Secure coding training

Secure development on all levels

Gerard Frankowski, PSNC

Poznań, 22-23 June 2010

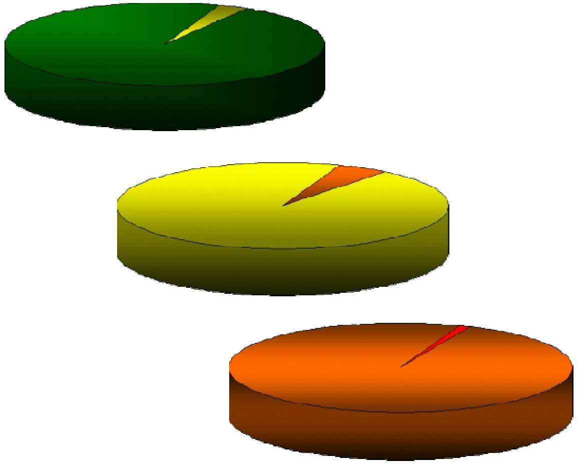
connect • communicate • collaborate

The goal of this presentation

- A smooth introduction to the training
  - Why secure programming?
  - Cost of software bugs
  - Some spectacular programming mistakes
  - What levels (layers) should be differentiated when talking about secure applications
    - ***Technologies*** *(the main subject of the Day 1)*
    - *Functionality*
    - *Configuration*
    - ***Coding patterns*** *(the main subject of the Day 2)*
- This presentation is intended also for the project leaders
  - Let your people have resources to consume on learning secure programming!

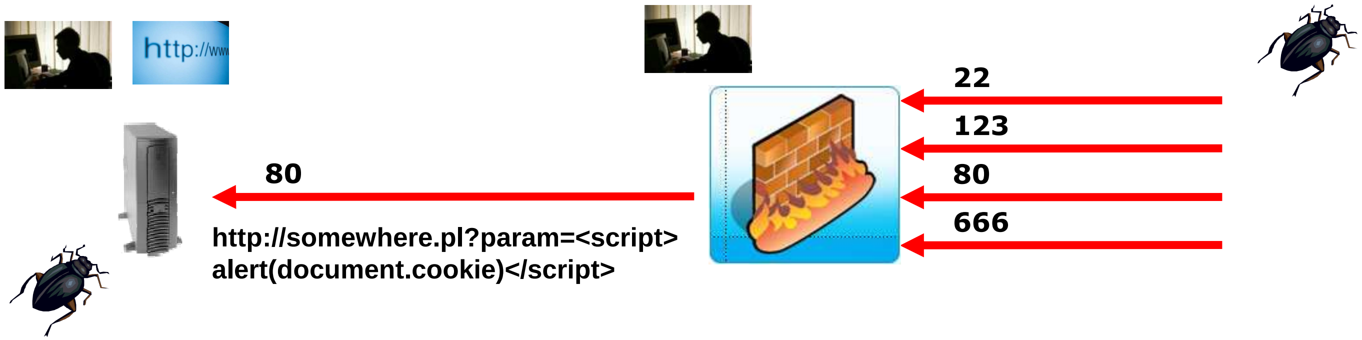
Why the code is insecure?

- We are not robots, we make mistakes
  - Let the software be 30 000 KLOC (30 millions lines) long
    - *Windows 2000 was of that size*
  - According to the Carnegie Mellon University’s CyLab, 1 KLOC (1000 lines of code) contains up to 30 software bugs
  - Let’s make some further assumptions:
    - *20 software bugs (of all kinds) in 1 KLOC*
    - *only 5% of them are security-related*
    - *only 1% of the latter give system access*
  - 30 000 000 * 0.02 * 0.05 * 0.01 = 300
  - The attacker needs to find only 1 out of those 300…



Why the code does matter?

- It is the administrator who should take care of the system security, isn't it?
  - Appropriate server configuration
  - Firewall policies…



80

http://somewhere.pl?param=<script>
alert(document.cookie)</script>

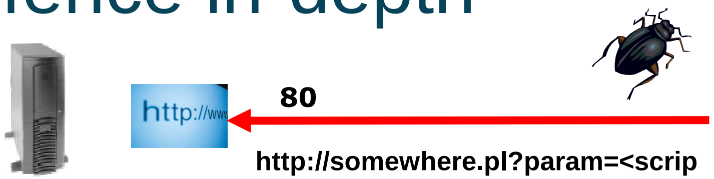
22

123

80

666

- The response must be “defence in-depth”
  - We defend on every level



80

http://somewhere.pl?param=<scrip
t>alert(document.cookie)</script>

But are the software bugs expensive?

