ETX Signal-X

Daily Intelligence Digest

Saturday, January 31, 2026

10	27
ARTICLES	TECHNIQUES

Article 1

GraphQL API hacking Series for Bug Hunters Part 02

Source: securitycipher

Date: 29-Apr-2025

URL: https://medium.com/@lancersiromony/graphql-api-hacking-series-for-bug-hunters-part-02-837e0bc3be06

T1 Exploiting GraphQL Field Aliasing to Bypass Blacklist Filtering

💉 Payload

```
{
  adminUser: user(id: 1) {
    id
    username
    email
    password
  }
}
```

⚔️ Attack Chain

1. Identify a GraphQL API endpoint that restricts access to sensitive fields (e.g., `adminUser` or `password`) via blacklist filtering.
2. Use GraphQL's field aliasing feature to rename the restricted field in the query (e.g., `adminUser: user`).
3. Submit the aliased query to the endpoint.
4. Observe if the response includes the sensitive data despite blacklist protections.

🔍 Discovery

Manual inspection of GraphQL schema and queries, noticing that field aliasing is supported and may not be covered by backend filtering logic.

🔓 Bypass

Field aliasing allows attackers to request blacklisted fields under a different name, bypassing naive blacklist implementations.

🔗 Chain With

Can be chained with introspection or enumeration to identify additional sensitive fields or escalate to privilege escalation if admin data is exposed.

T2 Using GraphQL Fragments to Circumvent Field Restrictions

💉 Payload

```
{
  ...userFields
}

fragment userFields on Query {
  user(id: 1) {
    id
    username
    email
    password
  }
}
```

⚔️ Attack Chain

1. Find a GraphQL API that restricts direct querying of sensitive fields (e.g., `password`).
2. Create a fragment that includes the sensitive fields.
3. Reference the fragment in the main query.
4. Send the query to the endpoint and check if the sensitive field is returned.

🔍 Discovery

Noticing that fragments are supported and may not be covered by field-level access controls.

🔓 Bypass

Fragments can be used to request restricted fields indirectly, bypassing direct field name checks.

🔗 Chain With

Combine with aliasing or introspection to maximize field exposure and data exfiltration.

T3 Overriding Default Query Arguments to Access Hidden Data

💉 Payload

```
{
  users(role: "admin") {
    id
    username
    email
  }
}
```

⚔️ Attack Chain

1. Analyze the schema for query arguments (e.g., `role`).
2. Attempt to override default or hidden arguments in the query (e.g., set `role: "admin"`).
3. Submit the query and observe if privileged data is returned.

🔍 Discovery

Schema enumeration and fuzzing query arguments to identify undocumented or weakly protected parameters.

🔓 Bypass

Directly setting arguments that are assumed to be internal or default can bypass access controls if not properly validated.

🔗 Chain With

Can be chained with user enumeration or privilege escalation attacks if admin or sensitive user data is exposed.

T4 Abuse of GraphQL Introspection to Map Sensitive Schema Elements

💉 Payload

```
{
  __schema {
    types {
      name
      fields {
        name
      }
    }
  }
}
```

⚔️ Attack Chain

1. Send an introspection query to the GraphQL endpoint.
2. Retrieve the full schema, including hidden or sensitive types and fields.
3. Use the schema information to craft targeted queries for sensitive data.

🔍 Discovery

Testing if introspection is enabled and not restricted on production endpoints.

🔓 Bypass

N/A (introspection is a feature, not a bypass, but exposes attack surface).

🔗 Chain With

Provides a blueprint for further exploitation, such as field aliasing, fragments, or argument manipulation.

Article 2

Finding more subdomains.(part 2)

Source: securitycipher

Date: 28-Nov-2024

URL: https://infosecwriteups.com/finding-more-subdomains-part-2-1850ead4dd92

⚠️ No actionable techniques found

Article 3

Google XSS Game Solution

Source: securitycipher

Date: 11-Dec-2025

URL: https://medium.com/@blueorionn/google-xss-game-solution-425ce2539a58

T1 Basic Script Injection via Input Value Attribute

💉 Payload

```
"><script>alert(1)</script>
```

⚔️ Attack Chain

1. Locate input field where user input is reflected inside the value attribute of an HTML element without proper encoding.
2. Submit the payload `"><script>alert(1)</script>` as the input value.
3. The input is rendered as: `<input value=""><script>alert(1)</script>" />`, breaking out of the attribute and injecting a script tag.
4. The script executes in the victim's browser.

