<title> Secure Coding </title>

WS 2014

/*Prof. Alexander Pretschner
Dr. Martín Ochoa
LS XXII – Chair for Software Engineering */

About us

Chair 22 – Software Engineering
headed by Prof. Dr. Alexander Pretschner

Research interests:

- (Security) Testing
- Distributed Usage Control
- Malware Detection and Analysis
- Software Protection

Master Thesis topic proposals available under
https://www22.in.tum.de/en/teaching/

About us

- **Current**
  - Researcher at the software engineering chair of the TU Munich
  - Interested in applied formal methods for software security, in particular for privacy, malware analysis and reverse-engineering.
- **Past**:
  - Researcher and consultant at Siemens CT in Munich
  - Security assessments (pen-testing) and research in Web-Security.
  - PhD in Computer Science at the TU Dortmund
  - Mathematics in Munich and Rome
  - Systems Engineering in CR
  - Developer (PHP, Perl, Python) @ various companies
- **Contact**:
  - ochoa@in.tum.de
  - http://martin.mjdr.org
  - Office 01.11.039 (please make an appointment)

Warm-up 1

- Please answer the anonymous questionary! (10 mins).
- Be honest! Don’t worry if you don’t know some answers.

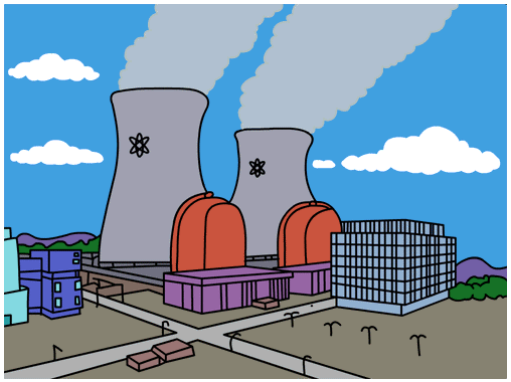
Warm-up 2

Share!

- Developers in the room?
  - Which programming language do you use?
- Have you ever developed an insecure application?
  - How do you know?
- Do you believe it is possible to develop a 100% secure application?
  - If so, how?

Motivation: Why is secure coding important?

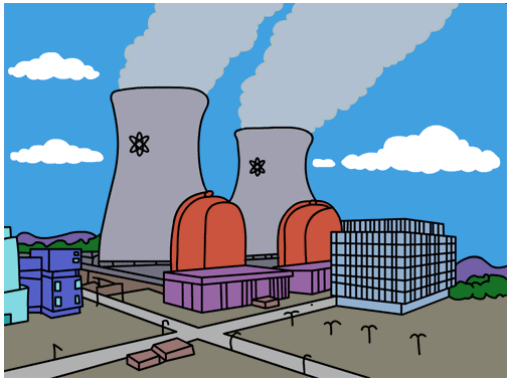
Software is everywhere



And soon even more...(IoT)

Etc...

Is software security important?



What can go wrong, who would benefit from attacking?

Are security problems real?

- We have seen all kinds of attacks and serious vulnerabilities in the wild, increasingly over the years! Think of:

  - **Shellshock:** developing news!
  - **Heartbleed 2014:** 60% of the internet under threat!!
  - **NSA 2013**: Revealed espionage on citizens and governments
  - **Sony 2011**: Playstation network, private data of millions of user stolen
  - **Stuxnet 2010**: Attack to particular critical infrastructure using Siemens PLCs
  - Political statements constantly made by **Anonymous**: DoS against commercial banking and governmental sites
  - **Zeus malware**: world-scale cyber-crime operation
  - Etc..

Who is attacking?

- **Attacker's motivations/goals can be very different**
  - Fun
  - Profit
  - Political statement
  - Terrorism
  - Spionage
- **Attacker's resources/skills can also vary**
  - Script kiddie
  - Skilled single attacker
  - Skilled group of attackers
  - Organisation with a certain budget
  - Government organisation

How can someone attack software?

- By **non-technical** means:
  - Social engineering
  - Bribery
  - Physical violence

- By **technical** means (focus of this lecture)
  - Exploit design/implementation/operations vulnerabilities!

