SOFTHOUSE

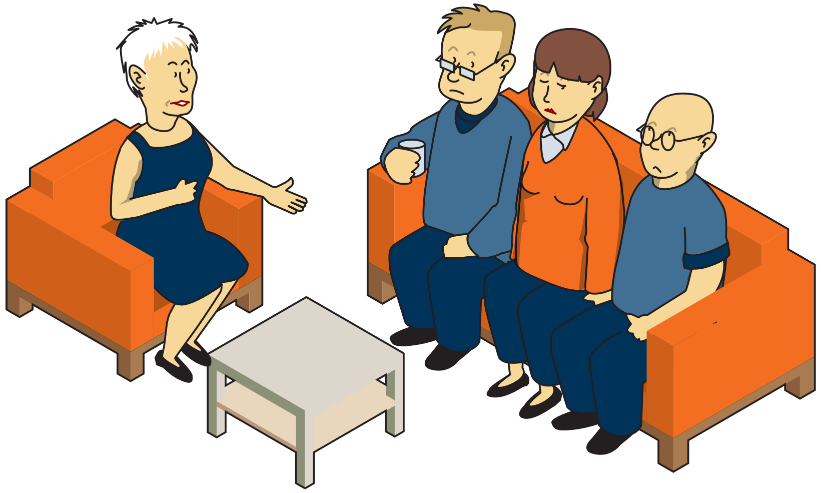
SECURE CODING

in five minutes

Intro:
MORNING MEETING AT ACME INC.

Anna, Scrum Master:	*Sorry to start the day with a bit of unpleasantness, but Janne at Delta-Q called me late yesterday. He says there was a hitch last week with the online store we're working on. Now he knows what the problem is. Judging by the e-mails coming in from customers, it seems that personal information and passwords have been leaking ...*
Jonas, developer:	*Oh no!*
Anna:	*Now he wants to meet the people working on the site. He believes that either we've left something open, or else it's even worse ...*
Simon, developer:	*Are you saying that he believes that we ...?*

Anna is grim-faced. She nods gravely.



HAVE YOU THOUGHT THROUGH SECURITY?

Hand on heart – what's your own approach to security?

- Do you avoid taking risky short cuts when writing code?
- Do you refrain from using templates or routines in your application whose origin you're uncertain of?
- Have you made a rational risk analysis and considered what threats and security issues it is sensible to take into account? Have you thought through what an inexperienced or destructive user might do with your system, and do you have a reliable process for restoring it?
- Have you or someone else backed everything up, so that you can confidently and safely restore your working environment after a crash?

If not, the following pages may serve as food for thought ...

OPTIMUM SECURITY LEVELS ARE BASED ON A TRADE-OFF

Total security is an unobtainable fantasy: there are no absolute measures of safety of the sort "if you do this, absolutely nothing can happen." That's how it is with everything in life. Compare, for example, your life as a road-user – it is built around your daily application of a balanced risk assessment where, for instance, you weigh up the protection afforded by your bicycle helmet against the dangers posed by drunk drivers.

Similarly, your life as a software developer or computer user must be based on a balanced risk assessment where you make trade-offs:

- **between efficiency and security**
- **between freedom of action and a sense of security**
- **between cost and the value of your work**
- **between fast, temporary solutions and time-consuming, robust solutions.**

You do not have to solve all the security problems in your product, but you must

- understand the risks you take
- understand that you are taking them

Safe suggestion #1:
TAKE RESPONSIBILITY

When we write robust and intelligible code, we get fewer security problems. Unfortunately, most developers will probably agree that they often have to hand over products that they are not entirely happy with – a consequence of the fact that management prioritises speed of production over long-term freedom from problems.

With luck, the worst thing that happens is that a colleague smirks at your clumsy code or groans at your substandard structure when debugging the code later on. If you are unlucky, it'll be worse – you might, for instance, have created a vulnerability that someone with malicious intent can exploit.

Here we can gain inspiration from the Software Craftsmanship movement and stress the importance of professional responsibility. Developers with professional pride should not have to accept deadlines so tight that there's insufficient time to do a good job!

Software Craftsmanship

The Software Craftsmanship movement aims to promote professionality in the field of software development, focusing on well-crafted software, knowledge sharing and cooperation.