🔍 Discovery

Observed user input reflected in the value attribute of an input element without encoding or sanitization.

🔓 Bypass

Breaks out of the attribute context using `">`.

🔗 Chain With

Can be chained with session fixation or cookie theft if the script can exfiltrate sensitive data.

T2 Script Injection via Image Tag Onerror Attribute

💉 Payload

```
"><img src=x onerror=alert(1)>
```

⚔️ Attack Chain

1. Find a location where user input is reflected in HTML without encoding.
2. Submit the payload `"><img src=x onerror=alert(1)>`.
3. The input is rendered as: `<div>"><img src=x onerror=alert(1)></div>`, breaking out of the context and injecting an image tag with an onerror handler.
4. The onerror handler executes JavaScript when the image fails to load.

🔍 Discovery

Tested for HTML injection and found that input is reflected directly into the DOM.

🔓 Bypass

Uses `">` to break out of the current tag and inject a new tag with an event handler.

🔗 Chain With

Can be used for session hijacking or further DOM-based attacks.

T3 Script Injection via SVG and Onload Attribute

💉 Payload

```
"><svg/onload=alert(1)>
```

⚔️ Attack Chain

1. Locate a reflection point where user input is placed in HTML context.
2. Submit the payload `"><svg/onload=alert(1)>`.
3. The input is rendered as: `<div>"><svg/onload=alert(1)></div>`, breaking out of the context and injecting an SVG tag with an onload handler.
4. The onload handler executes JavaScript as soon as the SVG loads.

🔍 Discovery

Tested for tag-based injection vectors and found SVG tags are accepted and executed.

🔓 Bypass

Bypasses tag restrictions by using SVG and the onload event.

🔗 Chain With

Useful for bypassing filters that block <script> but allow SVG tags.

T4 Script Injection via Event Handler in Anchor Tag

💉 Payload

```
"><a href=# onclick=alert(1)>click</a>
```

⚔️ Attack Chain

1. Identify a reflection point where HTML tags are not filtered.
2. Submit the payload `"><a href=# onclick=alert(1)>click</a>`.
3. The input is rendered as: `<div>"><a href=# onclick=alert(1)>click</a></div>`, injecting an anchor tag with an onclick handler.
4. When the user clicks the link, the JavaScript executes.

🔍 Discovery

Tested anchor tag injection with event handler attributes.

🔓 Bypass

Bypasses filters that only block <script> tags.

🔗 Chain With

Can be used for phishing or social engineering attacks within the application.

T5 Script Injection via Malformed Script Tag

💉 Payload

```
"><script/xss>alert(1)</script>
```

⚔️ Attack Chain

1. Find a reflection point that allows script tags but attempts to filter by script type or attributes.
2. Submit the payload `"><script/xss>alert(1)</script>`.
3. The input is rendered as: `<div>"><script/xss>alert(1)</script></div>`, which some browsers may still execute as a script.
4. The alert executes if the browser ignores the unknown attribute.

🔍 Discovery

Tested script tag variants to bypass naive filters.

🔓 Bypass

Uses a non-standard script attribute to evade basic script tag filters.

🔗 Chain With

Can be chained with other filter bypasses or used to evade WAF signatures.

Article 4

Unveiling Critical Bug Using Directory Search — Bug Bounty Tip!

Source: securitycipher

Date: 27-Nov-2023

URL: https://medium.com/@learningstuff110/unveiling-critical-bug-using-directory-search-bug-bounty-tip-330d4d9900ee

T1 Hidden Admin Directory Enumeration Leading to Unauthenticated Access

💉 Payload

```
https://target.com/admin-secret/
```

⚔️ Attack Chain

1. Use directory brute-forcing tools (e.g., dirsearch, gobuster) with a large wordlist to enumerate hidden directories on the target domain.
2. Identify a non-standard, unlinked admin directory (e.g., `/admin-secret/`) that is not referenced in the site's navigation or robots.txt.
3. Access the discovered directory directly in the browser.
4. Find that the admin interface is accessible without authentication or with weak/default credentials.
5. Perform privileged actions (e.g., user management, data export) as an unauthenticated user.