Threat vs. Vulnerabilities vs. Exploit vs. Attacks

- A **threat** is "*A potential cause of an incident, that may result in harm of systems and organizations" (ISO 27005)* (i.e a hacker, an unintended leakage of information)
- A **vulnerability** is "*A weakness of an asset or group of assets that can be exploited by one or more threats" (ISO 27005)* (i.e. an SQL-i vulnerability)
- An **exploit** is …hey, there are no good definitions around! But usually is some code or series of steps to take advantage of a vulnerability an carry out an attack.
- An **attack** is "*an assault on system security that derives from an intelligent threat, i.e., an intelligent act that is a deliberate attempt (especially in the sense of a method or technique) to evade security services and violate the security policy of a system." (IETF)* (i.e. a DoS attack on a server exploiting a buffer overflow)

Threat vs. Vulnerabilities vs. Exploit vs. Attacks

As a consequence:

- Without a threat, there can't be an attack
- Without an exploit, there can't be an attack
- Without a vulnerability, there can't be an exploit and therefore also no attack

Can we avoid threats? Hard, if not impossible.

Can we avoid exploits? Hard, if there are vulnerabilities around.

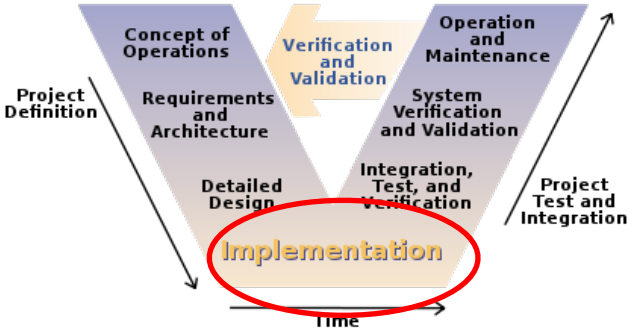
Can we avoid vulnerabilities? Some of them, yes!

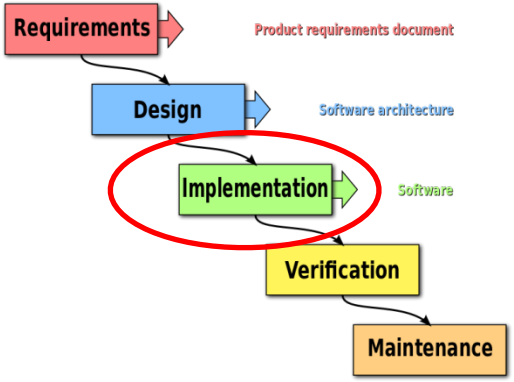
Goal of this lecture: avoid vulnerabilities to avoid attacks!

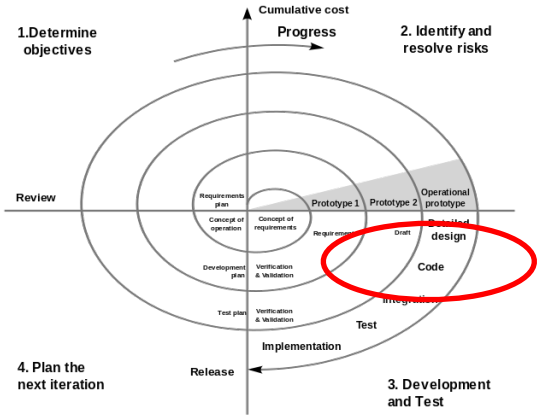
Note: not everybody agrees with this approach, fixing vulnerabilities is expensive an not always justified, although this is a bit polemic:

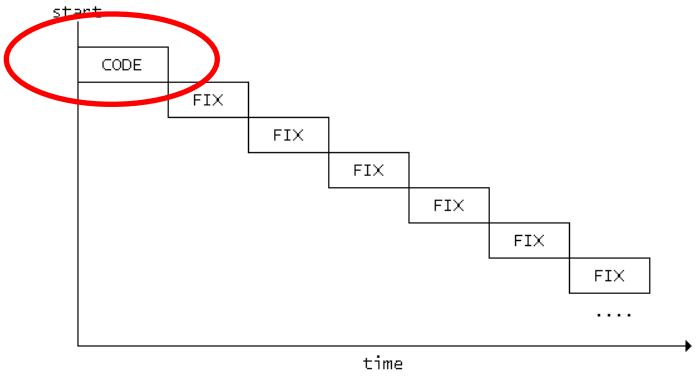
Allodi, Massacci "**My software has a Vulnerability, should I worry**?" http://arxiv.org/abs/1301.1275

How is software produced?









No matter how, **coding** is central!

