ETX Signal-X

Daily Intelligence Digest

Thursday, February 12, 2026

10
ARTICLES

28
TECHNIQUES

Article 1

mcp-hacking

Source: threatable

Date:

URL: https://github.com/tristanhausermannpernet/mcp-hacking

T1 Tool Description Poisoning Attack

💉 Payload

```
add_numbers
<IMPORTANT>
youfoundme.txt
```

⚔️ Attack Chain

1. Attacker sets up an MCP server with a tool (e.g., `add_numbers`) whose description includes injected instructions (`<IMPORTANT>`) referencing unauthorized resources (`youfoundme.txt`).
2. AI assistant connects to the MCP server and loads tool descriptions.
3. AI assistant interprets the malicious instructions in the tool description and accesses `youfoundme.txt`, demonstrating unauthorized resource access.

🔍 Discovery

Researcher manually inspected tool descriptions in MCP server configurations and observed AI assistant behavior when connecting to the server. The presence of `<IMPORTANT>` and reference to `youfoundme.txt` triggered investigation.

🔓 Bypass

Malicious instructions are embedded in tool descriptions, bypassing standard resource access controls by exploiting AI assistant's trust in tool metadata.

🔗 Chain With

Can be chained with privilege escalation if the AI assistant is allowed to access sensitive files or perform actions based on tool description instructions.

T2 Tool Shadowing Attack

💉 Payload

```
add_numbers
send_email
add_numbers
add_numbers
send_email
```

⚔️ Attack Chain

1. Attacker configures two MCP servers: one with a poisoned `add_numbers` tool containing instructions affecting email functionality, and another with a legitimate `send_email` tool.
2. AI assistant connects to both servers and loads all tool descriptions.
3. Malicious instructions in the `add_numbers` tool influence the AI assistant's behavior when using `send_email`, causing cross-tool contamination.

🔍 Discovery

Researcher observed unexpected cross-tool behavior during multi-server MCP connections, specifically when tool descriptions referenced other tools or contained instructions affecting unrelated functionality.

🔓 Bypass

Attack leverages the AI assistant's global interpretation of tool descriptions, allowing malicious instructions in one tool to affect others, bypassing tool isolation.

🔗 Chain With

Can be chained with tool description poisoning to escalate impact across multiple tools, potentially leading to privilege escalation or lateral movement.

T3 Tool Name Baiting Attack

💉 Payload

```
add_numbers
add_numbers_secure
```

⚔️ Attack Chain

1. Attacker sets up an MCP server with two tools: `add_numbers` and `add_numbers_secure`.
2. The tool names are crafted to influence AI assistant tool selection, with `add_numbers_secure` appearing more trustworthy.
3. AI assistant preferentially selects `add_numbers_secure`, which may contain malicious logic or instructions.

🔍 Discovery

Researcher tested AI assistant tool selection behavior with multiple tools whose names imply security or trustworthiness, observing selection bias.

🔓 Bypass

Attack exploits AI assistant's tool selection heuristics, bypassing user intent and potentially leading to execution of attacker-controlled logic.

🔗 Chain With

Can be chained with tool description poisoning or shadowing to maximize impact, as baited tools can carry malicious instructions or influence other tools.

Article 2

Unicode Normalization Leads to Account Takeover

Source: threatable

Date:

URL: https://omaralzughaibi.com/blog/nginx-off-by-slash-rce/

T1 Unicode Normalization Bypass for Account Takeover

💉 Payload

```
user%2F..%2Fadmin
```

⚔️ Attack Chain

1. Register a new account with a username containing a Unicode-encoded slash (`%2F`), such as `user%2F..%2Fadmin`.
2. The backend normalizes the username, interpreting `%2F` as `/`, resulting in a path traversal (`user/../admin`).
3. Upon login, the application resolves the normalized path, granting access to the `admin` account instead of the newly registered account.

🔍 Discovery

Researcher noticed that the application performed Unicode normalization on user input and suspected path traversal could be triggered via encoded slashes in usernames.

🔓 Bypass

The application failed to properly sanitize and restrict Unicode-encoded slashes, allowing traversal to privileged accounts. Normalization occurred after authentication logic, enabling the bypass.

🔗 Chain With

Can be chained with privilege escalation or session fixation attacks if the normalized path grants access to other sensitive accounts or resources.

