SOEN 321: Information Systems Security

SECURITY ANALYSIS OF SOCIAL MEDIA APPLICATIONS

Christelle Charles 40249246
Zixin Deng 40047744
Imen Khezzar 40246836
Hanseok Kim 40281850
Sonali Patel 40176580
Vanisha Patel 40242554

Concordia University

April 22, 2025

Abstract

This report explores the privacy and security risks present in various social media applications and their websites through the hybrid evaluation model, which incorporates both static and dynamic analysis. The report identifies critical vulnerabilities such as Flash cross-domain policy misconfigurations, insecure CORS implementations, and injection-based flaws including XPath, Python code, OS command, and SQL injection. The testing tools involved with this study include Burp Suite (with a custom Python plugin), Mobile Security Framework (MobSF), Wireshark for packet inspection, and behavioural analysis of background data transmission. The research demonstrates that sometimes, even well-known platforms may be subjected to legacy misconfigurations or overlooked flaws that expose users to risks such as session hijacking, data leakage, unauthorized access, and server-side exploitation. Through combining automated and manual penetration testing methods based on OWASP WSTG and PTES standards, this report aims to deliver insights into improving application and web resilience.

1 Introduction

With the ever-increasing use of social media platforms in daily communication, content sharing, and community engagement, ensuring the security and privacy of these platforms has become a critical concern. Users put their trust in social media websites with a vast amount of personal data, including identities, conversations, preferences, and connections. A breach or exploitation in these platforms can lead to significant consequences such as identity theft, privacy violations, platform-level misinformation, and account hijacking. This project analyzes the security risks of 43 social media websites, examining how these platforms handle authentication, session control, cross-domain requests, and input validation. The scope of the analysis stems from mobile apps to websites, allowing the detection of both client and server-side flaws using a hybrid testing method using tools such as Burp Suite Professional(with the addition of a custom-developed Python plugin adapted using jython), MobSF, Wireshark, and in-browser network logging to simulate real-world attacks. The use of both static and dynamic evaluation allowed for a much deeper analysis to dig up hidden vulnerabilities that standard scanning may overlook Lastly, this research aims to not only evaluate technical flaws but also raise awareness about various persistent risks in social media ecosystems and support for better security-by-design practices. By following the OWASP Web Security Testing Guide and the Penetration Testing Execution Standard (PTES), this report contributes to a more secure and privacy-conscious web environment.

2 Findings

2.1 Wireshark Background Activity Analysis

- **Facebook:** While the app was idle, query responses from endpoints such as graph.facebook.com and b-graph.facebook.com were detected.
- **Youtube:** Multiple instances of background activity initiated by the YouTube app and or Google were detected, such as frequent `QUIC` packets containing PING and CRYPTO frames and TLS 1.3 Client Hello message targeting play.googleapis.com.

- **Discord:** A TLS 1.3 connection was initiated to 'cdn.discordapp.com' (IP: 162.159.133.233), indicating that the app was actively communicating with Discord servers even when not in use.
- **Pinterest:** TLS connections to `www.pinterest.com` were observed while the app was idle, the app is likely maintaining a connection or sending telemetry in the background. It could be preloading data, syncing analytics or checking for new data.
- **Instagram:** The device initiated a DNS query for `gateway.instagram.com`, which resolved to 31.13.80.6 via the Meta-owned domain `dgw.c10r.facebook.com`. Shortly afterward, the device initiated a TLSv1.3 handshake with that server, explicitly identifying `gateway.instagram.com` in the SNI.
- **QQ:** Query responses from astrategy.beacon.qq.com endpoint were detected while the app was idle. These are often linked to behavioral tracking.
- **LinkedIn:** The `px.ads.linkedin.com` endpoint was accessed during idle time which can indicate analytics upload, session management or background session tracking.

2.2 Burp Suite Dynamic Analysis

- **Yiyo.io:** QR login and password reset endpoints lack CSRF protection, enabling unauthorized logins and user enumeration.
- **Tagged.com:** Exposes a permissive Flash policy (`crossdomain.xml`) and allows cross-origin requests with credentials, posing a session theft risk.
- **Mengchenghui.com:** Utilizes a vulnerable version of jQuery (v1.8.3), enabling cross-site scripting (XSS) via the `.load()` method.
- **Rumble.com (1):** Vulnerable to XPath injection due to unsanitized input in the URL path.
- **Rumble.com (2):** Sends user email and password in plaintext within both the request and response, exposing sensitive credentials.
- **Mingle2.com:** Accepts session cookies over unencrypted HTTP and lacks the `Secure` flag, making them susceptible to interception.
- **Plurk.com:** Allows universal Flash access through `crossdomain.xml`, introducing cross-origin security vulnerabilities.
- **Mastodon:** Transmits user email and password in plaintext, compromising sensitive user information.

3 Methodology

3.1 Approach

We conducted a hybrid application penetration test that combined manual testing and automated static and dynamic analysis. Our methodology is based on recognized industry standards, including the OWASP Web Security Testing Guide (WSTG) and the Penetration Testing Execution Standard (PTES). The objective was to identify security vulnerabilities related to authentication, authorization, session management, GraphQL APIs, WebSocket communications, and transport security.

3.2 Security Analysis Framework

The security issues considered in this assessment are grounded in previous research on the security of social media platforms [1, 2, 3] and refined to address a broad range of vulnerabilities relevant to modern web and mobile applications. To support the design of the security analysis framework, we systematically reviewed the academic literature on API security [4, 5], mobile application security [6], and recent industry threat reports [7]. The framework itself is aligned with established industry standards, including the OWASP Web Security Testing Guide (WSTG) [8], the Penetration Testing Execution Standard (PTES) [9], the OWASP API Security Top 10 [5], and the OWASP Mobile Security Testing Guide (MSTG) [10]. Our focus was on vulnerabilities that can be exploited by remote attackers, particularly those affecting authentication, authorization, session handling, API and WebSocket communication, input validation, and transport layer security across both mobile and web interfaces.

Overview: For the social media applications under evaluation, we conducted a comprehensive hybrid application penetration test. The process began with automated static analysis, followed by a dynamic assessment phase. We created dummy user accounts for each application and initiated an authenticated scan using Burp Suite. To enhance coverage beyond Burp's default capabilities, we developed custom scripts to test additional areas, including authentication flaws, session management issues, transport layer security weaknesses, GraphQL vulnerabilities, and WebSocket exposure. For every medium- and high-severity finding we received from the scanner, we leveraged Burp Suite's AI-powered analysis to perform deeper investigations by replaying and validating the corresponding requests. We then transitioned to manual dynamic analysis, where we used passive network monitoring to assess encryption strength, detect potential data leakage, and observe domain communication. This included reviewing POST/PUT traffic, unencrypted DNS queries, TLS handshake details, and background request patterns to uncover any silent or insecure transmissions.

3.2.1 Broken Authentication and Access Control

To detect broken authentication and access control vulnerabilities, we analyze how the application handles user identity tokens during requests to protected endpoints. Specifically, we send requests with no token, an invalid token, or a token belonging to a user with lower privileges. By comparing the server's response to those of valid requests—looking at status codes, headers, and content—we determine whether access is improperly granted. If a profile or admin endpoint is accessible without proper verification, this indicates

a failure in enforcing access control policies. This technique is effective in detecting unauthorized access to user data, administrative dashboards, and account modification functions, which are commonly misconfigured in web and mobile applications.

3.2.2 Cross-Site Request Forgery and CORS Misconfiguration

We assess CSRF vulnerabilities by examining whether state-changing operations—such as updating user settings or triggering password resets—are protected against unauthorized cross-origin requests. We specifically look for requests that use form-encoded data, do not require custom headers, and lack anti-CSRF tokens. If these conditions are met, an attacker could trick a logged-in user into executing unintended actions. For CORS misconfiguration, we spoof the `Origin` header with a malicious domain (e.g., `http://evil.com`) and observe if the server responds with `Access-Control-Allow-Origin: *` and `Access-Control-Allow-Credentials: true`. Such configurations may allow third-party sites to access sensitive data using the victim's authenticated session, compromising the browser's same-origin policy.

