

Python security

CL-PYS | Onsite / Virtual classroom | 3 days

Audience: Python developers, architects and testers

Preparedness: General Python development

Exercises: Hands-on

The Python language is used in many different settings – from command-line tools to complex Web applications. Many of these Python programs are exposed to attack, either by being directly accessible through the Internet or by directly processing user-provided data in a server environment. Developers must therefore be extremely cautious in how to use different technologies securely, and should also have a deep understanding in secure coding techniques and potential pitfalls.

This course covers the most critical security issues in Python applications. We cover vulnerabilities from the OWASP Top Ten list for the web as they concern Python web applications as well as the Django framework. The course also encompasses the most significant security issues for Python code in general (including many Python-specific issues such as function hijacking), while also presenting security solutions provided by the Python ecosystem – such as authentication, access control and encryption.

Understanding the security solutions provided by Python as well as the various security issues and vulnerabilities is a must for all programmers using these technologies to develop web, desktop or server applications.

Outline:

- IT security and secure coding
- Web application security (OWASP Top Ten 2017)
- Client-side security
- XML security
- Python security architecture
- Practical cryptography
- Common coding errors and vulnerabilities
- Denial of service
- Principles of security and secure coding
- Knowledge sources

Participants attending this course will:

Understand basic concepts of security, IT security and secure coding
Learn Web vulnerabilities beyond OWASP Top Ten and know how to avoid them
Learn about XML security
Learn client-side vulnerabilities and secure coding practices
Understand security concepts of Web services
Learn about JSON security
Learn about Python security architecture
Have a practical understanding of cryptography
Learn about typical coding mistakes and how to avoid them
Learn about denial of service attacks and protections
Get sources and further readings on secure coding practices

Related courses:

- CL-WSC - Web application security (Onsite / Virtual classroom, 3 days)
- CL-WSM - Web application security master course (Onsite / Virtual classroom, 5 days)
- CL-PSC - Secure coding in PHP (Onsite / Virtual classroom, 3 days)
- CL-NJS - Node.js and Web application security (Onsite / Virtual classroom, 3 days)

Detailed table of contents

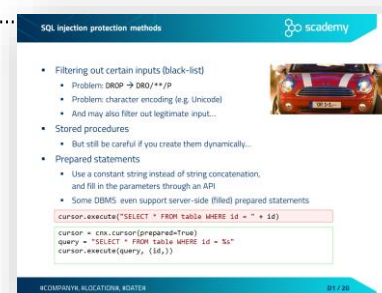
Day 1

IT security and secure coding

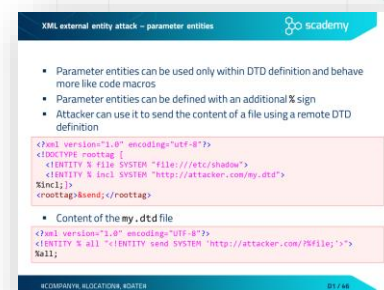
- Nature of security
- What is risk?
- IT security vs. secure coding
- From vulnerabilities to botnets and cybercrime
 - Nature of security flaws
 - From an infected computer to targeted attacks
 - The Seven Pernicious Kingdoms
 - OWASP Top Ten 2017

Web application security (OWASP Top Ten 2017)

- A1 - Injection
 - Injection principles
 - SQL injection
 - Exercise – SQL injection
 - Typical SQL Injection attack methods
 - Blind and time-based SQL injection
 - SQL injection protection methods
 - Database access in Python
 - ORM libraries in Python
 - Other injection flaws
 - Command injection
 - Command injection exercise – starting Netcat
 - Case study – ImageMagick
- A2 - Broken authentication
 - Session handling threats
 - Session handling best practices
 - Sessions in Django
 - Additional cookie security considerations
 - Setting cookie attributes – best practices
 - Cross site request forgery (CSRF)
 - CSRF prevention
 - CSRF prevention in Django