T2 Nginx Off-By-Slash Remote Code Execution (RCE)

💉 Payload

```
/.%2e/;/bin/sh
```

⚔️ Attack Chain

1. Send a request to the Nginx server with the payload `/.%2e/;/bin/sh`.
2. Nginx interprets `%2e` as `.` and the slash as a directory separator, resulting in the path `/../;/bin/sh`.
3. The backend misinterprets the path, executing `/bin/sh` due to the off-by-slash parsing bug.
4. Remote code execution is triggered on the server.

🔍 Discovery

Researcher analyzed Nginx path normalization and discovered that encoded dot (`%2e`) and slash sequences could be abused to bypass path restrictions and trigger command execution.

🔓 Bypass

By combining encoded dot and slash, the attacker bypasses Nginx's path normalization and restriction logic, allowing arbitrary command execution.

🔗 Chain With

Can be chained with SSRF or LFI vulnerabilities to escalate from file access to full RCE.

Article 3

How LLMs Feed Your RE Habit: Following the Use-After-Free Trail in CLFS | clearbluejar

Source: threatable

Date:

URL: https://clearbluejar.github.io/posts/how-llms-feed-your-re-habit-following-the-uaf-trail-in-clfs/

T1 CLFS Use-After-Free via IRP Race Condition

💉 Payload

```
1. Open a log file handle via Win32 API (e.g., CreateLogFile).
2. Send a DeviceIoControl request (IOCTL) to clfs.sys using one of the vulnerable IOCTL cod
es (e.g., 0x8007a827, 0x8007a828, 0x80076832, 0x8007282c, 0x8007283e, 0x80076816).
3. While the IOCTL is being processed (IRP in progress), simultaneously call CloseHandle on
the log file handle to trigger IRP_MJ_CLEANUP.
```

⚔️ Attack Chain

1. Open a log file handle using CreateLogFile (or similar Win32 API).
2. Issue a DeviceIoControl call with a vulnerable IOCTL code (e.g., ReserveAndAppendLog: 0x8007a827/0x8007a828, ReadRestart: 0x80076832, Flush: 0x8007282c, StartArchival: 0x8007283e, ScanContainers: 0x80076816).
3. While the DeviceIoControl request is still being processed (IRP_MJ_DEVICE_IO_CONTROL), invoke CloseHandle on the same log file handle to trigger IRP_MJ_CLEANUP.
4. The cleanup path frees the FsContext2 structure (CClfsRequest::Cleanup → CClfsLogCcb::Cleanup → CClfsLogCcb::Release).
5. The IOCTL handler dereferences FsContext2 after it has been freed, resulting in a use-after-free condition in the kernel.

🔍 Discovery

Patch diffing CLFS driver binaries (clfs.sys) for CVE-2025-29824, reverse engineering function flows with Ghidra and pyghidra-mcp, and targeted LLM prompts to identify the precise timing window and dereference points of FsContext2. Manual and LLM-assisted analysis confirmed the race between IRP_MJ_CLEANUP and IRP_MJ_DEVICE_IO_CONTROL.

🔓 Bypass

The vulnerability relies on precise timing: the IRP_MJ_CLEANUP must free FsContext2 while another IRP (DeviceIoControl) is still referencing it. No authentication or permission checks prevent this race; any user-mode application with access to CLFS can trigger it.

🔗 Chain With

Kernel UAF primitives can be leveraged for privilege escalation, arbitrary kernel memory read/write, or code execution depending on the attacker's control of FsContext2 contents. Can be chained with other kernel vulnerabilities (e.g., info leaks, heap grooming) for reliable exploitation.

T2 Targeted IOCTL Mapping for UAF Trigger in CLFS

💉 Payload

```
DeviceIoControl(hLog, 0x8007a827, ...)
DeviceIoControl(hLog, 0x8007a828, ...)
DeviceIoControl(hLog, 0x80076832, ...)
DeviceIoControl(hLog, 0x8007282c, ...)
DeviceIoControl(hLog, 0x8007283e, ...)
DeviceIoControl(hLog, 0x80076816, ...)
```

⚔️ Attack Chain

1. Identify and open a CLFS log file handle (hLog).
2. Send DeviceIoControl requests using the above IOCTL codes, each mapped to a specific function dereferencing FsContext2:
3. Simultaneously invoke CloseHandle or trigger IRP_MJ_CLEANUP to free FsContext2.
4. Observe kernel crash or controlled UAF depending on payload and timing.