3.2.3 Injection Attacks (XSS, XPath, SQL)

Injection vulnerabilities are identified by inserting crafted payloads into URL paths or input fields to test the robustness of client-side and server-side data handling. For cross-site scripting (XSS), we use payloads with various HTML and JavaScript encodings to determine if user inputs are reflected or stored without sanitization. For XPath injection, we analyze error messages and JavaScript logic that relies on functions like `XPathEvaluator()`, indicating client-side XML processing. SQL injection is tested by injecting expressions such as `' OR 1=1` or `sleep(5)` and observing timing delays or database error responses. These tests reveal the application's failure to properly validate and sanitize user inputs, potentially enabling data theft or session hijacking.

3.2.4 Session Management and Sensitive Data Exposure

We evaluate session management by observing how authentication tokens and session cookies are handled during client-server communication. Requests are sent over unencrypted HTTP as well as HTTPS to determine if cookies are transmitted securely. We also inspect the `Set-Cookie` headers to confirm whether flags like `Secure` and `HttpOnly` are properly set. Failure to enforce secure transmission or isolate session data exposes the application to session hijacking through man-in-the-middle attacks. Additionally, we examine POST request payloads and API responses to identify any transmission of sensitive data such as plaintext passwords, email addresses, or authentication tokens. These issues often surface during manual inspection of HTTP traffic using tools like Burp Suite or Wireshark.

3.2.5 Background Activity and Telemetry Tracking

To identify hidden background activity, we use Wireshark to monitor network traffic while the app is idle. Filters such as `dns`, `http.request.method == POST`, and `tls.handshake.extensions_server_name` are applied to detect DNS queries, data uploads, and TLS handshakes to analytics or telemetry services. We analyze whether the

app establishes persistent connections, sends telemetry, or uploads logs without user interaction. Repeated pings or large POST requests when the app is not actively used may indicate aggressive background tracking, which raises concerns about privacy and excessive data usage, particularly on mobile networks.

3.2.6 Legacy Flash Cross-Domain Policies

Although modern applications rarely rely on Flash, some legacy platforms may still include outdated Flash support files. We check for the presence of permissive Flash cross-domain policies by requesting `/crossdomain.xml` and inspecting its contents. If the file includes `<allow-access-from domain="*">`, it means any external domain can interact with the app's Flash components, which violates the same-origin policy. Attackers can exploit this to run Flash-based scripts that steal data or impersonate users. While this is a legacy issue, its presence indicates neglected security hygiene and may point to other overlooked vulnerabilities in older systems.

3.3 Tools

We used Burp Suite Professional [11] as the primary tool for dynamic testing, including authenticated scans, request interception, and vulnerability validation. To enhance our testing of authentication, session management, GraphQL, and WebSocket security, we incorporated custom scripts and the AuthMatrix extension. In particular, we developed a custom Burp Suite plugin using Python to automate the detection of several classes of vulnerabilities during live traffic inspection. This plugin inspected both HTTP requests and responses to identify security issues such as credentials submitted over insecure channels, potential IDOR (Insecure Direct Object Reference) patterns, missing HTTP security headers, and insecure cookie configurations. It also flagged dangerous behaviors such as GraphQL introspection being enabled and insecure WebSocket upgrades. Each detection routine was designed around known attack surfaces: for instance, authentication flaws were flagged when credentials were submitted over HTTP; authorization flaws were inferred from exposed user identifiers; and insecure session cookies were logged when they lacked Secure or HttpOnly attributes. This automation allowed for consistent detection and logging of issues across all interactions without manual intervention. Wireshark[12] was used for passive network monitoring, enabling the analysis of DNS queries, TLS handshakes, and potential data leakage. For injection testing, we utilized Burp Suite Intruder to perform targeted payload-based attacks and identify vulnerabilities such as cross-site scripting (XSS) and SQL injection. For static analysis, we are using mobSF [13] This focused toolset enabled a comprehensive assessment across both mobile and web interfaces.

4 Results

4.1 Background Activity Analysis Results

The background activity analysis revealed that several popular mobile applications, including Facebook, YouTube, Discord, Pinterest, Instagram and Q,Q engage in background network activity when they appear idle to the user. For Facebook, DNS queries to endpoints such as `graph.facebook.com`, `b-graph.facebook` and `edge-mqtt.facebook.`

com suggest ongoing GraphQL communications and persistent MQTT connections, likely used for real-time features like Messenger presence, feed updates and telemetry reporting. Similarly, YouTube maintains persistent QUIC connections and initiates TLS handshakes with Google servers (e.g. play.googleapis.com), indicating background data preloading and session management aimed at providing a seamless video streaming experience. Discord's idle time TLS connections to cdn.discordapp.com suggest synchronization of messages and assets, while Pinterest shows periodic TLS communications with its main domain, potentially for data synching and telemetry. Instagram also demonstrates active background communication through DNS and TLS handshakes with gateway.instagram.com, likely for preloading content and delivering notifications. The QQ app exhibits a similar pattern, accessing domains such as strategy.beacon.qq.com, indicating behavioural analytics and telemetry syncing. Notably, px.ads.linkedin.com was also accessed during idle time, pointing to background analytics or ad-related data exchange. While much of this behaviour is standard for modern apps aiming to enhance responsiveness and user experience, it may raise concerns for users focused on privacy.

4.2 A01: Broken Access Control

We identified critical broken access control vulnerabilities in multiple applications, which could allow attackers to bypass authorization mechanisms, escalate privileges, and access protected resources. These issues were validated through authenticated scans and manual testing using Burp Suite's AI-powered analysis.

Trust Exploitation via Flash Policy Misconfiguration. On Tagged.com, we discovered a permissive crossdomain.xml policy that allowed any subdomain of tagstat.com to perform cross-origin requests with arbitrary headers. This overly broad trust model poses a significant risk, as a compromised or malicious subdomain could exploit this trust to act on behalf of authenticated users. Similar policy files were found across multiple paths (e.g., /cdn/, /flash/), increasing the platform's attack surface.

Session Enforcement Failures. On Yiyo.io, the QR code login flow was found to be vulnerable to CSRF due to the /auth/qrcode_check endpoint accepting requests without validating a CSRF token. By submitting only a valid verification code (e.g., 873037) without the expected token, the server still returned a successful response, enabling unauthorized login attempts. Additionally, the password reset functionality lacked CSRF protection and leaked user account existence through varying response messages, allowing for user enumeration.

4.3 A02: Cryptographic Failures

The dynamic analysis revealed major issues with exposing sensitive data in the requests and responses. On rumble.com, the sign up process exposes confidential data such as email and password in plaintext in the request. In addition, the credentials are echoed back after a successful sign up in the response. Similarly, mastodon.social exposes email and password in the login process. This is a security risk because an attacker can monitor network traffic and intercept the credentials. The attacker can have access to the account. Many users reuse the same credentials for other services, therefore their other

accounts can also be compromised. It is important to encrypt sensitive encrypted data and to never echo back the data in the response.

4.4 A03: Injection

Vulnerable JQuery Library: In the security assessment of `mengchenghui.com`, we discovered that the webpage includes a vulnerable and outdated version of jQuery v.1.8.3. This allows cross-site scripting attacks using the `.load()` method, which fails to recognize and remove the `<script>` HTML tags that contain a whitespace character. During testing, many XSS payloads with whitespace variations were sent to target the `jQuery.load()` vulnerability. The response returned `200`, meaning that jQuery failed to handle whitespace characters and the request did not get blocked. If the vulnerability is exploited, the attacker can inject JavaScript into the page leading to unauthorized actions performed on the webpage.

4.5 A05: Security Misconfiguration

Insecure Flash Cross-Domain Policy: On `Plurk.com`, we discovered a permissive `crossdomain.xml` file that allowed any domain to perform cross-origin requests with arbitrary headers. This overly broad trust model poses a significant risk, as a malicious website could exploit this trust to act on behalf of authenticated users. The file was also publicly cached, making it easily discoverable and exploitable.

CORS misconfiguration: This was discovered on www.tagged.com where the server accepts requests from arbitrary origins, including `http://evil.com`, and responds with `Access-Control-Allow-Credentials: true`. This behavior was confirmed on both the main page and the authentication endpoint (`/secure_login.html`), which also set session cookies in the response. This insecure configuration allows attackers to potentially steal sensitive user data or session credentials via cross-origin requests from a malicious website.

Insecure Session Cookie Transmission. On `Mingle2.com`, session cookies were transmitted over unencrypted HTTP connections and lacked the `Secure` flag. This misconfiguration exposes session cookies to interception via man-in-the-middle attacks, allowing attackers to hijack user sessions and access sensitive information.

