

Cloud application security in Python for AWS

CYDPyAWSCld5d | 5 days | On-site or online | Hands-on

Your cloud application written in Python works as intended, 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.

The cloud has become a critical aspect of online services. No matter which model you're using (SaaS, PaaS, IaaS), part of your service is now operated by someone else. This may look like a net positive, but it also greatly expands the attack surface and brings in several new risks that may not be obvious. Have you configured all security settings correctly? Are you prepared for supply chain attacks? How can you protect the confidentiality of user data in the cloud? And most importantly: can the bad guys use your exposure to their advantage?

Handling security needs 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, all to substantially improve code hygiene. Mistakes, consequences, and best practices are our blood, sweat and tears.

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.

All this is put in the context of Python, and extended by core programming issues, discussing security pitfalls of the programming language and the AWS cloud platform.

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

So that nothing unexpected happens.

Nothing.

Cyber security skills and drills



35 LABS



16 CASE STUDIES

Audience

Python developers working on Web applications and AWS

Group size

12 participants

Preparedness

General Python and Web development

Outline

- Cyber security basics
- The OWASP Top Ten 2021
- Cloud security
- Input validation
- Wrap up

Standards and references

OWASP, CWE and Fortify Taxonomy

What you'll have learned

- Understand cloud security specialties
- Getting familiar with essential cyber security concepts
- Understanding how cryptography supports security
- Learning how to use cryptographic APIs correctly in Python
- 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
- Managing vulnerabilities in third party components
- Learn to deal with cloud infrastructure security
- Input validation approaches and principles
- Identify vulnerabilities and their consequences
- Learn the security best practices in Python

Table of contents

Day 1

› Cyber security basics

What is security?

Threat and risk

Cyber security threat types – the CIA triad

Cyber security threat types – the STRIDE model

Consequences of insecure software

Cloud security basics

- Cloud infrastructure basics
- The Cloud Cube Model and Zero Trust Architecture

› The OWASP Top Ten 2021

The OWASP Top 10 2021

A01 – Broken Access Control

- Access control basics
- Failure to restrict URL access
- Confused deputy
 - Insecure direct object reference (IDOR)
 *Lab – Insecure Direct Object Reference*
 - Authorization bypass through user-controlled keys
 *Case study – Authorization bypass on Facebook*
 *Lab – Horizontal authorization*
- File upload
 - Unrestricted file upload
 - Good practices
 *Lab – Unrestricted file upload*
- Open redirects and forwards
 -  *Case study – Unvalidated redirect at Epic Games*
 - Open redirects and forwards – best practices
- [Cross-site Request Forgery \(CSRF\)](#)
 *Lab – Cross-site Request Forgery*

- CSRF best practices
- CSRF defense in depth
- 🔗 *Lab – CSRF protection with tokens*

A02 – Cryptographic Failures

- Information exposure
 - Exposure through extracted data and aggregation
 - 📖 *Case study – Strava data exposure*
- Cryptography for developers
 - Cryptography basics
 - Cryptography in Python
 - Elementary algorithms
 - Random number generation
 - Pseudo random number generators (PRNGs)
 - Cryptographically strong PRNGs
 - Using virtual random streams
 - Weak PRNGs
 - Using random numbers
 - 🔗 *Lab – Using random numbers in Python*
 - 📖 *Case study – Equifax credit account freeze*
 - Hashing
 - Hashing basics
 - Hashing in Python
 - 🔗 *Lab – Hashing in Python*

Day 2

› The OWASP Top Ten 2021

A02 – Cryptographic Failures (continued)

- Cryptography for developers
 - Confidentiality protection
 - Symmetric encryption
 - [Block ciphers](#)
 - Modes of operation
 - Modes of operation and IV – best practices
 - Symmetric encryption in Python
 - 🔗 *Lab – Symmetric encryption in Python*
 - Asymmetric encryption
 - The RSA algorithm
 - Using RSA – best practices
 - RSA in Python

- Combining symmetric and asymmetric algorithms
- Some further key management challenges
- Certificates
 - Certificates and PKI
 - X.509 certificates
 - Chain of trust
 - PKI actors and procedures
 - PGP – Web of Trust
 - Certificate revocation
- Transport security
 - Transport security weaknesses
 - The TLS protocol
 - TLS basics
 - TLS features (changes in v1.3)
 - The handshake in a nutshell (v1.3)
 - TLS best practices
 - 🔗 *Lab – Using a secure socket in Python*
 - HTTP Strict Transport Security (HSTS)

A03 – Injection

- Injection principles
- Injection attacks
- [SQL injection](#)
 - SQL injection basics
 - 🔗 *Lab – SQL injection*
 - Attack techniques
 - Content-based blind SQL injection
 - Time-based blind SQL injection
- NoSQL injection
 - NoSQL injection basics
 - NoSQL injection in MongoDB
 - NoSQL injection in DynamoDB
- SQL injection best practices
 - Input validation
 - Parameterized queries
 - 🔗 *Lab – Using prepared statements*
 - Additional considerations
 - 📖 *Case study – Hacking Fortnite accounts*
- 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*

Day 3

› The OWASP Top Ten 2021

A03 – Injection (continued)

- 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*
 - Additional protection layers – defense in depth

A04 – Insecure Design

- The STRIDE model of threats
- Secure design principles of Saltzer and Schroeder
 - Economy of mechanism
 - Fail-safe defaults
 - Complete mediation
 - Open design
 - Separation of privilege
 - Least privilege
 - Least common mechanism
 - Psychological acceptability
- Client-side security

- Frame sandboxing
 - Cross-Frame Scripting (XFS) attacks
 -  *Lab - Clickjacking*
 - Clickjacking protection best practices
 -  *Lab - Using CSP to prevent clickjacking*

