

ETX Signal-X

Daily Intelligence Digest

Sunday, February 8, 2026

10

ARTICLES

13

TECHNIQUES

Article 1

PEAR is a framework and distribution system for reusable PHP components. Prior to version 1.33.0, a SQL injection vulnerability in category deletion can allow an attacker with access to the category manager workflow to inject SQL via a category id. This i

Source: threatable

Date:

URL: <https://github.com/pear/pearweb/security/advisories/GHSA-q28j-3p7r-6722>

T1 SQL Injection via Category Deletion Workflow

⚡ Payload

```
1; DROP TABLE users--
```

⚔️ Attack Chain

- 1 Attacker authenticates to the PEAR admin panel and accesses the category manager workflow ('public_html/admin/category-manager.php').
- 2 Attacker crafts a POST request to the category deletion endpoint, setting the `id` (or `parent/category id`) parameter to a malicious SQL payload (e.g., `1; DROP TABLE users--`).
- 3 The backend logic in `include/pear-database-category.php` directly interpolates the provided id into an SQL query without parameterization.
- 4 The injected SQL executes on the database, enabling arbitrary SQL execution (e.g., dropping tables, extracting data).

🔍 Discovery

Researcher reviewed the category deletion logic and identified that user-supplied category id was used directly in SQL queries. Inspection of request handling in the category manager revealed the vulnerable POST parameter.

🔒 Bypass

No authentication bypass described; attacker must have access to the category manager workflow. No mention of input filtering or WAF.

🔗 Chain With

Can be chained with privilege escalation (if attacker can gain admin access), or with lateral movement to pivot from SQLi to RCE if the database supports stacked queries or has access to sensitive configuration tables.

How a Forged JWT Token Exposed eGift Cards of all users worth Millions

Source: securitycipher

Date: 28-Nov-2025

URL: <https://codewithvamp.medium.com/how-a-forged-jwt-token-exposed-egift-cards-of-all-users-worth-millions-685f6cd20824>

T1 JWT Token Forgery via Missing Signature Validation

Payload

```
{
  "alg": "none",
  "typ": "JWT"
}.
{
  "user_id": "victim_user_id",
  "email": "victim@example.com"
}.
// No signature part
```

Attack Chain

- 1 Intercept a legitimate JWT token used for authentication in the eGift card API.
- 2 Replace the payload with any target user's information (e.g., user_id, email).
- 3 Set the JWT header to `{"alg": "none"}` and remove the signature part.
- 4 Submit the forged JWT token in the Authorization header to the API endpoint handling eGift card operations.
- 5 API accepts the token and returns all eGift cards for the target user.

Discovery

Observed that the API did not validate the JWT signature and accepted tokens with `alg: none`. Manual token manipulation and replay confirmed the flaw.

Bypass

The server did not enforce signature verification for JWT tokens, allowing attackers to forge tokens for any user by setting `alg: none` and omitting the signature.

Chain With

Can be chained with IDOR or privilege escalation if other endpoints accept forged JWTs for sensitive operations.

T2

Mass Enumeration of eGift Cards via Forged JWTs

⚡ Payload

```
{  
  "alg": "none",  
  "typ": "JWT"  
} .  
{  
  "user_id": "<enumerated_user_id>",  
  "email": "<enumerated_email>"  
} .  
  
// No signature part
```

✗ Attack Chain

- 1 Enumerate user IDs and emails from other sources (e.g., registration, leaked data, or predictable patterns).
- 2 For each enumerated user, craft a forged JWT token as above.
- 3 Use the forged token to query the eGift card API endpoint.
- 4 Collect eGift card details for all users in bulk.

🔍 Discovery

After confirming JWT forgery, tested mass enumeration by automating token generation for multiple users and observed unrestricted access to all eGift cards.

🔓 Bypass

No rate limiting or anti-automation controls were present. Lack of signature validation enabled full account takeover for eGift card assets.

🔗 Chain With

Can be combined with automated redemption or transfer endpoints for large-scale theft.

