Software Analysis

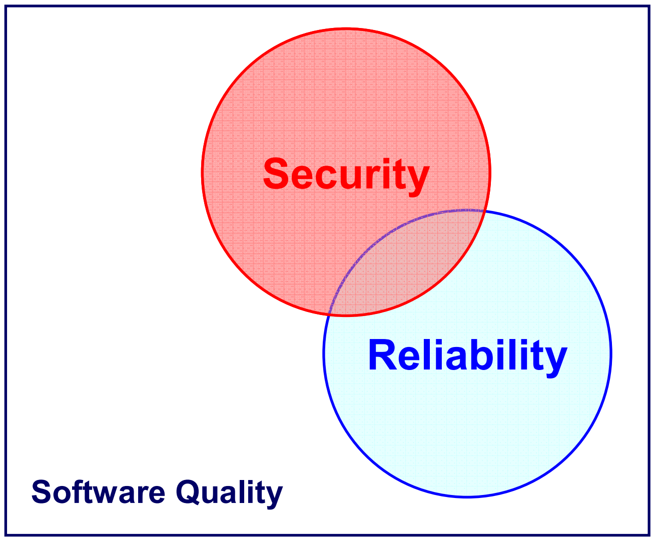
Security
1. Motivation

Mar 2008

Jonathan Aldrich
Carnegie Mellon University

Slides by Paulo Marques, University of Coimbra, Portugal

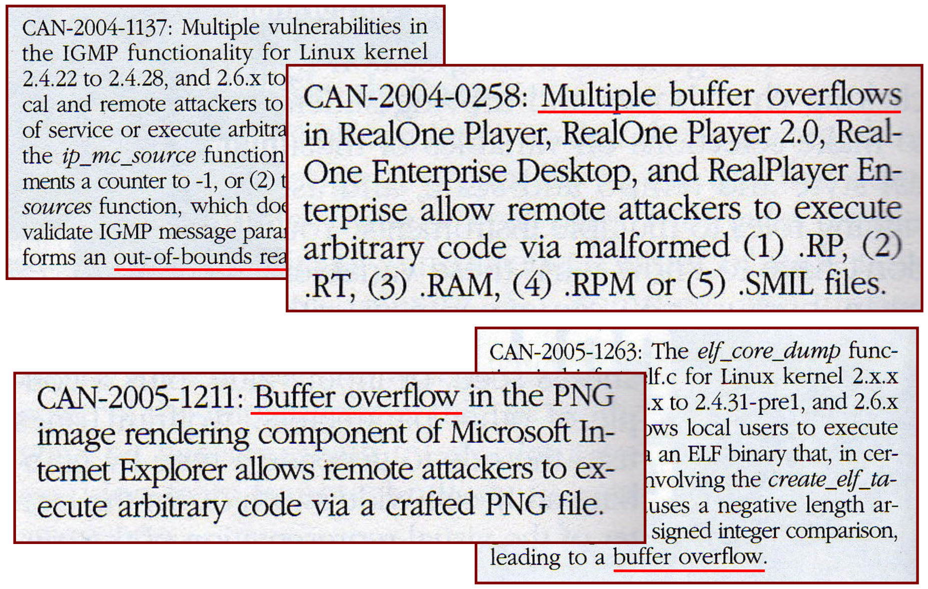
Why?



Strong Language

Ed Nisley

in Dr. Dobb's Journal, CMP Publishers, October 2005



CAN-2004-1137: Multiple vulnerabilities in the IGMP functionality for Linux kernel 2.4.22 to 2.4.28, and 2.6.x to
cal and remote attackers to
of service or execute arbitra
the *ip_mc_source* function
ments a counter to -1, or (2) t
sources function, which doe
validate IGMP message para
forms an out-of-bounds rea

CAN-2004-0258: Multiple buffer overflows in RealOne Player, RealOne Player 2.0, RealOne Enterprise Desktop, and RealPlayer Enterprise allow remote attackers to execute arbitrary code via malformed (1) .RP, (2) .RT, (3) .RAM, (4) .RPM or (5) .SMIL files.

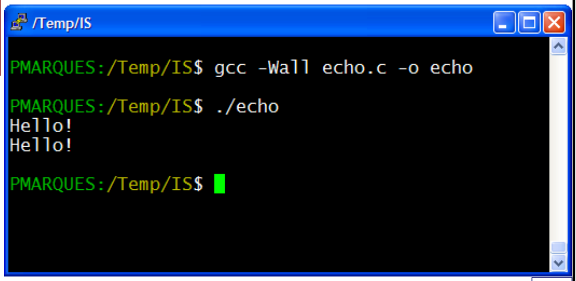
CAN-2005-1211: Buffer overflow in the PNG image rendering component of Microsoft Internet Explorer allows remote attackers to execute arbitrary code via a crafted PNG file.

CAN-2005-1263: The *elf_core_dump* func-
lf.c for Linux kernel 2.x.x
.x to 2.4.31-pre1, and 2.6.x
ows local users to execute
an ELF binary that, in cer-
nvolving the *create_elf_ta-*
uses a negative length ar-
signed integer comparison,
leading to a buffer overflow.

So, what exactly is a buffer overflow?

A very simple program

```c
#include <stdio.h>

void print_it(void)
{
    char s[8];

    scanf("%s", s);
    printf("%s\n", s);
}

int main(void)
{
    print_it();

    return 0;
}
```



```
PMARQUES:/Temp/IS$ gcc -Wall echo.c -o echo

PMARQUES:/Temp/IS$ ./echo
Hello!
Hello!

PMARQUES:/Temp/IS$
```

Extra code

- Our objective is to have hack() executed!
  - For now, don't worry about where hack() comes from…

```c
#include <stdio.h>
#include <stdlib.h>

void print_it(void)
{
    char s[8];

    scanf("%s", s);
    printf("%s\n", s);
}

void hack(void)
{
    printf("I've been hacked!\n");
    system("ls -l");
    printf("Done!\n");
}

int main(void)
{
    print_it();

    return 0;
}
```

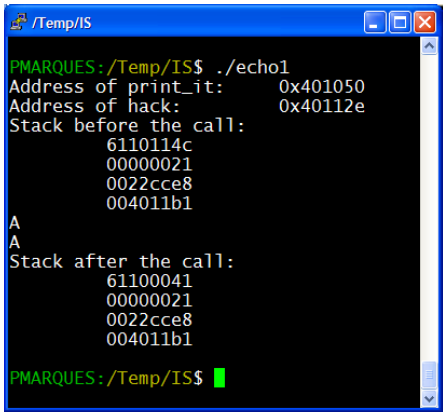
It could be a lot worse!

Examining some internal aspects of our program…

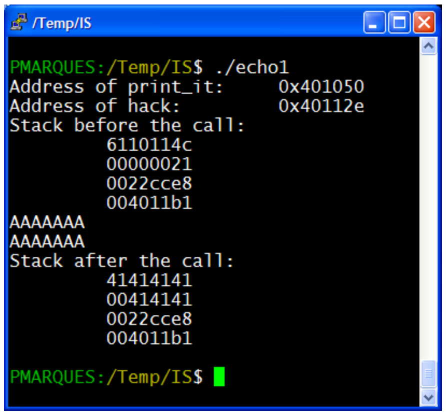
```c
#include <stdio.h>
#include <stdlib.h>

void print_it(void)
{
    char s[8];

    printf("Stack before the call:\n");
    printf("    \t %08x \n", * ( (int*)(s +  0) ) );
    printf("    \t %08x \n", * ( (int*)(s +  4) ) );
    printf("    \t %08x \n", * ( (int*)(s +  8) ) );
    printf("    \t %08x \n", * ( (int*)(s + 12) ) );

    scanf("%s", s);
    printf("%s\n", s);

    printf("Stack after the call:\n");
    printf("    \t %08x \n", * ( (int*)(s +  0) ) );
    printf("    \t %08x \n", * ( (int*)(s +  4) ) );
    printf("    \t %08x \n", * ( (int*)(s +  8) ) );
    printf("    \t %08x \n", * ( (int*)(s + 12) ) );
}

void hack(void)
{
    printf("I've been hacked!\n");
    system("ls -l");
    printf("Done!\n");
}

int main(void)
{
    printf("Address of print_it: \t %p \n", print_it);
    printf("Address of hack:     \t %p \n", hack);

    print_it();

    return 0;
}
```

7

Result of executing it



```
PMARQUES:/Temp/IS$ ./echo1
Address of print_it:     0x401050
Address of hack:         0x40112e
Stack before the call:
            6110114c
            00000021
            0022cce8
            004011b1
A
A
Stack after the call:
            61100041
            00000021
            0022cce8
            004011b1

PMARQUES:/Temp/IS$
```

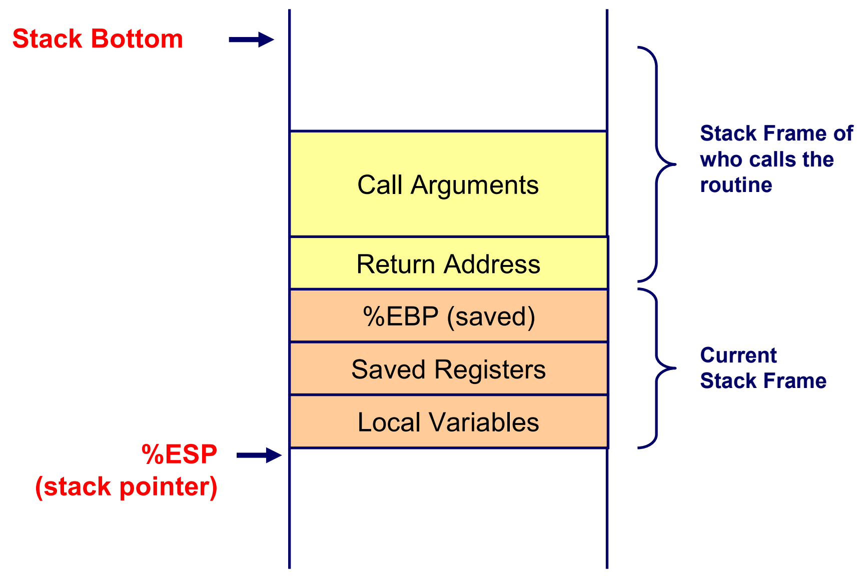


```
PMARQUES:/Temp/IS$ ./echo1
Address of print_it:     0x401050
Address of hack:         0x40112e
Stack before the call:
            6110114c
            00000021
            0022cce8
            004011b1
AAAAAAA
AAAAAAA
Stack after the call:
            41414141
            00414141
            0022cce8
            004011b1

PMARQUES:/Temp/IS$
```

The ASCII code of 'A' is 0x41!

