ETX Signal-X

Daily Intelligence Digest

Sunday, February 1, 2026

10
ARTICLES

20
TECHNIQUES

Article 1

From Cross-Subdomain Cookie Reuse to Becoming Super Admin: An Exploit Chain Walkthrough

Source: securitycipher

Date: 07-May-2025

URL: https://medium.com/@cyberpro151/from-cross-subdomain-cookie-reuse-to-becoming-super-admin-an-exploit-chain-walkthrough-32527caa2a11

T1 Cross-Subdomain Cookie Reuse for Privilege Escalation

💉 Payload

```
Set-Cookie: session=eyJ1c2VyX2lkIjoxLCJyb2xlIjoiYWRtaW4ifQ==; Domain=.target.com; Path=/; HttpOnly; Secure
```

⚔️ Attack Chain

1. Attacker logs in to a low-privileged account on subdomain1.target.com and captures the session cookie.
2. Attacker observes that the session cookie is set with `Domain=.target.com`, making it valid across all subdomains.
3. Attacker reuses the same session cookie on subdomain2.target.com, which has different access controls.
4. On subdomain2.target.com, the same session grants elevated privileges due to misaligned role checks.

🔍 Discovery

Manual inspection of cookie attributes during authentication flow; noticed the domain scope was set to the parent domain, not the specific subdomain.

🔓 Bypass

Leverages the fact that subdomain2 does not properly validate the session's origin or re-check user role, trusting the cookie blindly.

🔗 Chain With

Can be chained with IDOR or privilege escalation bugs on subdomain2 to gain further access.

T2 JWT Manipulation via Weak Secret for Privilege Escalation

💉 Payload

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2lkIjoxLCJyb2xlIjoic3VwZXJhZG1pbiJ9.DUMMY_SIGNATURE
```

⚔️ Attack Chain

1. Attacker discovers JWT is used for session management and is signed with a weak/guessable secret (e.g., 'secret').
2. Attacker decodes the JWT and modifies the `role` claim to `superadmin`.
3. Attacker re-signs the JWT with the weak secret and replaces their session token with the forged one.
4. Application accepts the forged JWT and grants super admin privileges.

🔍 Discovery

Tested JWT signature with common weak secrets using tools like jwt_tool; observed valid authentication with forged tokens.

🔓 Bypass

Bypasses backend role validation by forging a valid JWT with escalated privileges.

🔗 Chain With

Can be chained after cross-subdomain cookie reuse to escalate privileges from admin to super admin.

T3 Exploiting Unrestricted Admin Panel Access via Predictable URL

💉 Payload

```
https://admin.target.com/panel/superadmin
```

⚔️ Attack Chain

1. Attacker identifies that the admin panel is accessible at a predictable URL on a separate subdomain.
2. Attacker accesses the URL directly after obtaining elevated privileges (e.g., via JWT manipulation).
3. Application does not enforce additional access controls on the endpoint, granting full super admin access.

🔍 Discovery

Brute-forced or guessed admin panel URLs after privilege escalation; found no secondary authentication or access restrictions.

🔓 Bypass

Relies on security-through-obscurity; no actual privilege checks on the endpoint.

🔗 Chain With

Completes the exploit chain from initial low-privilege access to full super admin control.

Article 2

HTTP Request Smuggling: Basic CL.TE Vulnerability

Source: securitycipher

Date: 14-Aug-2025

URL: https://infosecwriteups.com/http-request-smuggling-basic-cl-te-vulnerability-2cadaa2d9640

T1 CL.TE HTTP Request Smuggling via Inconsistent Header Parsing

💉 Payload

```
POST / HTTP/1.1
Host: vulnerable-website.com
Content-Length: 13
Transfer-Encoding: chunked