HTTP Request Smuggling in Government Websites

Source: securitycipher

Date: 29-Aug-2025

URL: <https://osintteam.blog/http-request-smuggling-in-government-websites-30fcbe230415>

 No actionable techniques found

The “Blind Sniper” Attack: Spamming Thousands of Users Without Knowing Their Email Addresses

Source: securitycipher

Date: 08-Jan-2026

URL: <https://systemweakness.com/the-blind-sniper-attack-spamming-thousands-of-users-without-knowing-their-email-addresses-4d62ac327410>

T1 Blind Sniper Mass Notification Abuse

Payload

```
POST /api/notifications/send
{
  "user_id": "<target_user_id>",
  "message": "You've won a prize! Click here: http://malicious.site"
}
```

Attack Chain

- 1 Enumerate user IDs via predictable patterns, API responses, or indirect leaks.
- 2 Craft a POST request to the notification API endpoint using a valid session or token.
- 3 Set the `user_id` parameter to each enumerated user ID and supply a spam/phishing message in the `message` field.
- 4 Automate the process to send mass notifications to thousands of users without needing their email addresses.

Discovery

Researcher noticed the notification API accepted arbitrary user IDs and allowed custom messages, without verifying sender permissions or limiting mass actions.

Bypass

No email address or contact info required; attack leverages internal user identifiers and abuses lack of sender validation.

Chain With

Can be chained with user enumeration bugs, session fixation, or privilege escalation to increase reach or automate targeting.

T2

Notification API Rate Limit Bypass

⚡ Payload

```
for user_id in range(1000, 9999):
    POST /api/notifications/send
    {
        "user_id": user_id,
        "message": "Spam content"
    }
```

✗ Attack Chain

- 1 Identify absence of rate limiting on the notification API endpoint.
- 2 Script rapid-fire requests to the endpoint, iterating over large user ID ranges.
- 3 Deliver spam or phishing notifications at scale, overwhelming user inboxes or notification centers.

🔍 Discovery

Triggered by observing no throttling or rate limit headers after repeated API requests; confirmed via automated testing.

🔒 Bypass

Attack bypasses standard anti-spam measures by directly abusing internal notification infrastructure, not external email systems.

🔗 Chain With

Can be chained with credential stuffing, brute-force user ID discovery, or notification template injection for more sophisticated attacks.

18.2 Lab: Exploiting Java deserialization with Apache Commons | 2024

Source: securitycipher

Date: 14-Apr-2024

URL: <https://cyberw1ng.medium.com/18-2-lab-exploiting-java-deserialization-with-apache-commons-2024-7ba379e97b52>

T1 Java Deserialization RCE via Apache CommonsCollections Gadget

⚡ Payload

```
java -jar ysoserial-all.jar CommonsCollections4 'rm /home/carlos/morale.txt' | base64
```

⚔️ Attack Chain

- 1 Log in to the target web application and inspect the session cookie; identify it contains a serialized Java object.
- 2 Use the ysoserial tool to generate a CommonsCollections4 gadget chain payload that executes `rm /home/carlos/morale.txt`.
- 3 Pipe the serialized object output through `base64` to encode it for cookie injection.
- 4 Replace the session cookie value in Burp Repeater with the base64-encoded malicious payload.
- 5 URL-encode the entire cookie value ("Convert selection" → "URL Encode All Characters").
- 6 Send the request to the application to trigger deserialization and execute the command, deleting the target file.

🔍 Discovery

Manual inspection of session cookie after login revealed a serialized Java object. Knowledge of Apache Commons Collections presence and lack of source code access led to gadget chain exploitation using ysoserial.

🔒 Bypass

If issues arise with running ysoserial directly, use Docker to build and run ysoserial: `git clone https://github.com/frohoff/ysoserial.git cd ysoserial docker build -t ysoserial:latest . docker run ysoserial:latest CommonsCollections4 'rm /home/carlos/morale.txt' | base64` Additionally, for Java 16+, use JVM flags to bypass module restrictions: `java -jar ysoserial-all.jar \ --add-opens=java.xml/com.sun.org.apache.xalan.internal.xsltc.trax=ALL-UNNAMED \ --add-opens=java.xml/com.sun.org.apache.xalan.internal.xsltc.runtime=ALL-UNNAMED \ --add-opens=java.base/java.net=ALL-UNNAMED \ --add-opens=java.base/java.util=ALL-UNNAMED \ CommonsCollections4 'rm /home/carlos/morale.txt' | base64`

🔗 Chain With

Any endpoint accepting serialized Java objects (not just session cookies) can be targeted with similar gadget chain payloads for RCE. Can be chained with privilege escalation if the deserialized object runs as a privileged user. Potential for lateral movement if other files or commands can be executed via the same gadget chain.

