Web application security in Python

CELPyWeb | 1 year subscription | e-Learning | Online VM

The course provides a comprehensive exploration of secure coding principles and practices tailored specifically for Python developers. Starting off from the foundations of cybersecurity, you will understand the consequences of insecure code by examining threats through the lens of the CIA triad.

In the main part of the material, you will systematically walk through the various vulnerabilities outlined in the OWASP Top Ten. As you progress through the modules investigating the intricacies of authentication and authorization, through realizing the practical aspects of cryptography, to tackling injection attacks, you will gain a deep understanding of both theoretical concepts and practical skills for securing Python web applications. Further subjects include error handling, code quality or denial of service, as well as XML and JSON security, and security considerations of the Python platform.

These modules go beyond just the theory. Not only do they identify vulnerabilities, show their consequences, and detail the best practices, but - through hands-on labs and real-world case studies - they offer practical experience in identifying, exploiting, and mitigating these security risks within Python-based web applications.

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

So that nothing unexpected happens in your code.

Nothing.

Note: This course content is available as an e-learning subscription. We reserve a period of 3 months to digest the foundational material, after which we activate shorter learning units on a monthly basis. This gives secure coding efforts an initial boost, and builds up sustained readiness over time. These learning units are indicated in red in the table of contents below.

Cyber security skills and drills

Foundational material		Monthly learning units	
48 LABS	30 CASE STUDIES	2-3 LABS	CASE STUDY

Audience

Python developers working on Web applications

Preparedness

General Python and Web development

Outline

- Cyber security basics
- The OWASP Top Ten
- A01 - Broken Access Control
- A02 - Security Misconfiguration
- A03 - Software Supply Chain Failures
- A04 - Cryptographic Failures
- A05 - Injection
- A06 - Insecure Design
- A07 - Authentication Failures
- A08 - Software or Data Integrity Failures
- A09 - Logging and Alerting Failures
- A10 - Mishandling of Exceptional Conditions
- X02 - Memory Management Failures
- Wrap up

Standards and references

OWASP, CWE and Fortify Taxonomy

What you'll have learned

- Getting familiar with essential cyber security concepts
- Identify Web application vulnerabilities and their consequences
- Learn the security best practices in Python
- Input validation approaches and principles
- 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

Table of contents

Horizontal authorization

- Authorization bypass through user-controlled keys

Case study – Remote takeover of Nexx garage doors and alarms

Lab – Horizontal authorization

File upload

- Unrestricted file upload
- Good practices

Lab – Unrestricted file upload

Cross-site Request Forgery (CSRF)

Lab – Cross-site Request Forgery

Cross-site Request Forgery (CSRF) best practices

- CSRF best practices
- CSRF defense in depth
- CSRF (XSRF) protection in Angular

Lab – CSRF protection with tokens

SSRF

- Server-side Request Forgery (SSRF)

Case study – SSRF in Ivanti Connect Secure

› A02 - Security Misconfiguration

Misconfiguration and XML parsing

- Configuration principles
- XML Entities
- DTD and the entities
- Entity expansion

XML External Entity (XXE)

- File inclusion with external entities
- Server-Side Request Forgery with external entities

Lab – External entity attack

Case study – XXE vulnerability in Ivanti products

XXE best practices

- Preventing XXE

Lab – Prohibiting DTD

Python platform security (Unit 5)

- The Python ecosystem and its attack surface

- Python bytecode and security
- Security features offered by Python
- PEP 578 and audit hooks
- The difficulties of sandboxing untrusted code
- Sandboxing Python

Web security configuration issues

- Content Security Policy
- Resource control
- Fetch directives
- Source allowlisting
- Strict CSP: using nonces and hashes
- Navigation and reporting directives
- Document restrictions and sandboxing
- Browser support
- CSP best practices
- Cookie attributes

Secrets management

- Hard coded passwords
- Best practices

Lab – Hardcoded password

A03 - Software Supply Chain Failures

Vulnerable components and dependencies

- Using vulnerable components
- Assessing the environment
- Hardening
- Untrusted functionality import
- Malicious packages in Python
- Supply chain security and the Software Bill of Materials (SBOM)
- SBOM examples

Case study – The Polyfill.io supply chain attack

Vulnerability management

- Patch management
- Vulnerability management
- Vulnerability databases
- Vulnerability rating – CVSS

Lab – Finding vulnerabilities in third-party components
- Bug bounty programs

Build security
- DevOps, the CI / CD build process and Software Composition Analysis
- Dependency checking in Python

Lab – Detecting vulnerable components

Dangerous and obsolete language elements
- Using dangerous language elements
- Using obsolete language elements
- Security aspects of monkey patching in Python
- Dangers of compile(), exec() and eval()