0

POST /malicious HTTP/1.1
Host: vulnerable-website.com
Content-Length: 6

attack
```

⚔️ Attack Chain

1. Craft an HTTP request with both `Content-Length` and `Transfer-Encoding: chunked` headers (CL.TE ambiguity).
2. Set `Content-Length` to the length of the first body (e.g., 13), but include `Transfer-Encoding: chunked` to trigger parsing confusion.
3. After the `0\n\n` (end of chunked body), append a second HTTP request (e.g., `POST /malicious ...`).
4. If the front-end server uses `Content-Length` and the back-end uses `Transfer-Encoding`, the back-end interprets the second request as a new, valid HTTP request, enabling request smuggling.

🔍 Discovery

Manual inspection of HTTP request parsing logic, looking for dual presence of `Content-Length` and `Transfer-Encoding` headers and observing discrepancies in how front-end and back-end servers handle them.

🔓 Bypass

The attack exploits differences in header parsing between front-end and back-end servers. If the front-end honors `Content-Length` and the back-end honors `Transfer-Encoding`, the attacker can smuggle arbitrary requests past the front-end.

🔗 Chain With

Can be chained with session fixation or cache poisoning if the smuggled request manipulates authentication or cache headers. Potential for credential theft or privilege escalation if the smuggled request targets sensitive endpoints.

Article 3

Mastering Subdomain Takeover: Step-by-Step Guide with Real Tools and Techniques

Source: securitycipher

Date: 30-Oct-2025

URL: https://medium.com/@verylazytech/mastering-subdomain-takeover-step-by-step-guide-with-real-tools-and-techniques-8f1c8a4583ca

T1 Subdomain Takeover via Unresolved CNAME to Third-Party Service

💉 Payload

```
Add CNAME record pointing to non-existent.s3.amazonaws.com
```

⚔️ Attack Chain

1. Identify subdomain (e.g., `static.example.com`) with a CNAME record pointing to a third-party service (e.g., `non-existent.s3.amazonaws.com`).
2. Confirm that the referenced resource (e.g., S3 bucket) does not exist or is not claimed.
3. Register or create the missing resource (e.g., create the S3 bucket named `non-existent`).
4. Upload content to the resource to control responses for the subdomain.

🔍 Discovery

Manual DNS enumeration and tools like `subjack`, `tko-subs`, or `amass` to find subdomains with dangling CNAMEs pointing to cloud services.

🔓 Bypass

If the cloud provider blocks bucket creation for certain names, try variations or alternative providers with similar CNAME patterns.

🔗 Chain With

Combine with phishing, cookie theft, or credential harvesting by hosting malicious content on the taken-over subdomain.

T2 Subdomain Takeover via Unclaimed SaaS Service (e.g., Heroku, GitHub Pages)

💉 Payload

```
CNAME record: victim.example.com -> victim.herokuapp.com
```

⚔️ Attack Chain

1. Locate subdomains with CNAME records referencing SaaS platforms (e.g., `victim.herokuapp.com`, `victim.github.io`).
2. Check if the referenced SaaS resource/account is unclaimed or deleted.
3. Register the resource on the SaaS platform using the same name.
4. Deploy attacker-controlled content to the SaaS resource.

🔍 Discovery

Automated scanning of DNS records for known SaaS CNAME patterns using tools like `subjack` or custom scripts.

🔓 Bypass

If direct registration is blocked, attempt to find case-sensitive or Unicode variants accepted by the SaaS provider.

🔗 Chain With

Leverage for session hijacking, cookie scope abuse, or as a trusted domain for social engineering.

T3 Subdomain Takeover via Orphaned NS Records

💉 Payload

```
NS record: orphaned.example.com -> ns1.abandoned-nameserver.com
```

⚔️ Attack Chain

1. Enumerate subdomains with custom NS records pointing to expired or unregistered nameservers.
2. Register the abandoned nameserver domain (e.g., `abandoned-nameserver.com`).
3. Set up a nameserver at the registered domain.
4. Control DNS resolution for the target subdomain, allowing creation of arbitrary records (A, MX, TXT, etc.).

🔍 Discovery

DNS zone transfer attempts and analysis of NS records for subdomains, looking for unregistered nameserver domains.

🔓 Bypass

If the nameserver domain is unavailable, monitor for domain drop or target subdomains with multiple NS records where only one is orphaned.

🔗 Chain With

Facilitate further attacks such as email interception (MX), SPF/DKIM manipulation, or setting up phishing infrastructure.

T4 Subdomain Takeover via Unused Delegated Zone (Zone Apex Takeover)

💉 Payload