- NIST Report ”The Economic Impacts of Inadequate Infrastructure for Software Testing” (2002)
  - http://www.nist.gov/director/planning/upload/report02-3.pdf
  - This is more general, not only security bugs
    - *But they are bugs as well…*

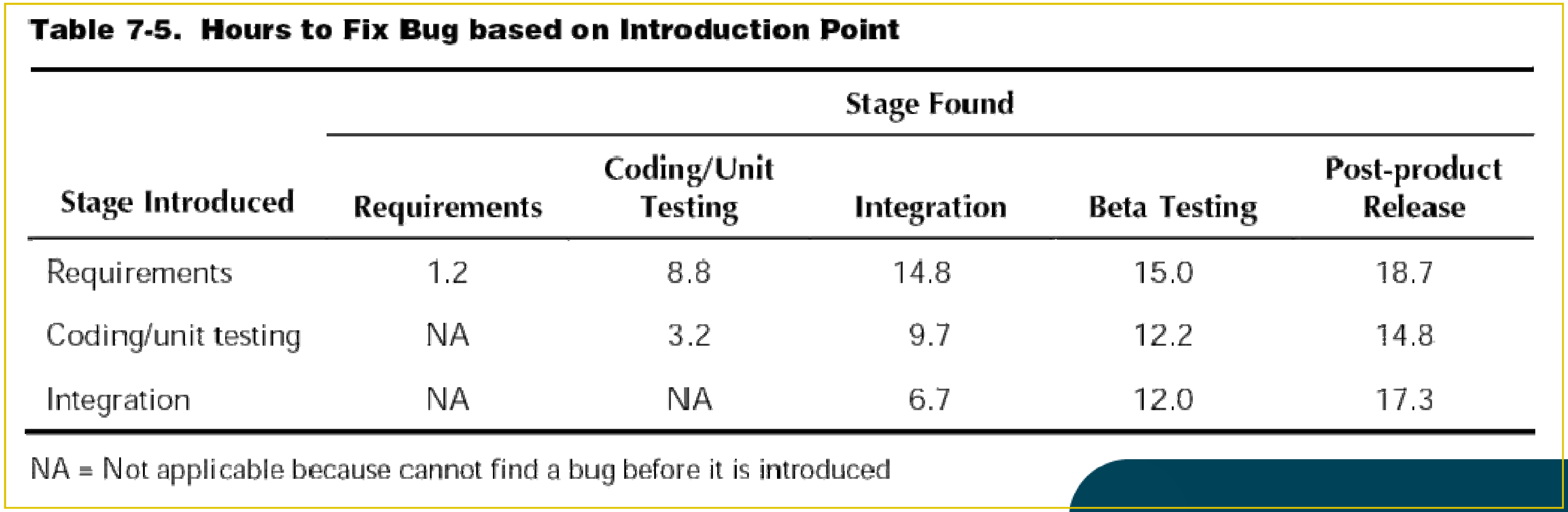
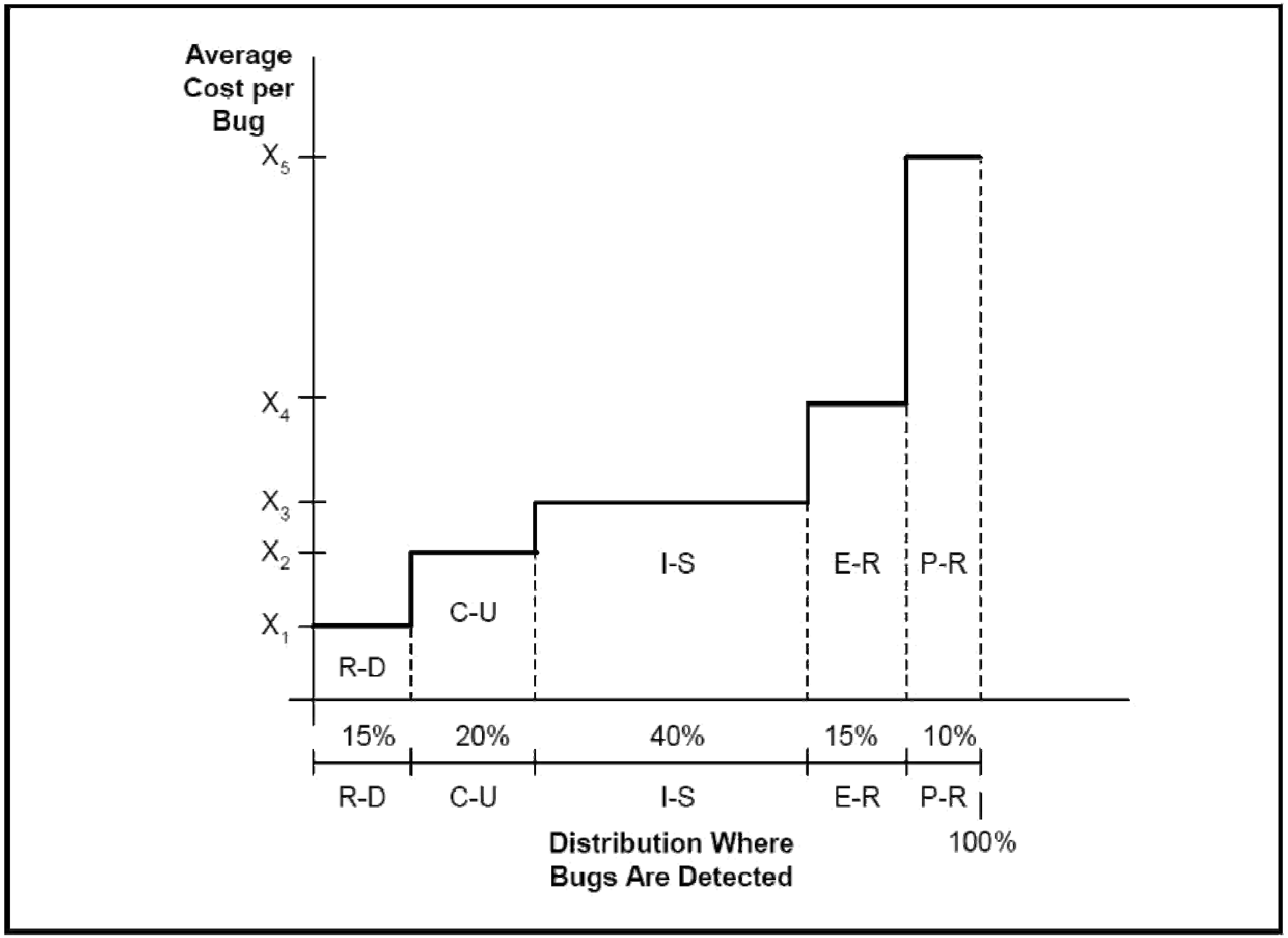