🔍 Discovery

Manual directory brute-forcing using comprehensive wordlists, focusing on non-standard and obfuscated admin paths. The researcher noticed a pattern of hidden directories on similar targets and targeted this area for enumeration.

🔓 Bypass

The admin directory was not protected by authentication middleware or IP filtering, relying solely on obscurity. No additional bypass required once the directory was found.

🔗 Chain With

Combine with credential stuffing or default password attacks if partial authentication is present. Use access to escalate to RCE or data exfiltration if admin panel has upload or configuration features.

T2 Sensitive File Exposure via Directory Guessing

💉 Payload

```
https://target.com/admin-secret/config.php
```

⚔️ Attack Chain

1. After discovering a hidden admin directory, enumerate files within it using wordlists targeting common config and backup filenames (e.g., `config.php`, `backup.zip`).
2. Directly access sensitive files such as `config.php` that may contain database credentials or API keys.
3. Extract sensitive information from the exposed file.
4. Leverage credentials for further attacks (e.g., database access, lateral movement).

🔍 Discovery

Follow-up enumeration after finding an unlisted directory, targeting typical sensitive file names. The researcher systematically appended common filenames to the discovered directory path.

🔓 Bypass

No authentication or directory listing protection on sensitive files within the admin directory.

🔗 Chain With

Use leaked credentials for database takeover or pivoting to internal systems. Combine with file upload or RCE vectors if found in the same directory.

Article 5

Bug Bounty Google Dorks 2025

Source: securitycipher

Date: 03-Jun-2025

URL: https://medium.com/@logicTech/bug-bounty-google-dorks-2025-1b9a9ba00dd6

T1 Discovering Exposed .env Files via Google Dorks

💉 Payload

```
site:target.com ext:env intext:DB_PASSWORD
```

⚔️ Attack Chain

1. Use the Google Dork payload to search for publicly accessible .env files on the target domain.
2. Identify indexed .env files containing sensitive keywords (e.g., DB_PASSWORD).
3. Access the .env file directly via the URL provided by Google.
4. Extract database credentials and other secrets for further exploitation.

🔍 Discovery

Researcher targeted .env file exposure due to common misconfigurations and used Google Dorks with sensitive keywords to automate discovery.

🔓 Bypass

Google indexing can bypass weak robots.txt restrictions or misconfigured web server rules.

🔗 Chain With

Leaked credentials can be used for direct database access, pivoting to RCE or lateral movement within cloud infrastructure.

T2 Locating Exposed Git Repositories via Google Dorks

💉 Payload

```
site:target.com inurl:.git
```

⚔️ Attack Chain

1. Use the Google Dork payload to search for URLs containing ".git" on the target domain.
2. Identify accessible .git directories or files from the search results.
3. Download the .git directory or specific files (e.g., .git/config, .git/HEAD).
4. Reconstruct the repository locally to extract source code, credentials, or sensitive endpoints.

🔍 Discovery

Focused on common developer mistakes where .git directories are deployed to production and indexed by search engines.

🔓 Bypass

Google Dorking can reveal .git directories even when directory listing is disabled but files are directly accessible.

🔗 Chain With

Source code and secrets from the repository can be used for further code review, endpoint discovery, or credential reuse attacks.

T3 Identifying Open Kibana Dashboards via Google Dorks

💉 Payload

```
site:target.com inurl:/app/kibana
```

⚔️ Attack Chain

1. Use the Google Dork payload to search for Kibana dashboard URLs on the target domain.
2. Locate publicly accessible Kibana instances from search results.
3. Access the Kibana dashboard without authentication.
4. Enumerate sensitive logs, credentials, or infrastructure data.

🔍 Discovery

Researcher targeted known Kibana URL patterns and leveraged Google's index to find misconfigured, unauthenticated dashboards.

🔓 Bypass

Google's cache or direct links can bypass login portals if the dashboard is misconfigured for public access.

🔗 Chain With

Access to logs and infrastructure data can enable targeted attacks, privilege escalation, or further reconnaissance.