Breaking Through the Limits: How I Bypassed Rate-Limiting with IP and Username Rotation

Source: securitycipher

Date: 23-Dec-2024

URL: <https://medium.com/@abdelrahmanhisham/breaking-through-the-limits-how-i-bypassed-rate-limiting-with-ip-and-username-rotation-d8de230aec2a>

T1

Rate-Limit Bypass via Alternating IP and Username Rotation

Payload

```
POST /login
{
  "username": "targetuser",
  "password": "guess"
}
```

Attack Chain

- 1 Attempt up to the rate limit (e.g., 100 login attempts per 10 minutes) using a single IP and username.
- 2 Change the IP address after reaching the limit for the current username, then continue login attempts (limit resets once).
- 3 When the IP-based limit is hit for the same username, switch to a new username and continue attempts (limit resets once).
- 4 Alternate between changing IP and username after each rate-limit block, allowing continuous brute-force attempts without interruption.

Discovery

Observed that changing either IP or username resets the rate limit once, but not twice in a row for the same parameter. Systematically alternated both to test if the limit could be bypassed indefinitely.

Bypass

Rate limit is only enforced per IP or per username, not both together. Alternating between IP and username circumvents the intended restriction logic.

Chain With

Can be chained with credential stuffing, brute-force, or account takeover attacks when combined with username enumeration.

T2

Username Enumeration via Sign-Up Page Feedback

⚡ Payload

```
POST /signup
{
  "username": "targetuser",
  "password": "irrelevant"
}
```

⚔️ Attack Chain

- 1 Submit sign-up requests with a list of potential usernames.
- 2 Observe the application's response for each request.
- 3 Note which requests return a message indicating the username is "already taken".
- 4 Compile a list of valid usernames for use in further attacks (e.g., brute-force, credential stuffing).

🔍 Discovery

Tested the sign-up page with various usernames and noticed distinct feedback for already-registered usernames, enabling enumeration.

🔒 Bypass

No bypass required; the application exposes enumeration via direct feedback.

🔗 Chain With

Enumerated usernames can be used in conjunction with rate-limit bypass to perform targeted brute-force or credential stuffing attacks.

How I Discovered an HTML Injection via a Signup Form

Source: securitycipher

Date: 27-Oct-2025

URL: <https://medium.com/@gehadr73/how-i-discovered-an-html-injection-via-a-signup-form-4aa29b7da2a0>

T1 HTML Injection via Signup Form

Payload

>

Attack Chain

- 1 Navigate to the signup form on the target application.
- 2 In the "Name" field, input the payload: ">"
- 3 Submit the form.
- 4 Upon successful registration, view the user profile or any page where the submitted name is rendered.
- 5 Observe that the payload is executed, confirming HTML injection.

Discovery

Manual fuzzing of form fields with special characters and HTML tags during signup, specifically testing for improper output encoding.

Bypass

The payload leverages the lack of output encoding and escapes out of any HTML attribute context, injecting a new tag and executing JavaScript.

Chain With

Can be chained with session hijacking, phishing, or privilege escalation if the injected HTML/JS is rendered in privileged contexts or admin panels.

No Session Expiry after log-out, attacker can reuse the old cookies

Source: securitycipher

Date: 08-Jan-2025

URL: <https://mknayek101.medium.com/no-session-expiry-after-log-out-attacker-can-reuse-the-old-cookies-b90a4a45032d>

T1 Session Persistence via Reusable Cookies After Logout

Payload

Exported session cookies via EditThisCookie browser extension (exact cookie values depend on target app)

Attack Chain

- 1 Login to target application using "Login via Google" or "Login via Facebook".
- 2 Use browser extension (e.g., EditThisCookie) to export active session cookies.
- 3 Logout from the application in the original browser.
- 4 Open a second browser (or incognito window) and import the previously exported cookies.
- 5 Access the application in the second browser; session is restored and user is authenticated as the victim.
- 6 Even after victim logs out and logs back in, attacker can reuse old cookies to regain access as long as the victim's session remains active on the server.

Discovery

Manual session management testing using cookie export/import after observing "Login via Facebook/Google" integration. Researcher noticed session did not expire after logout and tested cookie reuse across browsers.