```
Delegate zone: unusedzone.example.com -> ns1.attacker.com
```

⚔️ Attack Chain

1. Identify subdomains delegated as their own DNS zones but with unconfigured or unresponsive nameservers.
2. Register the nameserver domain (e.g., `attacker.com`) if unclaimed.
3. Set up authoritative DNS for the delegated zone.
4. Create arbitrary DNS records for the subdomain and its children.

🔍 Discovery

DNS enumeration for subdomains with SOA and NS records pointing to external, unresponsive, or unregistered nameservers.

🔓 Bypass

If the nameserver is partially responsive, attempt to race or poison cache with attacker-controlled responses.

🔗 Chain With

Allows for wildcard subdomain creation, mail interception, and hosting malicious content under the target's DNS hierarchy.

T5 Subdomain Takeover via Unused CloudFront Distribution

💉 Payload

```
CNAME record: cdn.example.com -> d1234.cloudfront.net
```

⚔️ Attack Chain

1. Find subdomains with CNAMEs to CloudFront distributions (e.g., `d1234.cloudfront.net`).
2. Check if the CloudFront distribution is deleted or unclaimed.
3. Create a new CloudFront distribution with the same domain alias.
4. Serve attacker-controlled content via the subdomain.

🔍 Discovery

Automated tools or manual checks for CNAMEs pointing to CloudFront and verification of distribution status via HTTP responses or AWS console.

🔓 Bypass

If the distribution cannot be claimed due to AWS restrictions, monitor for changes or attempt similar takeovers on other CDN-backed subdomains.

🔗 Chain With

Host malicious scripts, perform credential phishing, or abuse as a trusted CDN endpoint for supply chain attacks.

Article 4

From a Single Quote & a Space to a 1-Year AI Subscription

Source: securitycipher

Date: 10-Dec-2025

URL: https://medium.com/@shabutaher0/from-a-single-quote-a-space-to-a-1-year-ai-subscription-bc89a6671eff

T1 SQL Injection via Single Quote in Email Parameter

💉 Payload

```
'
```

⚔️ Attack Chain

1. Register a new account using an email address containing a single quote (e.g., `test'@example.com`).
2. Observe server error or abnormal behavior indicating SQL query breakage.
3. Leverage the injection point to manipulate SQL queries for further exploitation (e.g., authentication bypass, data extraction).

🔍 Discovery

Inputting a single quote in the email field during registration triggered a server error, indicating an unhandled SQL injection point.

🔓 Bypass

N/A

🔗 Chain With

Can be chained with authentication logic flaws or privilege escalation if SQLi leads to account takeover or admin access.

T2 Space Character in Email for Account Duplication and Subscription Abuse

💉 Payload

```
user @example.com
```

⚔️ Attack Chain

1. Register an account with an email address containing a space before the `@` symbol (e.g., `user @example.com`).
2. The application treats `user @example.com` and `user@example.com` as separate accounts due to improper email normalization.
3. Abuse this to register multiple accounts with the same underlying email, each eligible for a free trial or subscription benefit.

🔍 Discovery

Testing email input validation revealed that the application did not normalize or reject emails with spaces, allowing duplicate account creation.

🔓 Bypass

Bypassing email uniqueness checks by inserting whitespace, which is ignored by some email providers but not by the application.

🔗 Chain With

Can be chained with business logic flaws to obtain multiple free trials, discounts, or subscription periods on the same email.

T3 Subscription Abuse via Email Address Manipulation

💉 Payload

```
user+anything@example.com
```

⚔️ Attack Chain

1. Register multiple accounts using email aliasing (e.g., `user+1@example.com`, `user+2@example.com`).
2. The application treats each as a unique user, but all emails route to the same inbox.
3. Abuse this to repeatedly claim free subscription offers or trial periods.

🔍 Discovery

Testing for email aliasing revealed the application did not normalize or block `+` aliases, allowing repeated benefit claims.

🔓 Bypass

Bypassing account creation limits by leveraging email aliasing supported by common providers (e.g., Gmail).

🔗 Chain With

Combine with other registration logic flaws for mass exploitation of subscription or promotional systems.

Article 5

How I Found an Unauthenticated Jira API Endpoint Leaking Internal Build Data

Source: securitycipher

Date: 03-Nov-2025

URL: https://medium.com/@dipanshuchhanikar/how-i-found-an-unauthenticated-jira-api-endpoint-leaking-internal-build-data-2d1dcf10f181

T1 Unauthenticated Jira API Endpoint Leaking Internal Build Data

💉 Payload

