Software Security

Lennart Beckman
Europoint Networking
lennart@europoint.se
0707-323609

Security basics - CIA

- Confidentiality
- Integrity
- Availability
- Traceability

Security basics

- Nothing is 100% secure
- Time is limited – for developer
- Time is unlimited – for intruder

Problem

Where is the problem introduced?
Where does security come in?

- OWASP, Open Web Application Security Project
- CLASP, Comprehensive Lightweight Application Security 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

- Choose the best from the models
- Define roles
  - Security coach
  - Security assessor
- Continuous, 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 privilege
- Separation of data and functionality

Defence in depth

Programming

Two kinds of problems:

1. Input
2. Other problems

Buffer overrun

- Typical problem
- Easy to create...
- Common in low level languages, C and 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

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

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?

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

Tips

- Tänk på första och sista tecknet

Exercise – what is wrong?

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

Tips

- Prova med 1024 tecken

How do you use this?

Only one byte ...
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 sizes 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

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>signed char</td>
<td>1 byte</td>
<td>-128</td>
<td>127 (0x7f)</td>
</tr>
<tr>
<td>unsigned char</td>
<td>1 byte</td>
<td>0</td>
<td>255 (0xff)</td>
</tr>
<tr>
<td>short</td>
<td>2 byte</td>
<td>-32768</td>
<td>32767 (0x7f7f)</td>
</tr>
<tr>
<td>unsigned short</td>
<td>2 byte</td>
<td>0</td>
<td>65535 (0xffff)</td>
</tr>
<tr>
<td>int</td>
<td>4 byte</td>
<td>-2147483648</td>
<td>2147483647</td>
</tr>
<tr>
<td>unsigned int</td>
<td>4 byte</td>
<td>0</td>
<td>4294967295</td>
</tr>
<tr>
<td>long</td>
<td>4 byte</td>
<td>-2147483648</td>
<td>2147483647</td>
</tr>
<tr>
<td>unsigned long</td>
<td>4 byte</td>
<td>0</td>
<td>4294967295</td>
</tr>
<tr>
<td>long long</td>
<td>8 byte</td>
<td>0</td>
<td>0 (max)</td>
</tr>
<tr>
<td>unsigned long long</td>
<td>8 byte</td>
<td>0</td>
<td>0 (max)</td>
</tr>
</tbody>
</table>

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

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

But...

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

Typecasting

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

Example, cont.

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

Example 2, checking for overflow

```c
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:

```c
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 all arithmetic (can be 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

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

**Exemple**

“Read Id from the user”
string sql = “SELECT cardno FROM” + “cust WHERE id = “ + Id; 
“Print his cardnumber, cardno”

**Problem**

Try with Id = 1 or 2>1 -- , gives the search criteria:

id = 1 or 2>1 --

Always true. Produces all cardnumbers...

**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

**Countermeasures**

- Input control
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…)

Cross Site Scripting

- Web based systems
- Input control missing
- Common

Description

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

XSS, example

Let someone's browser execute e.g.

```html
```

Easy…
What is the problem?

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

How to find

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

Countermeasures

- Input control
- Output control

Test

- 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(window.location);</script>`
    - `\";alert(document.cookie);\"`
    - `</a><script>alert(document.cookie);</script>`
    - Persistenta kan vara mer svårfunna

Compiler optimization
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

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...

Example, cont.

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

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

How to find?

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

Even worse

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

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

Cont.

- Turn the optimization off for sensitive parts

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

Example 2

```c
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

Explanation

- Overflow is a non-defined state in the standard. A compiler can choose 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)

Countermeasures?

- Code review
- Input control

Input control
Use only a few entrances

Use input validation

Input control between systems

Explicit validation

Validate with regular expressions

In many programming languages, e.g.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>^</td>
<td>start av string</td>
</tr>
<tr>
<td>$</td>
<td>slut av string</td>
</tr>
<tr>
<td>?</td>
<td>0 eller flera upprepningar</td>
</tr>
<tr>
<td>+</td>
<td>1 eller flera upprepningar</td>
</tr>
<tr>
<td>(n,m)</td>
<td>Mellan n och m uppr.</td>
</tr>
<tr>
<td>[ab]</td>
<td>a eller b</td>
</tr>
<tr>
<td>[abc]</td>
<td>någon av a,b eller c</td>
</tr>
<tr>
<td>[a-z]</td>
<td>ingår i a till z</td>
</tr>
<tr>
<td>\</td>
<td>escape char</td>
</tr>
<tr>
<td>etc.</td>
<td></td>
</tr>
</tbody>
</table>
Example

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

semesterbild.jpg
is an example of a string in this

Example 2

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

Programming rules

CERT, Top 10 Secure Coding Practices

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

Tio i topp

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

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.
Bonus

1. Define security requirements
2. Model threats.