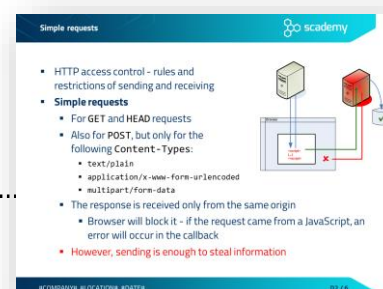
- A4 - XML external entity (XXE)
 - XML Entity introduction
 - XML external entity attack (XXE) – resource inclusion
 - XML external entity attack – URL invocation
 - XML external entity attack – parameter entities
 - Exercise – XXE attack
 - Preventing entity-related attacks
 - Case study – XXE in Google Toolbar
- A5 - Broken access control
 - Typical access control weaknesses
 - Insecure direct object reference (IDOR)
 - Exercise – Insecure direct object reference
 - Protection against IDOR
 - Case study – Facebook Notes
- A7 - Cross-Site Scripting (XSS)
 - Persistent XSS
 - Reflected XSS
 - DOM-based XSS
 - Exercise – Cross Site Scripting
 - XSS prevention
 - XSS prevention in Python
- A8 - Insecure deserialization
 - Serialization and deserialization basics
 - Security challenges of deserialization
 - Security issues when using Pickle
 - Code injection via overriding `__reduce__` in Pickle
 - Code injection via YAML deserialization
 - Issues with deserialization – JSON



Day 2

Client-side security

- JavaScript security
- Same Origin Policy
- Simple requests
- Preflight requests
- Exercise – Same-Origin Policy



- JavaScript usage
- JavaScript Global Object
- Dangers of JavaScript
- Exercise – Client-side authentication
- Client-side authentication and password management
- Protecting JavaScript code
- Clickjacking
 - Exercise – IFrame, Where is My Car?
 - Protection against Clickjacking
 - Anti frame-busting – dismissing protection scripts
 - Protection against busting frame busting
- AJAX security
 - XSS in AJAX
 - Script injection attack in AJAX
 - Exercise – XSS in AJAX
 - XSS protection in AJAX
 - Exercise CSRF in AJAX – JavaScript hijacking
 - CSRF protection in AJAX
- HTML5 security
 - New XSS possibilities in HTML5
 - HTML5 clickjacking attack – text field injection
 - HTML5 clickjacking – content extraction
 - Form tampering
 - Exercise – Form tampering
 - Cross-origin requests
 - HTML proxy with cross-origin request.....
 - Exercise – Client side include

XML security

- Introduction
- XML parsing
- XML parsing in Python
- XML bomb
- Exercise – XML bomb
- JSON security
 - Introduction
 - Embedding JSON server-side.....

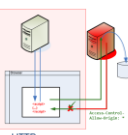
HTML proxy with cross-origin request

Can we do a HTTP proxy with this?

```
function xhr() {
  xhr.onreadystatechange=function()
  {
    if (xhr.readyState == 4)
    {
      document.getElementById("go").innerHTML =
        xhr.responseText;
    }
  }
}
```

Reverse Web shell: a Javascript (XSS) tunneling HTTP

- XSS + COR can be used for tunneling HTTP traffic between a server and the attacker through the client's page
- With the injected script an attacker can access vulnerable sites via the victim's browser by sending requests over the channel



Embedding JSON server-side

When an initial block of JSON is put to the page

Do not insert the JSON into the page directly

```
<script> var initData = <data.to_json %>; </script>
```

Always use escaping!