```
GET /rest/bitbucket/latest/build-status/1.0/commits/{commit_hash}
```

⚔️ Attack Chain

1. Identify Jira instance and test for the Bitbucket build status API endpoint: `/rest/bitbucket/latest/build-status/1.0/commits/{commit_hash}`
2. Send unauthenticated GET request to the endpoint, substituting `{commit_hash}` with a valid or brute-forced commit hash.
3. Receive internal build data (e.g., build status, timestamps, possibly environment details) in the response without authentication.

🔍 Discovery

Researcher noticed Bitbucket integration in Jira UI and probed related REST API endpoints for authentication requirements. Fuzzed common Bitbucket API paths and found this endpoint did not enforce authentication.

🔓 Bypass

No authentication or session required; endpoint is accessible to anyone with the commit hash.

🔗 Chain With

Use build metadata to enumerate internal project structure, branch names, and CI/CD pipeline details. Leverage leaked data for further social engineering or to identify weak points in the build process for supply chain attacks.

Article 6

$22,300 Bug Bounty: Cloning Private GitLab Repositories via Import Feature

Source: securitycipher

Date: 10-Jan-2026

URL: https://osintteam.blog/22-300-bug-bounty-cloning-private-gitlab-repositories-via-import-feature-15bcabb62530

⚠️ Failed to fetch content

Article 7

Uncovering Host Header Injection Vulnerabilities in 5 Apex Domain Hosts Part Two how to chain

Source: securitycipher

Date: 14-Mar-2024

URL: https://javroot.medium.com/uncovering-host-header-injection-vulnerabilities-in-5-apex-domain-hosts-part-two-how-to-chain-0abe308a4807

T1 Host Header Injection for Password Reset Link Manipulation

💉 Payload

```
Host: attacker.com
```

⚔️ Attack Chain

1. Initiate a password reset on the target application.
2. Intercept the password reset request and modify the `Host` header to `attacker.com`.
3. The application generates a password reset link using the supplied `Host` header value.
4. The victim receives an email with a password reset link pointing to `attacker.com`, allowing the attacker to capture the reset token.

🔍 Discovery

Manual review of password reset flows and observation of the `Host` header's influence on generated links.

🔓 Bypass

If the application uses the `Host` header directly without validation or canonicalization, this attack is possible even if the application is behind a proxy or load balancer.

🔗 Chain With

Can be chained with phishing or open redirect to escalate from token theft to account takeover or further lateral movement.

T2 Host Header Injection to Poison Cache and Serve Malicious Content

💉 Payload

```
Host: attacker.com
```

⚔️ Attack Chain

1. Send a crafted request with `Host: attacker.com` to a cache-enabled endpoint.
2. The application or CDN caches the response under the attacker-controlled host.
3. Subsequent legitimate requests may be served the cached, attacker-influenced response, leading to content spoofing or session fixation.

🔍 Discovery

Testing cacheable endpoints with varying `Host` headers and observing cache keys and response content.

🔓 Bypass

If the cache key includes the `Host` header and the application does not validate it, this attack works even when standard anti-cache-poisoning headers are present.

🔗 Chain With

Can be chained with reflected XSS or open redirect to serve persistent malicious payloads to users.

T3 Host Header Injection for SSRF via Internal Service Discovery

💉 Payload

```
Host: 127.0.0.1
```

⚔️ Attack Chain

1. Identify an endpoint that makes server-side HTTP requests based on the `Host` header value.
2. Send a request with `Host: 127.0.0.1` or another internal IP.
3. The server makes a backend request to the internal service, potentially exposing sensitive data or internal APIs.

🔍 Discovery

Fuzzing internal IPs and localhost values in the `Host` header and monitoring for SSRF indicators or internal error messages.

🔓 Bypass

Works if the application does not restrict or sanitize the `Host` header, even if SSRF protections are in place elsewhere.

🔗 Chain With

Can be chained with privilege escalation if internal admin endpoints are exposed, or with data exfiltration attacks.

Article 8

$250 Bounty: Poisoning the Prototype: Exploiting Lodash’s Hidden Attack Surface

Source: securitycipher

Date: 20-Jul-2025

URL: https://medium.com/meetcyber/250-bounty-poisoning-the-prototype-exploiting-lodashs-hidden-attack-surface-bbc092de974c

T1 Lodash zipObjectDeep() Prototype Pollution via Crafted Key

💉 Payload