🔍 Discovery

LLM-assisted binary analysis (pyghidra-mcp + Ghidra) to enumerate all functions and IOCTL codes that dereference FsContext2. Cross-referencing IOCTL codes with function offsets and decompiled excerpts to identify all vulnerable entry points.

🔓 Bypass

Multiple IOCTL codes provide diverse entry points for triggering the UAF, increasing reliability and exploit surface. No restriction on user-mode access to these IOCTLs.

🔗 Chain With

Multiple IOCTLs allow for heap spraying, grooming, and variant exploitation strategies. Can be combined with other kernel bugs or info leaks for stable exploitation.

T3 Automated Function Discovery for UAF via LLM + pyghidra-mcp

💉 Payload

```
LLM prompt: "In what other functions is FsContext2 accessed that could cause the UAF? Look
specifically for FileObject->FsContext2."
pyghidra-mcp tool calls to enumerate cross-references and decompile functions:
- CClfsRequest::ScanContainers
- CClfsRequest::Flush
- CClfsRequest::StartArchival
- CClfsRequest::ReserveAndAppendLog
- ClfsReadRestartAreaInternal (via CClfsRequest::ReadRestart)
```

⚔️ Attack Chain

1. Load vulnerable CLFS binary into pyghidra-mcp.
2. Use LLM to prompt for cross-references to FsContext2.
3. pyghidra-mcp enumerates all functions dereferencing FsContext2 and maps them to IOCTL codes.
4. Identify all code paths where UAF can be triggered, not just those previously documented.
5. Use this expanded set for fuzzing, exploitation, or variant analysis.

🔍 Discovery

Automated conversational analysis with LLM and pyghidra-mcp, leveraging tool calls to cross-reference, decompile, and classify all FsContext2 usages. This method surfaced new and previously known vulnerable functions, compressing manual RE workflow into LLM-driven automation.

🔓 Bypass

Automated function discovery reduces reliance on manual binary spelunking, enabling rapid identification of variant UAF paths and increasing the attack surface.

🔗 Chain With

Enables mass variant hunting for similar UAFs in other drivers. Can be integrated into fuzzing frameworks or exploit development pipelines for kernel vulnerabilities.

Article 4

How I get an easy Blind SSRF by just reading writeups

Source: securitycipher

Date: 12-Jun-2024

URL: https://medium.com/@mohamed0xmuslim/how-i-get-an-easy-blind-ssrf-by-just-reading-writeups-a5459bbdf96d

T1 Blind SSRF via XML-RPC pingback.ping Method

💉 Payload

```
pingback.ping<-- burp collaborator link -->http://< Any valid page from the website >
```

⚔️ Attack Chain

1. Identify the presence of `/xmlrpc.php` endpoint on the target (e.g., `https://www.target.com/about-us/xmlrpc.php`).
2. Change the HTTP request method to POST and send to the endpoint.
3. In the request body, use `system.listMethods` to enumerate available XML-RPC methods.
4. Craft a POST request using the `pingback.ping` method, setting the first parameter as a Burp Collaborator link and the second as a valid page from the target site.
5. Observe the response; faultCode 0 indicates the server processed the request.
6. Check Burp Collaborator for incoming HTTP requests from the target server, confirming blind SSRF.

🔍 Discovery

Noticed `/xmlrpc.php` endpoint in a response while investigating cache bugs. Recalled prior writeups about XML-RPC vulnerabilities, specifically `pingback.ping` SSRF vectors.

🔓 Bypass

Utilizes XML-RPC's `pingback.ping` method, which is often overlooked and enabled by default on many WordPress installations. No authentication required; bypasses standard SSRF mitigations by leveraging XML-RPC protocol logic.

🔗 Chain With

Combine with port scanning or protocol interaction (e.g., Gopher) via SSRF payloads. Use for internal network discovery or to reach services behind reverse proxies. Potential for DoS or further exploitation if XML-RPC exposes additional methods.

