Security testing Python Web applications

CYDWebPyTst3d | 3 days | On-site or online | Hands-on

Your Web application written in Python is tested functionally, so you are done, right? But did you consider feeding in incorrect values? 16Gbs of data? A null? An apostrophe? Negative numbers, or specifically -1 or -2^31? Because that's what the bad guys will do – and the list is far from complete.

Testing for security needs a remarkable software security expertise and a healthy level of paranoia, and this is what this course provides: a strong emotional engagement by lots of hands-on labs and stories from real life.

The curriculum goes through the common Web application security issues following the OWASP Top Ten but goes far beyond it both in coverage and the details.

A special focus is given to finding all discussed issues during testing, and an overview is provided on security testing methodology, techniques and tools.

So that you are prepared for the forces of the dark side.

So that nothing unexpected happens.

Nothing.

Cyber security skills and drills

27 LABS 8 CASE STUDIES

Audience

Python developers and testers working on Web applications

Group size

12 participants

Preparedness

General Python and Web development, testing and QA

Outline

- Cyber security basics
- The OWASP Top Ten 2021
- Security testing
- Wrap up

Standards and references

OWASP, CWE and Fortify Taxonomy

What you'll have learned

- Getting familiar with essential cyber security concepts
- Understanding Web application security issues
- Detailed analysis of the OWASP Top Ten elements
- Putting Web application security in the context of Python
- Going beyond the low hanging fruits
- Input validation approaches and principles
- Understanding security testing methodology and approaches
- Managing vulnerabilities in third party components
- Getting familiar with security testing techniques and tools

Table of contents

Day 1

- Injection attacks
- SQL injection
  - SQL injection basics
  - *Lab – SQL injection*
- SQL injection best practices
  - Input validation
  - Parameterized queries
  - *Lab – Using prepared statements*
  - *Case study – Hacking Fortnite accounts*
  - Testing for SQL injection
- Code injection
  - Code injection via input()
  - OS command injection
    - *Lab – Command injection*
    - OS command injection best practices
    - Avoiding command injection with the right APIs
    - *Lab – Command injection best practices*
      - *Case study – Shellshock*
    - *Lab - Shellshock*
    - Testing for command injection

Day 2

› The OWASP Top Ten 2021

A03 - Injection (continued)

- Input validation
  - Input validation principles
  - Denylists and allowlists
  - What to validate – the attack surface
  - Where to validate – defense in depth
  - When to validate – validation vs transformations
  - Output sanitization
  - Encoding challenges
  - Unicode challenges
  - *Lab – Encoding challenges*
  - Validation with regex
  - Regular expression denial of service (ReDoS)
  - *Lab – ReDoS in Python*
  - Dealing with ReDoS

- HTML injection - Cross-site scripting (XSS)
  - Cross-site scripting basics
  - Cross-site scripting types
    - Persistent cross-site scripting
    - Reflected cross-site scripting
    - Client-side (DOM-based) cross-site scripting
  - *Lab – Stored XSS*
  - *Lab – Reflected XSS*
  - *Case study – XSS in Fortnite accounts*
  - XSS protection best practices
    - Protection principles - escaping
    - XSS protection APIs in Python
    - XSS protection in Jinja2
    - *Lab – XSS fix / stored*
    - *Lab – XSS fix / reflected*
    - Client-side protection principles
    - Additional protection layers – defense in depth
  - Testing for XSS

› Security testing

Security testing vs functional testing

Manual and automated methods

Security testing methodology

- Security testing – goals and methodologies
- Overview of security testing processes
- Identifying and rating assets
  - Preparation
  - Identifying assets
  - Identifying the attack surface
  - Assigning security requirements
  - *Lab – Identifying and rating assets*
- Threat modeling
  - SDL threat modeling
  - Mapping STRIDE to DFD
  - DFD example
  - Attack trees
  - Attack tree example
  - *Lab – Crafting an attack tree*
  - Misuse cases
  - Misuse case examples

- Risk analysis
- *Lab – Risk analysis*
- Reporting, recommendations, and review

Day 3

› The OWASP Top Ten 2021

A04 - Insecure Design

- Client-side security
  - Frame sandboxing
    - Cross-Frame Scripting (XFS) attacks
    - *Lab - Clickjacking*
    - Clickjacking protection best practices
    - *Lab – Using CSP to prevent clickjacking*
  - Testing for client-side security weaknesses

A06 - Vulnerable and Outdated Components

- Using vulnerable components
- Untrusted functionality import
- Malicious packages in Python
- Vulnerability management
  - Patch management
  - Vulnerability management
  - Vulnerability databases

A07 - Identification and Authentication Failures

- Authentication
  - Authentication basics
  - Multi-factor authentication
  - *Case study – PayPal 2FA bypass*
- Testing for weak authentication
- Session management
  - Session management essentials
  - Why do we protect session IDs – Session hijacking
  - Session fixation
  - Session handling in Flask
  - Testing for session management issues
- Password management
  - Inbound password management
    - Storing account passwords

- Password in transit
- *Lab – Is just hashing passwords enough?*
- Dictionary attacks and brute forcing
- Salting
- Adaptive hash functions for password storage
- Password policy
- NIST authenticator requirements for memorized secrets
  - *Case study – The Ashley Madison data breach*
  - *The dictionary attack*
  - *The ultimate crack*
  - *Exploitation and the lessons learned*
- Testing for password management issues

A08 - Software and Data Integrity Failures

- Subresource integrity
  - Importing JavaScript
  - *Lab – Importing JavaScript*
  - *Case study – The British Airways data breach*
- Insecure deserialization
  - Serialization and deserialization challenges
  - Integrity – deserializing untrusted streams
  - Deserialization with pickle
  - *Lab – Deserializing with Pickle*
  - PyYAML deserialization challenges
  - Integrity – deserialization best practices
  - Testing for insecure deserialization

A10 - Server-Side Request Forgery (SSRF)

- Server-side Request Forgery (SSRF)
- *Case study – SSRF and the Capital One breach*

› Security testing

Security testing techniques and tools

- Code analysis
  - Static Application Security Testing (SAST)
  - *Lab – Using static analysis tools*
- Dynamic analysis
  - Security testing at runtime
  - Penetration testing
  - Stress testing
  - Dynamic analysis tools
    - Dynamic Application Security Testing (DAST)

- Web vulnerability scanners
  - *Lab – Using web vulnerability scanners*
- SQL injection tools
  - *Lab – Using SQL injection tools*
- Fuzzing

› Wrap up

Secure coding principles

- Principles of robust programming by Matt Bishop

And now what?

- Software security sources and further reading
- Python resources
- Security testing resources