacks such as xss. However during testing it was observed that csp configuration still allows certain scripts sources that could potentially be abused if a trusted source is compromised, the policy still permit scripts from specific domains or allow certain inline behaviors depending on the configuration. Proof of concept:

Payload: <script>alert(1)</script>

Impact: It leads to bypass restriction under certain conditions, executes malicious scripts if a trusted sources become compromised, steal session cookie or authentication tokens, perform on behalf of legitimate users, deliver malicious content through trusted domains.

Remediation: To mitigate, use strict csp policy that only allows trusted sources, avoid using insecure directive such as 'unsafe-inline' , 'unsafe-eval', restrict other resource such as frames, objects, and external content.

proof:



40. Tools used

- Burp suite
- Web Browser

41. Conclusion

The Vulnerability Assessment and Penetration Testing (VAPT) conducted on the Damn Vulnerable Web Application (DVWA) identified multiple security weakness across different security level, including sql injection, cross-site scripting (XSS), Cross-Site Request Forgery (CSRF), Brute Force, Weak Session ID, Insecure captcha, Authorization Bypass, Open HTTP redirect, Content Security Policy misconfigurations and Command Injection.

These vulnerabilities demonstrate how improper input validation, weak authentication mechanism, insufficient session management, and inadequate security controls can expose web application to various attacks. If such vulnerabilities exist , attacker could potentially exploit them to gain unauthorized, compromise sensitive data, perform malicious actions on behalf of legitimate users, and disrupt application availability.

During the assessment, vulnerabilities were identified at low, medium and high configuration. So, critical, high, medium severity vulnerabilities are fixed immediately first. To Strength the overall security posture of the application. It is recommended to implement the Remediation mention above.