```
zipObjectDeep(['__proto__.polluted'], ['yes'])
```

⚔️ Attack Chain

1. Identify usage of Lodash's `zipObjectDeep()` in the target application (especially v4.17.15 or similar).
2. Craft an array input with a key containing `__proto__`, e.g., `['__proto__.polluted']`.
3. Pass this array as the first argument to `zipObjectDeep()`, with a corresponding value array (e.g., `['yes']`).
4. The resulting object will have its prototype polluted: `{}.polluted === 'yes'` will now return true globally.
5. Leverage this prototype pollution to affect application logic, escalate to denial of service, or chain with other gadgets for further exploitation (e.g., RCE if the application uses polluted properties unsafely).

🔍 Discovery

Researcher reviewed Lodash's deep object creation helpers for unsafe handling of special property names. Focused on `zipObjectDeep()` due to its pattern of accepting user-controlled property paths.

🔓 Bypass

No explicit blacklist or sanitization of `__proto__` or similar keys in the vulnerable version, allowing direct prototype pollution.

🔗 Chain With

Combine with gadgets that rely on polluted properties (e.g., merge, clone, or template rendering functions) for code execution or logic manipulation. Use polluted properties to bypass security checks or escalate privileges in downstream logic.

Article 9

PHP Type Juggling Vulnerabilities: How Attackers Exploit Loose Comparisons

Source: securitycipher

Date: 06-Feb-2025

URL: https://0xkratos.medium.com/php-type-juggling-vulnerabilities-how-attackers-exploit-loose-comparisons-e4e0c78ec9e6

T1 Loose Comparison Bypass with "0e" Notation

💉 Payload

```
0e1234567
```

⚔️ Attack Chain

1. Identify a PHP application using loose comparison (==) to validate user input against a stored hash (e.g., password or token).
2. Submit a payload such as `0e1234567` as the input value.
3. PHP interprets both `0e1234567` and a hash like `0e9876543` as floats in scientific notation (0 × 10^1234567), both evaluating to 0.
4. The loose comparison returns true, bypassing authentication or validation.

🔍 Discovery

Noticed application using `==` instead of `===` for hash comparison; tested with known magic values ("0e" + digits) to trigger type juggling.

🔓 Bypass

By exploiting PHP's type juggling, any string matching the regex `^0e[0-9]+$` will be interpreted as 0 in scientific notation, allowing bypass if the stored hash also matches this pattern.

🔗 Chain With

Can be chained with weak hash algorithms (e.g., md5, crypt) to generate predictable magic hashes for further privilege escalation or account takeover.

T2 Array to String Comparison Bypass

💉 Payload

```
input[]=test
```

⚔️ Attack Chain

1. Locate a parameter that is compared loosely to a string (e.g., `if ($_GET['input'] == 'password')`).
2. Submit the parameter as an array: `input[]=test`.
3. PHP compares the array to the string, resulting in `false`, but if the logic is inverted (e.g., `!=`), it may unintentionally grant access.

🔍 Discovery

Tested input parameters by sending them as arrays instead of strings, observing logic misbehavior in authentication or access control.

🔓 Bypass

PHP's loose comparison between arrays and strings always returns false, which can be abused if the application expects a true result for valid credentials and uses inverted logic.

🔗 Chain With

Can be used to bypass access controls or logic checks, especially when combined with other parameter pollution or logic flaw vulnerabilities.

T3 Numeric String and Integer Comparison Bypass

💉 Payload

```
"123abc" == 123
```

⚔️ Attack Chain

1. Find a place where user input is compared to an integer using `==` (e.g., `if ($_POST['id'] == 123)`).
2. Submit a payload such as `id=123abc`.
3. PHP converts the string to integer ("123abc" → 123) during loose comparison, causing the check to pass.

🔍 Discovery

Identified integer comparisons using loose equality; tested with numeric strings appended with alphabetic characters.

🔓 Bypass

PHP's type juggling converts any string starting with a number to that number during loose comparison, allowing bypass with crafted input.

🔗 Chain With

Can be used to bypass ID-based access controls, or to escalate privileges if user roles or permissions are validated in this manner.

Article 10

Hidden in the Noise: How to Tweak Dirbuster, Gobuster & FFUF Wordlists for Better Hits

Source: securitycipher

Date: 30-Jun-2025

URL: https://medium.com/h7w/hidden-in-the-noise-how-to-tweak-dirbuster-gobuster-ffuf-wordlists-for-better-hits-c47d572141d5

⚠️ Methodology article - no vulnerabilities