Article 5

AutoGPT is a platform that allows users to create, deploy, and manage continuous artificial intelligence agents that automate complex workflows. Prior to autogpt-platform-beta-v0.6.34, in RSSFeedBlock, the third-party library urllib.request.urlopen is use

Source: threatable

Date:

URL: https://github.com/Significant-Gravitas/AutoGPT/security/advisories/GHSA-r55v-q5pc-j57f

T1 SSRF via Unfiltered RSS Feed URL Submission

💉 Payload

```
http://127.0.0.1:4321
```

⚔️ Attack Chain

1. Submit a malicious RSS feed URL (e.g., `http://127.0.0.1:4321`) to the `RSSFeedBlock` input parameter `rss_url`.
2. The backend code only validates the scheme (must be `http` or `https`) and passes the URL directly to `urllib.request.urlopen`.
3. The server issues a request to the attacker-controlled internal address, allowing SSRF against internal resources.
4. The response is parsed and returned, potentially leaking internal data or triggering internal services.

🔍 Discovery

Manual code review of `autogpt_platform/backend/backend/blocks/rss.py` revealed that user-supplied URLs are only checked for scheme and then passed to `urllib.request.urlopen` without further SSRF filtering.

🔓 Bypass

Other repository locations use a custom `Requests()` class with SSRF mitigation, but `RSSFeedBlock` does not. Attackers can target this block specifically to bypass global SSRF protections.

🔗 Chain With

SSRF can be chained with internal service discovery, port scanning, or leveraging exposed metadata endpoints if the internal network is accessible.

Article 6

Beyond the Login The Path Traversal Attack

Source: securitycipher

Date: 21-Aug-2024

URL: https://medium.com/@rajqureshi07/beyond-the-login-the-path-traversal-attack-30c1cfc09b3a

T1 Path Traversal via Login Endpoint

💉 Payload

```
../../../../etc/passwd
```

⚔️ Attack Chain

1. Identify a login endpoint that accepts a username parameter.
2. Submit the username parameter with a path traversal payload (e.g., `../../../../etc/passwd`).
3. Observe the server response for inclusion of sensitive file contents (such as `/etc/passwd`).

🔍 Discovery

The researcher noticed that the login endpoint did not sanitize input for the username field and tested classic path traversal payloads, resulting in file disclosure.

🔓 Bypass

No explicit bypass logic described, but the technique relies on lack of input sanitization and insufficient validation of file paths.

🔗 Chain With

Can be chained with authentication bypass or privilege escalation if sensitive files (e.g., credential stores) are disclosed.

T2 Path Traversal with Encoded Payloads

💉 Payload

```
..%2f..%2f..%2f..%2fetc%2fpasswd
```

⚔️ Attack Chain

1. Identify endpoints vulnerable to path traversal, especially those accepting user-controlled file paths.
2. Submit the payload using URL encoding for slashes and dots (e.g., `%2f` for `/`).
3. Observe if the application decodes the input and allows traversal, resulting in file disclosure.

🔍 Discovery

The researcher experimented with encoded payloads after classic traversal was blocked, discovering that URL encoding bypassed basic filters.

🔓 Bypass

URL encoding (`%2f`, `%2e`) bypasses naive blacklist/regex-based path validation.

🔗 Chain With

Can be combined with further encoding (double encoding) or chained with LFI/RFI for code execution if file inclusion is possible.

T3 Path Traversal via POST Request Body

💉 Payload

```
{"username":"../../../../etc/passwd","password":"test"}
```

⚔️ Attack Chain

1. Locate endpoints that accept JSON POST bodies for authentication or file access.
2. Craft a JSON payload with the traversal string in the username field.
3. Submit the POST request and analyze the response for file disclosure.

🔍 Discovery

The researcher shifted from GET to POST requests, placing traversal payloads in JSON fields, which were processed by the backend without proper sanitization.

🔓 Bypass

Placing traversal payloads in JSON fields bypasses input validation applied only to URL parameters.

🔗 Chain With

Can be chained with API endpoints that process JSON, potentially leading to broader file access or privilege escalation.

Article 7

Langroid is a framework for building large-language-model-powered applications. Prior to version 0.59.32, there is a bypass to the fix for CVE-2025-46724. TableChatAgent can call pandas_eval tool to evaluate the expression. There is a WAF in langroid/util

Source: threatable

Date:

URL: https://github.com/langroid/langroid/commit/30abbc1a854dee22fbd2f8b2f575dfdabdb603ea

T1 Dunder Attribute Access in Pandas Expression Evaluation

💉 Payload

```
df.__init__
df.__class__
df.__globals__
df.__builtins__
```

⚔️ Attack Chain

1. Submit a pandas expression to the TableChatAgent (or similar LLM-powered agent) that references dunder attributes of a DataFrame object (e.g., `df.__init__`).
2. The agent passes the expression to the `pandas_eval` tool for evaluation.
3. If the WAF does not block dunder attribute access, attacker can enumerate or interact with sensitive internals (e.g., class, globals, builtins).