A05 - Security Misconfiguration

- Cookie security
 - Cookie attributes
- XML entities
 - DTD and the entities
 - Entity expansion
 -  *Lab - Billion laughs attack*
 - External Entity Attack (XXE)
 - File inclusion with external entities
 - Server-Side Request Forgery with external entities
 -  *Lab - External entity attack*
 -  *Case study - XXE vulnerability in SAP Store*
 - Preventing XXE
 -  *Lab - Prohibiting DTD*

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*
- Session management
 - Session management essentials
 - Why do we protect session IDs - Session hijacking
 - Session fixation
 - Session handling in Flask
- Single sign-on (SSO)
 - Single sign-on concept
 - OAuth 2.0

- OAuth 2.0 basics
- OAuth 2.0 prerequisites
- OAuth 2.0 flow for Authorization Code Grant – Step #1
- OAuth 2.0 flow for Authorization Code Grant – Step #2
- OAuth 2.0 flow for Authorization Code Grant – Step #3
- OAuth 2.0 flow for Authorization Code Grant – Step #4
- OAuth 2.0 flow for Authorization Code Grant – Step #5
- OAuth 2.0 flow for Authorization Code Grant – Step #6
- OAuth 2.0 flow for Authorization Code Grant – Step #7
- OAuth 2.0 grant types
- Best practices
 - 📖 *Case study – Stealing SSO tokens from Epic Games accounts*
- SAML
 - SAML basics
 - SAML profiles
 - SAML security
- Identity and access management (IAM)
 - Identity and access management in AWS
 - Groups, roles and credentials
 - Access tokens

Day 4

› The OWASP Top Ten 2021

A07 – Identification and Authentication Failures (continued)

- 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](#)
 - Password hardening
 - Using passphrases
 - 📖 *Case study – The Ashley Madison data breach*
 - 📖 *The dictionary attack*
 - 📖 *The ultimate crack*
 - 📖 *Exploitation and the lessons learned*

- Password database migration
- (Mis)handling None passwords
- Outbound password management
 - Hard coded passwords
 - Best practices
 - 🔗 *Lab – Hardcoded password*
 - Protecting sensitive information in memory
 - Challenges in protecting memory

A08 – Software and Data Integrity Failures

- Integrity protection
 - Message Authentication Code (MAC)
 - MAC in Python
 - 🔗 *Lab – Calculating MAC in Python*
 - Digital signature
 - Digital signature with RSA
 - Elliptic Curve Cryptography
 - ECC basics
 - Digital signature with ECC
 - Digital signature in Python
 - 🔗 *Lab – Digital signature with ECDSA in Python*
- 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

A09 – Security Logging and Monitoring Failures

- Logging and monitoring principles
- Insufficient logging
- 📖 *Case study – Plaintext passwords at Facebook*
- Logging best practices
- Monitoring best practices
- Detection and monitoring
 - Utilizing AWS monitoring for security
 - Protecting logs

- The AWS Security Hub

A10 – Server-Side Request Forgery (SSRF)

- Server-side Request Forgery (SSRF)

 *Case study – SSRF and the Capital One breach*

> Cloud security

AWS security

- Security considerations
 - AWS and security
 - The AWS shared responsibility model
 - AWS cloud compliance
 - AWS hardening
 - Security tools for AWS

Container security

- Container security concerns
- Containerization, virtualization and security
- The attack surface
- Docker security
 - Docker and security
 - Common Docker security mistakes
 - Docker security best practices
-  *Lab – Static analysis of Docker images*
 - Hardening Docker
- Kubernetes security
 - The Kubernetes architecture and security
 - Securing Kubernetes hosts
 - Best practices for Kubernetes access control
 - Building secure Kubernetes images
 - Secure deployment of Kubernetes containers
 - Protecting Kubernetes deployments at runtime

Day 5

> Cloud security

Data security in the cloud

- Data confidentiality and integrity in the cloud

- Data privacy in the cloud
- Compliance considerations
- Data security in AWS
- Policies
- Storing cryptographic keys
- Protecting data at rest
- Protecting data in transit
- JSON security
 - JSON validation
 - JSON injection
 - Dangers of JSONP
 - JSON/JavaScript hijacking
 - Best practices
- 📖 *Case study – ReactJS vulnerability in HackerOne*

› The OWASP Top Ten 2021

Web application security beyond the Top Ten

- Code quality
 - Language elements
 - Using dangerous language elements
 - Using obsolete language elements
 - Portability flaw
 - Module injection and monkey patching
 - Dangers of `compile()`, `exec()` and `eval()`
 - The difficulties of sandboxing untrusted code
 - Object oriented programming pitfalls
 - Accessibility modifiers
 - Private attributes and name mangling
 - Multiple inheritance and security
 - Mutability
- Denial of service
 - Flooding
 - Resource exhaustion
 - Sustained client engagement
 - Infinite loop
 - Economic Denial of Sustainability (EDoS)
 - Algorithm complexity issues
 - Regular expression denial of service (ReDoS)
 - 📖 *Lab – ReDoS in Python*
 - Dealing with ReDoS

› Input validation

Input validation principles

Denylist and allowlist

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

Integer handling problems

- Representing signed numbers
- Integer visualization
- Integers in Python
- Integer overflow
- Integer overflows in ctypes and numpy

Files and streams

- Path traversal

Lab – Path traversal

- Path traversal-related examples
- Additional challenges in Windows
- Virtual resources
- Path traversal best practices

Lab – Path canonicalization

Format string issues

Unsafe native code

- Native code dependence

Lab – Unsafe native code

- Best practices for dealing with native code

› Wrap up

Secure coding principles

- Principles of robust programming by Matt Bishop

And now what?

- Software security sources and further reading
- Python resources