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

Audience

Python developers working on desktop applications

Group size

12 participants

Outline

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

Preparedness

General Python development

21 labs

8 case studies

What you'll have learned

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

Table of contents

- Content-based blind SQL injection
- Time-based blind SQL injection

- SQL injection best practices
  - Input validation
  - Parameterized queries
  - Additional considerations
  - *Lab – SQL injection best practices*
  - *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*
    - *Case study – Command injection via ping*
    - Python module hijacking
    - *Lab – Module hijacking*
- General protection best practices

Day 2

› Input validation

Integer handling problems

- Representing signed numbers
- Integer visualization
- Integers in Python
- Integer overflow
- Integer overflow with ctypes and numpy
- *Lab – Integer problems in Python*
- Other numeric problems
  - Division by zero
  - Other numeric problems in Python
  - Working with floating-point numbers

Files and streams

- Path traversal

- Path traversal-related examples

Lab – Path traversal

- Additional challenges in Windows
- Virtual resources
- Path traversal best practices
- 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
- Authentication weaknesses - spoofing

Case study – PayPal 2FA bypass

- 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 length
    - Password hardening
    - Using passphrases
    - Password change
    - Forgotten passwords

Lab – Password reset weakness

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

Python platform security

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

› Using vulnerable components

Assessing the environment

Hardening

Malicious packages in Python

Vulnerability management

- Patch management
- Vulnerability management
- Bug bounty programs
- Vulnerability databases
- Vulnerability rating – CVSS
- 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 and strong PRNGs
  - Using random numbers in Python
  - *Case study – Equifax credit account freeze*
  - *Lab – Using random numbers in Python*
- 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
    - Elliptic Curve Cryptography
    - The ECC algorithm
    - Using ECC – best practices
    - ECC in Python
    - Combining symmetric and asymmetric algorithms
    - 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
  - Digital signature with ECC
  - Digital signature in Python

Public Key Infrastructure (PKI)

- Some further key management challenges
- Certificates
  - Chain of trust
  - PGP – Web of Trust
  - Certificate management – best practices

› Common software security weaknesses

Time and state

- Race conditions
  - File race condition
    - Time of check to time of usage – TOCTTOU
    - Insecure temporary file
  - Avoiding race conditions in Python
    - Thread safety and the Global Interpreter Lock (GIL)
    - Avoiding race conditions in Python
  - *Case study: TOCTTOU in Calamares*

Errors

- Error and exception handling principles
- Error handling
  - Returning a misleading status code
  - Information exposure through error reporting
- Exception handling
  - In the except,catch block. And now what?
  - Empty catch block
  - The danger of assert statements
  - *Lab – Exception handling mess*

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()
- Sandboxing Python

Denial of service

- Denial of Service
- Resource exhaustion
- Cash overflow
- Flooding
- Algorithm complexity issues

› Wrap up

Secure coding principles

- Principles of robust programming by Matt Bishop
- Secure design principles of Saltzer and Schröder

And now what?

- Software security sources and further reading
- Python resources