› A04 - Cryptographic Failures

Information exposure
- Exposure through extracted data and aggregation

Case study – Strava data exposure
- Leaking system information

Cryptography for developers
- Cryptography basics
- Cryptography in Python

Hashing
- Hashing basics
- Common hashing mistakes
- Hashing in Python

Lab – Hashing in Python

PRNG
- Random number generation
- Pseudo random number generators (PRNGs)
- Cryptographically secure PRNGs
- Using virtual random streams

Case study – Weak randomness in OpenVPN Access Server

PRNG in Python
- Weak PRNGs
- Using random numbers

Lab – Using random numbers in Python

Encryption

- Confidentiality protection
- Symmetric encryption
- Block ciphers
- Modes of operation
- Modes of operation and IV – best practices

Encryption in Python

- Symmetric encryption in Python
- *Lab – Symmetric encryption in Python*

Asymmetric encryption

- The RSA algorithm
- Using RSA – best practices
- RSA in Python
- *Case study – RSA attacks: Bleichenbacher, ROBOT, and Marvin*
- Combining symmetric and asymmetric algorithms

Key exchange and agreement

- Key exchange
- Diffie-Hellman key agreement algorithm
- Key exchange pitfalls and best practices

A05 - Injection

Injection problems

- Injection principles
- Injection attacks

SQL injection

- SQL injection basics
- *Lab – SQL injection*

SQL injection attack techniques

- Attack techniques
- Content-based blind SQL injection
- Time-based blind SQL injection

SQL injection best practices

- Input validation
- Parameterized queries
- *Lab – Using prepared statements*

SQL injection additional considerations

- Database defense in depth
- *Case study – SQL injection against US airport security*

Beyond SQL injection - ORM and NoSQL (Unit 6)

- SQL injection protection and ORM
- NoSQL injection basics

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*
- *Case study – Command injection in Ivanti security appliances*

HTML injection - Cross-site scripting (XSS)

- Cross-site scripting basics
- Persistent cross-site scripting
- Reflected cross-site scripting
- Client-side (DOM-based) cross-site scripting

XSS attacks

- *Lab – Stored XSS*
- *Lab – Reflected XSS*
- *Case study – XSS to RCE in Teltonika routers*

XSS best practices 1

- Protection principles - escaping
- XSS protection APIs in Python
- *Lab – XSS fix / stored*

XSS best practices 2

- XSS protection in Jinja2
- *Lab – XSS fix / reflected*
- Additional protection layers – defense in depth
- XSS protection in Angular
- XSS protection in React

Case study – XSS vulnerabilities in DrayTek Vigor routers

Template injection (Unit 6)

- Script injection

Case study – Script injection in Kubernetes NGINX Ingress

- Server-side template injection (SSTI)

Lab – Template injection

- Client-side template injection (CSTI) in Angular
- Client-side template injection (CSTI) in React

Input validation principles 1 (Unit 1)

- Input validation principles
- Denylists and allowlists

Case study – Denylist failure in urllib.parse.urlparse()

- Data validation techniques

Lab – Input validation

Input validation principles 2 (Unit 1)

- What to validate – the attack surface
- Where to validate – defense in depth
- When to validate – validation vs transformations

Input validation principles 3 (Unit 1)

- Output sanitization
- Encoding challenges
- Unicode challenges

Lab – Encoding challenges

Lab – Dealing with Unicode homoglyph attacks

- Validation with regex

Path traversal and file validation (Unit 3)

- Path traversal

Lab – Path traversal

- Path traversal-related examples
- Additional challenges in Windows

Case study – File spoofing in WinRAR

- Virtual resources
- Path traversal best practices

Lab – Path canonicalization

Native code (CFFI) issues (Unit 4)

- Native code dependence
- *Lab – Unsafe native code*
- Best practices for dealing with native code

A06 - Insecure Design

Insecure design

- The STRIDE model of threats
- Secure design principles of Saltzer and Schroeder

Insecure design - Saltzer and Schroeder 1

- Economy of mechanism
- Fail-safe defaults
- Complete mediation
- Open design

Insecure design - Saltzer and Schroeder 2

- Separation of privilege
- Least privilege
- Least common mechanism
- Psychological acceptability

Client-side security (Unit 5)

- Same Origin Policy
- Simple request
- Preflight request
- Cross-Origin Resource Sharing (CORS)
- *Lab – Same-origin policy demo*

Clickjacking

- Frame sandboxing
- Cross-Frame Scripting (XFS) attacks
- *Lab – Clickjacking*
- Clickjacking beyond hijacking a click

Anti-clickjacking best practices

- Clickjacking protection best practices
- *Lab – Using CSP to prevent clickjacking*