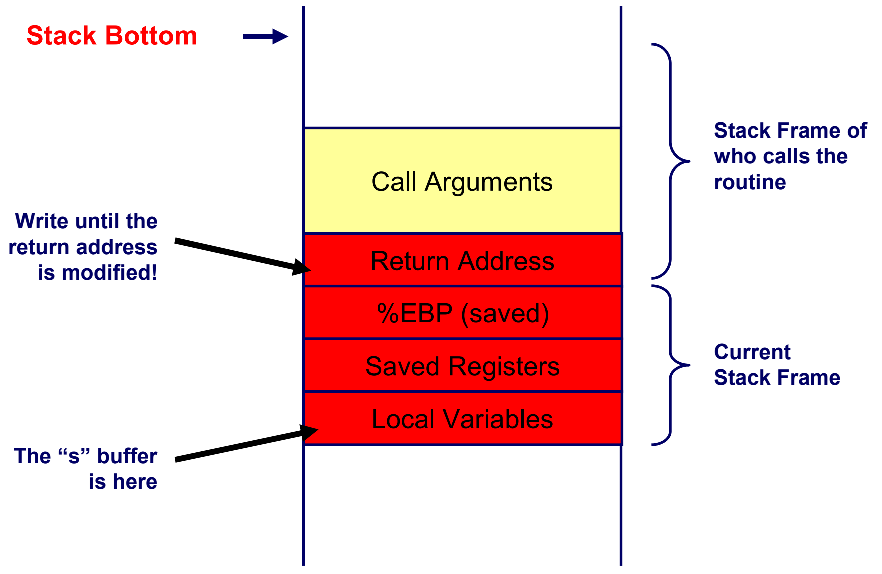
8

Execution Stack on x86



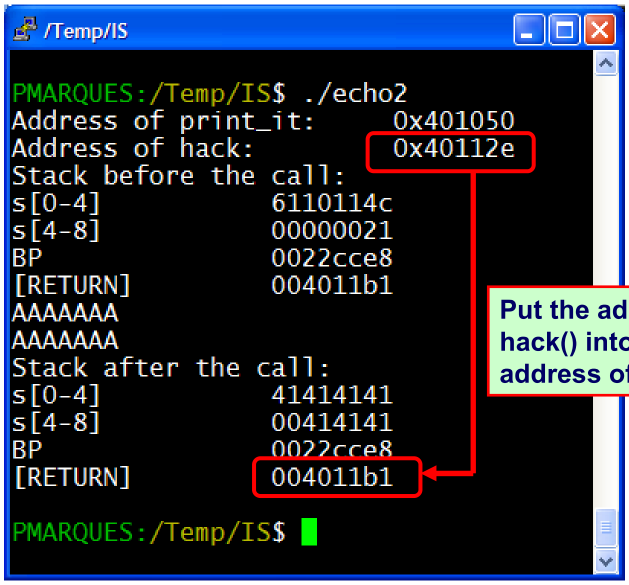
9

The Attack!



10

The execution revisited

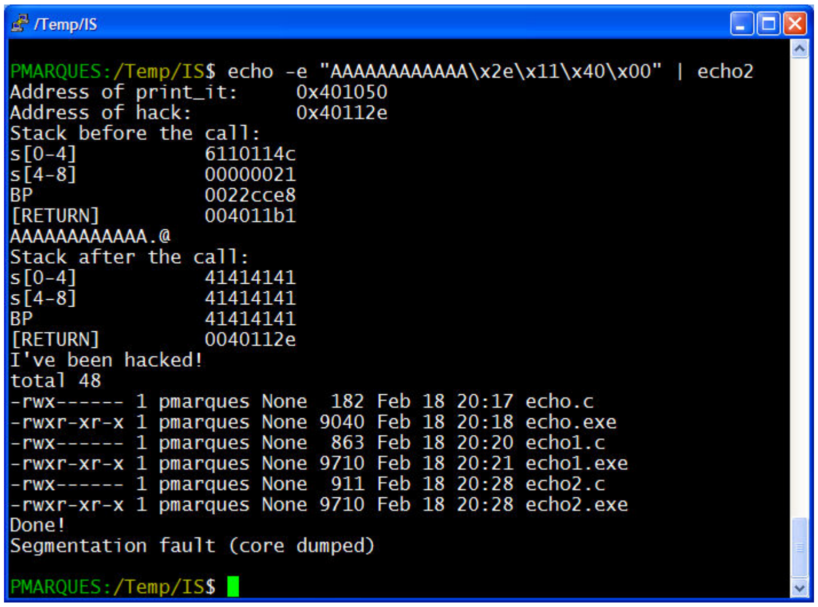


```
PMARQUES:/Temp/IS$ ./echo2
Address of print_it:     0x401050
Address of hack:         0x40112e
Stack before the call:
s[0-4]                   6110114c
s[4-8]                   00000021
BP                       0022cce8
[RETURN]                 004011b1
AAAAAAA
AAAAAAA
Stack after the call:
s[0-4]                   41414141
s[4-8]                   00414141
BP                       0022cce8
[RETURN]                 004011b1

PMARQUES:/Temp/IS$
```

Put the address of hack() into the return address of the function!

11

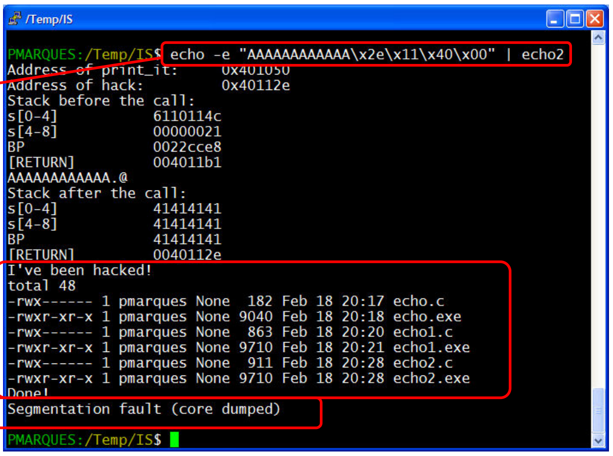
Quite Simple…



```
PMARQUES:/Temp/IS$ echo -e "AAAAAAAAAAAA\x2e\x11\x40\x00" | echo2
Address of print_it:     0x401050
Address of hack:         0x40112e
Stack before the call:
s[0-4]                   6110114c
s[4-8]                   00000021
BP                       0022cce8
[RETURN]                 004011b1
AAAAAAAAAAAA.@
Stack after the call:
s[0-4]                   41414141
s[4-8]                   41414141
BP                       41414141
[RETURN]                 0040112e
I've been hacked!
total 48
-rwx------ 1 pmarques None  182 Feb 18 20:17 echo.c
-rwxr-xr-x 1 pmarques None 9040 Feb 18 20:18 echo.exe
-rwx------ 1 pmarques None  863 Feb 18 20:20 echo1.c
-rwxr-xr-x 1 pmarques None 9710 Feb 18 20:21 echo1.exe
-rwx------ 1 pmarques None  911 Feb 18 20:28 echo2.c
-rwxr-xr-x 1 pmarques None 9710 Feb 18 20:28 echo2.exe
Done!
Segmentation fault (core dumped)

PMARQUES:/Temp/IS$
```

12

What's happening?



```
PMARQUES:/Temp/IS$ echo -e "AAAAAAAAAAAA\x2e\x11\x40\x00" | echo2
Address of print_it:      0x401050
Address of hack:          0x40112e
Stack before the call:
s[0-4]           6110114c
s[4-8]           00000021
BP               0022cce8
[RETURN]         004011b1
AAAAAAAAAAAA.@
Stack after the call:
s[0-4]           41414141
s[4-8]           41414141
BP               41414141
[RETURN]         0040112e
I've been hacked!
total 48
-rwx------ 1 pmarques None  182 Feb 18 20:17 echo.c
-rwxr-xr-x 1 pmarques None 9040 Feb 18 20:18 echo.exe
-rwx------ 1 pmarques None  863 Feb 18 20:20 echo1.c
-rwxr-xr-x 1 pmarques None 9710 Feb 18 20:21 echo1.exe
-rwx------ 1 pmarques None  911 Feb 18 20:28 echo2.c
-rwxr-xr-x 1 pmarques None 9710 Feb 18 20:28 echo2.exe
Done!
Segmentation fault (core dumped)
PMARQUES:/Temp/IS$
```

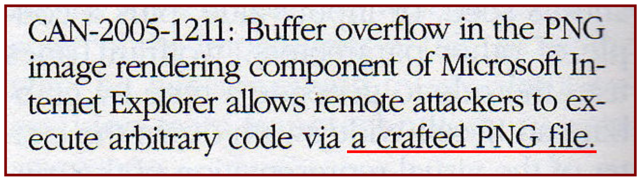
String is longer than the buffer size. Return address is overwritten.