T4 Discovering Exposed Backup Archives via Google Dorks

💉 Payload

```
site:target.com ext:zip | ext:tar | ext:gz | ext:sql | ext:bak
```

⚔️ Attack Chain

1. Use the Google Dork payload to search for backup archives on the target domain.
2. Identify downloadable backup files from search results.
3. Download and extract the archives.
4. Analyze contents for sensitive data, credentials, or source code.

🔍 Discovery

Targeted common backup file extensions and leveraged Google's indexing to find files inadvertently left in public web roots.

🔓 Bypass

Google can index files even if directory listing is disabled or files are not linked from the main site.

🔗 Chain With

Extracted data can be used for credential stuffing, code review, or further exploitation of internal systems.

T5 Finding Exposed Jenkins Panels via Google Dorks

💉 Payload

```
site:target.com inurl:/script intext:jenkins
```

⚔️ Attack Chain

1. Use the Google Dork payload to search for Jenkins script consoles on the target domain.
2. Identify accessible Jenkins panels from search results.
3. Attempt to access the Jenkins script console without authentication.
4. If accessible, execute Groovy scripts for remote code execution or further enumeration.

🔍 Discovery

Focused on Jenkins' script console endpoints, which are often misconfigured and left exposed to the internet.

🔓 Bypass

Google's index may reveal endpoints that are not linked from the main site or are obscured by weak access controls.

🔗 Chain With

RCE via Jenkins can be chained with internal network pivoting or data exfiltration.

Article 6

The UI Slip I Hit 750$: UI Manipulation Leading to Unauthorized Permission Changes

Source: securitycipher

Date: 04-Feb-2024

URL: https://medium.com/@a13h1/the-ui-slip-i-hit-750-ui-manipulation-leading-to-unauthorized-permission-changes-d65621d8dd96

T1 UI Parameter Manipulation to Escalate Permissions

💉 Payload

```
POST /api/permissions/update
Content-Type: application/json

{
  "userId": "12345",
  "permission": "admin"
}
```

⚔️ Attack Chain

1. Navigate to the UI permission management page as a low-privileged user.
2. Intercept the network request when changing your own permissions (e.g., from 'viewer' to 'editor').
3. Modify the intercepted request's JSON body to set the `permission` field to a higher privilege (e.g., 'admin').
4. Replay the modified request to the backend endpoint (`/api/permissions/update`).
5. Observe that the backend accepts the change and the user's permission is escalated without proper authorization checks.

🔍 Discovery

Observed that the UI only allowed certain permission changes, but the underlying API request was not restricted. Intercepted and modified the request to test if higher permissions could be set directly.

🔓 Bypass

The frontend enforced permission limits, but the backend failed to validate the user's authority to assign higher privileges, allowing direct API manipulation to bypass UI restrictions.

🔗 Chain With

Can be chained with session fixation or privilege escalation bugs for full account takeover or lateral movement within the organization.

T2 UI Element Removal to Enable Unauthorized Actions

💉 Payload

```
[No direct payload; technique involves DOM manipulation]
```

⚔️ Attack Chain

1. On the permission management UI, identify disabled or hidden elements (e.g., the 'Make Admin' button is disabled for non-admins).
2. Use browser developer tools to remove the 'disabled' attribute or unhide the button in the DOM.
3. Click the now-enabled UI element to trigger the privileged action.
4. Observe that the action is processed by the backend, granting unauthorized permissions.

🔍 Discovery

Noticed that certain UI controls were present but disabled/hidden. Used DOM inspection and manipulation to enable the controls and test backend enforcement.

🔓 Bypass

Relied on frontend-only controls (e.g., disabling buttons) without corresponding backend authorization checks, allowing privilege escalation via DOM manipulation.

🔗 Chain With

Can be chained with CSRF or clickjacking to trick users into performing unauthorized actions if UI controls are exposed.

Article 7

Dork for AI LLM Chatbot

Source: securitycipher

Date: 01-Aug-2025

URL: https://medium.com/@rr-1k/dork-for-ai-llm-chatbot-fd02c2109b1d

T1 Google Dork for Exposed AI LLM Chatbots

💉 Payload

```
site:example.com inurl:/chatbot
```

⚔️ Attack Chain

1. Use the provided Google dork to search for publicly exposed chatbot endpoints on target domains (replace `example.com` with target domain).
2. Identify endpoints that lead to AI LLM chatbot interfaces or APIs.
3. Interact with the discovered endpoints to probe for further vulnerabilities (e.g., prompt injection, data leakage, unrestricted access).