5 Discussion

5.1 Practical Implications of Background Activity Findings

The background network activity observed across apps like Facebook, YouTube, Discord, Instagram, and QQ highlights a trade-off between user convenience and resource consumption. These apps maintain persistent connections to support real-time features such as instant messaging, content preloading and push notifications. While this enhances responsiveness and user experience, it can lead to increased mobile data usage, battery drain, and reduced device performance.

From a privacy standpoint, many of the background communications involve telemetry, analytics, and behavioural tracking. Endpoints like `strategy.beacon.qq.com`

and `px.ads.linkedin.com` suggest passive data collection, raising concerns about transparency and informed consent.

These findings emphasize the need for greater transparency, user control, and informed consent around background data operations. Platform providers should disclose telemetry practices clearly and give users fine-grained control over which background services are enabled.

5.2 Limitations

1. Scope of Evaluation: We assessed only publicly available or free-tier social media applications. Platforms requiring organizational identity verification or paid access were not included, which may limit the generalization of our results.

2. Ethical Boundaries: We did not attempt automated mass enumeration or brute-force attacks to avoid violating terms of service or ethical guidelines.

3. Tool reliance and Manual Work: While Burp Suite's AI modules aided in injection and CSRF detection, certain logic flaws, such as broken object-level authorization, still required manual investigation.

4. Observation Window: Our passive monitoring setup captured background activity over a limited observation window; extended or long-term data collection might reveal additional patterns of concern.

5. Content-Related Risk Area: Due to the scope of this study, we did not analyze content moderation systems, recommendation algorithms, or abuse reporting mechanisms, which may also carry security or privacy implications.

5.3 Lessons Learned

From our study, several recurring and noteworthy observations emerged:

1. Background Data Collection Is Ubiquitous: Many applications maintain persistent connections for telemetry, tracking, and content preloading, even when idle. Users have minimal visibility or control over this behaviour, which can impact privacy and device resources.

2. Broken Access Control Is Still Prevalent: Vulnerabilities such as when CSRF protections are missing from sensitive endpoints, or when authorization tokens are not properly validated. Vulnerabilities like those found in Yiyo.io and Tagged.com highlight how common these flaws still are.

3. Legacy and Deprecated Configurations Persist: Permissive Flash crossdomain policies were still present in platforms like Plurk, demonstrating that deprecated features can remain exploitable years after their official obsolescence.

4. Input Validation and Injection Risks Are Ongoing: Weak input validation, including XPath injection and the use of outdated client-side libraries (e.g., jQuery 1.8.3), remain prevalent. These could allow attackers to manipulate DOM behavior or backend logic if not remediated.

5. Third-Party Domains Are Widely Used Without Transparency: Apps routinely communicate with third-party domains (e.g., ad networks and telemetry services) in the background. These background communications are not always clearly disclosed and may raise regulatory compliance issues under frameworks like GDPR and CCPA.

6. Sensitive Data Often Transmitted Insecurely: Sensitive data may not be handled securely, and an account can be compromised through network traffic analysis. It is important to have different passwords for each service.

5.4 Recommendation

1. Enhance User Control Over Background Activity: To address concerns around background activity and user control, developers and platform providers should give the users control over background activity. This can be achieved by offering various options to limit background activity, and platform-level tools should allow the users to manage how and when apps connect in the background. Additionally, privacy and resource management should be enhanced. This can be achieved by auditing app permissions and implementing restrictions on background data usage, particularly for users who are privacy-conscious.

2. Encrypt and Minimize Sensitive Data Exposure: For insecure data handling, developers need to ensure that credentials and sensitive data are never included in plaintext in an API request and response. The data needs to be securely transmitted.

3. Update Legacy Components and Libraries: To address the injection vulnerability caused by using an outdated jQuery library, developers and upgrade jQuery to the latest stable version. In addition, content security policy (CSP) can be used to help prevent XSS attacks by controlling which code is allowed to execute in the page. The CSP header stops the browser from running harmful scripts when they are injected [14].

References

[1] E. S. Alashwali and K. B. Rasmussen, "Security of social media applications," in *IEEE Symposium on Security and Privacy Workshops (SPW)*, 2018.

[2] OWASP Foundation, "Owasp top 10: Web application security risks," https://owasp.org/www-project-top-ten/, 2021.

[3] Verizon, "2023 data breach investigations report," https://www.verizon.com/business/resources/reports/dbir/, 2023.

[4] OWASP Foundation, "Owasp api security top 10," https://owasp.org/www-project-api-security/, 2023.

[5] Salt Security, "Api security: A guide to api threats and solutions," https://salt.security/, 2022, whitepaper.

[6] ENISA, "Enisa threat landscape for mobile," https://www.enisa.europa.eu/publications/enisa-threat-landscape-for-mobile-devices, 2021.

[7] PortSwigger Research, "Exploiting graphql apis: An overview of techniques and attacks," https://portswigger.net/research/exploiting-graphql-apis, 2022.

[8] OWASP Foundation, "Owasp web security testing guide (wstg)," https://owasp.org/www-project-web-security-testing-guide/, 2021.

[9] Penetration Testing Execution Standard (PTES), "Technical guidelines," http://www.pentest-standard.org/index.php/PTES_Technical_Guidelines, 2023.

[10] OWASP Foundation, "Mobile security testing guide (mstg)," https://owasp.org/www-project-mobile-security-testing-guide/, 2022.

[11] PortSwigger Ltd., "Burp suite professional," https://portswigger.net/burp/pro, 2025.

[12] The Wireshark Foundation, "Wireshark: The world's foremost network protocol analyzer," https://www.wireshark.org/, 2025.

[13] MobSF Developers, "Mobile security framework (mobsf)," https://github.com/MobSF/Mobile-Security-Framework-MobSF, 2025.

[14] MDN Web Docs, "Content security policy (csp)," n.d., accessed: 2025-04-22. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/CSP

A List of Applications analyzed

Table 1: List of Applications Analyzed

Application	Application	Application	Application
Badoo	Bluesky	Bumble	Clubhouse
Discord	Facebook	imo	Instagram
Likee	Line	LinkedIn	Mastodon
Messenger	MeWe	mengchenghui	mingle2.com
Parler	Pinterest	Plurk	QQ
Quora	Reddit	Rumble	Skype
Snapchat	Steemit	Tango	Tapatalk
tagged	Teams	Telegram	Threads
TikTok	Tinder	Truth Social	Tumblr
Twitch	WhatsApp	X	Yiyo
YouTube	Yubo	Zello	

B Dynamic analysis (Wireshark)

Figure 1: Facebook query responses received while idle

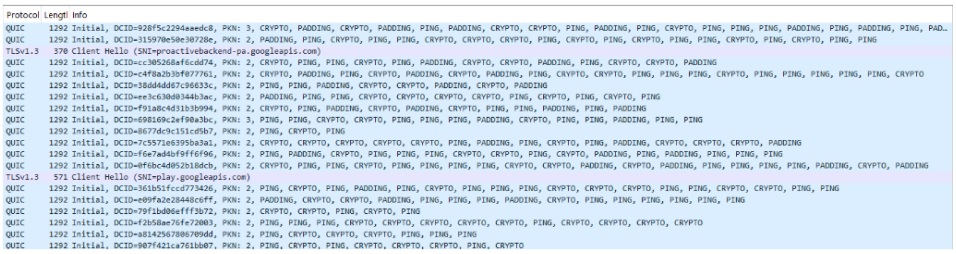


Figure 2: Frequent QUIC packets containing PING and CRYPTO frames received while YouTube was idle

Figure 3: Query responses received from QQ while the app was idle

Figure 4: The px.ads.linkedin.com domain being accessed while LinkedIn app was idle