Malicious code executing

It crashes at the end. But… the harm is already done!

13

But where does the hack() function comes from???



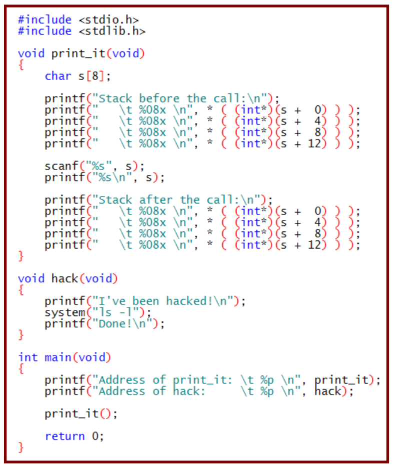
CAN-2005-1211: Buffer overflow in the PNG image rendering component of Microsoft Internet Explorer allows remote attackers to execute arbitrary code via a crafted PNG file.

Normally, buffers are not long enough to store the code directly (e.g. s[] is only 8 bytes long – not enough for hack()). But, if your program reads data from some other source, storing it in memory, it's normally easy to find out the address of that buffer. In that case, the return address of the exploitable function is to be filled in with the address of the relevant buffer. E.g. an image!

14

But…

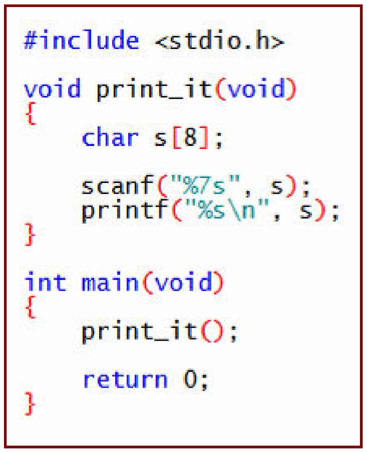
- Hackers do not have access to the source code to find out the size of buffers. How do they do it?



```c
#include <stdio.h>
#include <stdlib.h>

void print_it(void)
{
    char s[8];

    printf("Stack before the call:\n");
    printf("   \t %08x \n", * ( (int*)(s +  0) ) );
    printf("   \t %08x \n", * ( (int*)(s +  4) ) );
    printf("   \t %08x \n", * ( (int*)(s +  8) ) );
    printf("   \t %08x \n", * ( (int*)(s + 12) ) );

    scanf("%s", s);
    printf("%s\n", s);

    printf("Stack after the call:\n");
    printf("   \t %08x \n", * ( (int*)(s +  0) ) );
    printf("   \t %08x \n", * ( (int*)(s +  4) ) );
    printf("   \t %08x \n", * ( (int*)(s +  8) ) );
    printf("   \t %08x \n", * ( (int*)(s + 12) ) );
}

void hack(void)
{
    printf("I've been hacked!\n");
    system("ls -l");
    printf("Done!\n");
}

int main(void)
{
    printf("Address of print_it: \t %p \n", print_it);
    printf("Address of hack:     \t %p \n", hack);

    print_it();

    return 0;
}
```

Just increase the size of the data you feed to the application until it crashes. Then, the buffer size becomes known!

Corrected Code

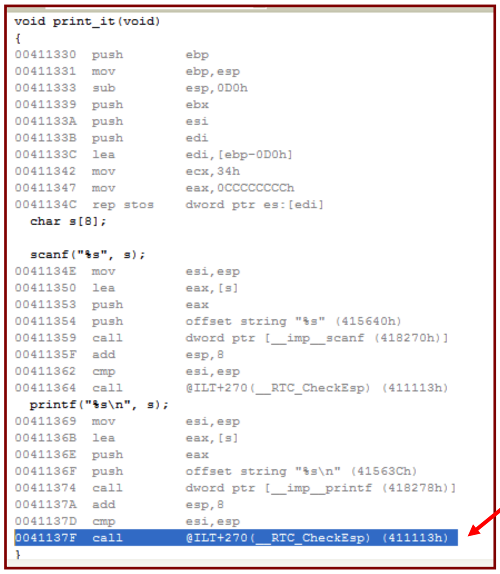


```c
#include <stdio.h>

void print_it(void)
{
    char s[8];

    scanf("%7s", s);
    printf("%s\n", s);
}

int main(void)
{
    print_it();

    return 0;
}
```

Do not use "unsecure" functions!!!
Be very careful!

gets()/fgets()
scanf()/fscanf()
strcat()/strcpy()
…

Some compilers are getting “smarter”...



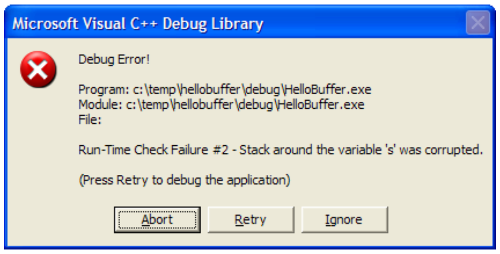
```
void print_it(void)
{
00411330  push        ebp
00411331  mov         ebp,esp
00411333  sub         esp,0D0h
00411339  push        ebx
0041133A  push        esi
0041133B  push        edi
0041133C  lea         edi,[ebp-0D0h]
00411342  mov         ecx,34h
00411347  mov         eax,0CCCCCCCCh
0041134C  rep stos    dword ptr es:[edi]
  char s[8];

  scanf("%s", s);
0041134E  mov         esi,esp
00411350  lea         eax,[s]
00411353  push        eax
00411354  push        offset string "%s" (415640h)
00411359  call        dword ptr [__imp__scanf (418270h)]
0041135F  add         esp,8
00411362  cmp         esi,esp
00411364  call        @ILT+270(__RTC_CheckEsp) (411113h)
  printf("%s\n", s);
00411369  mov         esi,esp
0041136B  lea         eax,[s]
0041136E  push        eax
0041136F  push        offset string "%s\n" (41563Ch)
00411374  call        dword ptr [__imp__printf (418278h)]
0041137A  add         esp,8
0041137D  cmp         esi,esp
0041137F  call        @ILT+270(__RTC_CheckEsp) (411113h)
}
```

Microsoft Visual Studio 2005

Check if the stack has been corrupted!

17

Some compilers are getting “smarter”...



Debug Error!

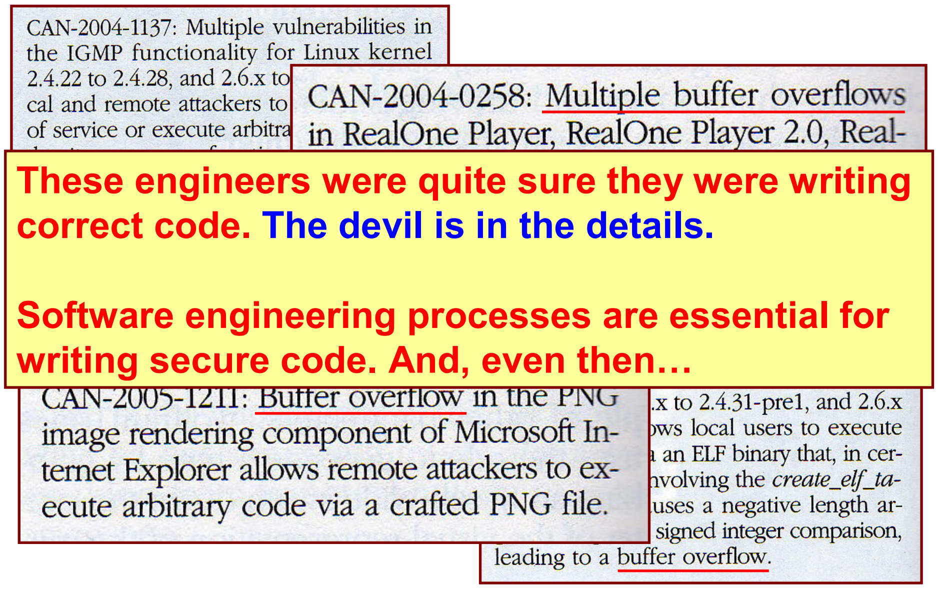
Program: c:\temp\hellobuffer\debug\HelloBuffer.exe
Module: c:\temp\hellobuffer\debug\HelloBuffer.exe
File:

Run-Time Check Failure #2 - Stack around the variable 's' was corrupted.

(Press Retry to debug the application)

18

So…



CAN-2004-1137: Multiple vulnerabilities in the IGMP functionality for Linux kernel 2.4.22 to 2.4.28, and 2.6.x to
cal and remote attackers to
of service or execute arbitra

CAN-2004-0258: Multiple buffer overflows in RealOne Player, RealOne Player 2.0, Real-

These engineers were quite sure they were writing correct code. The devil is in the details.

Software engineering processes are essential for writing secure code. And, even then…

CAN-2005-1211: Buffer overflow in the PNG image rendering component of Microsoft Internet Explorer allows remote attackers to execute arbitrary code via a crafted PNG file.

.x to 2.4.31-pre1, and 2.6.x
ows local users to execute
a an ELF binary that, in cer-
nvolving the *create_elf_ta-*
uses a negative length ar-
signed integer comparison,
leading to a buffer overflow.

So, why Security?

