New issue

Stored XSS in Maxkb via Middleware HTML Injection #23

Open

AnalogyC0de opened last month Owner

Stored XSS in Maxkb via Middleware HTML Injection

Identification

- **Project:** Maxkb
- **Repository:** https://github.com/1Panel-dev/MaxKB
- **Affected Version/Commit:** <= v2.2.1

CVE Description

Maxkb is vulnerable to Stored Cross-Site Scripting (XSS) due to a lack of HTML escaping when processing application names and icons. An authenticated user can create an application with a malicious payload in the application name. When any user visits the public chat interface (`/ui/chat/{access_token}`), `StaticHeadersMiddleware` performs unescaped string replacement to inject the application data directly into the HTML response. This allows the attacker to break out of the `<title>` tag and execute arbitrary JavaScript in the victim's browser context.

Affected Component

- **File(s):** `apps/application/serializers/application_serializers.py` , `apps/common/middleware/static_headers_middleware.py`
- **Function / Method:** `to_application_model` , `process_response`
- **Entry Point:** `POST /api/application/` (`name` and `icon` parameters)

Reproduction Summary

1. An authenticated attacker sends a `POST` request to `/api/application/` to create an application, setting the `name` field to `</title><script>alert('XSS')</script><title>`.
2. A victim navigates to the public chat URL at `/ui/chat/<access_token>`.
3. The server's middleware directly inserts the payload into the HTML response's `<title>` tag.
4. The victim's browser renders the page and executes the injected `<script>` tag.

Technical Details

Entry Point (Write Phase - T0)

`apps/application/serializers/application_serializers.py:517-558`

```
@staticmethod
def to_application_model(user_id: str, application: Dict):
    return Application(
        id=uuid.uuid1(),
        name=application.get('name'),  # ← NO HTML ESCAPING
        desc=application.get('desc'),
        # ... other fields
    )
```

Consumption Point (Read Phase - T1)

`apps/common/cache_data/application_access_token_cache.py:20-27`

```
def get_application_access_token(access_token, use_get_data):
    application_access_token = QuerySet(ApplicationAccessToken).filter(
        access_token=access_token).first()
    if application_access_token is None:
        return None
    return {
        'white_active': application_access_token.white_active,
        'white_list': application_access_token.white_list,
        'application_icon': application_access_token.application.icon,  # ← FROM DB
        'application_name': application_access_token.application.name  # ← FROM DB
    }
```

Sink (HTML Injection)

`apps/common/middleware/static_headers_middleware.py:14-33`

```
def process_response(self, request, response):
    if request.path.startswith('/ui/chat/'):
        access_token = request.path.replace('/ui/chat/', '')
        application_access_token = get_application_access_token(access_token, True)
        if application_access_token is not None:
            white_active = application_access_token.get('white_active', False)
            white_list = application_access_token.get('white_list', [])
```

```
            application_icon = application_access_token.get('application_icon')
            application_name = application_access_token.get('application_name')

            # ... CSP header code ...

            response.content = (response.content.decode('utf-8').replace(
                '<link rel="icon" href="/ui/favicon.ico" />',
                f'<link rel="icon" href="{application_icon}" />')  # ← UNESCAPED
                .replace('<title>MaxKB</title>',
                        f'<title>{application_name}</title>')  # ← UNESCAPED
                .encode("utf-8"))
    return response
```

Proof of Concept - Step 1: Create Malicious Application

```
POST /api/application/ HTTP/1.1
Host: maxkb.example.com
Authorization: Bearer <attacker_token>
Content-Type: application/json

{
  "name": "</title><script>alert('XSS')</script><title>",
  "desc": "Malicious application",
  "dialogue_number": 0,
  "type": "SIMPLE",
  "dataset_setting": { ... },
  "model_setting": { ... },
  "problem_optimization": false
}
```

Proof of Concept - Step 2 & 3: Victim Visits Chat URL & HTML Response Contains

```
<link rel="icon" href="/ui/favicon.ico" />
<title></title><script>alert('XSS')</script><title></title>
```

Validation Notes

Reviewer Analysis & Code Verification

```
# File: apps/application/models/application.py
# Line 48

name = models.CharField(max_length=128, verbose_name="应用名称")
```

> The `name` field is a vanilla CharField with no custom save() method or preprocessing. Data is stored exactly as provided.

```
# File: apps/common/middleware/static_headers_middleware.py
# Lines 28-32

response.content = (response.content.decode('utf-8').replace(
    '<link rel="icon" href="/ui/favicon.ico" />',
    f'<link rel="icon" href="{application_icon}" />')  # ← UNESCAPED
    .replace('<title>MaxKB</title>',
             f'<title>{application_name}</title>')  # ← UNESCAPED
    .encode("utf-8"))
```

> **Critical Observation**: The sink is NOT a JSON API (which would be safe), but actual HTML content modification. This is a classic stored XSS vulnerability. The middleware decodes the HTML response, performs string replacement using f-string interpolation (which does NOT escape HTML), and re-encodes the response. The application name is directly inserted into the `<title>` tag without any HTML escaping. The payload `"</title><script>alert('XSS')</script><title>"` is syntactically correct.

shaohuzhang1 3 weeks ago

Fixed: 1Panel-dev/MaxKB#4919

Metadata

Assignees

No one assigned

Labels

No labels

Projects

No projects

Milestone

No milestone

Relationships

None yet

Development

Code with agent mode

No branches or pull requests

Participants