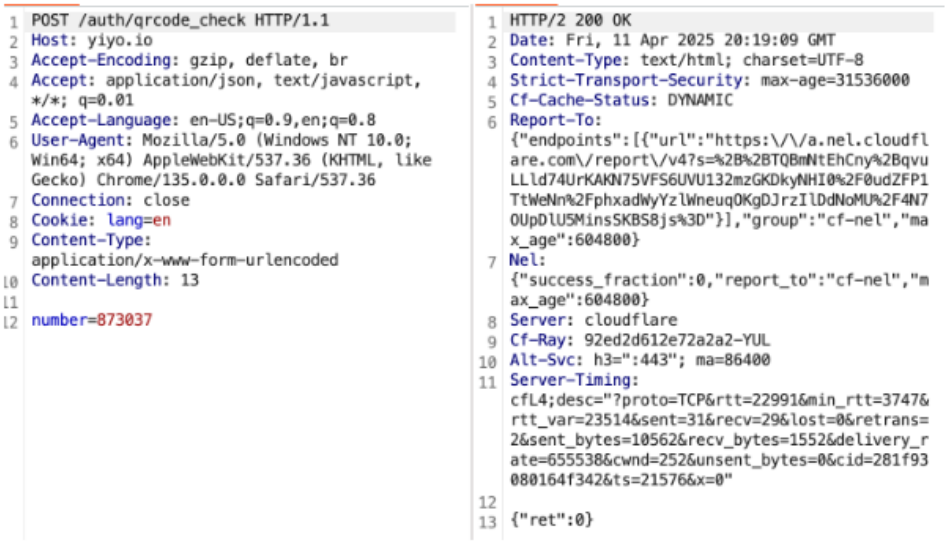
C Dynamic analysis (Burpsuite)

Figure 5: Request to /auth/login successfully returned the login page, confirming the authentication interface is accessible

```
GET /auth/qrcode_check HTTP/1.1
Host: yiyo.io
Accept-Encoding: gzip, deflate, br
Accept:
text/html,application/xhtml+xml,application/x
ml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US;q=0.9,en;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 10.0;
Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/135.0.0.0 Safari/537.36
Connection: close
Cookie: lang=en
```

```
HTTP/2 200 OK
Date: Fri, 11 Apr 2025 20:18:59 GMT
Content-Type: text/html; charset=UTF-8
Vary: Accept-Encoding
Strict-Transport-Security: max-age=31536000
Cf-Cache-Status: DYNAMIC
Report-To:
{"endpoints":[{"url":"https:\/\/a.nel.cloudfl
are.com\/report\/v4?s=jfZXnYhFdCs6Z4nNYCuMVVo
EQa4K%2F%2B2tPXvYn6dDDW1pMQ6KGbi0V7LFDQdNG0or
9ynhhwZItDvvtqBPNctqITIGpOmMeqivKJmwt3%2B97VE
jBHzuFU8fHsM%3D"}],"group":"cf-nel","max_age"
:604800}
Nel:
{"success_fraction":0,"report_to":"cf-nel","m
ax_age":604800}
Server: cloudflare
Cf-Ray: 92ed2d1f3ce4a2a2-YUL
Alt-Svc: h3=":443"; ma=86400
Server-Timing:
cfL4;desc="?proto=TCP&rtt=29056&min_rtt=3747&
rtt_var=32573&sent=23&recv=23&lost=0&retrans=
2&sent_bytes=9013&recv_bytes=1324&delivery_ra
te=655538&cwnd=252&unsent_bytes=0&cid=281f930
80164f342&ts=11097&x=0"

<!DOCTYPE html>
```

Figure 6: Request to /auth/qrcode_check without parameters also returned HTTP 200, indicating the endpoint is reachable but inactive without input



```
POST /auth/qrcode_check HTTP/1.1
Host: yiyo.io
Accept-Encoding: gzip, deflate, br
Accept: application/json, text/javascript,
*/*; q=0.01
Accept-Language: en-US;q=0.9,en;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 10.0;
Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/135.0.0.0 Safari/537.36
Connection: close
Cookie: lang=en
Content-Type:
application/x-www-form-urlencoded
Content-Length: 13

number=873037
```

```
HTTP/2 200 OK
Date: Fri, 11 Apr 2025 20:19:09 GMT
Content-Type: text/html; charset=UTF-8
Strict-Transport-Security: max-age=31536000
Cf-Cache-Status: DYNAMIC
Report-To:
{"endpoints":[{"url":"https:\/\/a.nel.cloudfl
are.com\/report\/v4?s=%2B%2BTQBmNtEhCny%2Bqvu
LLld74UrKAKN75VFS6UVU132mzGKDkyNHI0%2F0udZFP1
TtWeNn%2FphxadWyYzlWneuqOKgDJrzIlDdNoMU%2F4N7
OUpDlU5MinsSKBS8js%3D"}],"group":"cf-nel","ma
x_age":604800}
Nel:
{"success_fraction":0,"report_to":"cf-nel","m
ax_age":604800}
Server: cloudflare
Cf-Ray: 92ed2d612e72a2a2-YUL
Alt-Svc: h3=":443"; ma=86400
Server-Timing:
cfL4;desc="?proto=TCP&rtt=22991&min_rtt=3747&
rtt_var=23514&sent=31&recv=29&lost=0&retrans=
2&sent_bytes=10562&recv_bytes=1552&delivery_r
ate=655538&cwnd=252&unsent_bytes=0&cid=281f93
080164f342&ts=21576&x=0"

{"ret":0}
```

Figure 7: Request to /auth/qrcode_check with only a valid code (873037) and no token returned a success response, confirming that the endpoint does not validate CSRF tokens

```
POST /password/reset HTTP/1.1
Host: yiyo.io
Accept: application/json, text/javascript, */*; q=0.01
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/135.0.0.0 Safari/537.36
Cookie: lang=zh
Origin: https://yiyo.io
X-Requested-With: XMLHttpRequest
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Content-Length: 15

email=§payload§
```

Figure 8: Request sent with various email address that we brute forced

Request	Payload	Status code	Length	Content type
0	admin@yiyo.io	200		text/html; charset...
1	administrator@yiyo.io	200		text/html; charset...
2	support@yiyo.io	200		text/html; charset...
3	info@yiyo.io	200		text/html; charset...
4	contact@yiyo.io	200		text/html; charset...
5	help@yiyo.io	200		text/html; charset...
6	service@yiyo.io	200		text/html; charset...
7	test@yiyo.io	200		text/html; charset...
8	user@yiyo.io	200		text/html; charset...
9	webmaster@yiyo.io	200		text/html; charset...

Figure 9: Successfully find valid emails in database

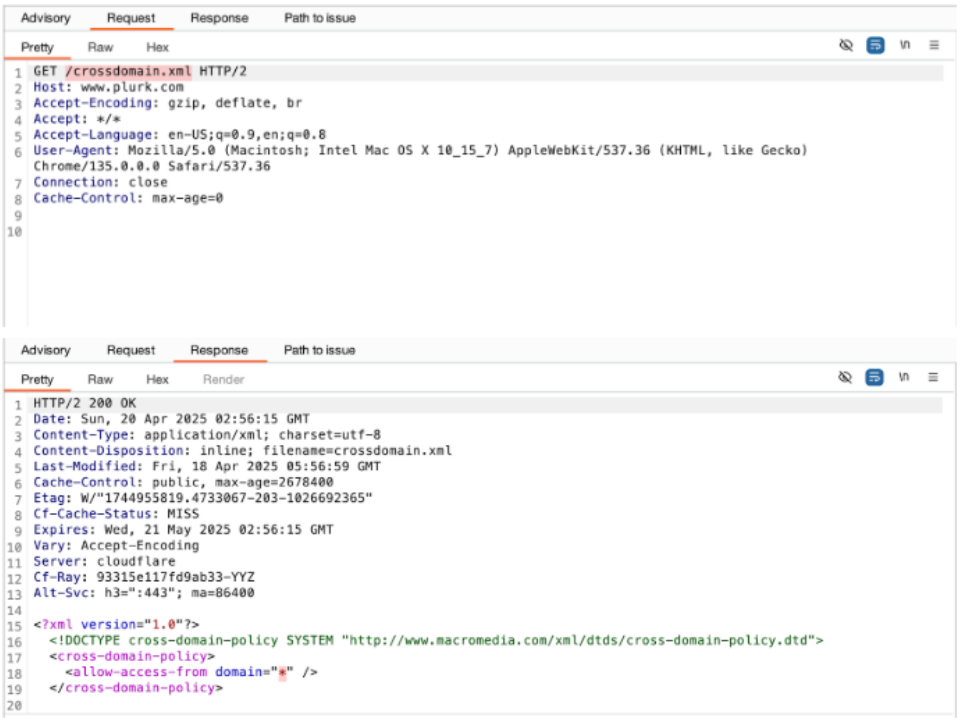


Figure 10: The GET /crossdomain.xml request asks for the Flash security policy file. HTTP/2 200 OK response confirms that the file exists and allows all domains