- Applications nowadays run on the "Wild-West" of the Internet, they will be attacked
- Security problems are expensive to fix!
  - Cost of code patching (developing, documenting, testing, re-designing)
  - Cost of redistributing
  - Loss of credibility
  - Loss of revenue
  - Loss of productivity
  - Possible lawsuits

«The Microsoft Security Response Center believes a security bug that requires a security bulletin costs in the neighborhood of $100.000.»
[Howard2002]

Software Analysis

Security
2. Real security attacks

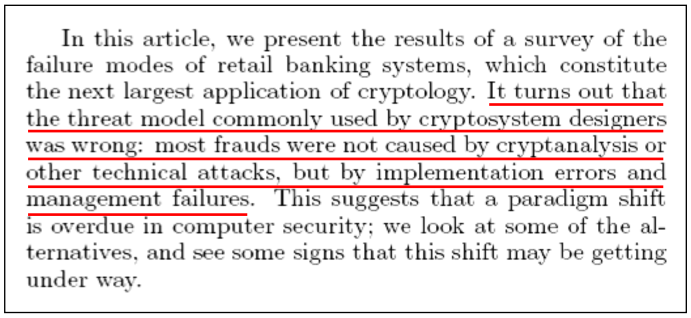
Mar 2008

Jonathan Aldrich
Carnegie Mellon University

Slides by Paulo Marques, University of Coimbra, Portugal

Why Cryptosystems Fail

Ross Anderson, "**Why Cryptosystems Fail**", in Proc. of the 1st ACM conference on Computer and Communications Security, Fairfax, Virginia, United States, 1993



In this article, we present the results of a survey of the failure modes of retail banking systems, which constitute the next largest application of cryptology. It turns out that the threat model commonly used by cryptosystem designers was wrong: most frauds were not caused by cryptanalysis or other technical attacks, but by implementation errors and management failures. This suggests that a paradigm shift is overdue in computer security; we look at some of the alternatives, and see some signs that this shift may be getting under way.

22

Some Examples *(Inside Jobs)*

Many frauds are carried out with some inside knowledge or access, and ATM fraud turns out to be no

In a recent case, a housewife from Hastings, England, had money stolen from her account by a bank clerk who issued an extra card for it. The

Technical staff also steal clients' money, knowing that complaints will probably be ignored. At one bank in Scotland, a maintenance engineer fitted an ATM with a handheld computer, which recorded customers' PINs and account numbers. He then made up counterfeit cards and looted their accounts [C1] [C2]. Again, customers who complained were stonewalled; and the bank was

One bank had a well managed system, in which the information systems, electronic banking and internal audit departments cooperated to enforce tight dual control over unissued cards and PINs in the branches. This kept annual theft losses down, until one day a protegé of the deputy managing director sent a circular to all branches announcing that to cut costs, a number of dual control procedures were being abolished, including that on cards and PINs. This was done without consultation, and without taking any steps to actually save money by reducing staff. Losses increased tenfold; but managers in the affected departments were unwilling to risk their careers by making a fuss. This seems to be a typical example of how computer security breaks down in real organisations.

Some Examples (*Outside Jobs*)

In a recent case at Winchester Crown Court in England [RSH], two men were convicted of a simple but effective scam. They would stand in ATM queues, observe customers' PINs, pick up the discarded ATM tickets, copy the account numbers from the tickets to blank cards, and use these to loot the customers' accounts.

This trick had been used (and reported) several years previously at a bank in New York. There the culprit was an ATM technician, who had been fired, and who managed to steal over $80,000 before the bank saturated the area with security men and caught him in the act.

These attacks worked because the banks printed the full account number on the ATM ticket, and because there was no cryptographic redundancy on the magnetic strip. One might have thought

The fastest growing modus operandi is to use false terminals to collect customer card and PIN data. Attacks of this kind were first reported from the USA in 1988; there, crooks built a vending machine which would accept any card and PIN, and dispense a packet of cigarettes. They put their invention in a shopping mall, and harvested PINs and magnetic strip data by modem. A more recent instance of this in Connecticut got substantial press publicity [J2], and the trick has spread to other countries too: in 1992, criminals set up a market stall in High Wycombe, England, and customers who wished to pay for goods by credit card were asked to swipe the card and enter the PIN at a terminal which was in fact hooked up to a PC. At the time of writing, British banks had still not warned their customers of this threat.

Some Examples (*Technical*)

Another technical attack relies on the fact that most ATM networks do not encrypt or authenticate the authorisation response to the ATM. This means that an attacker can record a 'pay' response from the bank to the machine, and then keep on replaying it until the machine is empty. This technique, known as 'jackpotting', is not limited to outsiders - it appears to have been used in 1987 by a bank's operations staff, who used network control devices to jackpot ATMs where accomplices were waiting.

Another bank's systems had the feature that when a telephone card was entered at an ATM, it believed that the previous card had been inserted again. Crooks stood in line, observed customers' PINs, and helped themselves. This shows how even the most obscure programming error can lead to serious problems.

Test transactions have been another source of trouble. There was a feature on one make of ATM which would output ten banknotes when a fourteen digit sequence was entered at the keyboard. One bank printed this sequence in its branch manual, and three years later there was a sudden spate of losses. These went on until all the banks using the machine put in a software patch to disable the transaction.

One small institution issued the same PIN to all its customers, as a result of a simple programming error. In yet another, a programmer arranged things so that only three different PINs were issued, with a view to forging cards by the thousand. In neither case was the problem detected until some considerable time had passed: as the

Some Examples (*Technical*) (2)

Programmers at one bank did not even go to the trouble of setting up master keys for its encryption software. They just directed the key pointers to an area of low memory which is always zero at system startup. The effect of this was that the live and test systems could use the same cryptographic key dataset, and the bank's technicians found that they could work out customer PINs on their test equipment. Some of them used to

Some brands of security module make particular attacks easier. Working keys may, for example, be generated by encrypting a time-of-day clock and thus have only 20 bits of diversity rather than the expected 56. Thus, according to probability theory, it is likely that once about 1,000 keys have been generated, there will be two of them which are the same. This makes possible a number of

When the bank wished to establish a zone key with VISA using their security module, they found they had no terminal which would drive it. A contractor obligingly lent them a laptop PC, together with software which emulated a VT100. With this the various internal auditors, senior managers and other bank dignitaries duly created the required zone keys and posted them off to VISA.

However, none of them realised that most PC terminal emulation software packages can be set to log all the transactions passing through, and this is precisely what the contractor did. He captured the clear zone key as it was created, and later used it to decrypt the bank's PIN key. Fortunately for them (and VISA), he did this only for fun and did not plunder their network

We know of cases where a bank subcontracted all or part of its ATM system to a 'facilities management' firm, and gave this firm its PIN key. There have also been cases where PIN keys have been shared between two or more banks. Even if all bank staff could be trusted, outside firms may not share the banks' security culture: their staff are not always vetted, are not tied down for life with cheap mortgages, and are more likely to have the combination of youth, low pay, curiosity and recklessness which can lead to a novel fraud being conceived and carried out.

Quite Serious!

Cryptanalysis may be one of the less likely threats to banking systems, but it cannot be completely ruled out.

Security by Obscurity

Since world war two, a curtain of silence has descended on government use of cryptography. This is not surprising, given not just the cold war, but also the reluctance of bureaucrats (in whatever organisation) to admit their failures. But it does put the cryptosystem designer at a severe disadvantage compared with engineers working in other disciplines; the post-war years are precisely the period in which modern cryptographic systems have been developed and brought into use. It is as if accident reports were only published for piston-engined aircraft, and the causes of all jet aircraft crashes were kept a state secret.

Misunderstanding Technology

Underlying many of these control failures is poor design psychology. Bank branches (and computer centres) have to cut corners to get the day's work done, and only those control procedures whose purpose is evident are likely to be strictly observed. For example, sharing the branch safe keys between the manager and the accountant is well understood: it protects both of them from having their families taken hostage. Cryptographic keys are often not packaged in as user-friendly a way, and are thus not likely to be managed as well. Devices which actually look like keys (along the lines of military crypto ignition keys) may be part of the answer here.

Wrong Risk Model

The basic assumptions behind this program are that implementation and operation will be essentially error-free, and that attackers will possess the technical skills which are available in a government signals security agency. It would therefore seem to be more relevant to military than civilian systems, although we will have more to say on this later.

Human Factor

Corporate politics can have an even worse effect, as we saw above: even where technical staff are aware of a security problem, they often keep quiet for fear of causing a powerful colleague to lose face.

Software Analysis

Security
3. Basic Concepts

Mar 2008

Jonathan Aldrich
Carnegie Mellon University

Slides by Paulo Marques, University of Coimbra, Portugal

Security – Basic Aspects

- **Authentication**: Mechanisms that allow to identify a user in the system
- **Authorization**: Guaranteeing that only certain authenticated users can perform certain operations
- **Protection**: Guaranteeing that certain resources are not used by non-authorized users
- **Security Management**: What policies are in place for managing security concerns; how are security breaches handled?

29

Protection – Main Concerns

- **Confidentiality**:
  - Protection against revealing secrets
- **Integrity**:
  - Protection against tampering with data
- **Non-Repudiation**:
  - Protection against denying actions previously performed
- **Denial of Service**:
  - Protection against a service being unavailable for legitimate users
- **Software Faults**:
  - Protection against undue access because of coding errors
- **Physical Protection**:
  - Protection against systems being physically compromised

30

Some Typical Examples

Concern	Ways to Address it (Examples)
Authentication	Username/Password; Kerberos
Authorization	Access Control Lists (ACL); Permission Tokens
Protection	
Confidentiality	Data Encryption
Integrity	Digital Signatures
Non Repudiation	Digital Signatures
Denial of Service Attacks	Dynamic Firewalls; "intelligent" *routers*
Software Faults	Software Engineering; Proofs
Physical Protection	Locked doors; Encryption
Security Management	Security Policies; Service Level Agreements

This is not a security course, thus we will not address these issues in detail.

Check Your Understanding (1)

Provide examples of each concept for Amazon.com:

- **Authentication**
- **Authorization**
- **Confidentiality**
- **Integrity**
- **Non-Repudiation**
- **Denial of Service**

Security is not something you “add” to a system. Security is something that must be considered from the beginning, during requirement gathering and analysis!