Bypass

Session tokens are not invalidated server-side upon logout; old cookies remain valid for authentication until the session is explicitly expired by the server.

Chain With

Can be chained with any vulnerability that enables cookie theft (e.g., XSS, MITM, insecure storage) to achieve persistent account takeover even after user logout. Also enables lateral movement if session tokens grant access to other linked services or OAuth scopes.

DOM XSS in document.write Sink Using Source location.search Inside a Element

Source: securitycipher

Date: 29-Jul-2025

URL: <https://infosecwriteups.com/dom-xss-in-document-write-sink-using-source-location-search-inside-a-select-element-6df5304d9b11>

T1 DOM XSS via document.write Sink with location.search Source Inside <select> Element

Payload

```
https://target.com/page?input=<select><option>test</option><option>test2</option></select><img src=x onerror=alert(1)>
```

Attack Chain

- 1 Identify a page where `document.write` is used to insert user-supplied data from `location.search` into the DOM.
- 2 Craft a payload that includes a `<select>` element with options, followed by an `` tag with an `onerror` event handler executing JavaScript (e.g., `alert(1)`).
- 3 Supply the payload in the `input` query parameter, causing the browser to parse and execute the injected script when the page loads.

Discovery

Manual inspection of JavaScript sources revealed that `document.write` was used with unsanitized `location.search` data, and the context allowed HTML tags inside a `<select>` element to be parsed, enabling script execution via event handlers.

Bypass

The use of a `<select>` element bypasses naive filtering that only expects text or simple tags, leveraging browser parsing quirks to allow script execution even inside complex HTML structures.

Chain With

Can be chained with session hijacking, cookie theft, or privilege escalation if the XSS occurs on authenticated pages or in privileged contexts.

T2

Event Handler Injection via Malformed Tag After <select> Element

⚡ Payload

```
<select><option>test</option></select><img src=x onerror=alert(document.domain)>
```

⚔️ Attack Chain

- 1 Inject a `<select>` element to pass through basic HTML tag filtering.
- 2 Immediately follow with an `` tag containing a JavaScript event handler (e.g., `onerror=alert(document.domain)`).
- 3 The browser parses the malformed HTML and executes the event handler, triggering the payload.

🔍 Discovery

Testing revealed that the parser did not properly isolate the `<select>` context, allowing subsequent tags and event handlers to be executed. This was discovered by experimenting with different tag combinations and observing browser behavior.

🔒 Bypass

Combining form elements with scriptable tags (like ``) circumvents filters that only block direct script tags or inline event handlers outside form contexts.

🔗 Chain With

Useful for escalating from reflected XSS to stored XSS if the payload can be persisted, or for bypassing CSP if inline event handlers are not blocked.

Privilege Escalation Turned a Regular User Into an Admin [part-1]

Source: securitycipher

Date: 26-Oct-2025

URL: <https://senoritaahunter.medium.com/privilege-escalation-turned-a-regular-user-into-an-admin-part-1-fbe3d82440ca>

T1 Inconsistent Backend Permission Checks Allow Regular User to Perform Admin Actions

Payload

```
POST /api/team/add_member
{
  "team_id": "<team_id>",
  "email": "xyz@gmail.com"
}
```

Attack Chain

- 1 Log in to xyz.com with a regular User account (no admin privileges).
- 2 Create a new team (e.g., "Team Testers United") using the UI or API.
- 3 Use the API endpoint to add a member to the team via a POST request, despite UI restrictions.
- 4 Edit and remove team members using corresponding API endpoints, confirming full admin-like control.
- 5 Validate against documentation that Users should not have these permissions.

Discovery

Triggered by curiosity after noticing the UI allowed team creation and member addition with a regular user account. Used browser dev tools to inspect network requests and observed successful POSTs for add/edit/remove member actions. Cross-referenced with public documentation to confirm the intended restrictions.

Bypass

The backend API only checked if the user was a member of the team, not their global role. This allowed any authenticated user within a team context to perform admin-level actions regardless of their actual role.

Chain With

Use this flaw to escalate privileges for any user in any team. Add malicious accounts to exfiltrate data or create persistence. Remove legitimate users, causing denial of service or sabotage. Potentially chain with weak role assignment endpoints to grant global admin rights if available.

Automated with ❤️ by ethicxlhuman