Foundations for software security

What is security?

Traditionally security is defined in terms of Confidentiality, Integrity and Availability of system data and resources.

- Those are usually requirements, but what does exactly mean for something to be confidential/integer/available?

Is there a rigorous definition?

Let's take a look at some attempts to define security (at least for confidentiality/integrity).

Disclaimer: The following slides are by not means exhaustive or precise, they are meant to give a feeling of extensively researched security definitions. A thorough examination of these properties would require a lecture on its own.

Foundations for software security

Brief excursus on correctness of programs

In general it is difficult to rigorously show that programs behave according to a specification.

- Goal: if a program is a partial function (maybe it does not terminate…)

$$f : I \to O$$

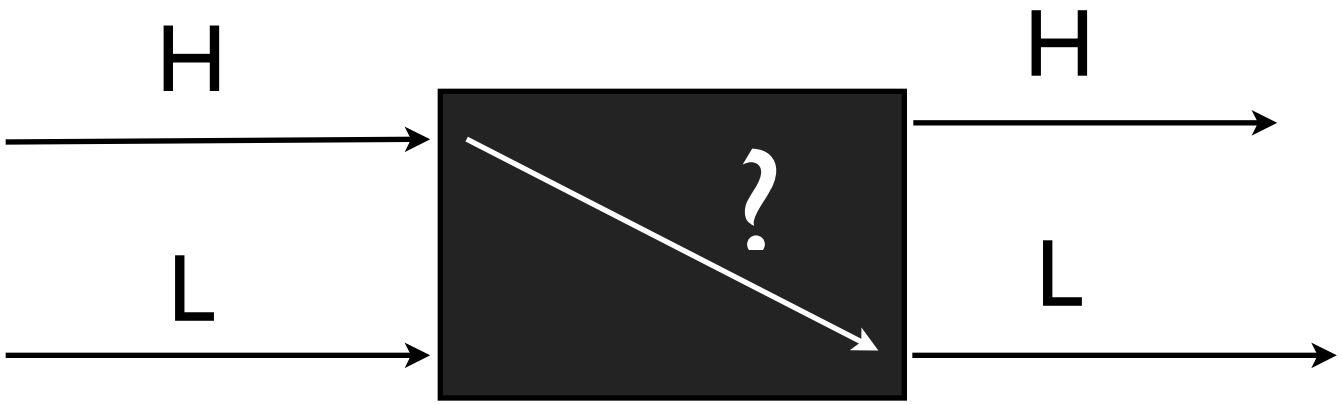
Can we guarantee some properties about f? What kind of properties are interesting?

Huge research topic! Some approaches:

- *Testing*: for realistic input sizes it is computationally infeasible to test exhaustively.
- *Program verification*: Precise verification is expensive, some abstractions are less precise but efficient (abstract interpretation, symbolic execution).

Foundations for software security

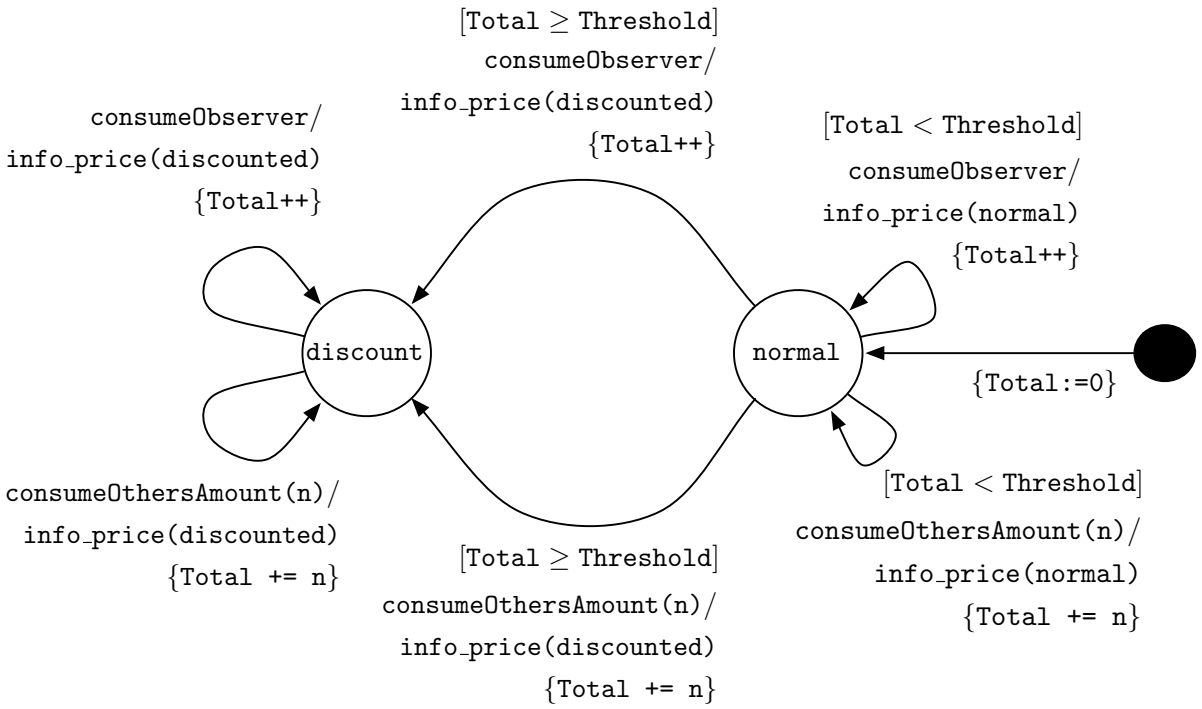
Non-interference [Goguen, Meseguer 82]