Fundamentals

- Any security analysis starts by considering that an intelligent hostile entity, which appropriate resources, will analyze and try to attack the system
- A security analysis is always a cost analysis:
  - A number of intelligent motivated attackers, with time and a potential infinite set of resources will be able to attack my system. Thus:
  - **How much does it cost for the attacker to break into my system?**
  - **How much time it take to break into the system?**
  - **What do I lose if the attacker is able to break in?**

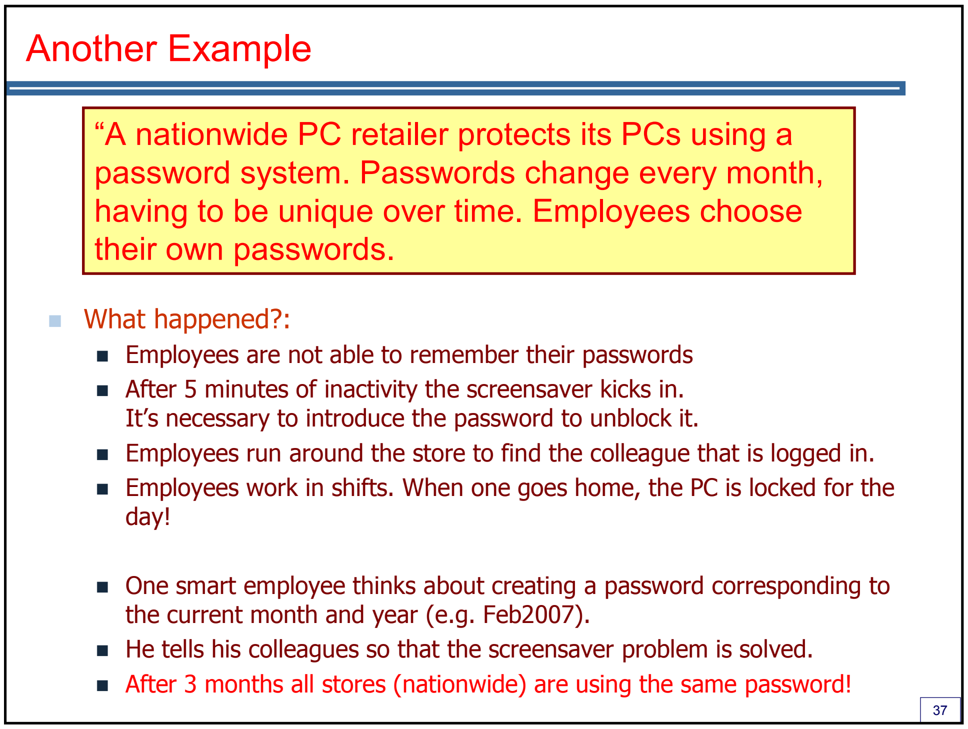
Fundamentals (2)

- Every time a security system is deployed, it has to consider an attack/risk model
- A security analysis must always make these models explicit!
  - Ensuring that you are considering the right problem
  - Ensuring that the correct resources are projected
  - Ensuring that the system is not "over-designed"

35

Human-Computer Interaction is Critical

36



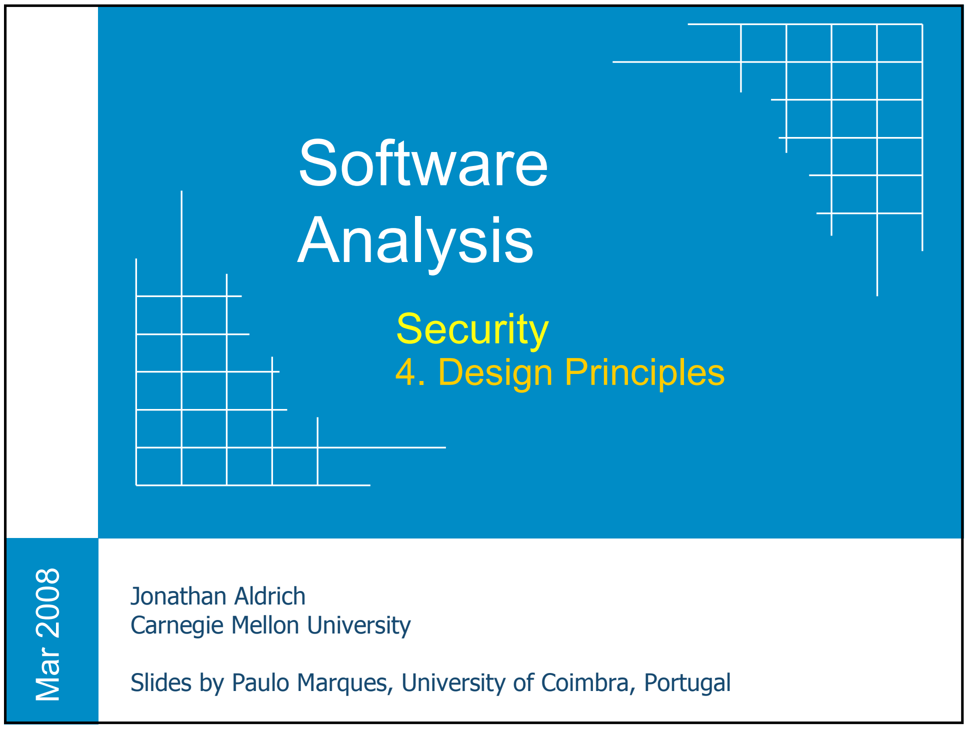
Another Example

“A nationwide PC retailer protects its PCs using a password system. Passwords change every month, having to be unique over time. Employees choose their own passwords.

- What happened?:
  - Employees are not able to remember their passwords
  - After 5 minutes of inactivity the screensaver kicks in.
    It’s necessary to introduce the password to unblock it.
  - Employees run around the store to find the colleague that is logged in.
  - Employees work in shifts. When one goes home, the PC is locked for the day!

  - One smart employee thinks about creating a password corresponding to the current month and year (e.g. Feb2007).
  - He tells his colleagues so that the screensaver problem is solved.
  - After 3 months all stores (nationwide) are using the same password!

37



Software Analysis

Security
4. Design Principles

Mar 2008

Jonathan Aldrich
Carnegie Mellon University

Slides by Paulo Marques, University of Coimbra, Portugal

Security Principles to Live By

1. Establish a security process
2. Define the product security goals
3. Consider security as a product feature
4. Learn from mistakes
5. Use the "principle of least privilege"
6. Use "defense in depth"
7. Assume external systems are insecure
8. Plan on Failure
9. Fail to a secure mode
10. Employ secure defaults
11. Remember (*security features* != *secure features*)
12. Never depend on security through obscurity

39

1. Establish a Security Process

- Your software development process should contemplate how to:
  - Design, Code, Test, Deploy and Fix systems in a secure way
- If not…
  - The system will never be acceptably secure
  - Enormous amounts of time and money will be spent in trying to fix the system each time a security problem occurs

40

2. Define the Product Security Goals

Requirements Gathering

- Who is the application audience?
  - Anonymous internet users? Bank managers? Sys-admins?
- What does security means for the target audience?
  - Do different users have different requirements?
- In what environment will the application run?
  - Internet? Cell-phone? Bank? Thousands of uncontrolled PCs?
- What are you attempting to protect?
  - Confidentiality? Integrity? Non-repudiation?

41

2. Define the Product Security Goals (2)

Requirements Gathering

- What are the implications for the users if the objects you are trying to secure are compromised?
  - Lost of privacy? Loss of money? Annoyance?
- Who will manage the application?
  - The user? The system administrator?
- What are the communication needs of the application?
  - Talk to other applications of the company? The internet? Both?
- What security infrastructure services does the system and environment provides?
  - Hardware cryptograph? OS-based user management? Steel doors?

42

3. Consider security as a product feature

- What do you think of this plan?

Milestone 0:	Design Complete
Milestone 1:	Add core features
Milestone 2:	Add secondary features
Milestone 3:	Add security
Milestone 4:	Fix bugs
Milestone 5:	Ship Product

- Key issues:
  - It’s almost impossible to retrofit a system to make it secure
  - Properly including security is extremely expensive
  - Security bug fixing will be expensive and time consuming
- Make sure that security is a design consideration in your product.
  - Specifications and Designs must outline the security requirements and threats to the system

4. Learn from Mistakes

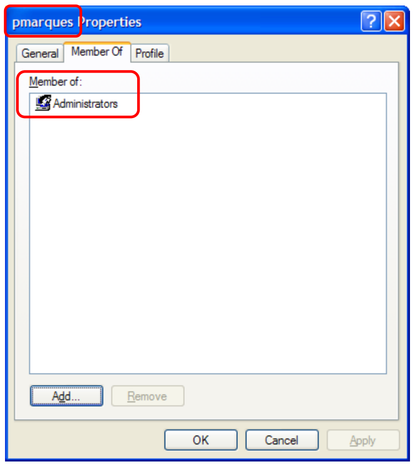
« What doesn’t kill you makes you stronger. »

- Build up the knowledge in the organization and team.
- Put procedures in place that make sure that identified security problems don’t happen again (hopefully)
- What to ask
  - How did the security error occur?
  - Is the same error replicated in other parts of the code?
  - How could we have prevented this error from occurring?
  - How can we make sure this will never happen again?

5. Use Least Privilege

« An application should always execute with the minimum privileges possible. »

- Running with the minimum privileges possible makes it much harder for a security attach to occur
  - ONE bug in a Windows application that runs in an administrator account is all that's needed for the machine to be compromised.



Why do this?

45

6. Use "defense in depth"

Defense in Depth principle:

Imagine that your application is the last one still standing. Every other defense mechanism has been destroyed.