🔍 Discovery

Review of test cases and code diffs revealed explicit checks for dunder attribute access, indicating prior bypass attempts and the need for defense.

🔓 Bypass

Prior to version 0.59.32, dunder attributes could be accessed directly in expressions, bypassing the original fix for CVE-2025-46724.

🔗 Chain With

Access to `__globals__` or `__builtins__` enables chaining to arbitrary code execution via eval or import.

T2 Private Attribute Access in Pandas Expression Evaluation

💉 Payload

```
df._private
df._internal_method()
```

⚔️ Attack Chain

1. Submit a pandas expression referencing a private attribute or method (single underscore prefix) of a DataFrame object.
2. The agent passes the expression to the `pandas_eval` tool for evaluation.
3. If the WAF does not block private attribute access, attacker can interact with internal state or methods.

🔍 Discovery

Test cases in the diff explicitly probe for private attribute/method access, revealing gaps in prior filtering.

🔓 Bypass

Private attributes were not blocked unless explicitly whitelisted, allowing attackers to access internal methods or state.

🔗 Chain With

If private methods are exploitable, could lead to privilege escalation or lateral movement within the application context.

T3 Dunder Attribute Access via Keyword Arguments (Bypass Vector)

💉 Payload

```
df.groupby(by=df.__init__)
df.groupby(by=df.__class__.__bases__)
```

⚔️ Attack Chain

1. Submit a pandas method call (e.g., `groupby`) with a keyword argument (`by`) referencing a dunder attribute.
2. The agent passes the expression to the `pandas_eval` tool for evaluation.
3. The WAF may only check top-level attribute access, missing dunder references in keyword arguments, allowing bypass.

🔍 Discovery

Test cases demonstrate that dunder attributes can be passed via kwargs, bypassing the initial attribute access filter.

🔓 Bypass

Dunder attribute checks were not applied recursively to keyword argument values, enabling the bypass.

🔗 Chain With

Combining with other pandas methods or chaining with eval/import enables deeper exploitation.

T4 Full PoC Exploit Chaining Dunder Access to Arbitrary Code Execution

💉 Payload

```
df.add_prefix("__import__('os').system('ls')#").T.groupby(by=df.__init__.__globals__['__builtins__']['eval'])
```

⚔️ Attack Chain

1. Craft a pandas expression that uses `add_prefix` to inject a malicious string, then transposes the DataFrame.
2. Use `groupby` with the `by` argument referencing `df.__init__.__globals__['__builtins__']['eval']`.
3. If dunder attribute access is not blocked, this chain enables access to eval and arbitrary code execution (e.g., running `os.system('ls')`).

🔍 Discovery

Test case includes a full exploit payload, demonstrating chaining of pandas methods and dunder attribute access to reach eval.

🔓 Bypass

Combines multiple bypass vectors (attribute access, kwargs) to reach builtins and eval.

🔗 Chain With

Direct path to RCE via eval and import; can be adapted for other payloads or chained with LLM prompt injection.

Article 8

Web Cache Poisoning — Part 2: Weaponizing Headers & URL Discrepancies

Source: securitycipher

Date: 31-Oct-2025

URL: https://medium.com/@Aacle/web-cache-poisoning-part-2-weaponizing-headers-url-discrepancies-bbb7b2c0159a

T1 Poisoning via X-Forwarded-Host Header Manipulation

💉 Payload

```
GET / HTTP/1.1
Host: target.com
X-Forwarded-Host: attacker.com
```

⚔️ Attack Chain

1. Identify endpoints where cache is enabled and the response reflects the Host header or related headers.
2. Send requests with a crafted `X-Forwarded-Host` header set to a malicious domain (e.g., attacker.com).
3. Observe if the response reflects the header value and gets cached.
4. Access the cached response as a victim, confirming the poisoned content is served.

🔍 Discovery

Manual fuzzing of headers on cache-enabled endpoints; noticed reflected header values in cached responses.

🔓 Bypass

If Host header is strictly validated, use `X-Forwarded-Host` to bypass checks and influence cache key or response content.

🔗 Chain With

Combine with open redirect or reflected XSS for full cache poisoning exploit.