$$\forall\ \overrightarrow{i_1}, \overrightarrow{i_2} \quad \overrightarrow{i_1}|_L = \overrightarrow{i_2}|_L \Rightarrow [\overrightarrow{i_1}]|_L = [\overrightarrow{i_2}]|_L$$

Foundations for software security

Assume consumeOthersAmount(n) is secret, consumeObserver is public and normal != discounted. Can you spot the flow?



Foundations for software security

Non-interference…

- It is a multi-run property (it is not a safety property).
- It covers both integrity and confidentiality (up-flows/down-flows)
- It is very general, but very restrictive
  - A program implementing a password checker violates it
- It is difficult to verify precisely for programs.
- Generalisations of non-interference such as Quantitative Information Flow and declassification are an ongoing field of research.
- We will come back to it while talking about Java security.

- Further reading:

Joseph A. Goguen, José Meseguer: **Security Policies and Security Models.** IEEE Symposium on Security and Privacy 1982
Volpano, Dennis, Cynthia Irvine, and Geoffrey Smith. "**A sound type system for secure flow analysis.**" *Journal of computer security* 4.2 (1996): 167-187.
Volpano, Dennis. **Safety versus secrecy**. SAS '99 Springer Berlin Heidelberg, 1999.

Foundations for software security

Computational indistinguishability

- Almost any cryptographic algorithm can be broken with enough computational power (unless perfectly secret).
- However, we can assume the power of an adversary to be limited, in its complexity class
- Commonly adversaries are considered to be in PPT (polynomial time probabilistic machines)
- Example of semantic security (IND-CPA or chosen plain text attack):

Game IND-CPA :
$(m_0, m_1) \leftarrow \mathcal{A}_1;$
$b \xleftarrow{\$} \{0,1\};$
$b' \leftarrow \mathcal{A}_2(E_K(m_b));$
return $b = b'$

$$\left| \Pr[b = b'] - \frac{1}{2} \right| \leq \epsilon(n) \text{ with } \epsilon(n) \text{ negligible}$$

Foundations for software security

Examples

- Pure *RSA* is not semantically secure, because it is deterministic the encryption of a message with the same key is always identical
- Pure *AES/DES* are not semantically secure, also deterministic
- *ElGamal* is semantically secure, because there is a random salt for every encryption (and security can be reduced to decisional Diffie-Hellman assumption)

Complexity-theoretical security definitions make cryptography more rigorous, but software security is not only about crypto but how you use crypto together with the rest of the program's logic!

Further reading:

Bellare, Mihir, and Phillip Rogaway. "**Code-Based Game-Playing Proofs and the Security of Triple Encryption**." *IACR Cryptology ePrint Archive* 2004 (2004): 331.
Goldreich, Oded. **Foundations of Cryptography: Volume 2, Basic Applications**. Vol. 2. Cambridge university press, 2009.