- Induction problem:
  - A programmer assumes that the rest of the system takes care of security for him.
  - If all programmers assume that, who takes cares of security?
- Defense in depth:
  - Reduces the possibility of a Single Point of Failure problem.

46

7. Assume that external systems are insecure

- It's a form of defense in depth
- In particular, assume that data being received or passed to a system where you haven't complete control may be compromised.
  - It may well be!
- Example:

```
st := "SELECT * FROM users WHERE name = '" + userName + "';"
```

In the web form, username is introduced as:
a' or 't'='t

```
st := "SELECT * FROM users WHERE name = 'a' or 't'='t';"
```

8. Plan on Failure

«The only safe computer is a dead computer. Or, at least, a disconnected one. And even so… »

- What happens when a system fails?
- Systems do fail…
  - What happens if the web site is defaced?
  - What happens if someone is able to enter the server room?
  - What happens if the application is compromised?
- An integral part of designing secure systems is planning on how to address failure
  - Plan on how to address the failure
  - Plan on how to correct the problem

9. Fail to a Secure Mode

- What happens when the application detects a problem?
- It's possible to fail securely or insecurely
  - Failing securely means not disclosing any data that it would not normally do; that data still cannot be tempered with; etc.
- The systems should be designed so that they fail securely
- Examples:
  - In Unix, passwords are now encrypted using one-way hash functions (e.g. MD5)
  - In Windows, the password file is further encrypted using a symmetrical encryption algorithm

49

10. Employ Secure Defaults

- An application should install with defaults that are "reasonably" secure for the majority of its use cases
  - E.g. Administrator password defined during installation; Guest account disabled; Firewall enabled; etc.
- An applications should execute with the least possible functionality enabled
  - Something that isn't running cannot be broken in

Very hard to accomplish in practice!

50

11. Security features != Secure features

« My site uses SSL with 1024-bit RSA encryption! »

But the user's credit-card number is stored in a plain-text file…

- One of the most common causes of problems is that developers don't understand security
  - Security is being added as "features"/"technology", not as a way of addressing a particular threat model
  - Developers try to re-implement security mechanisms without proper understanding security
- Companies add "security technology stuff" just because it looks good on the product datasheet

51

12. Never depend on security through obscurity

- Always assume that an attacker knows everything you do
- Assume that an attacker has access to all the source code and design
- Assume that an attacker knows the infrastructure where the application is going to be deployed in
- Why?
  - In most cases is relatively trivial to reverse engineer the needed information from the application and network
  - Time and patience is on the attacker's side
  - There are a lot more potential attackers than people designing and coding the application

52

Check Your Understanding (2)

Provide examples of each concept for Amazon.com:

1. Define the product security goals
2. Learn from mistakes
3. Use the "principle of least privilege"
4. Use "defense in depth"
5. Assume external systems are insecure
6. Plan on Failure
7. Fail to a secure mode
8. Employ secure defaults

53

Software Analysis

Security
5. Threat Based Security Design

Mar 2008

Jonathan Aldrich
Carnegie Mellon University

Slides by Paulo Marques, University of Coimbra, Portugal

Requirement Analysis

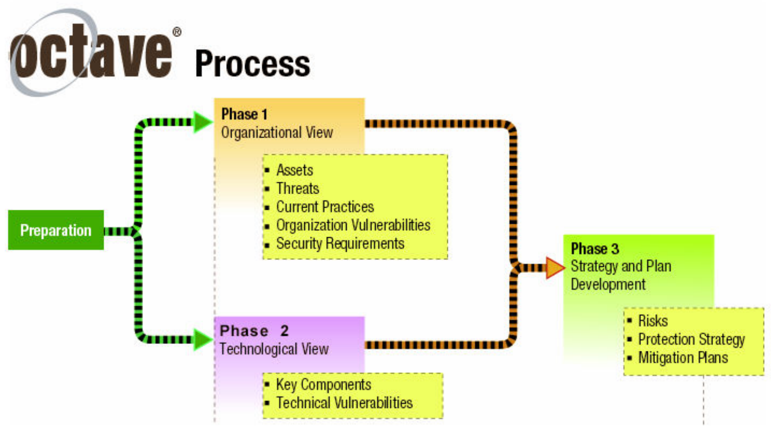
Security Design by Threat Modeling

- Threat Modeling and Analysis in a nutshell:
  1. Brainstorm the known threats to the system
  2. Rank the threats by decreasing risk
  3. Choose how to respond to these threats
  4. Chose techniques to mitigate the threats
  5. Chose appropriate technologies from the identified techniques

More info:
• Horward and LeBand's "*Writing Secure Code*", Chapter 2.
• SEI/CERT OCTAVE: Operationally Critical Threat, Asset, and Vulnerability Evaluation: http://www.cert.org/octave

55

SEI/CERT OCTAVE



C. Alberts et. al., "*An Introduction to the OCTAVE Approach*", CERT/SEI, Carnegie Mellon University, August 2003

56

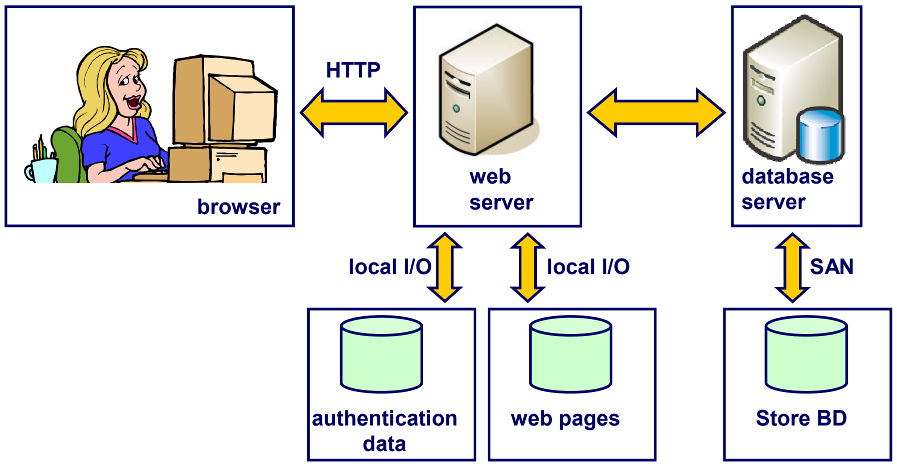
Analysis – What do you want to look at?

- Typically:
  - Core Processes, Services and Servers
  - Persistent Data
  - Non-Persistent (Ephemeral) Data
  - Communication Channels
  - Physical Infrastructure
- Key Criteria:
  - Identify as many security threats as possible
- Approach:
  - Structured Brainstorming Technique
    (Document everything! Make sure to have a scribe!)