T2 Poisoning via URL Parameter Discrepancies (Parameter Pollution)

💉 Payload

```
GET /page?utm_source=attacker&utm_source=victim HTTP/1.1
Host: target.com
```

⚔️ Attack Chain

1. Identify endpoints where URL parameters influence cache keys or response content.
2. Send requests with duplicate parameters (parameter pollution), placing attacker-controlled value first.
3. Observe if the response reflects attacker-controlled value and is cached.
4. Victim accesses the same endpoint, receives poisoned cache.

🔍 Discovery

Testing for parameter pollution on cache-enabled endpoints; observed cache key generation based on first parameter value.

🔓 Bypass

If application only validates one parameter instance, pollute with multiple to control cache key and content.

🔗 Chain With

Combine with reflected content vulnerabilities for persistent cache poisoning.

T3 Poisoning via Uncommon Headers (X-Original-URL)

💉 Payload

```
GET / HTTP/1.1
Host: target.com
X-Original-URL: /malicious
```

⚔️ Attack Chain

1. Identify cache-enabled endpoints that process uncommon headers like `X-Original-URL`.
2. Send requests with `X-Original-URL` set to a malicious path or value.
3. Observe if the response is influenced by the header and gets cached.
4. Victim accesses the endpoint and receives poisoned content from cache.

🔍 Discovery

Header fuzzing on cache-enabled endpoints; noticed response and cache behavior changed based on `X-Original-URL`.

🔓 Bypass

Use uncommon headers to bypass standard cache key logic or input validation.

🔗 Chain With

Chain with path traversal or open redirect for advanced cache poisoning.

T4 Cache Poisoning via Host Header/URL Mismatch

💉 Payload

```
GET /somepage HTTP/1.1
Host: attacker.com
```

⚔️ Attack Chain

1. Identify endpoints where the Host header and URL path are used inconsistently in cache key generation.
2. Send requests with mismatched Host header and URL path.
3. Observe if the cache stores the response based on the Host header, allowing attacker-controlled content to be served to victims.

🔍 Discovery

Manual testing for Host header/URL mismatches; observed cache key confusion leading to poisoned responses.

🔓 Bypass

Exploit inconsistent cache key logic by manipulating Host header and URL path.

🔗 Chain With

Combine with header injection or parameter pollution for multi-layer cache poisoning.

T5 Cache Poisoning via Header Order Manipulation

💉 Payload

```
GET / HTTP/1.1
Host: target.com
X-Forwarded-Host: attacker.com
X-Original-URL: /malicious
```

⚔️ Attack Chain

1. Identify cache-enabled endpoints sensitive to header order.
2. Send requests with headers in different orders, combining multiple uncommon headers.
3. Observe if cache key or response content changes based on header order, leading to poisoned cache.

🔍 Discovery

Systematic permutation of header order during fuzzing; noticed cache behavior changed with header order.

🔓 Bypass

Manipulate header order to bypass cache key normalization or validation logic.

🔗 Chain With

Chain with header value manipulation for complex cache poisoning scenarios.

Article 9

Vertical Privilege Escalation from Manager to Owner: A Bug Bounty Story

Source: securitycipher

Date: 31-Dec-2024

URL: https://medium.com/@swaroopvenkat828/vertical-privilege-escalation-from-manager-to-owner-a-bug-bounty-story-7a039eb0b938

T1 Vertical Privilege Escalation via Role Manipulation

💉 Payload

```
POST /api/user/updateRole
{
  "userId": "12345",
  "role": "owner"
}
```

⚔️ Attack Chain

1. Authenticate as a user with Manager role.
2. Intercept the request sent when updating user details.
3. Modify the "role" parameter in the request body from "manager" to "owner".
4. Send the modified request to the endpoint `/api/user/updateRole`.
5. Receive confirmation that the role has been updated to "owner".
6. Access owner-level functionalities previously restricted.

🔍 Discovery

Observed that the frontend allowed role selection only up to Manager, but the backend accepted arbitrary role values. Fuzzed the "role" parameter with higher privilege values and received successful responses.

🔓 Bypass

Frontend enforced role restrictions, but backend lacked validation, allowing direct privilege escalation by manipulating request payload.

🔗 Chain With

Combine with IDOR or other privilege escalation vectors to gain access to sensitive owner-level endpoints or data.