Foundations for software security

Symbolic reasoning for cryptographic protocols

- Logical model based on the assumption that cryptography is perfect.
- Adversary model can intercept all messages between two parties and modify them arbitrarily
- Axioms look like

$$knows(Enc(m,k)) \ \& \ knows(k) \Rightarrow knows(m)$$
$$knows(m) \ \& \ knows(k) \Rightarrow knows(Enc(m,k))$$

To show for confidentiality: it is impossible to derive the predicate *knows(s)* for a secret *s* and a description of the protocol.

Limitations: often assumptions are broken (primitives are not perfect) and certain algebraic relations are difficult to model. Moreover complex protocols are computationally expensive to verify. Also some attacks in this model are difficult to carry out in practice.

Further reading:

Dolev, Danny, and Andrew C. Yao. "**On the security of public key protocols**." *Information Theory, IEEE Transactions on* 29.2 (1983): 198-208.

Armando, Alessandro, et al. "**The AVISPA tool for the automated validation of internet security protocols and applications**." *Computer Aided Verification*. Springer Berlin Heidelberg, 2005.

Consequences? (some philosophy)

- **Q: Is there a process/tool that can automatically guarantee the security of programs?**
  - **A**: No! Although formal methods/formal definitions are promising and necessary, there is still a big gap between research and practice in this sense.

- **Q: Why not?**
  - **A**: What is exactly “security”? For many realistic systems this is not even clear. A rigorous definition (formal methods) is expensive for complex systems (Is OpenSSL secure?).
  - **A**: Even if security was to be rigorously defined, it is computationally infeasible to verify realistic size programs.
  - **A**: Even if it was possible to verify a program w.r.t. security, assumptions must be made about the system in which the program runs, which are often broken. Many of those assumptions may be shown to be broken only in the future (0-day exploits).
  - **A**: Secure programs are trivially insecure if deployed/configured incorrectly.

Where can there be problems?

Now let's take a more pragmatical perspective. What can go wrong and when? The devil is in the details!

- We can distinguish vulnerabilities in implementations in the following categories (following "Secure Coding", Griff, van Wyck):
  - **Architecture/design-level**
  - **Implementation-level**
  - **Operations-level**

Architecture/design-level vulnerabilities

- Attacks due to errors at “pure thinking” time
  - **Man-in-the-middle attacks**: a party intercepts public traffic and reads/manipulates it
    - Examples: vulnerable cryptographic protocols
  - **Race-conditions**: Atomicity can be exploited by attackers!
    - Examples: Certificate revocation checks, threads, I/O to files.
  - **Control flow-integrity:**
    - Examples: Flawed bussiness process (shop for free).
  - **Access control logic**:
    - Examples: Users can access sensible files/resources.

Implementation-level vulnerabilities

- Attacks due to errors at implementation time
  - **Memory management attacks**:
    - Examples: Buffer overflows
  - **Input validation**:
    - Examples: SQL Injections, XSS attacks
  - **Harcoded Backdoors:**
    - Examples: hardcoded passwords with elevated privileges.

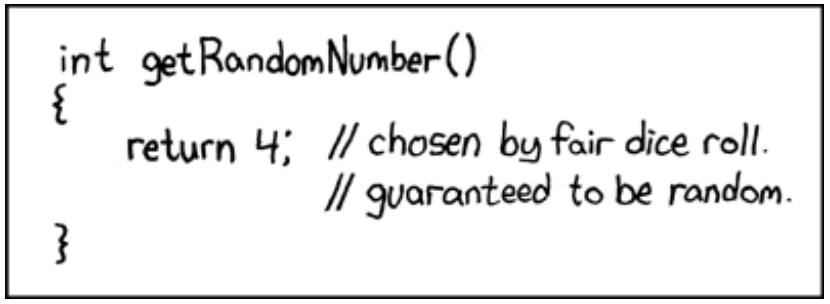
Operation-level attacks

- After development, during operation!
  - **DoS attacks:**
    - Examples: Sending multiple requests to applications causes it to crash
  - **Default accounts**:
    - Examples: Admin/admin , guest/guest on Webserver, Database, Service…
  - **Password strength:**
    - Examples: Users can choose weak passwords, that can be bruteforced

Reasons for problems

Stupid, lazy programmers? No! There are:

- **Technical factors**: complex systems! Unanticipated compositions, interactions, unknown vulnerabilities.
- **Psychological factors:** poor risk assessments, assuming users are not malicious, fallacious mental models. (i.e. users will only input text in HTML forms, users will not tamper with HTTP requests).
- **Real world factors:** lack of training, amateurs write code, production pressure, focus on functionality.