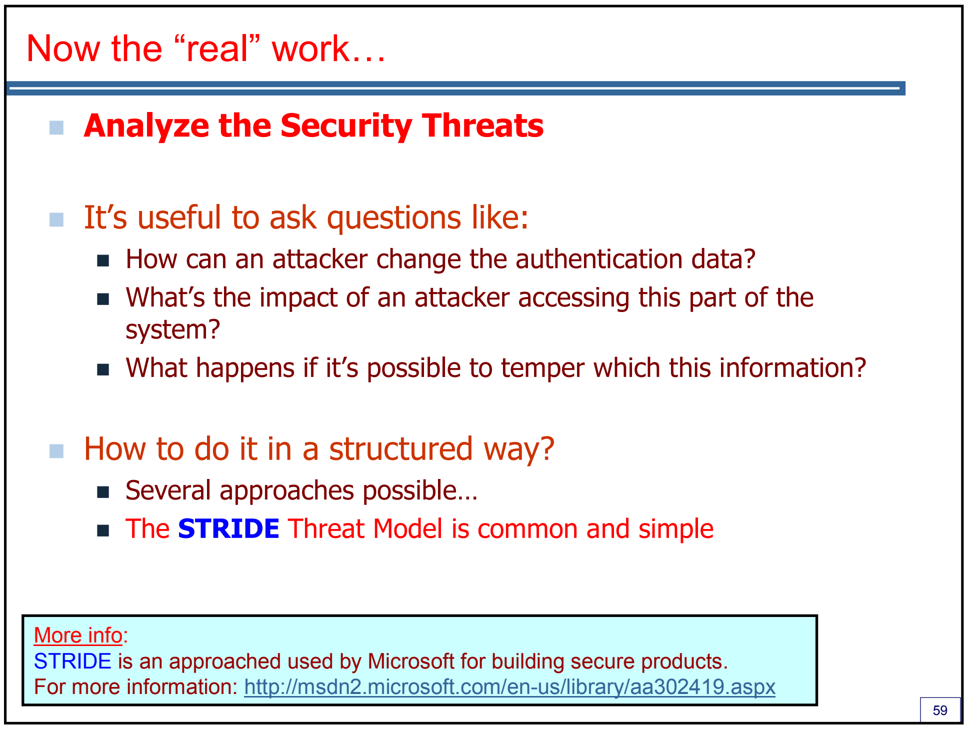
57

1. Analysis

- Start by drawing the overall architecture of the system



58

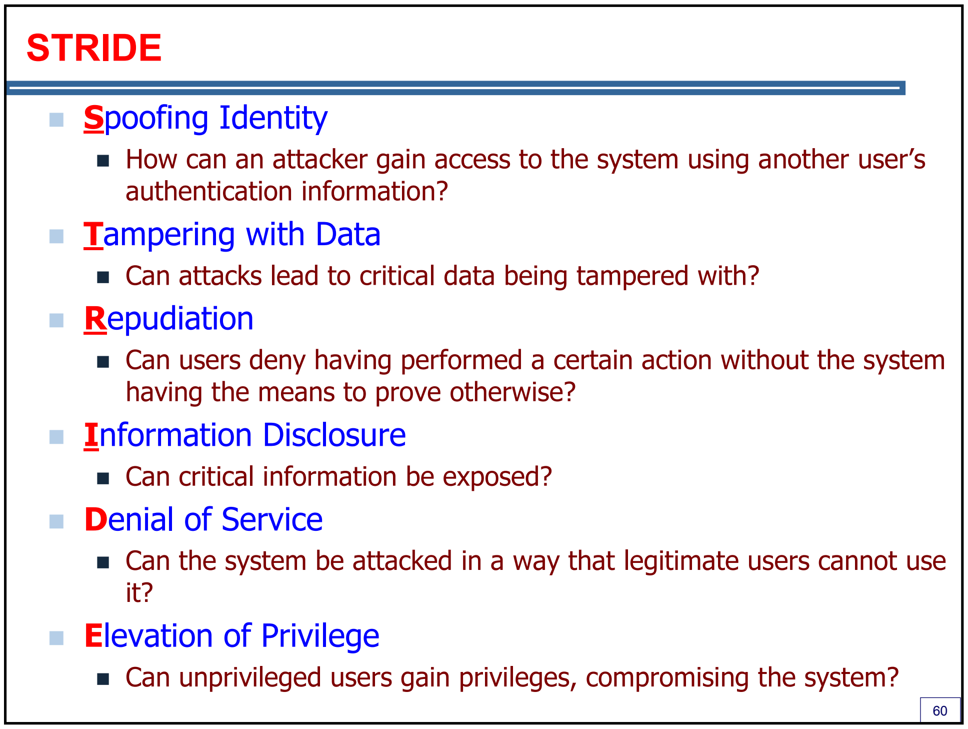


Now the “real” work…

- **Analyze the Security Threats**
- It’s useful to ask questions like:
  - How can an attacker change the authentication data?
  - What’s the impact of an attacker accessing this part of the system?
  - What happens if it’s possible to temper which this information?
- How to do it in a structured way?
  - Several approaches possible…
  - The **STRIDE** Threat Model is common and simple

More info:
STRIDE is an approached used by Microsoft for building secure products.
For more information: http://msdn2.microsoft.com/en-us/library/aa302419.aspx

59



STRIDE

- **S**poofing Identity
  - How can an attacker gain access to the system using another user’s authentication information?
- **T**ampering with Data
  - Can attacks lead to critical data being tampered with?
- **R**epudiation
  - Can users deny having performed a certain action without the system having the means to prove otherwise?
- **I**nformation Disclosure
  - Can critical information be exposed?
- **D**enial of Service
  - Can the system be attacked in a way that legitimate users cannot use it?
- **E**levation of Privilege
  - Can unprivileged users gain privileges, compromising the system?

60

Examples of Threats

- Threat #1
  - A malicious user views or tempers with information on route between the web server and the client browser
- Threat #2
  - A malicious user views or defaces the web site by changing the web page files on the web server
- Threat #3
  - A malicious employee steals credit card numbers by intercepting information flowing on the Storage Area Network
- Threat #4
  - An attacker denies access to the system by flooding the web server with TCP/IP packets (DoS)
- Threat #5
  - An attacker gains access to a large number of passwords by accessing the authentication data files
- …

[Howard05] About 20-40 threats per 2-hour meeting.

61

2. Rank Threats by Decreasing Risk

- Simple approach:

Risk = Impact × Probability

- Use levels, not "fine grain numbers"
  - e.g. 5 levels for impact, 5 levels for probability
- It's hard to judge Impact and Probability
  - What do I lose if this risk materializes?
    (Time, Money, Reputation, etc.)
  - Probability – Rule of thumb:
    It's not encryption: it's bugs and human nature!
- Check the material on the "Risk Management Course"

62

Rank Threats by Decreasing Risk (2)

Risk	Probability	Impact	ID	Descrition
			#1	A malicious user views or tempers with information on route between the web server and the client browser
			#2	A malicious user views or defaces the web site by changing the web page files on the web server
			#3	A malicious employee steals credit card numbers by intercepting information flowing on the Storage Area Network
			#4	An attacker denies access to the system by flooding the web server with TCP/IP packets (DoS)
			#5	An attacker gains access to a large number of passwords by accessing the authentication data files

63

Rank Threats by Decreasing Risk (2)

Risk	Probability	Impact	ID	Descrition
10	5	2	#1	A malicious user views or tempers with information on route between the web server and the client browser
9	3	3	#2	A malicious user views or defaces the web site by changing the web page files on the web server
5	1	5	#3	A malicious employee steals credit card numbers by intercepting information flowing on the Storage Area Network
3	1	3	#4	An attacker denies access to the system by flooding the web server with TCP/IP packets (DoS)
16	4	4	#5	An attacker gains access to a large number of passwords by accessing the authentication data files

64

Rank Threats by Decreasing Risk (3)

Risk	Probability	Impact	ID	Descrition
16	4	4	#5	An attacker gains access to a large number of passwords by accessing the authentication data files
10	5	2	#1	A malicious user views or tempers with information on route between the web server and the client browser
9	3	3	#2	A malicious user views or defaces the web site by changing the web page files on the web server
5	1	5	#3	A malicious employee steals credit card numbers by intercepting information flowing on the Storage Area Network
3	1	3	#4	An attacker denies access to the system by flooding the web server with TCP/IP packets (DoS)

65

3. Choose how to respond to these threats

- Four basic approaches on how to respond to threats
  1. Do Nothing (i.e. assume the risk)
  2. Inform the user of the threat
  3. Remove the problem
  4. Fix the problem
- The decision is mostly managerial. It implies careful considering what are the consequences if a given threat materializes.
- If the risk is high enough, only options 3 and 4 should be used.
  - Many times, it's hard for project managers to opt for 3 (removing an implemented feature from the project) in order to cope with a serious threat

66

4. Chose techniques to mitigate the threats

- How to mitigate the threats identified?

Threat Type	Examples of Mitigation Techniques
Spoofing Identity	Authentication, Protect Secrets, Don't store passwords
Tampering with Data	Authorization, Hashes, Message Authentication Codes, Digital Signatures
Repudiation	Digital Signatures, Audit Trails
Information Disclosure	Authorization, Encryption, Privacy Enhanced Protocols, Don't store secrets
Denial of Service	Back-off protocol, Filtering, Quality of Service
Elevation of Privilege	Run with least privilege, Sandboxes

5. Chose appropriate technologies

Technique	Ways to Address it (Examples)
Authentication	Username/Password; Kerberos; IPSec; X.509 Certificate Authentication; etc.
Authorization	Access Control Lists (ACL); Permission Tokens; Location-Based Authorization; IP-based authorization; etc.
Tampering/Repudiation/Disclosure	
Confidentiality	Data Encryption (RSA, AES, etc.)
Integrity	Digital Signatures (DSA, SHA-RSA.), Hashes (MD5, SHA1, etc.)
Non Repudiation	Digital Signatures
Privacy Enhanced Protocols	SSL, TLS, SSH, IPSec, etc.
Denial of Service Attacks	Progressive Back-off, IP-based filtering, DiffServ, etc.
Elevation of Privilege	*chroot*, OS-containers, virtual machines

Back to the Example

Risk	ID	Descrition	Mitigation
16	#1	An attacker gains access to a large number of passwords by accessing the authentication data files	Only store SHA1 hashes of the passwords; Password file symetrically encrypted (AES) with a key stored on a smart card.
10	#2	A malicious user views or tempers with information on route between the web server and the client browser	Use SSL encrypted connections; Use external signed server certificate.
9	#3	A malicious user views or defaces the web site by changing the web page files on the web server	Web-site directory is read-only to general users and writable only by the webmaster. Use a trip-wire system for early warning.
5	#4	A malicious employee steals credit card numbers by intercepting information flowing on the Storage Area Network	*No storage of credit-card numbers.*
3	#5	An attacker denies access to the system by flooding the web server with TCP/IP packets (DoS)	*Assume Risk*

69

Software Analysis

Security
6. Security Testing

Mar 2008

Jonathan Aldrich
Carnegie Mellon University

Slides by Paulo Marques, University of Coimbra, Portugal

Security Testing

Why is Security Testing Different from Functional Testing?

- Functional testing is about showing to a certain degree of confidence that a feature works as expected.
- Security testing is about demonstrating:
  - That a user cannot spoof another user's identity
  - That a user cannot tamper with data
  - That a user cannot repudiate an action that was performed
  - That a user cannot see data the user shouldn't have access to
  - That a user cannot cause denial of service
  - That a user cannot gain more privileges through malicious use

71

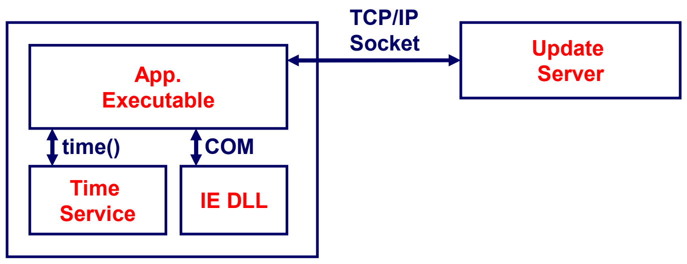
How to test for security problems

1. Decompose the application into its major functional components
2. Identify all component interfaces
3. Rank interfaces by potential vulnerability
4. Ascertain the data used by each interface
5. Find security problems by injecting faulty data

- In most cases tests are designed and performed as black-box testing
- Even so, the tester should have access to the application's source code
  - Why?

72

1. Decompose the application



- Identify all components that make part, directly or indirectly, of the application
  - Executable Files, including DLLs and EXEs
  - Services
  - Files
  - Scripts
  - etc.