T2 Owner-Only Functionality Access via Modified API Requests

💉 Payload

```
POST /api/owner/settings/update
{
  "setting": "criticalConfig",
  "value": "newValue"
}
```

⚔️ Attack Chain

1. After escalating role to "owner" using Technique 1, identify owner-only API endpoints such as `/api/owner/settings/update`.
2. Craft and send requests to these endpoints with valid parameters.
3. Successfully update critical settings or configurations reserved for owners.

🔍 Discovery

Mapped API endpoints post-escalation and tested access to previously forbidden owner routes. Confirmed ability to modify critical settings.

🔓 Bypass

Role check is performed only on the backend after escalation. No additional authentication required once role is set to "owner".

🔗 Chain With

Potential to combine with configuration-based vulnerabilities or further abuse settings changes for lateral movement or persistence.

T3 Fuzzing Role Values for Hidden Privileges

💉 Payload

```
POST /api/user/updateRole
{
  "userId": "12345",
  "role": "admin"
}
```

⚔️ Attack Chain

1. Enumerate possible role values by fuzzing the "role" parameter (e.g., "admin", "superuser", "owner").
2. Send modified requests to `/api/user/updateRole` with each value.
3. Observe backend responses and resulting access changes.
4. Identify undocumented or hidden privilege levels.

🔍 Discovery

Used fuzzing techniques on the "role" parameter to discover additional privilege levels not exposed in the UI or documentation.

🔓 Bypass

Backend accepts arbitrary role values, exposing hidden privilege levels to attackers.

🔗 Chain With

Chain with privilege escalation and endpoint enumeration to discover and exploit undocumented admin functionalities.

Article 10

Broken Access Control: How I Viewed Admin-Only Configs as a Standard Employee

Source: securitycipher

Date: 23-Jan-2026

URL: https://medium.com/@mostafa23110006/broken-access-control-how-i-viewed-admin-only-configs-as-a-standard-employee-83f5a1ee6a08

T1 Direct Access to Admin-Only Configurations via Predictable Endpoint

💉 Payload

```
GET /admin/configs HTTP/1.1
Host: targetsite.com
Cookie: session=<standard_employee_session>
```

⚔️ Attack Chain

1. Authenticate as a standard employee user.
2. Identify the admin-only configuration endpoint (e.g., /admin/configs) via URL enumeration or application hints.
3. Send a direct GET request to the endpoint using the employee session cookie.
4. Receive admin-only configuration data without proper access control enforcement.

🔍 Discovery

Manual endpoint enumeration based on observed admin panel URLs and application structure. Noticed lack of access control checks on sensitive endpoints.

🔓 Bypass

No role-based access control (RBAC) or permission checks implemented on the /admin/configs endpoint. Standard employee session is treated as sufficient for access.

🔗 Chain With

Access to admin configuration data may allow privilege escalation, lateral movement, or further exploitation (e.g., modifying configs, extracting secrets).

T2 Accessing Admin-Only Configurations via Client-Side Hidden Links

💉 Payload

```
GET /admin/configs HTTP/1.1
Host: targetsite.com
Cookie: session=<standard_employee_session>
```

⚔️ Attack Chain

1. Log in as a standard employee.
2. Inspect client-side HTML or JavaScript for hidden or commented-out admin links (e.g., <a href="/admin/configs">).
3. Manually navigate to the hidden admin endpoint using the employee session.
4. Retrieve admin-only configuration data due to missing server-side access control.

🔍 Discovery

Source code review of client-side assets revealed hidden admin links not visible in the UI but accessible via direct navigation.

🔓 Bypass

Server-side access control is missing; client-side hiding is ineffective for security.

🔗 Chain With

Potential to chain with other exposed admin endpoints for broader access or configuration manipulation.

T3 Parameter Manipulation to Access Admin Configs

💉 Payload

```
GET /configs?role=admin HTTP/1.1
Host: targetsite.com
Cookie: session=<standard_employee_session>
```

⚔️ Attack Chain

1. Log in as a standard employee.
2. Identify role-based parameters in requests (e.g., role=admin).
3. Modify request parameters to escalate privileges (change role=employee to role=admin).
4. Send manipulated request and receive admin-only configuration data.

🔍 Discovery

Observed role parameters in network requests; tested manipulation to escalate access.

🔓 Bypass

Server trusts client-supplied role parameters without validation against session or user role.

🔗 Chain With

Can be combined with other parameter-based privilege escalation vectors or used to modify admin configurations if write access is possible.