Table 7-5. Hours to Fix Bug based on Introduction Point

	Stage Found				
Stage Introduced	Requirements	Coding/Unit Testing	Integration	Beta Testing	Post-product Release
Requirements	1.2	8.8	14.8	15.0	18.7
Coding/unit testing	NA	3.2	9.7	12.2	14.8
Integration	NA	NA	6.7	12.0	17.3

NA = Not applicable because cannot find a bug before it is introduced

Another chart from the NIST report



Legend:
R-D: Requirements Gathering and Analysis/Architectural Design
C-U: Coding/Unit Test
I-S: Integration and Component/RAISE System Test
E-R: Early Customer Feedback/Beta Test Programs
P-R: Post-product Release

Were there any serious software bugs?

- Therac25 equipment
- Ariane 5 rocket
- Air-Traffic Control System in LA
- Mars exploration problems
- More:
  - http://computingcases.org/case_materials/therac/therac_case_intro.html
  - http://www.cse.lehigh.edu/~gtan/bug/softwarebug.html
- The specifics of security bugs
  - Example: MS Blaster (Lovesan)

Therac 25

- Cancer treatment with radiotherapy
- 1985-1987: many radiation overdosing, at least 5 people died
- Numerous errors in control software
  - Race condition error – if an operator worked too fast (!) several parameters could not be initialized properly
  - Source code from the previous version was applied, but the older version had additional hardware protections
  - Overflow error: a flag was incremented and accidentaly zeroed
  - No one thought about an independent code review

Ariane 5

- European Space Agency project for putting satellites into Earth orbit
  - 10 years, 7.000.000.000 USD
- 36.7 seconds after the launch, the first Ariane rocket crashed into the ground
  - The rocket cost: 500M USD
- Overflow error
  - 64-bit value (sideways velocity of the rocket) was intended to be saved into 16-bit variable
  - An error occured in the primary working unit
  - The secondary unit took control, but the same error occurred

NASA Mars Climate Orbiter

- A key part of NASA Mars exploration program
  - Worth 125M USD
- In 1999 it went too low and was never heard again
- Lack of conversion check
  - Two unit systems were used in the orbiter software: metric and English
  - In the routines managing the orbit change, someone forgot to assure appropriate conversion check
  - Too little tests
- Another Mars probe lost due to software errors
  - Mars Global Surveyor (2006, a series of errors started with two bad memory addresses)

Air-Traffic control system, LA

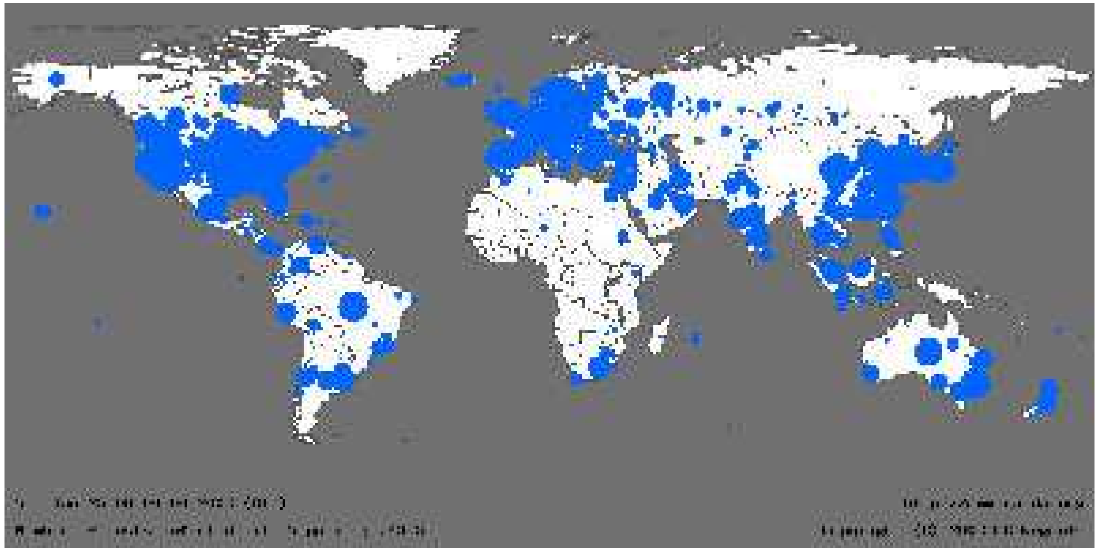
- In 2004, air-traffic control system in LA lost voice contact with about 400 airplanes
  - Happily, no one suffered – currently there are onboard anti-collision systems like TCAS
- The system unexpectedly shut down
  - Control unit contained a counter which measured milliseconds
  - The greatest number that could be stored in the system, was 2^{32} (2^{32} milliseconds is ca. 50 days)
  - There were special procedures to reset the software every 30 days, which apparently had not been done that time

