FuzzyLab

Nibby - Controle de Gastos

Sumário

Visão geral
Vulnerabilidades
Testes Realizados
    Foram realizados testes automatizados utilizando a ferramenta Snyk CLI, e testes manuais.
    1. Teste automatizado
    2. Testes manuais
        Open Redirect
        Self XSS
Conclusão

Visão geral

Foi realizada uma auditoria de segurança no sistema Nibby. Foram realizados testes nos modelos black-box e white-box, a fim de identificar vulnerabilidades e pontos fracos no sistema para que seja feita a mitigação desses pontos.

Vulnerabilidades

Vulnerabilidade	Descrição	Criticidade
Self-XSS	Permite que um atacante controle o javascript de uma página HTML. [1]	Baixa
Open Redirect	Permite que um atacante crie um link malicioso redirecionando a vítima para uma URL de sua escolha. [2]	Baixa

Testes Realizados

Foram realizados testes automatizados utilizando a ferramenta Snyk CLI, e testes manuais.

1. Teste automatizado

   O teste automatizado não identificou nenhuma vulnerabilidade no código-fonte do sistema:

```
→ nibby-controle-gastos-master snyk code test --org=eaf85080-b3c7-4d21-a99a-73d45af2cf62

Testing /home/art/Projects/fuzzylab/nibby/src/nibby-controle-gastos-master ...

 ✗ [Low] Use of Hardcoded Credentials
   Path: djangoapp/dashboard/tests/view_test/test_forms_views.py, line 23
   Info: Do not hardcode credentials in code. Found hardcoded credential used in username.

 ✗ [Low] Use of Hardcoded Credentials
   Path: djangoapp/dashboard/tests/view_test/test_forms_views.py, line 24
   Info: Do not hardcode credentials in code. Found hardcoded credential used in username.

 ✗ [Low] Use of Hardcoded Credentials
   Path: djangoapp/dashboard/tests/models_test/test_models_receitas.py, line 10
   Info: Do not hardcode credentials in code. Found hardcoded credential used in username.

 ✗ [Low] Use of Hardcoded Credentials
   Path: djangoapp/dashboard/tests/form_test/test_forms_dashboard_nova_receira_insert.py, line 10
   Info: Do not hardcode credentials in code. Found hardcoded credential used in username.

 ✗ [Low] Use of Hardcoded Credentials
   Path: djangoapp/dashboard/tests/form_test/test_forms_dashboard_nova_receira_insert.py, line 11
   Info: Do not hardcode credentials in code. Found hardcoded credential used in username.

 ✗ [Low] Use of Hardcoded Credentials
   Path: djangoapp/dashboard/tests/models_test/test_models_categorias.py, line 10
   Info: Do not hardcode credentials in code. Found hardcoded credential used in username.

 ✗ [Low] Use of Hardcoded Credentials
   Path: djangoapp/dashboard/tests/models_test/test_models_gastos.py, line 10
   Info: Do not hardcode credentials in code. Found hardcoded credential used in username.

 ✗ [Low] Use of Hardcoded Credentials
   Path: djangoapp/dashboard/tests/form_test/test_forms_dashboard_quantia_insert.py, line 10
   Info: Do not hardcode credentials in code. Found hardcoded credential used in username.


✓ Test completed

Organization:      eaf85080-b3c7-4d21-a99a-73d45af2cf62
Test type:         Static code analysis
Project path:      /home/art/Projects/fuzzylab/nibby/src/nibby-controle-gastos-master

Summary:

  8 Code issues found
  8 [Low]
```

(Apenas credenciais hardcoded em arquivos de testes, o que não representa uma ameaça)

2. Testes manuais

Nos testes manuais, foram identificados alguns pontos fracos que podem ser corrigidos, listados a seguir.

Open Redirect:

```
POST /accounts/login/?next={input} HTTP/2
Host: cloud.fuzzylab.tech
Cookie: csrftoken=bOuWnKTcCncjUXJpCTtAHlXl19X8wAG7


csrfmiddlewaretoken=T73xkWYunVD6nL0EZMTcbouzMfUEjOeGULnjxwHwP8Ff7yzTrvcCIzh
KDeHCFeKD&username={input}&password={input}


(parâmetro vulnerável: next)
```

Nos endpoints de Login e de Registro, é possível enviar um parâmetro “next” contendo uma URL onde o sistema irá redirecionar o usuário após efetuar a operação.

Se esse dado não for bem validado, permite que um atacante possa enviar uma URL para um usuário que após efetuar a operação no sistema, será redirecionado para um local arbitrário.

No caso do Nibby, o framework Django faz validações desse parâmetro baseado em configurações do desenvolvedor, onde na configuração atual, permite que o nibby redirecione usuários para qualquer sistema que rode no mesmo IP (89.116.225.21).

Esse comportamento pode fazer com que um usuário seja redirecionado para algum outro sistema hospedado na mesma VPS, que contenha alguma falha de XSS, permitindo que um atacante controle o navegador da vítima ou até mesmo roube a sessão da vítima.

Sugestão: Nas configurações do Django, definir rigidamente para quais hosts o sistema Nibby pode redirecionar seus usuários, e não deixar livre para todos no mesmo IP.

Self XSS

```
POST /categoria HTTP/2
Host: nibby.fuzzylab.tech
Cookie: csrftoken=twAIOXIS15Ljux5N9mkix7uJz9ULkpVo;
sessionid=jnnq054io0wy1xb2j6dbzk1m08hxyvv2


csrfmiddlewaretoken=T73xkWYunVD6nL0EZMTcbouzMfUEjOeGULnjxwHwP8Ff7yzTrvcCIzh
KDeHCFeKD&nome={input}

(parâmetro vulnerável: nome)
```

O usuário pode cadastrar uma nova categoria para seus gastos, porém, esse dado é usado para montar um javascript utilizado em /dashboard/, sem as devidas validações, permitindo que o javascript da página seja manipulado.

Porém, essa versão de XSS contém uma criticidade baixa, visto que o atacante consegue manipular o javascript apenas da própria página, não afetando outros usuários. De qualquer forma, é um ponto fraco importante de ser corrigido, pois pode ser encadeado com outras vulnerabilidades para gerar um impacto negativo significativo no sistema.

Nessa auditoria, não foram encontradas outras vulnerabilidades que possam ser encadeadas com essa, porém, com a evolução e manutenção do sistema, novas falhas podem ser inseridas, portanto, é de extremo interesse que esse ponto fraco seja mitigado para evitar que esse cenário ocorra no futuro do sistema.

Sugestão: Remoção de caracteres especiais ou encoding dos mesmos no momento em que esses valores serão usados para compor o javascript, ou no momento do input do usuário.

Conclusão

O sistema, apesar de simples, é bem seguro e possui uma superfície de ataque pequena.. O framework Django cuida de diversos aspectos da segurança, protegendo contra diversos ataques. "foram testados ataques como ClickJacking, porém, o Django por padrão define o header X-Frame-Options como Deny, impedindo iframes da página. A inserção de gastos e receita aceita apenas inputs numéricos, reduzindo drasticamente as possibilidades de ataques. O sistema de template é usado corretamente, evitando vulnerabilidades de template-injection. O ORM do Django evita por padrão ataques de SQL Injection, o WSGI valida todas as respostas, evitando CRLF Injection, a escolha do framework trouxe muita segurança para o sistema, os cuidados devem ser tomados quando alguma funcionalidade precisa de um código mais customizado e mais complexo.