Cel szkolenia:

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?

This advanced 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.

Participants attending this course will:

- Understand basic concepts of security, IT security and secure coding
- Realize the severe consequences of unsecure buffer handling
- Understand the architectural protection techniques and their weaknesses
- Learn about typical coding mistakes and how to avoid them
- Be informed about recent vulnerabilities in various platforms, frameworks and libraries
- Understand the requirements of secure communication
- Have a practical understanding of cryptography
- Learn about XML security
- Understand essential security protocols
Understand some recent attacks against cryptosystems
Understand security considerations in the SDLC
Understand security testing approaches and methodologies
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

Audience:
C and C++ developers, software architects and testers

Plan szkolenia:

- 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
    - Reasons of difficulty
    - From an infected computer to targeted attacks
  - Classification of security flaws
    - Landwehr’s taxonomy
    - The Seven Pernicious Kingdoms
- ARM machine code, memory layout and stack operations
  - ARM Processors – main registers
  - ARM Processors – most important instructions
  - ARM Processors – flags and condition fields
  - ARM Processors – control instructions
  - ARM Processors – stack handling instructions
  - Understanding complex ARM instructions
  - The function calling mechanism in ARM
  - The local variables and the stack frame
  - Function calls – prologue and epilogue of a function (ARM)
  - Stack frame of nested calls
  - Stack frame of recursive functions
- Buffer overflow
  - Stack overflow
Buffer overflow on the stack
- Overwriting the return address
- Exercises – introduction
- Exercise BOFIntro
- Exercise BOFShellcode

Protection against stack overflow
- Specific protection methods
- Protection methods at different layers
- The protection matrix of software security
- Stack overflow – Prevention (during development)
- Stack overflow – Detection (during execution)
- Fortify compiler option (FORTIFY_SOURCE)
- Exercise BOFShellcode – Using the Fortify compiler option

Stack smashing protection
- Stack smashing protection variants
- Stack smashing protection in GCC
- Exercise BOFShellcode – Stack smashing protection
- Effects of stack smashing protection – prologue
- Effects of stack smashing protection – epilogue
- Bypassing stack smashing protection – an example
- Overwriting arguments – Mitigation

Address Space Layout Randomization (ASLR)
- Randomization with ASLR
- Practical weaknesses and limitations to ASLR
- Circumventing ASLR: NOP sledding

Non executable memory areas – the NX bit
- Access control on memory segments
- The Never eXecute (NX) bit
- Exercise BOFShellcode – Enforcing NX memory segments