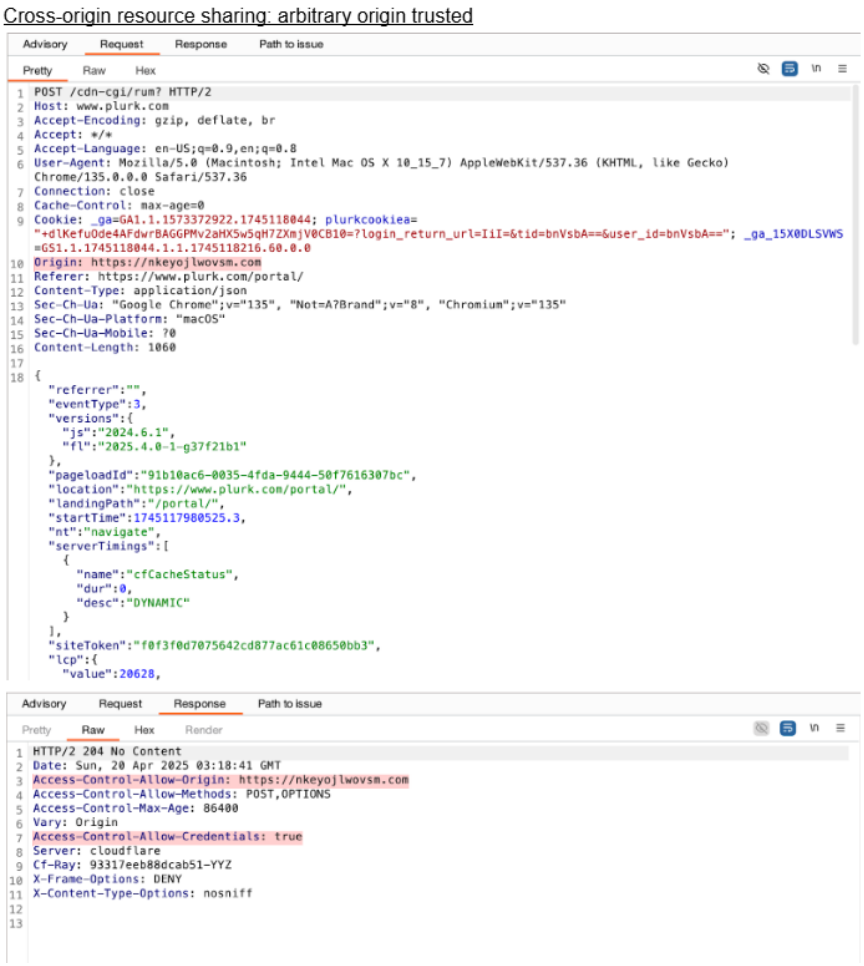


Figure 11: The server trusts the untrusted domain (nkeyojlwovsm.com) and allows it to access sensitive responses

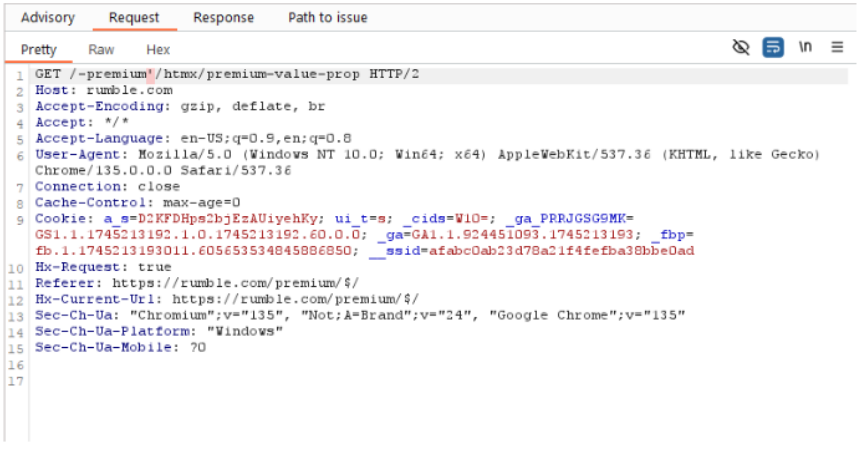


Figure 12: The user input was submitted directly in the URL path

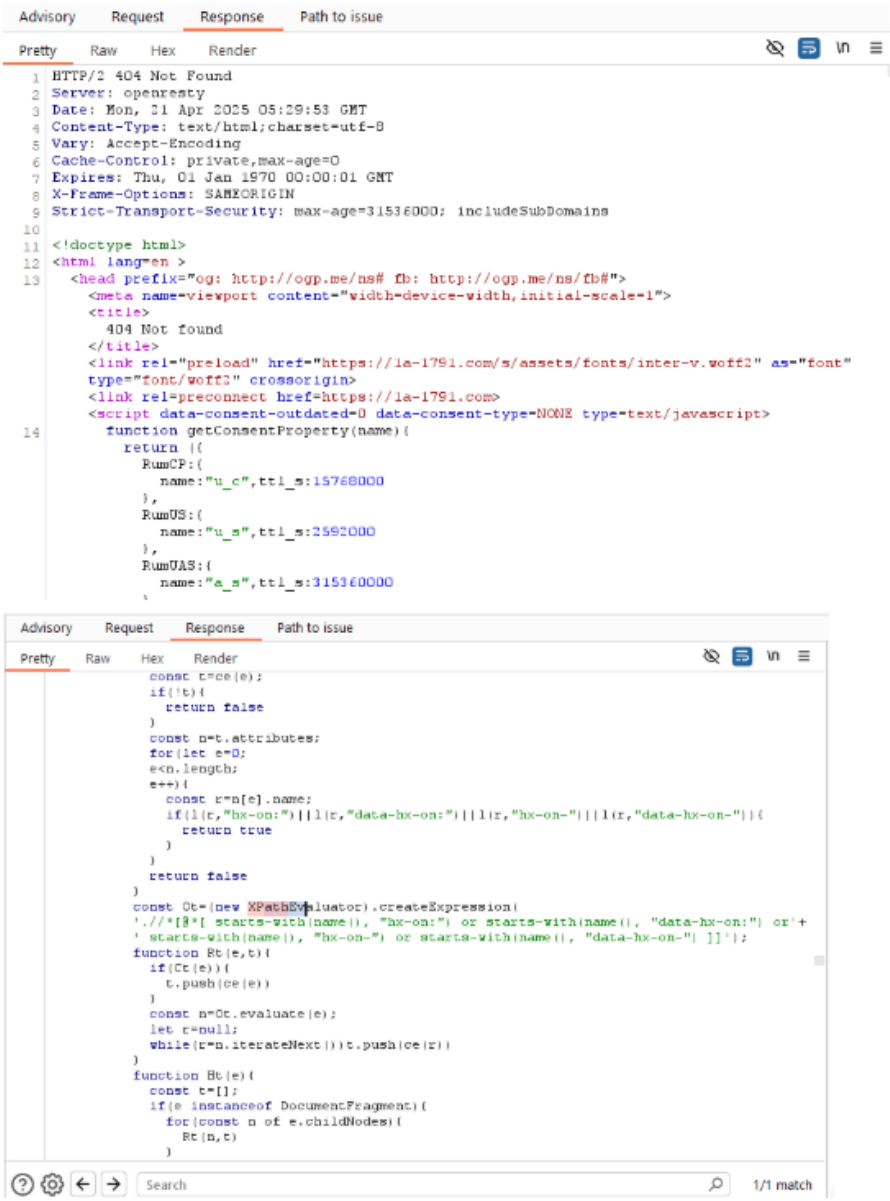


Figure 13: The user input is submitted directly in the URL path and the response returned 404 with HTML showing an XPath evaluator in the script

Request	Payload	Status code	Length	Content type
0	crossdomain.xml	200	349	text/xml
1	core/crossdomain.xml	200	43629	text/html; charset...
2	app/crossdomain.xml	200	43629	text/html; charset...
3	flex2/crossdomain.xml	200	43629	text/html; charset...
4	flexapp/crossdomain.xml	200	43629	text/html; charset...
5	static/crossdomain.xml	200	43629	text/html; charset...
6	cdn/crossdomain.xml	200	43629	text/html; charset...
7	flash/crossdomain.xml	200	74763	text/html; charset...
8	flex/crossdomain.xml	200	74763	text/html; charset...
9	assets/crossdomain.xml	404	18549	text/html; charset...