Also, ensure that the returned Content-Type header is application/json and not text/html

Otherwise the browser may execute an injected script

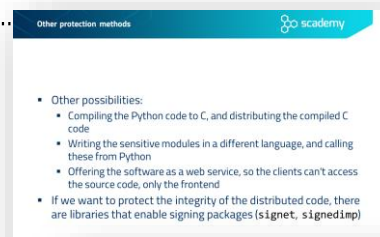
```
<script id="init_data" type="application/json">
  <data.to_json %>
</script>
```

```
//Client-side JavaScript code handling this
var dataElement = document.getElementById('init_data');
var jsonText = dataElement.textContent;
var initData = JSON.parse(jsonText);
```

- JSON injection
- JSON hijacking
- Case study – XSS via spoofed JSON element

Python security architecture

- Python architecture
- Python applications and their attack surfaces
- Authentication and authorization
 - Authentication in Python
 - Authorization in Python
 - Authentication and authorization in Django
 - Authentication and authorization in Flask
- Code protection in Python
 - Python bytecode
 - Obfuscation
 - Modifying the Python runtime
 - Weaknesses in the techniques
 - Other protection methods
 - Related security issues
 - Sandboxing



Other protection methods

- Other possibilities:
 - Compiling the Python code to C, and distributing the compiled C code
 - Writing the sensitive modules in a different language, and calling these from Python
 - Offering the software as a web service, so the clients can't access the source code, only the frontend
 - If we want to protect the integrity of the distributed code, there are libraries that enable signing packages (signet, signedimp)

Practical cryptography

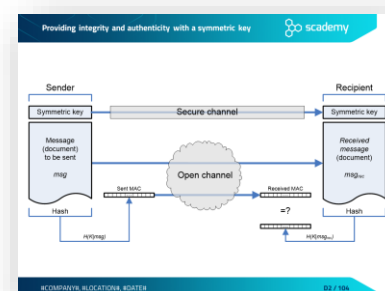
- Rule #1 of implementing cryptography.....
- Cryptosystems
 - Elements of a cryptosystem
 - Cryptographic libraries in Python – overview
- Symmetric-key cryptography
 - Providing confidentiality with symmetric cryptography
 - Symmetric encryption algorithms
 - Modes of operation
 - Symmetric encryption
- Other cryptographic algorithms
 - Hash or message digest
 - Hash algorithms
 - SHattered
 - Hashing
 - Message Authentication Code (MAC)
 - Providing integrity and authenticity with a symmetric key.....



Rule #1 of implementing cryptography

"Don't do it!"

- Don't invent your own algorithms
 - "It will be more secure because nobody knows how it works" is a common misconception
 - This bad approach is called **security by obscurity**
- Don't implement existing algorithms either
 - Using available implementations from established libraries is more secure and more efficient anyway

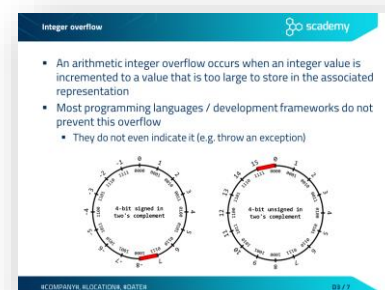


- Random number generation
 - Random numbers and cryptography
 - Cryptographically-strong PRNGs
 - Weak PRNGs in Python
 - Random numbers
 - Hardware-based TRNGs
- Asymmetric (public-key) cryptography
 - Providing confidentiality with public-key encryption
 - Rule of thumb – possession of private key
 - The RSA algorithm
 - Introduction to RSA algorithm
 - Encrypting with RSA
 - Combining symmetric and asymmetric algorithms
 - Digital signing with RSA
 - Asymmetric encryption
- Public Key Infrastructure (PKI)
 - Man-in-the-Middle (MitM) attack
 - Digital certificates against MitM attack
 - Certificate Authorities in Public Key Infrastructure
 - X.509 digital certificate

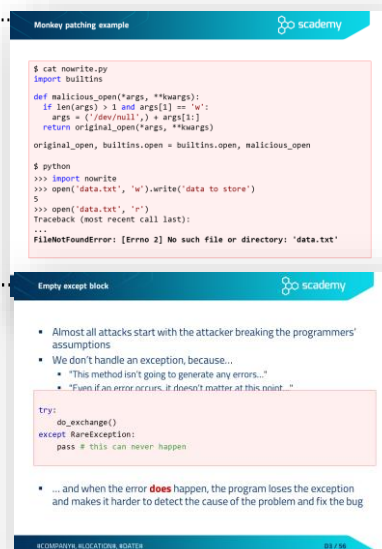
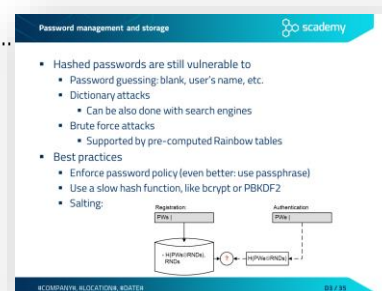
Day 3