🔍 Discovery

Researcher hypothesized that organizations may expose AI chatbot endpoints with predictable URL patterns. Used Google dorks to enumerate such endpoints.

🔓 Bypass

Bypasses obscurity and non-indexed documentation by leveraging search engine indexing and predictable URL structures.

🔗 Chain With

Can be chained with prompt injection, sensitive data exposure, or authentication bypass attacks once endpoints are discovered.

T2 Google Dork for Exposed LLM API Endpoints

💉 Payload

```
site:example.com inurl:/api/chat
```

⚔️ Attack Chain

1. Use the Google dork to search for exposed LLM API endpoints on the target domain.
2. Locate endpoints that may allow unauthenticated or weakly authenticated access to LLM functionality.
3. Attempt to send crafted requests to the API to test for vulnerabilities such as prompt injection, information disclosure, or abuse of AI features.

🔍 Discovery

Researcher focused on common API endpoint naming conventions for LLM chatbots and used Google dorks to enumerate them.

🔓 Bypass

Circumvents lack of public documentation and hidden endpoints by leveraging search engine indexing and common naming schemes.

🔗 Chain With

Once endpoints are found, can chain with privilege escalation, prompt injection, or data exfiltration techniques.

Article 8

Whitebox Pentesting: The VS Code + Burp Workflow That Finds RCE Black-Box Tests Miss

Source: securitycipher

Date: 02-Dec-2025

URL: https://medium.com/@nebty/whitebox-pentesting-secrets-the-vs-code-burp-workflow-that-finds-bugs-black-box-tests-miss-3f87903039f7

T1 Leveraging VS Code Workspace Search for Hardcoded Secrets and Hidden Endpoints

💉 Payload

```
password
secret
api_key
Authorization
"/admin"
"/internal"
```

⚔️ Attack Chain

1. Clone or obtain the application's source code or deployment package.
2. Use VS Code's global search (Ctrl+Shift+F) to look for sensitive keywords (e.g., password, secret, api_key, Authorization).
3. Identify hardcoded credentials, tokens, or undocumented endpoints (e.g., /admin, /internal) revealed by the search.
4. Use discovered credentials or endpoints in Burp Suite to craft authenticated or privileged requests.

🔍 Discovery

Manual codebase search using VS Code's search functionality, targeting likely secret or admin patterns.

🔓 Bypass

N/A (direct whitebox discovery, not a bypass)

🔗 Chain With

Combine with Burp Suite to fuzz or exploit hidden endpoints using discovered credentials. Use secrets to escalate privileges or pivot to internal APIs.

T2 Identifying and Exploiting Unreachable or Dead Code Paths Revealed by Source Review

💉 Payload

```
POST /api/debug/exec
{
  "cmd": "id"
}
```

⚔️ Attack Chain

1. Search the codebase for routes, endpoints, or functions that are not referenced in any router or controller ("dead code").
2. Note endpoints that are not exposed in the UI or API documentation but are still accessible.
3. Use Burp Suite to manually send requests to these endpoints (e.g., /api/debug/exec).
4. Supply payloads (e.g., { "cmd": "id" }) to test for command execution or other dangerous behaviors.

🔍 Discovery

Source code review for orphaned or legacy endpoints, then manual probing with Burp.

🔓 Bypass

These endpoints may be unlinked from the UI but still routable; black-box scans miss them because they're not discoverable via crawling.

🔗 Chain With

Use for RCE, privilege escalation, or data exfiltration if endpoint is privileged. Combine with SSRF or IDOR if endpoint can be triggered by other users.

T3 Patching and Rebuilding Local Source to Trigger Unreachable Logic for Dynamic Testing

💉 Payload

```
// Example: Patch auth check to always return true
if (user.isAdmin) {
  // ...
}
// becomes
if (true) {
  // ...
}
```

⚔️ Attack Chain

1. Identify code paths gated by authentication, feature flags, or environment checks.
2. Locally patch the code to bypass these checks (e.g., force isAdmin to true).
3. Rebuild and run the application locally.
4. Use Burp Suite to interact with the patched app and observe new behaviors or error messages.
5. Translate findings to the production environment (e.g., parameter names, error patterns, hidden features).