Security bugs

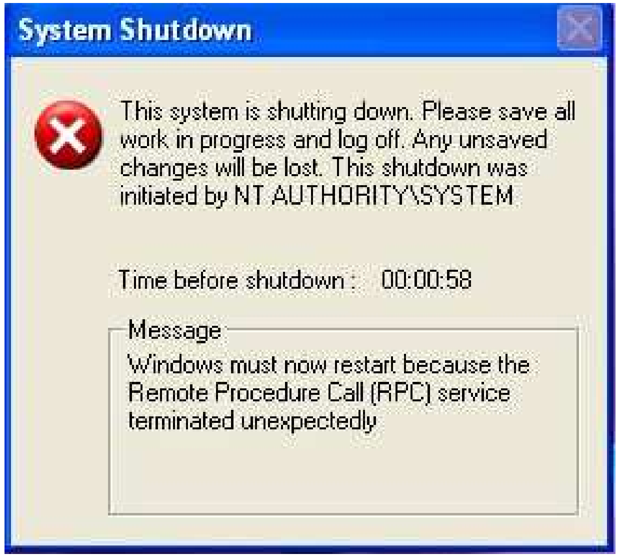
- Security bugs have slightly different nature
  - Usually, they are more oriented on stealing data, money, research results
  - However, accessibility is one of the security facets…
- On the other hand, they introduce other threats
  - Loses on trust
  - May even kill whole enterprises
    - *Blue Security and Blue Frog antispam system case (2006)*

MS Blaster (Lovesan)

- An example of spectacular security bug
  - Microsoft DCOM RPC buffer overrun (found by our PSNC colleagues)
- MS Blaster virus (2003)
  - Ca. 25M computers infected (2005)
  - Direct loses: 1.500.000.000 USD



http://www.security.nl/image/246



System Shutdown

This system is shutting down. Please save all work in progress and log off. Any unsaved changes will be lost. This shutdown was initiated by NT AUTHORITY\SYSTEM

Time before shutdown : 00:00:58

Message

Windows must now restart because the Remote Procedure Call (RPC) service terminated unexpectedly

http://pl.wikipedia.org

Top excuses for not writing secure code

- No one will do that!
- Why would anyone do that?
- We’ve never been attacked
- We’re secure – we use cryptography
- We’re secure – we use ACLs
- We’re secure – we use a firewall
- We’ve reviewed the code, and there are no security bugs
- We know it’s the default, but the administrator can turn it off
- If we don’t run as administrator, stuff breaks

Should we care if the software will never be perfect?

- We should care – there is security economy!
  - The economic factors become more meaningful for everyone, including attackers
  - Your system is in danger when

Attack cost <= Value of your data

- Therefore we should make the attacker’s goal more difficult
  - Better security systems
  - Less software errors

Should the developer be security specialist then?

- Obviously not – everyone has got an own job
  - We do not expect the developers to learn about network attacks or exploiting vulnerabilities
- But we think we can expect the developers…
  - To know the basics of secure coding (including simple examples of attacks for better understanding)
  - To apply secure coding practices in their favourite programming language (or the one they have to work with)
  - To create well-commented and easy-to-understand code
  - To apply simple tools detecting the most obvious security flaws
  - Last but not least, a „secure coder” will be more competitive ;)

Security layers of secure programming

- Similarly to IT security as a whole, creating software may be divided onto layers
  - Defence-in-depth principle should be applied as well
- At least 4 layers might be differentiated
  - Technologies
  - Functionality
  - Configuration
  - Coding patterns
- Remember about other groups that do matter
  - Project leaders
  - System, network and server administrators
  - Users (awareness!)

Security layers (1)

- Technologies
  - There are useful solutions which may help (e.g. HTTPS, SSL, VPN, PKI)
  - You may apply them or not
  - Will be mentioned quite thoroughly during Day 1
- Functionality
  - Your application may lack necessary security-related functionality…
    - *Lack of character hiding when typing login passwords*
  - … or this functionality may be improperly implemented
    - *Too expressive error handling mechanism*
    - *Different error messages for bad username and bad password*
  - A general subject for Day 1

Security layers (2)

- Configuration
  - The user must be able to configure the application securely
  - The default configuration must be carefully designed (apparent security/market usability tradeoff)
  - Another general issue to be mentioned during Day 1
- Coding patterns
  - Even the best functionality and the most secured configuration may be implemented in an insecure way
  - The subject of the whole Day 2

Summary

- Software will contain vulnerabilities (including security bugs)
  - However their number should be minimized
- Software bugs will matter even if we have other layers properly secured
- There were many cases of spectacular software bugs
- Security economy encourages us not to make applications 100% secure (which is impossible), but to make breaking them cost-ineffective for attacker
- Secure programming is one of the layer of defense
  - Itself it may be divided into different layers: technologies, functionality, configuration, coding patterns