Figure 14: Response returned HTTP 200 with large file sizes, indicating the presence of permissive cross-domain policies

```
GET / HTTP/1.1
Host: www.tagged.com
Accept-Encoding: gzip, deflate, br
Accept:
text/html,application/xhtml+xml,application/x
ml;q=0.9,image/avif,image/webp,image/apng,*/*
;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Language: en-US;q=0.9,en;q=0.8
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac
 OS X 10_15_7) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/135.0.0.0 Safari/537.36
Connection: close
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Sec-CH-UA: "Google Chrome";v="135",
"Not=A?Brand";v="8", "Chromium";v="135"
Sec-CH-UA-Platform: "macOS"
Sec-CH-UA-Mobile: ?0
Origin: http://evil.com
```

```
HTTP/1.1 200 OK
Date: Wed, 16 Apr 2025 13:28:28 GMT
Server: Apache
Set-Cookie: S=pu5tv5ukcs5lkegl8u467cu1sd;
path=/; domain=.tagged.com; secure; HttpOnly
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache,
must-revalidate
Pragma: no-cache
Set-Cookie: B=b=B9F4E129F93B7025;
expires=Sat, 14-Apr-2035 13:28:28 GMT;
Max-Age=315360000; path=/;
domain=.tagged.com; secure; HttpOnly
Vary: Accept-Encoding
Access-Control-Allow-Credentials: true
Access-Control-Allow-Headers: *
Content-Length: 42272
Content-Type: text/html; charset=UTF-8
Connection: close

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:fb="http://www.facebook.com/2008/fbml"
>
  <head id="html_head">
```

Figure 15: Request with Origin http://evil.com Response with HTTP 200 OK with cookies and CORS headers (Access-Control-Allow-Credentials: true), confirming CORS misconfiguration.

```
GET /secure_login.html HTTP/1.1
Host: www.tagged.com
Accept-Encoding: gzip, deflate, br
Accept:
text/html,application/xhtml+xml,application/x
ml;q=0.9,image/avif,image/webp,image/apng,*/*
;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Language: en-US;q=0.9,en;q=0.8
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac
 OS X 10_15_7) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/135.0.0.0 Safari/537.36
Connection: close
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Sec-CH-UA: "Google Chrome";v="135",
"Not=A?Brand";v="8", "Chromium";v="135"
Sec-CH-UA-Platform: "macOS"
Sec-CH-UA-Mobile: ?0
Origin: http://evil.com
```

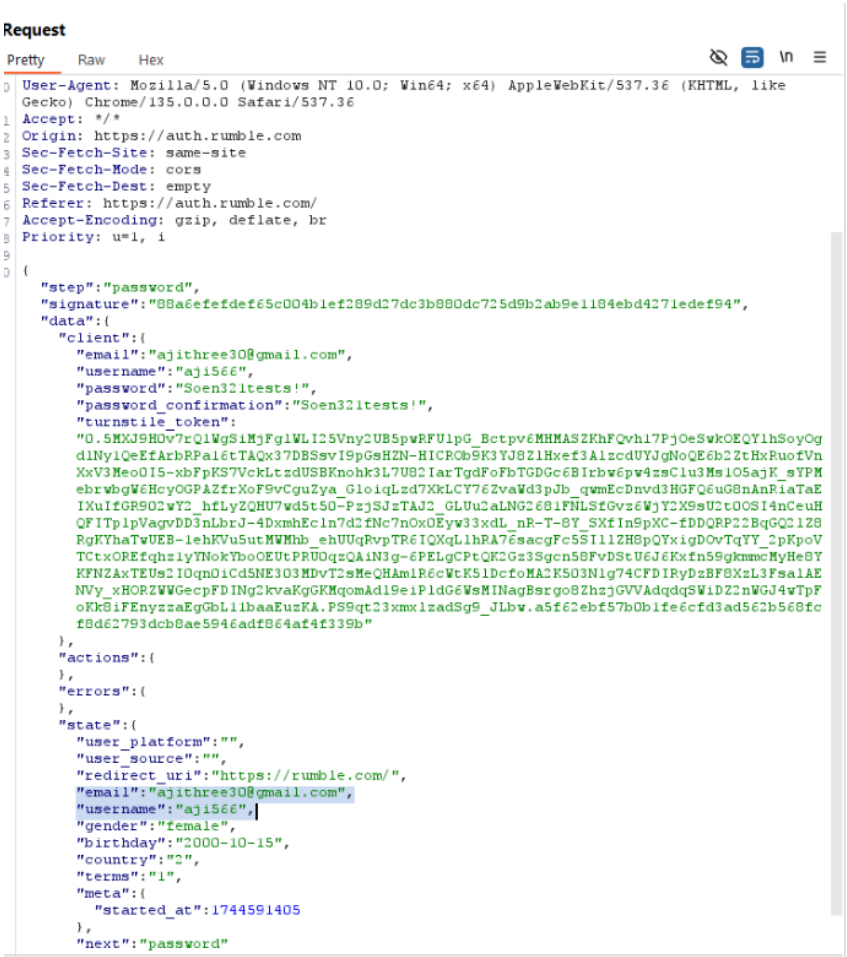
```
HTTP/1.1 200 OK
Date: Wed, 16 Apr 2025 13:28:43 GMT
Server: Apache
Set-Cookie: S=3eq0gsv39mvsslj4raq1p5fg96;
path=/; domain=.tagged.com; secure; HttpOnly
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache,
must-revalidate
Pragma: no-cache
Set-Cookie: B=b=FE83A1E6AFE9AD39;
expires=Sat, 14-Apr-2035 13:28:43 GMT;
Max-Age=315360000; path=/;
domain=.tagged.com; secure; HttpOnly
Vary: Accept-Encoding
Access-Control-Allow-Credentials: true
Access-Control-Allow-Headers: *
Content-Length: 2081
Content-Type: text/html; charset=UTF-8
Connection: close

<!DOCTYPE HTML>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
```

Figure 16: Request with Origin **http://evil.com** received a 200 OK response with auth cookies and insecure CORS headers, confirming a CORS vulnerability on the login endpoint.

```
GET /online-dating/neuchatel/chat HTTP/1.1
Host: mingle2.com
Accept-Encoding: gzip, deflate, br
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Language: en-US;q=0.9,en;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36
Connection: close
Cookie: _session_id=
68a55f90ede06cec02b1557408951da9
```

```
HTTP/2 200 OK
Date: Sun, 20 Apr 2025 21:16:22 GMT
Content-Type: text/html; charset=utf-8
Vary: Accept-Encoding
X-Frame-Options: SAMEORIGIN
X-Frame-Options: DENY
X-Xss-Protection: 1; mode=block
X-Xss-Protection: 1; mode=block
X-Content-Type-Options: nosniff
X-Content-Type-Options: nosniff
Cache-Control: max-age=0, private,
must-revalidate
Set-Cookie: tracker=
id%3D%3E%7Cuser_id%3D%3E%7Ccp%3D%3E%7Cs1%3D%
3E%7Cs2%3D%3E%7Ccr%3D%3E%7Clp%3D%3Ehttp%3A%2
F%2Fmingle2.com%2Fonline-dating%2Fneuchatel%
2Fchat%7Creferring_url%3D%3E%7Cinitial_click
_at%3D%3E2025-04-20+14%3A16%3A22+-0700%7Csub
scribed_at%3D%3E%7Cinternal_source%3D%3E%7Ck
w%3D%3E%7Cmt%3D%3E%7Cactual_kw%3D%3E%7Csite%
3D%3E%7Csearch_engine%3D%3E%7Csource_domain%
3D%3E%7Clp_category%3D%3Egeo%7Clp_subcategor
y%3D%3Ecities%7Cregistration_site_id%3D%3E1%
7Cdevice%3D%3E; path=/; expires=Mon, 20 Apr
2026 21:16:22 -0000
Set-Cookie: cleared_gta_version_1=true;
path=/
```

Figure 17: session cookies were accepted over unencrypted HTTP.



