

# Desktop application security in Python

**CYDPyDsk3d | 3 days | On-site or online | Hands-on**

Your 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.

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.

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

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

So that nothing unexpected happens.

Nothing.

## Cyber security skills and drills



19 LABS



7 CASE STUDIES

### Audience

Python developers working on desktop applications

### Group size

12 participants

### Preparedness

General Python development

### Outline

- Cyber security basics
- Input validation
- Security features
- Using vulnerable components
- Cryptography for developers
- Common software security weaknesses
- Wrap up

### Standards and references

CWE and Fortify Taxonomy

### What you'll have learned

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

### › 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)

 *Lab – ReDoS in Python*

Dealing with ReDoS

### **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
- 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*
      - Python module hijacking
    - 🔗 *Lab – Library hijacking in Python*

## Day 2

---

### › Input validation

#### **Integer handling problems**

- Representing signed numbers
- Integer visualization
- Integers in Python
- Integer overflow
- Integer overflows in ctypes and numpy
- Other numeric problems
  - Working with floating-point numbers

#### **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

## > Security features

### Authentication

- Authentication basics
- Multi-factor authentication
- Time-based One Time Passwords (TOTP)
- Authentication weaknesses
- 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
  - Outbound password management
    - Hard coded passwords
    - Best practices
    -  *Lab – Hardcoded password*
      - Protecting sensitive information in memory
      - Challenges in protecting memory

### Information exposure

- Exposure through extracted data and aggregation
-  *Case study – Strava data exposure*
- System information leakage
  - Leaking system information
- Information exposure best practices

### Platform security

- Python platform security

- The Python ecosystem and its attack surface
- Python bytecode and security
- Security features offered by Python
- PEP 578 and audit hooks
- Sandboxing Python

## › Using vulnerable components


Assessing the environment

Hardening

Malicious packages in Python

 *Case study – The British Airways data breach*

### **Vulnerability management**

- Patch management
- [Vulnerability management](#)
- Vulnerability databases
- [DevOps, the build process and CI / CD](#)
- Dependency checking in Python
-  *Lab – Detecting vulnerable components*

## Day 3



---

## › 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
  - Common hashing mistakes


- Hashing in Python

 *Lab – Hashing in Python*

## **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

- Key exchange and agreement

- Key exchange
- Diffie-Hellman key agreement algorithm
- Key exchange pitfalls and best practices
- Integrity protection
- Authenticity and non-repudiation


- 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*

- Some further key management challenges

- Certificates


- Certificates and PKI
- X.509 certificates
- Chain of trust
- PKI actors and procedures
- Certificate revocation

## › Common software security weaknesses

### Time and state

- Race conditions
  - File race condition
    - Time of check to time of usage – TOCTTOU
    - TOCTTOU attacks in practice
    - Insecure temporary file

### Errors

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

## › Wrap up

### 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