Web Application Security

Security and Network Systems project presentation

C. Simões N. Hopf R. Gomes V. Dutra

MEEC-FEUP

May 2023

Objectives

- The world is exceedingly reliant on the Internet and web security is still a critical challenge in the corporate world.
- From Banking to E-Commerce, several applications today are high-value targets and cannot be prone to exploits

Understanding the concerns and best practices is fundamental for a modern web application developer

Methodology

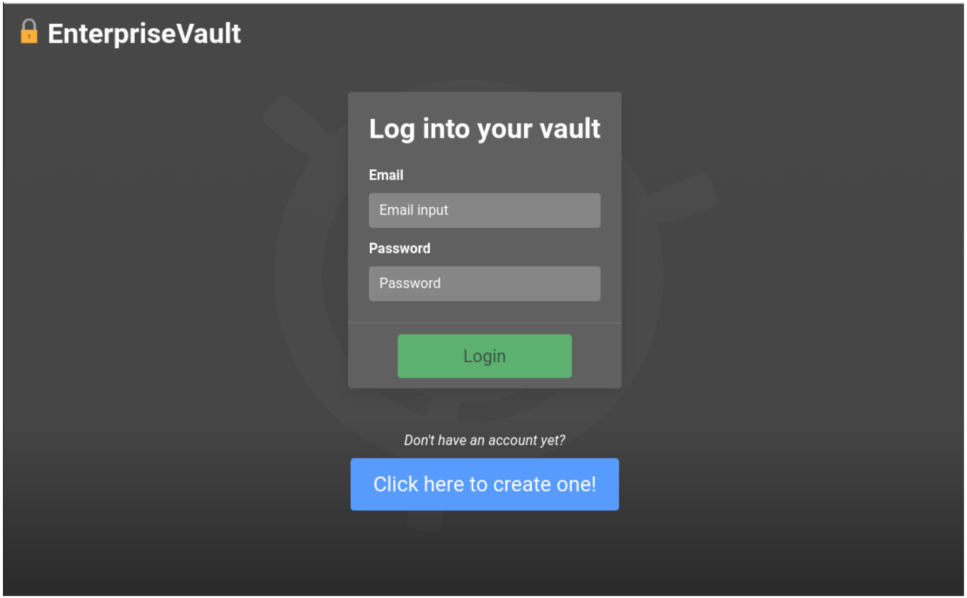
Since web security is an extensive domain, our first concern was defining a strict scope for this project:

1. Conceptualize a product that we will use as context for exploring the security procedures
2. Create a simple (vulnerable) implementation for the product
3. Exploit the vulnerabilities of the first application through unit tests
4. Modify the vulnerable application to make it resilient against all exploits (TDD)
5. Recreate the application using industry-standard solutions instead of relying on our own implementation

Concept

Create a **vault** in which visitors can:

- Register with a name, email and a password
- Log in with that email and password
- Upload an image and a secret phrase, when logged in
- See their image and secret phrase, when logged in
- Change (only) their image and secret phrase, when logged in
- See other people's names, but not their image or secret phrase


EnterpriseVault

Log into your vault

Email

Email input

Password

Password

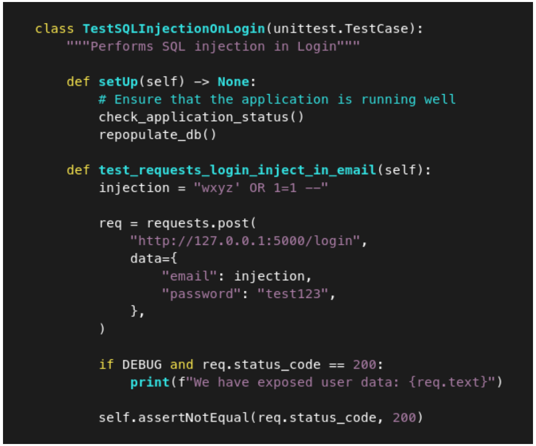
Login

Don't have an account yet?

Click here to create one!

Exploits

- SQL injection on sign up
- Dictionary attack on login
- XSS on sign up
- Image content exploit
- Cross-site request forgery
- Timing attacks
- Server access compromising



```python
class TestSQLInjectionOnLogin(unittest.TestCase):
    """Performs SQL injection in Login"""

    def setUp(self) -> None:
        # Ensure that the application is running well
        check_application_status()
        repopulate_db()

    def test_requests_login_inject_in_email(self):
        injection = "wxyz' OR 1=1 --"

        req = requests.post(
            "http://127.0.0.1:5000/login",
            data={
                "email": injection,
                "password": "test123",
            },
        )

        if DEBUG and req.status_code == 200:
            print(f"We have exposed user data: {req.text}")

        self.assertNotEqual(req.status_code, 200)
```

Security Fixes

- Templating in Flask and SQLite sanitization
- Do not allow users to input simple passwords and use fail2ban on the server
- Use the default Jinja2 template engine, as it escapes special characters such as <, > and /
- Image metadata is removed to prevent code execution in older browsers
- Create a single-use authentication process (no session cookie)
- Perform hashing and salting for every password and use fail2ban on the server
- Configure SSH keys and UFW (firewall)

Demonstração

Referências

1. L. Ma, D. Zhao, Y. Gao and C. Zhao, “Research on SQL Injection Attack and Prevention Technology Based on Web,” 2019 International Conference on Computer Network, Electronic and Automation (ICCNEA), Xi’an, China, 2019, pp. 176-179, doi: 10.1109/ICCNEA.2019.00042.
2. S. Kumar, R. Mahajan, N. Kumar and S. K. Khatri, “A study on web application security and detecting security vulnerabilities,” 2017 6th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), Noida, India, 2017, pp. 451-455, doi: 10.1109/ICRITO.2017.8342469.

3. K. Zhang, “A Machine Learning Based Approach to Identify SQL Injection Vulnerabilities,” 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), San Diego, CA, USA, 2019, pp. 1286-1288, doi: 10.1109/ASE.2019.00164.