Buffer overflow
- Return oriented programming (ROP)
  - Circumventing memory execution protection
  - Return-to-libc attack in ARM
  - Exercise Return-to-libc
  - ROP gadget - Register fill with constants
  - ROP gadget – Memory write
Combining the ROP gadgets
Real ROP attack scenarios
ROP mitigation
  Mitigation techniques of ROP attack
Heap overflow
  Memory allocation managed by a doubly-linked list
  Buffer overflow on the heap
  Steps of freeing and joining memory blocks
  Freeing allocated memory blocks
  Case study – Heartbleed
    TLS Heartbeat Extension
    Heartbleed – information leakage in OpenSSL
    Heartbleed – fix in v1.0.1g
  Protection against heap overflow
Common coding errors and vulnerabilities
  Input validation
    Input validation concepts
  Integer problems
    Representation of negative integers
    Integer ranges
    Integer overflow
    Integer problems in C/C++
    The integer promotion rule in C/C++
    Arithmetic overflow – spot the bug!
    Exercise IntOverflow
    What is the value of abs(INT_MIN)?
    Signedness bug – spot the bug!
    Integer truncation – spot the bug!
    Integer problem – best practices
    Case study – Android Stagefright
Injection
  Injection principles
  SQL Injection exercise
  Typical SQL Injection attack methods
  Blind and time-based SQL injection
  SQL Injection protection methods
- Command injection
- Command injection exercise – starting Netcat
- Printf format string bug
  - Printf format strings
  - Printf format string bug – exploitation
  - Exercise Printf
- Printf format string problem – best practices
- Some other input validation problems
  - Array indexing – spot the bug!
  - Off-by-one and other null termination errors
  - The Unicode bug
- Path traversal vulnerability
  - Path traversal – weak protections
  - Path traversal – best practices
- Log forging
  - Some other typical problems with log files
- Time and state problems
  - Time and state related problems
  - Serialization errors
  - Exercise TOCTTOU
  - Best practices against TOCTTOU
  - Problems with temp files
  - Requirements for creating temp files
  - Requirements explained
  - Creating temp files on POSIX systems
  - Creating temp files portably
  - Deleting temp files
- 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
○ Error detection and correction codes
  ○ Authenticity - Malicious message manipulation
    ○ Modeling message manipulation
    ○ Practical integrity protection (detection)
○ Non-repudiation
  ○ Non-repudiation
○ Summary
  ○ Detecting integrity violation
○ 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
○ Cryptosystems
  ○ Elements of a cryptosystem
○ Symmetric-key cryptography
  ○ Providing confidentiality with symmetric cryptography
  ○ Symmetric encryption algorithms
  ○ Modes of operation
  ○ Symmetric encryption with OpenSSL: encryption
  ○ Symmetric encryption with OpenSSL: decryption
○ Other cryptographic algorithms
  ○ Hash or message digest
  ○ Hash algorithms
  ○ SHAttered
Hashing with OpenSSL
Message Authentication Code (MAC)
Providing integrity and authenticity with a symmetric key
Random number generation
  Random numbers and cryptography
  Cryptographically-strong PRNGs
  Weak PRNGs in C and C++
  Stronger PRNGs in C
  Generating random numbers with OpenSSL
  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 with OpenSSL
    Digital signatures with OpenSSL

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

XML security
  XML injection
    Exercise – XML injection
    Protection through sanitization and XML validation
    XML parsing in C++
  Abusing XML Entity
    XML Entity introduction
    Exercise – XML bomb
    XML bomb
    XML external entity attack (XXE) – resource inclusion
    Exercise – XXE attack
○ Preventing entity-related attacks
○ Case study – XXE in Google Toolbar

○ Security protocols
  ○ Secure network protocols
  ○ Specific vs. general solutions
  ○ IPSEC protocol family
    ○ IPSEC standards
    ○ Security Association (SA)
    ○ Message formats
    ○ AH packet structure
    ○ ESP packet structure
    ○ Protected channels
    ○ More complex set-ups
  ○ Traffic control
  ○ SA structure
  ○ Key management

○ The TLS protocol
  ○ SSL and TLS
  ○ Usage options
  ○ Security services of TLS
  ○ SSL/TLS handshake

○ Protocol-level vulnerabilities
  ○ BEAST
  ○ FREAK
  ○ FREAK – attack against SSL/TLS
  ○ Logjam attack

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

○ 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
  ○ Argon2 and PBKDF2 implementations in C/C++
  ○ bcrypt and scrypt implementations in C/C++
  ○ Case study – the Ashley Madison data breach
  ○ Typical mistakes in password management
  ○ Exercise – Hard coded passwords

○ Sensitive information in memory
  ○ Protecting secrets in memory
  ○ Sensitive info in memory - minimize the attack surface
  ○ Your secrets vs. dynamic memory
  ○ Zeroisation
  ○ Zeroisation vs. optimization – Spot the bug!
  ○ Copies of sensitive data on disk
  ○ Core dumps
  ○ Disabling core dumps
  ○ Swapping
  ○ Memory locking - preventing swapping
  ○ Problems with page locking
  ○ Best practices