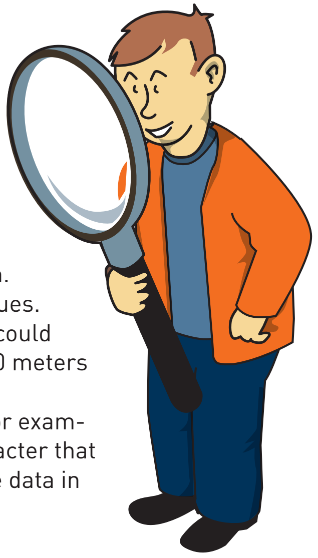
Read more at http://manifesto.softwarecraftsmanship.org

Safe suggestion #2:
NEVER TRUST DATA

Studies have shown that 60–80 % of all attacks are possible because we do not validate that the data is in the correct format, even if it was input directly by the user. The principle is never to trust data that comes from somewhere other than our own system and that we can keep track of.

All data input must therefore be validated in some way. One principle is to locate any defects or abnormalities, but then you constantly have to add new rules as the intruders become more skilful. It is actually better to define what authentic and accurate data should look like.

Also remember that even if you validate data on the client side, you need to revalidate it on the server side. A malicious user can circumvent validation in clients and send bad data straight to the server.



Some simple examples:

- If you have a field where you have to fill in a phone number, check that it does not contain any characters other than those that a phone number can contain.
- Many fields can be checked in terms of reasonable values. Input of quantities should be positive and e.g. heights could be checked for plausability; a person is probably not 10 meters tall.
- Some fields have to allow non-standard characters. For example, a surname might be *O'Neil*, which contains a character that some systems do not handle well. Make sure you store data in a sensible format so that problems do not arise.

Examples of vulnerabilities

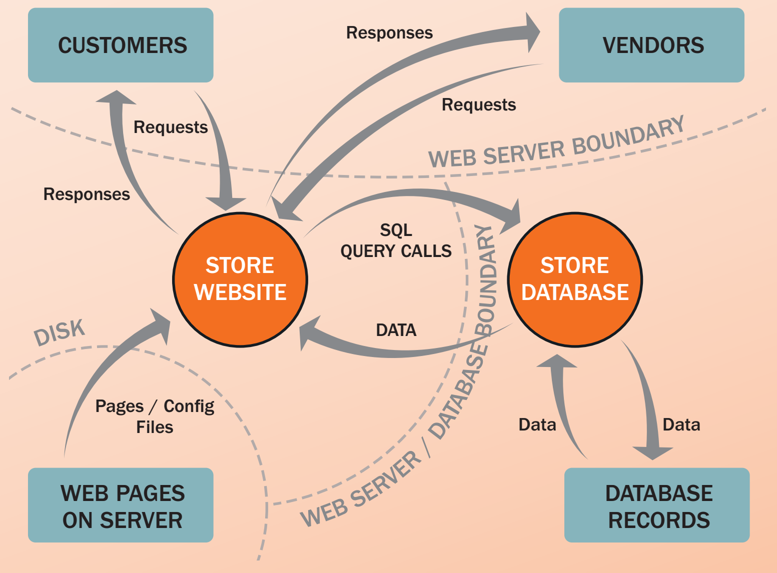
- Web forms where the hacker, for example, can enter JavaScript instead of his surname (and can thereby create Cross-Site Scripting).
- When the user enters SQL code to dump data in a field where you can search for products (so-called *SQL injection*).

Safe suggestion #3:
CREATE A THREAT MODEL

Don't skimp on documentation. Make sure you at the very least have a clear and updated overview of the system architecture at a high level. Besides making communication relating to the system much simpler, this also provides a good basis for threat modeling.

When you go through the architectural model together, you can discuss questions such as these:

- Between which parts is it possible to carry out an attack?
- Where is transferred data encrypted and unencrypted?
- What is vulnerable to an attack from the internet and where can an insider strike?
- What level of security are we willing to pay for?



Data flow diagrams of this type make it possible to get an overview of how data moves through the system, where it ends and where it might be good to protect the data (see page 14 for program tips).

Safe suggestion #4:
KEEP YOURSELF UPDATED

Professional developers should always keep themselves up-to-date on all fronts with both the big picture and the details.

You have to know what's going on in the technology niche in which you operate – the language you're working in, modified standards, newly launched products. If you are a tightly knit team you may not all need to keep track of everything – it may be enough for you to select coverage areas and keep each other informed about things that are worth paying attention to. This is linked to taking pride in your program development, which is what the Software Craftmanship movement advocates. You need to know what is good code in order to see what is bad code.