› A07 - Authentication Failures

Authentication

- Authentication basics
- Multi-factor authentication (MFA)
- *Case study – The InfinityGauntlet attack*
- Authentication weaknesses
- *Case study – Bypassing authentication on Ivanti security appliances*

Session security

- Session management essentials
- Why do we protect session IDs – Session hijacking
- Session fixation
- Session invalidation
- Session ID best practices
- Session handling in Flask
- Session handling security considerations in single-page applications

Password management

- Storing account passwords
- Password in transit
- *Lab – Is just hashing passwords enough?*

Password storage

- Dictionary attacks and brute forcing
- Salting
- Adaptive hash functions for password storage
- *Lab – Using adaptive hash functions in Python*

Password policy

- NIST authenticator requirements for memorized secrets

Password storage - a case study

- *Case study – The Ashley Madison data breach*
- *The dictionary attack*
- *The ultimate crack*
- *Exploitation and the lessons learned*

Additional password management challenges

- Password database migration
- (Mis)handling None passwords

Password auditing (Unit 8)

- Using password cracking tools
- Password cracking in Windows
- *Lab – Password audit with John the Ripper*
- *Lab – On-line password brute forcing*
- Password recovery issues
- Password recovery best practices
- *Lab – Password reset weakness*
- *Case study – GitLab account takeover*

Protecting secrets in memory (Unit 8)

- Challenges in protecting memory
- *Case study – Microsoft secret key theft via dump files*

› A08 – Software or Data Integrity Failures

Integrity protection and MAC

- Integrity protection
- Message Authentication Code (MAC)
- Calculating HMAC in Python
- *Lab – Calculating MAC in Python*

Digital signatures

- Digital signature
- Digital signature with RSA
- ECC basics
- Digital signature with ECC
- Digital signature in Python
- *Lab – Digital signature with ECDSA in Python*

Subresource integrity

- Importing JavaScript
- *Lab – Importing JavaScript*
- Subresource integrity in Angular
- *Case study – The British Airways data breach*

Insecure deserialization

- Serialization and deserialization challenges
- Integrity – deserializing untrusted streams
- Deserialization with pickle
- *Lab – Deserializing with Pickle*

- *Case study – The security of the machine learning supply chain*
- *Case study – The first wave of supply chain attacks: RCE via pickle (2022)*
- *Case study – Compromising the Hugging Face Hub repository*
- PyYAML deserialization challenges
- Integrity – deserialization best practices

A09 - Logging and Alerting Failures

Logging

- Logging and monitoring principles
- Insufficient logging
- *Case study – Plaintext passwords at Facebook*
- Logging best practices

Log forging (Unit 7)

- Newline injection
- Log forging
- Web log forging
- *Lab – Log forging*
- Log forging – best practices

Monitoring (Unit 7)

- Monitoring best practices
- Firewalls and Web Application Firewalls (WAF)
- Intrusion detection and prevention
- *Case study – The Marriott Starwood data breach*

A10 - Mishandling of Exceptional Conditions

Principles

- Error and exception handling principles
- Information exposure through error reporting
- Information leakage via error pages
- *Lab – Flask information leakage*
- Returning a misleading status code

Exception handling

- In the except block. And now what?
- Empty except block
- *Lab – Exception handling mess*

Control flow and error handling

- Incorrect block delimitation
- Dead code
- Using if-then-else and switch defensively

› X01 – Lack of Application Resilience (Unit 9)

Denial of service

- Flooding
- Resource exhaustion

Sustained client engagement

- Infinite loop
- Economic Denial of Sustainability (EDoS)

Amplification

- Some amplification examples

Algorithmic complexity issues

- Regular expression denial of service (ReDoS)
- *Lab – ReDoS*
- Dealing with ReDoS
- *Case study – ReDoS vulnerabilities in Python*

› X02 – Memory Management Failures

Integer handling problems (Unit 2)

- Representing signed numbers
- Integer visualization
- Integers in Python
- Integer overflow
- Integer overflows in ctypes and numpy
- *Lab – Integer problems in Python*
- Working with floating-point numbers

Race conditions (Unit 4)

- Time and state
- Race conditions
- Time of check to time of usage – TOCTTOU
- TOCTTOU attacks in practice
- *Lab - TOCTTOU*

Case study – Arbitrary file deletion via TOCTTOU in N-Able Agent
- Insecure temporary file
- Thread safety and the Global Interpreter Lock (GIL)
- Avoiding race conditions in Python

Case study: TOCTTOU in Calamares (CVE-2019-13178)

Locking and deadlocks (Unit 4)
- Mutual exclusion and locking
- Deadlocks
- Synchronization and thread safety

› Wrap up

Software security principles
- Principles of robust programming by Matt Bishop

Sources and further readings
- Software security sources and further reading
- Python resources