○ Insufficient anti-automation
  ○ Captcha
  ○ Captcha weaknesses

○ Improper error and exception handling
  ○ Typical problems with error and exception handling
  ○ Empty catch block
  ○ Overly broad catch
  ○ Exercise ErrorHandling – spot the bug!
  ○ Exercise – Error handling
  ○ Case study – "e;#iamroot"e; authentication bypass in macOS
    ○ Authentication process in macOS (High Sierra)
    ○ Incorrect error handling in opendirectoryd
    ○ The #iamroot vulnerability (CVE-2017-13872)
- Information leakage through error reporting
- Code quality problems
  - Dangers arising from poor code quality
  - Poor code quality – spot the bug!
  - Unreleased resources
  - Type mismatch – spot the bug!
- Poor code quality – spot the bug!
- Exercise TypeMismatch
- Memory allocation problems
  - Smart pointers
  - Zero length allocation
  - Double free
  - Mixing delete and delete[]
- Use after free
  - Use after free – instance of a class
  - Spot the bug
  - Use after free – dangling pointers
- Case study - WannaCry
  - The WannaCry ransomware
  - The vulnerability behind WannaCry – spot the bug!
  - Lessons learned

- Security in the software development lifecycle
  - Building Security in Maturity Model (BSIMM)
  - Software Assurance Maturity Model (SAMM)
  - Microsoft Security Development Lifecycle (SDL)

- Security testing
  - Functional testing vs. security testing
  - Security vulnerabilities
  - Prioritization – risk analysis
  - Security assessments in various SDLC phases
  - Security testing methodology
    - Steps of test planning (risk analysis)
    - Scoping and information gathering
      - Stakeholders
      - Assets
      - Security objectives for testing
  - Threat modeling
○ Attacker profiles
○ Threat modeling
○ Threat modeling based on attack trees
○ Threat modeling based on misuse/abuse cases
○ Misuse/abuse cases – a simple example
○ SDL threat modeling
○ The STRIDE threat categories
○ Diagramming – elements of a DFD
○ Data flow diagram – example
○ Threat enumeration – mapping STRIDE to DFD elements
○ Risk analysis – classification of threats
○ The DREAD risk assessment model

○ Testing steps
○ Deriving test cases
○ Accomplishing the tests
○ Processing test results
○ Mitigation concepts
○ Standard mitigation techniques of MS SDL
○ Review phase

○ Security testing techniques and tools
○ General testing approaches
○ Design review
  ○ Assessment of security requirements
  ○ Identifying security-critical aspects – hotspots
○ Source code review
  ○ Code review for software security
  ○ Taint analysis
  ○ Heuristic-based
  ○ Static code analysis
    ○ Static code analysis
    ○ Exercise – Static code analysis using FlawFinder
○ Testing the implementation
  ○ Manual vs. automated security testing
  ○ Penetration testing
  ○ Stress tests
  ○ Binary and memory analysis
Exercise – Binary analysis with strings
  Instrumentation libraries and frameworks
    Exercise – Using Valgrind

Fuzzing
  Automated security testing - fuzzing
  Challenges of fuzzing
  Exercise – Fuzzing with AFL (American Fuzzy Lop)

Deployment environment
  Assessing the environment
    Assessing the environment
    Searching for online devices with SHODAN
    Exercise – using SHODAN
    Finding weaknesses with search engines
    Exercise – Finding weaknesses with search engines
    Password audit
    Exercise – using John the Ripper
    Testing random number generators
    Exercise – Testing random number generators

Configuration management
  Configuration management

Hardening
  Hardening
  Network-level hardening
  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

Case study - Shellshock
  Shellshock – basics of using functions in bash
  Shellshock – vulnerability in bash
  Exercise - Shellshock
- Shellshock fix and counterattacks
- Exercise - Command override with environment variables

- 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
  - Recommended books - C/C++

Wymagania:

Advanced C/C++ development

Poziom trudności

Certyfikaty:

The participants will obtain certificates signed by SCADEMY (course completion).

Prowadzący:

Authorized SCADEMY Trainer

Informacje dodatkowe:

Training come with a number of easy-to-understand exercises providing live hacking fun. By accomplishing these exercises with the lead of the trainer, participants can analyze vulnerable code snippets and commit attacks against them in order to fully understand the root causes of certain security problems. All exercises are prepared in a plug-and-play manner by using a pre-set desktop virtual machine, which provides a uniform development environment.

SCADEMY together with online application security educational platform AVATAO (more about AVATAO [www.avatao.com](http://www.avatao.com)) for each of participant SCADEMYs authorized training adds the 30 days business AVATAO trial holds the following package:

- 30-day customized free trial