It is also obvious that your system platform and your developer tools must be kept updated to work efficiently. Sometimes people reckon that you should not change what works, but in the case of security, there is good reason to look at what is most current, so that you can know if you should upgrade purely for safety reasons. Therefore, always make sure someone on your team keeps abreast of news relating to your developer tools or any third-party code you are using.

Checklist

- ☑ One team member should follow release notes for each language, platform and framework that you use.
- ☑ Read blogs of industry specialists.
- ☑ Follow official and unofficial mailing lists and discussion groups on the language you're working in.
- ☑ Follow sources that feature new versions of the tools you use.
- ☑ Take a couple of relevant courses each year
- ☑ Update this check-list.

Safe suggestion #5:
MAKE A FUZZ

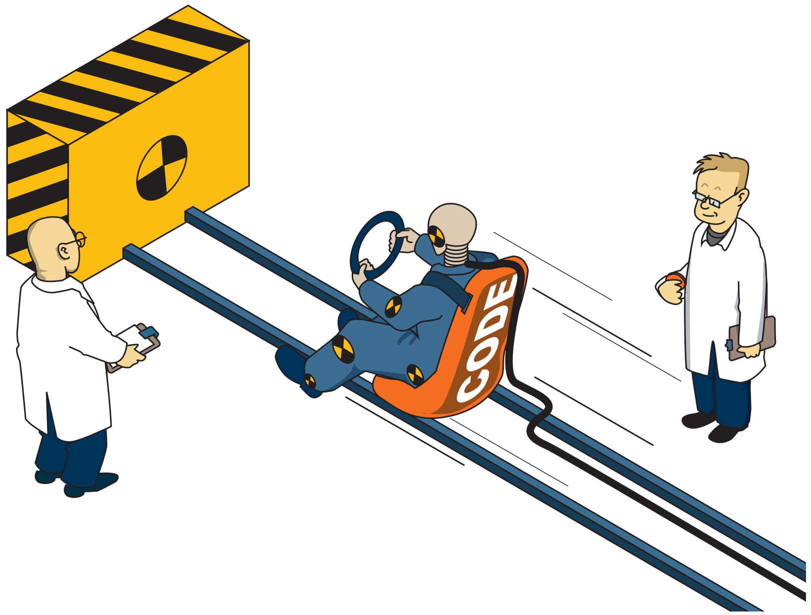
When you carry out normal component and system testing of an application, it can be hard to trigger certain potentially devastating types of fault. An error that makes the application behave unexpectedly may be the way in for a hacker.

What you can then do is fuzz testing. This means that you allow a tool to test limits and error handling by bombarding the application with data, both valid and invalid, and see what happens. If a malfunction occurs, you investigate how it happened.

This is something that can run in the background in a test environment and produce results before the release of new versions of the application.

It's a way to get answers to these questions:

- Can the system cope with error types that we haven't defined?
- Can we really handle the load we've said that we can manage?
- What happens when things go wrong?



Safe suggestion #6:
STAY PROUD OF YOUR CODE

Experienced hackers often exploit older parts of the application. The bit that no longer looks like a daily problem may in fact be the biggest problem that you have. The rule "don't fix it if it ain't broke" or "if it works, don't touch it" is not always valid in the security context.

Old code must not be forgotten. If we neglect it, it can't be reused or used as a starting point for new applications, and instead we may end up duplicating parts of the code base.

However, there are two approaches to handling the problem:

1. Make sure that code can grow old gracefully.
You do this by following Safe Suggestion # 1: Take responsibility. Be proud of what you have written in the past! Make sure it's recycled and kept up-to-date with new developments.

2. Give up and isolate the damage
Declare that "we don't trust this code" and shut it into a black box. You then manage the code just like an external system that you have no control over, i.e. you validate everything that comes out of it and make sure that you send the right things into it.

The Boy Scout Rule says:

Leave the campground cleaner than you found it.

In software it's a matter of micro improvements, i.e. improving the code each time we modify it, instead of letting it degenerate when we make changes and fix bugs.

Safe suggestion #7:

USE THE BEST TOOLS

As a professional developer, you should always use the best tools, both in the development environment and as a framework for the product. Many security issues can be resolved if the developers have good ways to test and get feedback on the problems. This robustness is the alpha and omega of security.