73

2. Identify component Interfaces

- Identify all mechanisms by which the application exchange data with the outside world
  - TCP/IP and UDP Sockets
  - Command line arguments
  - Console Input
  - Graphical Input
  - Files
  - Named Pipes
  - Shared Memory
  - Databases
  - COM calls
  - HTTP Requests
  - …

74

3. Rank interfaces according to their vulnerability

- For each one of the interfaces, assign it points according to its vulnerability (*table below*)
- Highly vulnerable interfaces should be tested first and more thoroughly

Characteristic	Points
The process hosting the interface runs on a high privileded account	2
The interface handling data is written in C or C++	1
The interface takes arbitrarily sized buffers or strings	1
The recipient buffer is stack based	2
The interface has no access control list (ACL) or a weak one	1
The interface has good appropriate ACLs	-2
The interface does not require authentication	1
The interface is server-based	1
The feature is installed by default	1
The feature is running by default	1
The feature has already had security vulnerabilities	1

75

4. Ascertain data used by each interface

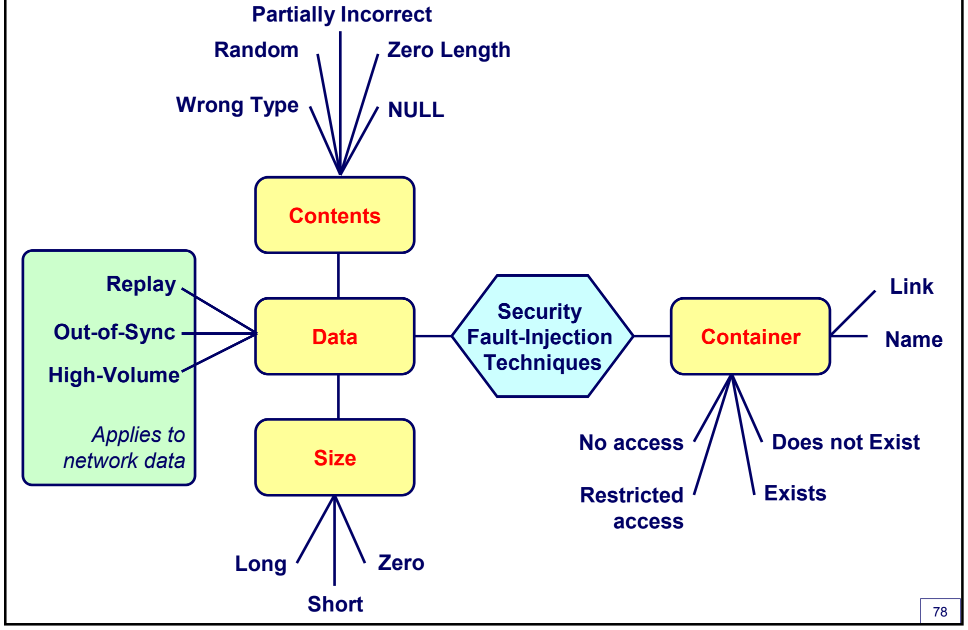
- Determine the data accessed by each interface, where it's coming from, its format, and how it's used
  - This is the data that is modified to exposed security bugs
- Examples
  - Sockets:
    - Data arriving from the network
  - Files:
    - File contents
  - Registry:
    - Registry data
  - Environment:
    - Environment variables
  - HTTP:
    - HTTP headers and HTTP data
  - Command Line Arguments:
    - Parameters

76

5. Find security problems by injecting faulty data

- Fault injection involves perturbing the environment such that the code handling the data that enters the interface behaves in an insecure manner.
- Once a vulnerability is discovered, the objective is to find an exploit that exposes the security fault.
  - E.g. not all buffer overflows are exploitable

Finding Security Vulnerabilities

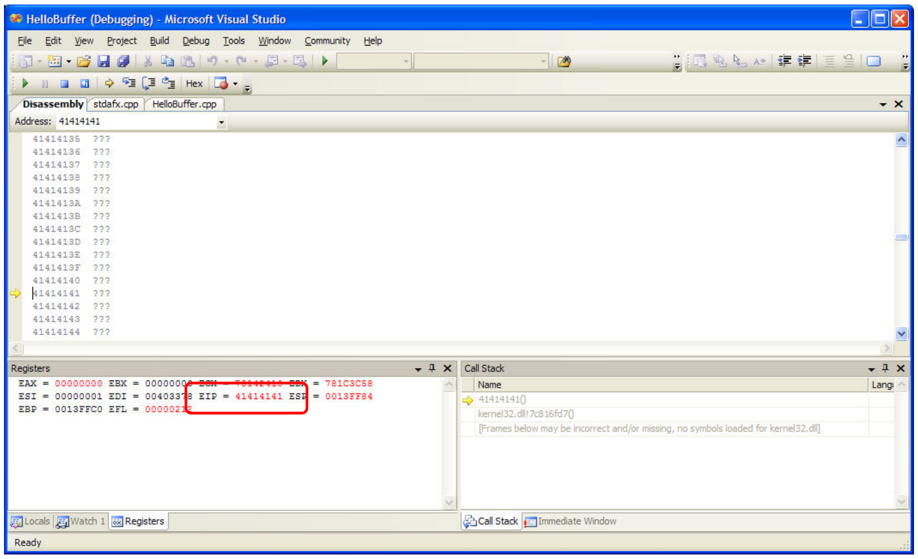


Remember this?

```
#include <stdio.h>

void print_it(void)
{
    char s[8];

    scanf("%s", s);
    printf("%s\n", s);
}

int main(void)
{
    print_it();

    return 0;
}
```

79

Clearly an exploitable bug!


HelloBuffer (Debugging) - Microsoft Visual Studio

Address: 41414141

```
41414135  ???
41414136  ???
41414137  ???
41414138  ???
41414139  ???
4141413A  ???
4141413B  ???
4141413C  ???
4141413D  ???
4141413E  ???
4141413F  ???
41414140  ???
41414141  ???
41414142  ???
41414143  ???
41414144  ???
```

Registers

```
EAX = 00000000 EBX = 00000000 ECX = [illegible] EDX = 781C3C58
ESI = 00000001 EDI = 00403378 EIP = 41414141 ESP = 0013FF84
EBP = 0013FFC0 EFL = 00000212
```

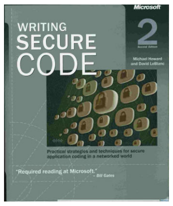
Call Stack

Name	Lang
41414141()	
kernel32.dll!7c816fd7()	
[Frames below may be incorrect and/or missing, no symbols loaded for kernel32.dll]	

Ready

80

Bibliography



- **Writing Secure Code, Second Edition**
  by Michael Howard, David C. LeBlanc
  - Microsoft Press, ISBN 0735617228
    December 2002
  - Chapters 1, 2, 3 and 14

- R. Anderson, "**Why Cryptosystems Fail**", in Proc. of the 1st ACM conference on Computer and Communications Security, Fairfax, Virginia, United States, 1993

- E. Nisley, "**Strong Language**", in Dr. Dobb's Journal, CMP Publishers, October 2005

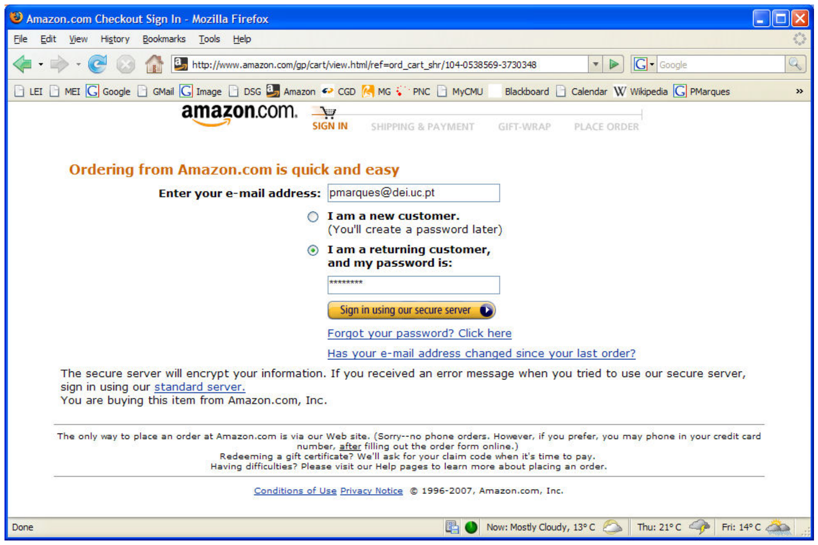
- Microsoft's **STRIDE** method:
  http://msdn2.microsoft.com/en-us/library/aa302419.aspx

- CERT/SEI **OCTAVE**:
  http://www.cert.org/octave

81

82

Am I safe?


Amazon.com Checkout Sign In - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://www.amazon.com/gp/cart/view.html/ref=ord_cart_shr/104-0538569-3730348
LEI MEI Google GMail Image DSG Amazon CGD MG PNC MyCMU Blackboard Calendar Wikipedia PMarques
SIGN IN SHIPPING & PAYMENT GIFT-WRAP PLACE ORDER

Ordering from Amazon.com is quick and easy

Enter your e-mail address: pmarques@dei.uc.pt

○ I am a new customer. (You'll create a password later)

[✓ I am a returning customer, and my password is:] ********

Sign in using our secure server

Forgot your password? Click here

Has your e-mail address changed since your last order?

The secure server will encrypt your information. If you received an error message when you tried to use our secure server, sign in using our standard server.
You are buying this item from Amazon.com, Inc.

The only way to place an order at Amazon.com is via our Web site. (Sorry--no phone orders. However, if you prefer, you may phone in your credit card number, after filling out the order form online.)
Redeeming a gift certificate? We'll ask for your claim code when it's time to pay.
Having difficulties? Please visit our Help pages to learn more about placing an order.

Conditions of Use Privacy Notice © 1996-2007, Amazon.com, Inc.