Request

Pretty Raw Hex

```
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/135.0.0.0 Safari/537.36
Accept: */*
Origin: https://auth.rumble.com
Sec-Fetch-Site: same-site
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://auth.rumble.com/
Accept-Encoding: gzip, deflate, br
Priority: u=1, i

{
  "step":"password",
  "signature":"88a6efefdef65c004b1ef289d27dc3b880dc725d9b2ab9e1184ebd4271edef94",
  "data":{
    "client":{
      "email":"ajithree30@gmail.com",
      "username":"aji566",
      "password":"Soen321tests!",
      "password_confirmation":"Soen321tests!",
      "turnstile_token":
      "0.5MXJ9HOv7rQlWgSiMjFglWLI25Vny2UB5pwRFUlpG_Bctpv6MHMASZKhFQvh17PjOeSwkOEQYlhSoyOg
      dlNy1QeEfArbRPal6tTAQx37DBSsvI9pGsHZN-HICRObSK3YJ8Zlhxef3AlzcdUYJgNoQE6b2ZtHxRuofVn
      XxV3MeoOI5-xbFpKS7VckLtzdUSBKnohk3L7U82IarTgdFoFbTGDGc6BIrbw6pw4zsClu3Ms1O5ajK_sYPM
      ebrwbgW6HcyOGPAZfrXoF9vCguZya_GloiqLzd7XkLCY76ZvaWd3pJb_qwmEcDnvd3HGFQ6uG8nAnRiaTaE
      IXuIfGR902wY2_hfLyZQHU7wd5t50-PzjSJzTAJ2_GLUu2aLNG26BiFNLSfGvz6WjY2X9sU2tOOSI4nCeuH
      QFITp1pVagvDD3nLbrJ-4DxmhEcln7d2fNc7nOxOEyw33xdL_nR-T-8Y_SXfIn9pXC-fDDQRP22BqGQ21Z8
      RgKYhaTwUEB-1ehKVu5utMWMhb_ehUUqRvpTR6IQXqL1hRA76sacgFc5SI11ZH8pQYxigDOvTqYY_2pKpoV
      TCtxOREfqhz1yYNokYboOEUtPRUOqzQAiN3g-6PELgCPtQK2Gz3Sgcn58FvDStU6J6Kxfn59gkmmcMyHe8Y
      KFNZAxTEUs2IOqnOiCd5NE3O3MDvT2sMeQHAmlR6cWtK5lDcfoMA2K5O3N1g74CFDIRyDzBF8XzL3FsalAE
      NVy_xHORZWWGecpFDING2kvaKgGKMqomAd19eiPldG6WsMINagBsrgo8ZhzjGVVAdqdqSWiDZ2nWGJ4wTpF
      oKk8iFEnyzzaEgGbL1lbaaEuzKA.PS9qt23xmxlzadSg9_JLbw.a5f62ebf57bOb1fe6cfd3ad562b568fc
      f8d62793dcb8ae5946adf864af4f339b"
    },
    "actions":{
    },
    "errors":{
    },
    "state":{
      "user_platform":"",
      "user_source":"",
      "redirect_uri":"https://rumble.com/",
      "email":"ajithree30@gmail.com",
      "username":"aji566",
      "gender":"female",
      "birthday":"2000-10-15",
      "country":"2",
      "terms":"1",
      "meta":{
        "started_at":1744591405
      },
      "next":"password"
```