Some things to consider when choosing your environment:

- **In which environment are our developers most effective, and what support can help with security?**

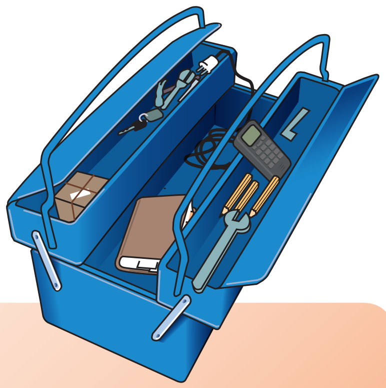
An environment that provides quick feedback and can assist in finding "code that smells bad" is a great help.

- **If you find something that doesn't work, how can you replace it?**

Can users of the system do it themselves? Can you force an end-end user to update an insecure version of your application?

- **Do you have an easy way of detecting when one of your third-party products needs to be replaced?**

And can you replace them easily?



Tools that can help you:

WebScarab	– Suite of applications to test web applications
OWASP AntiSamy	– Validation in Java and .Net
OWASP ESAPI	– Library of security solutions for Java
CodeSpy	– Find vulnerabilities through reflection in J2EE
JbroFuzz	– Fuzzing for JBoss
RFuzz	– Tool for fuzzing in Ruby

WHAT CAN WE EXPECT?

A cloud-based future
Applications move into the cloud as *Software as a Service*, which transfers infrastructure risks to the service providers. Threats to access will have to be included in contracts. What guarantees do we have that their platform can handle an attack?

Shared future
Many users are moving towards the philosophy "Sharing is Caring". They want to share their knowledge and experiences through social media. If the application is included in the social flora, we need in certain situations to reflect on the integrity of and access to our application in other ways. In this way, we make sure that there are clear limits as to what can and can't be shared.

What is important
In the future we'll have to think more about what is important in our business idea. It may be necessary to protect this in the best way we can. It could be some kind of data, technical solutions or perhaps reputation which is most important.

A hacker is not always bad
A vulnerability in the product can be a big problem, but it is even bigger if it is used, and disastrous if you do not know about it. Therefore make it easy for users or outsiders to tell you about any errors in the product and make sure you have a proper dialogue with them about what happens. Many people who find errors in a product are not out to exploit them, but want to help to create a usable product.

LITERATURE

Writing Secure Code (2nd Ed) – Michael Howard, David LeBlanc
Book on the basics of secure systems development

Software Security: Building Security In – Gary McGraw
Detailed book on how to write more secure applications

The Security Development Lifecycle – Michael Howard and Steve Lipner
Process for secure systems development used at Microsoft

LINKS

Open Web Application Security Project
Lots of articles, checklists and tools for you as a developer
http://www.owasp.org

SAFECode
Forum with several major software companies promoting industry standards for secure systems development
http://www.safecode.org/

Microsoft Security Bulletins
Follow the latest bulletins from Microsoft about vulnerabilities in their products
http://technet.microsoft.com/en-us/security/bulletin

Oracle Security
Security Patches for Java and frameworks
http://www.oracle.com/technetwork/topics/security/whatsnew/index.html

Security issues are increasingly important in the IT world, both for developers and users. Burying your head in the sand is just as counterproductive as being paranoid. Instead we should keep the discussion alive as to what is a reasonable level of security within our organization, our projects and for ourselves as individuals. At the same time there are of course practical steps we can take that can achieve a lot through relatively little sacrifice.

This publication can serve as a basis for internal discussion about how you and your colleagues can develop security within your organization, based on the following seven tips:

• Take responsibility
• Never trust data
• Create a threat model
• Keep yourself updated
• Make a fuzz
• Stay proud of your code
• Use the best tools

Softhouse Consulting

Stockholm
Tegnérgatan 37
SE-111 61 Stockholm
Phone: +46 8 410 929 50
info.stockholm@softhouse.se

Göteborg
Lilla Bommen 1
SE-411 04 Göteborg
Phone: +46 31 760 99 00
info.goteborg@softhouse.se

Malmö
Stormgatan 14
SE-211 20 Malmö
Phone: +46 40 664 39 00
info.malmo@softhouse.se

Karlskrona
Campus Gräsvik 3A
SE-371 75 Karlskrona
Phone: +46 455 61 87 00
info.karlskrona@softhouse.se

www.softhouse.se

SOFTHOUSE