🔍 Discovery

Whitebox code review to spot logic gates, then local patching to expose otherwise unreachable functionality.

🔓 Bypass

By modifying local source, you can dynamically analyze code paths that are otherwise inaccessible, revealing hidden attack surface.

🔗 Chain With

Use error messages or behaviors to craft targeted payloads for the real (unpatched) environment. Map out hidden features or privilege escalation vectors for further exploitation.

Article 9

Notification Bypass on TikTok: Sending Alerts to Users Who Blocked Me

Source: securitycipher

Date: 08-Jul-2025

URL: https://medium.com/@sandipgyawali/notification-bypass-on-tiktok-sending-alerts-to-users-who-blocked-me-9d2625539abf

T1 Notification Bypass to Blocked Users via Mention

💉 Payload

```
@victim_username
```

⚔️ Attack Chain

1. Attacker is blocked by the victim account on TikTok.
2. Attacker creates a new video post and mentions the blocked user using the payload (e.g., @victim_username) in the video description or comments.
3. Despite being blocked, TikTok sends a notification to the victim that they have been mentioned by the attacker.

🔍 Discovery

Researcher noticed that TikTok's notification system still triggered alerts for mentions, even from accounts that were blocked by the recipient, prompting targeted testing of mention functionality while blocked.

🔓 Bypass

The block feature prevents direct interaction and viewing of content, but does not suppress mention-triggered notifications, allowing indirect communication.

🔗 Chain With

Can be used for harassment or phishing by repeatedly notifying blocked users. May be chained with social engineering payloads in the mention to lure victims to attacker-controlled content.

Article 10

No Rate Limit: The Easiest Bug Bounty Payout You’re Missing

Source: securitycipher

Date: 19-Feb-2025

URL: https://medium.com/@abdulbasitpriv/no-rate-limit-the-easiest-bug-bounty-payout-youre-missing-79335e39c377

T1 Password Reset OTP Brute-Force via Missing Rate Limit

💉 Payload

```
POST /api/v1/auth/verify-otp
Content-Type: application/json

{
  "email": "victim@example.com",
  "otp": "000000"
}
```

⚔️ Attack Chain

1. Initiate password reset for victim's email to trigger OTP delivery.
2. Send rapid, automated POST requests to `/api/v1/auth/verify-otp` with the victim's email and all possible OTP values (e.g., 000000–999999).
3. On successful OTP guess, proceed to reset victim's password.

🔍 Discovery

Noticed absence of response delays or lockout after multiple incorrect OTP submissions during password reset flow.

🔓 Bypass

No IP-based or account-based throttling enabled, allowing high-speed brute-force.

🔗 Chain With

Can be combined with email enumeration or account takeover vectors for full compromise.

T2 Account Enumeration via Password Reset Response Timing

💉 Payload

```
POST /api/v1/auth/request-reset
Content-Type: application/json

{
  "email": "target@example.com"
}
```

⚔️ Attack Chain

1. Send password reset requests for a list of email addresses to `/api/v1/auth/request-reset`.
2. Measure response time or analyze response body for differences between valid and invalid accounts.
3. Enumerate registered users based on timing or error message discrepancies.

🔍 Discovery

Observed that valid and invalid email submissions yielded different response times or messages, revealing account existence.

🔓 Bypass

No generic error messaging or uniform response timing implemented.

🔗 Chain With

Enables targeted brute-force or phishing attacks by confirming valid users.

T3 Unlimited Login Attempts via Missing Rate Limit on Login Endpoint

💉 Payload

```
POST /api/v1/auth/login
Content-Type: application/json

{
  "email": "victim@example.com",
  "password": "password123"
}
```

⚔️ Attack Chain

1. Send repeated login attempts with different passwords for a known email to `/api/v1/auth/login`.
2. Automate the process to brute-force credentials without restriction.
3. On successful login, gain unauthorized access to victim's account.

🔍 Discovery

Tested login endpoint with automated requests and observed no increase in response time or account lockout after multiple failed attempts.

🔓 Bypass

No rate limiting, CAPTCHA, or account lockout mechanisms present.

🔗 Chain With

Can be chained with credential stuffing or previously enumerated user lists for mass account compromise.