Figure 18: Sending Sign Up Request on rumble.com

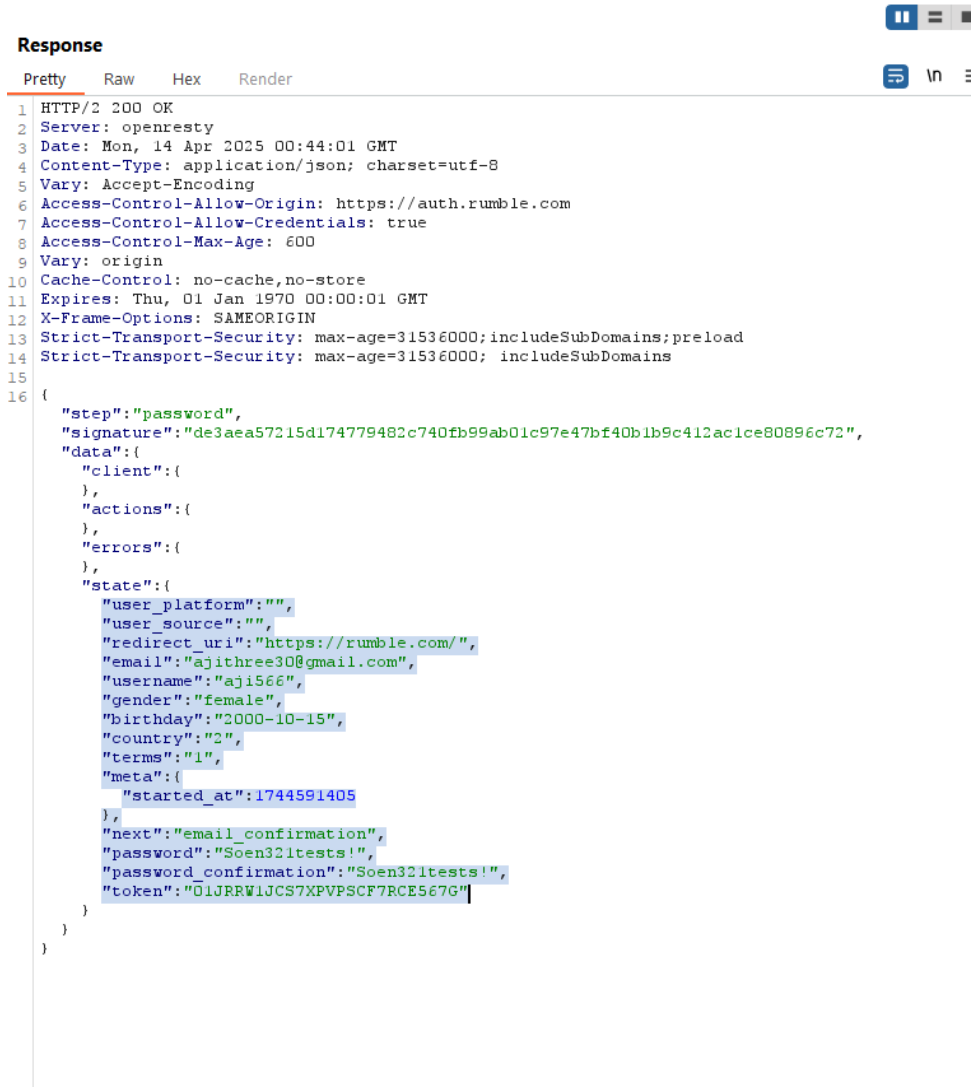


Figure 19: Response with email and password echoed on rumble.com

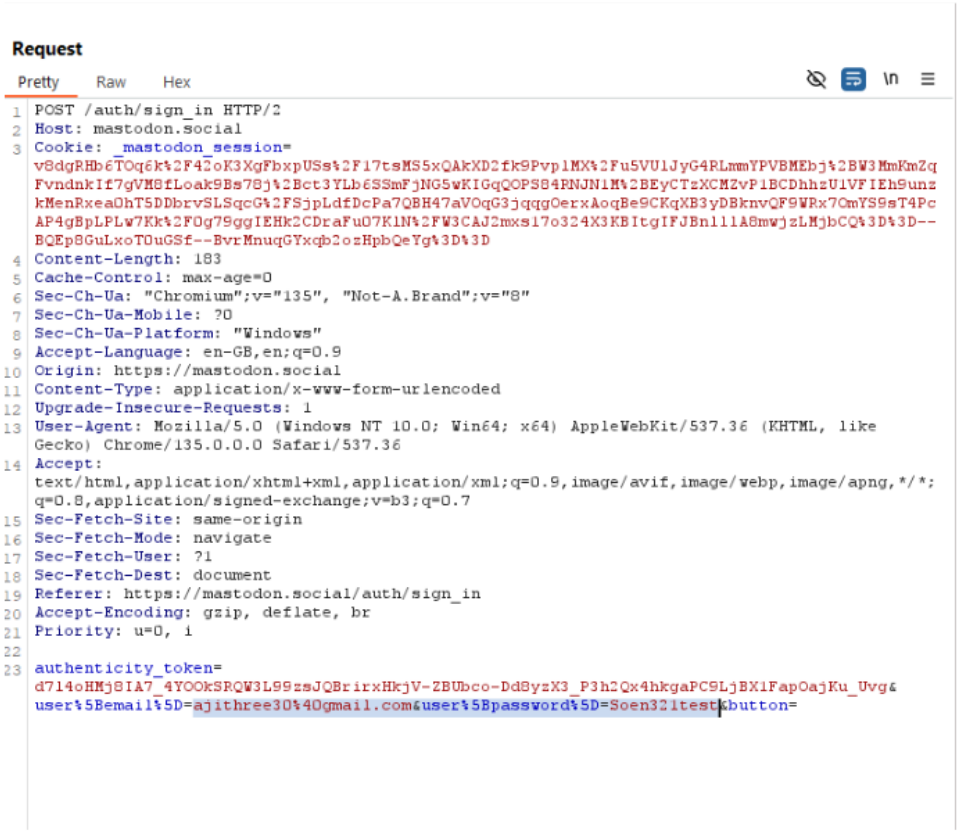


Figure 20: Login Request in mastodon.social

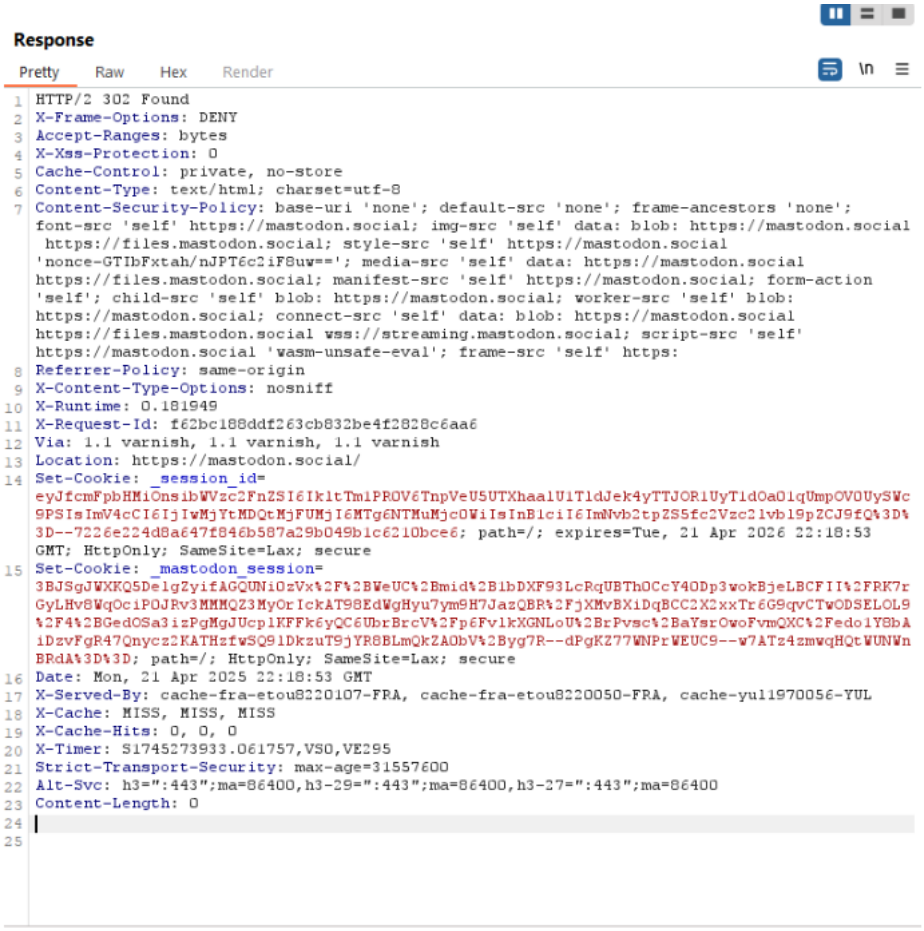


Figure 21: Login Response in mastodon.social

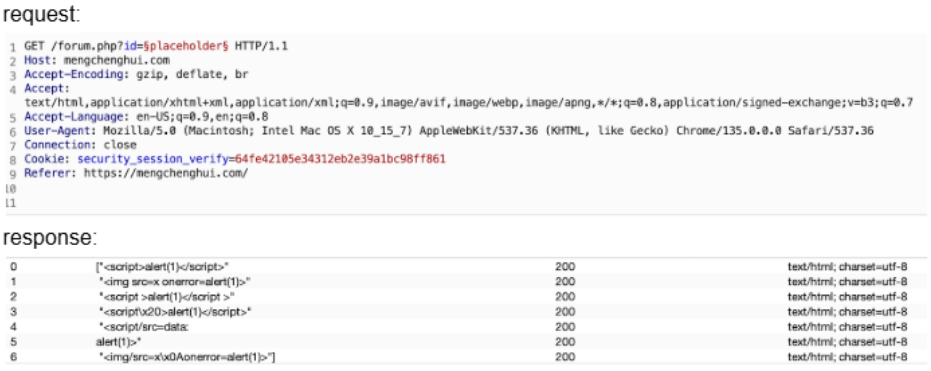


Figure 22: Testing XSS payloads exploiting jQuery.load() in version 1.8.3 using script tags with whitespace, revealing sanitization failure (200 OK response).

D Python plugin

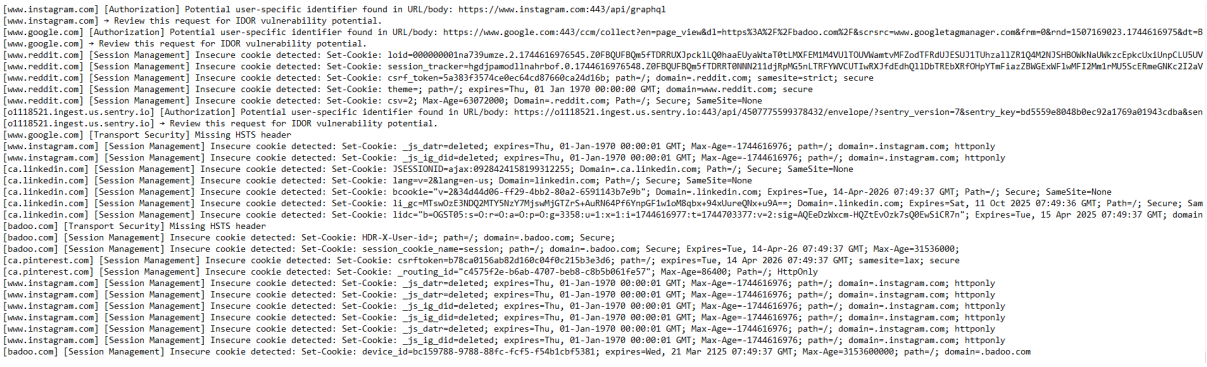


Figure 23: Output from the Burp Suite custom plugin identifying missing HSTS headers, insecure cookies, and potential IDOR vectors during dynamic analysis of multiple websites.

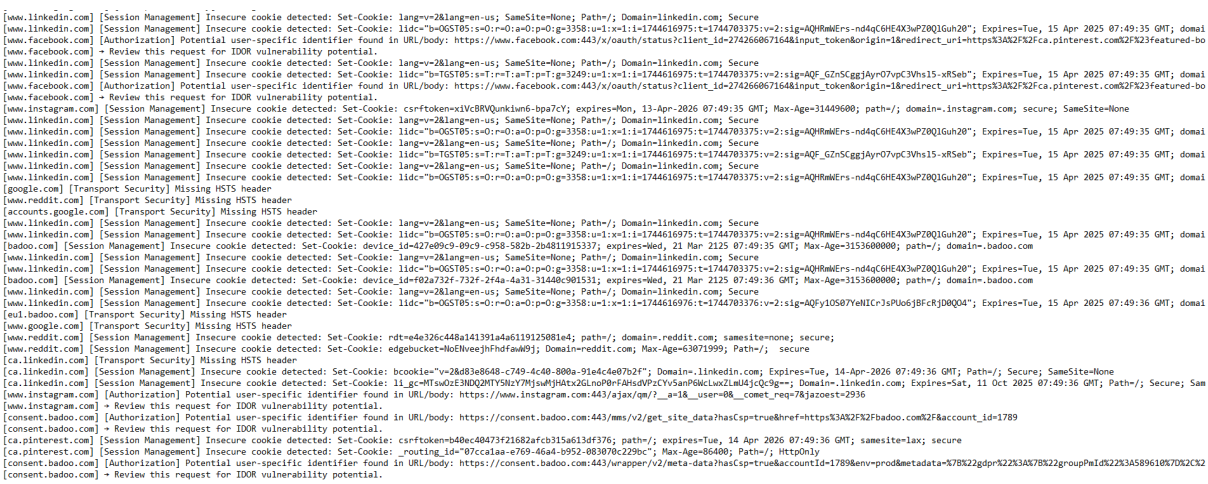


Figure 24: Consolidated plugin output highlighting session management flaws, transport security issues, and authorization weaknesses across various endpoints.