Common coding errors and vulnerabilities

- Input validation
 - Input validation concepts
 - Integer problems
 - Representation of negative integers
 - Integer overflow
 - Integers in Python
 - Integer problems in Python
 - Exercise IntOverflow
- Path traversal vulnerability
 - Path traversal – weak protections
 - Path traversal – best practices
 - Path traversal mitigation
- Unvalidated redirects and forwards
 - Redirects in Django
- Log forging
 - Some other typical problems with log files



- Executing user-controlled code in Python
 - Dangerous code execution – eval() and exec()
 - Dangerous code execution – input()
- String formatting issues in Python
 - Format string problems in Python
 - Information leakage – can you guess the output?
- Improper use of security features
 - Typical problems related to the use of security features
 - Password management
 - Exercise – Weakness of hashed passwords
 - Password management and storage
 - Special purpose hash algorithms for password storage
 - PBKDF2 and scrypt implementations in Python
 - Argon2 and bcrypt implementations in Python
 - Case study – the Ashley Madison data breach
 - Typical mistakes in password management
 - Dangers of reflection in Python
 - Python introspection and reflection – features and risks
 - Dynamic loading
 - Module injection
 - Monkey patching
 - Monkey patching example
 - Function hijacking
 - Exercise – Module injection in Python
- Improper error and exception handling
 - Typical problems with error and exception handling
 - Exception handling in Python
 - Empty except block
 - Overly broad except
 - Using multi-except
 - Returning from finally block – spot the bug!
 - Exercise ErrorHandling – spot the bug!
 - Exercise – Error handling
 - Relying on assertions for error checking – spot the bug!
- Time and state problems
 - Concurrency and threading
 - Concurrency issues in Python
 - Concurrency modules in Python
 - The threading module
 - Synchronization options
 - The Global Interpreter Lock
 - Performance and the GIL



- GIL management
- Bypassing the GIL
- A quote from Guido van Rossum
- Pools
- Exercise – GIL performance
- Exercise – Concurrency issues
- Time-of-check-to-time-of-use (TOCTTOU)
 - Serialization errors
 - Preventing file I/O TOCTTOU in Python
 - Exercise - TOCTTOU
- Code quality problems
 - Dangers arising from poor code quality
 - Immutability
 - Immutability – guess the output!
 - Context managers
 - Resource management in Python
 - Releasing resources – spot the bug!
 - An even better solution
 - Behind the with statement
 - @contextmanager decorator – spot the bug!
 - @contextmanager decorator – handling error



Denial of service

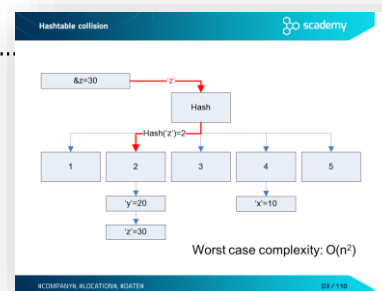
- DoS introduction
- Asymmetric DoS
- Regular expression DoS (ReDoS)
 - Exercise ReDoS
 - ReDoS mitigation
 - Case study – ReDos in Stack Exchange
- Hashtable collision attack
 - Using hashtables to store data
 - Hashtable collision
 - Hash tables in Python

Principles of security and secure coding

- Matt Bishop's principles of robust programming
- The security principles of Saltzer and Schroeder

Knowledge sources

- Secure coding sources – a starter kit



- Vulnerability databases
- Python secure coding resources
- Recommended books – Python security