

**Python Security**

**Course Number:** SEC-132
**Duration:** 3 days

**Overview**

This Python Security training class teaches attendees how to develop secure Python applications. Participants learn how to identify vulnerabilities and code with Python security best practices in mind to mitigate threats.

**Note:** To ensure ample one-on-one engagement with the instructor, this class is capped at 12 people, overriding Accelebrate’s default cap of 15.

**Prerequisites**

All Python Security training students must have general Python development experience.

**Materials**

All attendees receive comprehensive courseware.

**Software Needed on Each Student PC**

Attendees will not need to install any software on their computer for this class. The class will be conducted in a remote environment that Accelebrate will provide; students will only need a local computer with a web browser and a stable Internet connection. Any recent version of Microsoft Edge, Mozilla Firefox, or Google Chrome will be fine.

**Objectives**

* Understand essential cyber security concepts
* Work with input validation approaches and principles
* Identify vulnerabilities and their consequences
* Incorporate security best practices in Python
* Manage vulnerabilities in third-party components
* Understand how cryptography supports security
* Use cryptographic APIs correctly in Python

**Outline**

* Introduction
* 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
* 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
	+ Validation with regex
	+ Regular expression denial of service (ReDoS)
	+ ReDoS in Python
	+ Dealing with ReDoS
	+ Injection
		- Injection principles
		- Injection attacks
		- SQL injection
		- SQL injection best practices
		- Code injection
	+ Integer handling problems
		- Representing signed numbers
		- Integer visualization
		- Integers in Python
		- Integer overflow
		- Integer overflows in ctypes and numpy
		- Other numeric problems
	+ Files and streams
		- Path traversal
		- Path traversal-related examples
		- Additional challenges in Windows
		- Virtual resources
		- Path traversal best practices
		- Path canonicalization
	+ Format string issues
	+ Unsafe native code
		- Native code dependence
		- Lab – Unsafe native code
		- Best practices for dealing with native code
* Security Features
	+ Authentication
		- Authentication basics
		- Multi-factor authentication
		- Time-based One Time Passwords (TOTP)
		- Authentication weaknesses
		- Password management
		- Outbound password management
	+ Information exposure
		- Exposure through extracted data and aggregation
		- Case study – Strava data exposure
		- System information leakage
	+ Platform security
		- Python platform security
* Using Vulnerable Components
	+ Assessing the environment
	+ Hardening
	+ Malicious packages in Python
	+ Case study – The British Airways data breach
	+ Vulnerability management
		- Patch management
		- Vulnerability databases
		- DevOps, the build process, and CI/CD
		- Dependency checking in Python
		- Detecting vulnerable components
* Cryptography for Developers
	+ Cryptography basics
	+ Cryptography in Python
	+ Elementary algorithms
		- Random number generation
		- Hashing
	+ Confidentiality protection
		- Symmetric encryption
		- Asymmetric encryption
		- Combining symmetric and asymmetric algorithms
		- Key exchange and agreement
		- Message Authentication Code (MAC)
		- Digital signature
		- Certificates
* Common Software Security Weaknesses
	+ Time and state
		- Race conditions
	+ Errors
* Conclusion
	+ Secure coding principles
		- Principles of robust programming by Matt Bishop
		- Secure design principles of Saltzer and Schroeder
	+ And now what?
		- Software security sources and further reading
		- Python resources