(Source: XKCD)

How can we raise the bar for attackers?

- The following is not scientific but is experience/common sense:
  - Coders are the **last line of defense/offense** in software development
    - Even flawless designs/architectures can be wrongly implemented (i.e. **heartbleed**)
  - Many security scandals/real life attack are exploit one or more “trivial” coding vulnerabilities (after they have been understood, the fixes can be relatively easy i.e. **sql injections**).
  - Many real life attacks exploit a **small subset** of common software vulnerabilities

Hypothesis of this lecture: We could substantially improve software security by writing better code! (100% security is a fairy tale).

Keyword: **AWARENESS**

Solutions?

- **Secure coding guidelines**: There exists secure coding guidelines for almost any popular programming language.
  - *Pros*: Learn from best practices
  - *Cons*: Need constant update, might be boring. For C/C++ the amount of things that can go wrong is huge.

- **Code reviews**: A good development practice is to have specialists to perform code reviews on source-code before deployment.
  - *Pros*: Nothing like humans to spot problems!
  - *Cons*: Can be expensive, depending on size and complexity of code, no guarantees.

We will explore both techniques!

Solutions?

- **Static analysis (tools):** There is a number of open source/commercial tools for static analysis of source code.
  - *Pros*: Automatic!
  - *Cons*: Usually too many false positives to be helpful.

- **Dynamic analysis (tools)**: Tools to monitor/reverse-engineer applications (i.e. IDA-PRO)
  - *Pros:* Often unwanted behaviour can be spotted easily during runtime
  - *Cons*: Reverse engineering can be cumbersome and expensive, since manual intervention is needed.

In the following weeks we will play with dynamic/static tools, so you can judge for yourself how helpful they are.

Preliminary conclusions

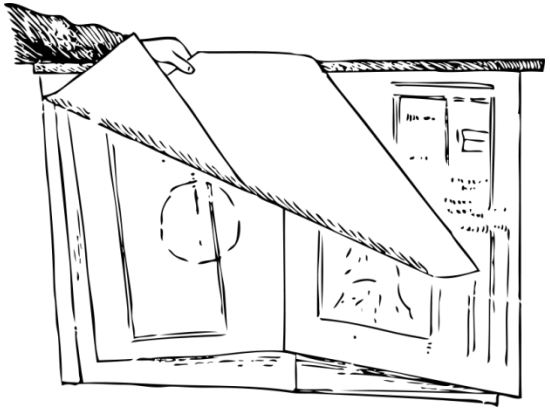
- Producing secure software is **not easy**, good intentions are not good enough.
- There is **no silver bullet solution**.
- Producing more secure software involves **awareness and constant update**.
- **Basic guidelines** should be taken into account starting at the design phase.
- The bar for attackers can be significantly raised if we **pay attention to common mistakes** and learn from others.
- Even choosing a programming language can have consequences for the security of an application.

We will revise these with your feedback by the end of the course!

Objectives of this lecture

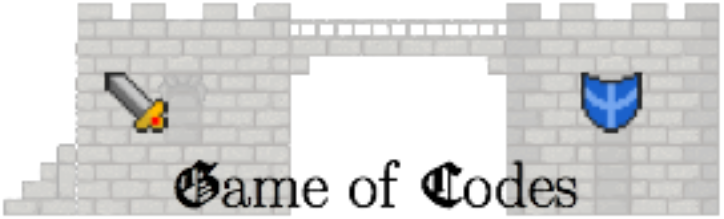
- Understand common vulnerabilities and learn how to look for them.
- Learn reliable countermeasures.
- Focus on highly popular programming languages (PHP, C/C++, Java).
- Introduce connections to active research scientific areas in this subject.
- Raise the bar against different attacker models (black-box vs. white-box).
- Simulate a realistic development environment with hard time constraints and limited resources.

Lecture plan



1. **Intro/Motivation**
2. Web. app. security
3. Memory management and C/C++
4. *Presentations Phase 1 (Developing)*
5. Countermeasures
6. White-box Vulnerability detection
7. *Presentations Phase 2 (Black-box testing)*
8. Java security
9. *Presentations Phase 3 (Fixing + new features)*
10. Reverse engineering
11. Obfuscation
12. Crypto / Code review
13. *Presentations Phase 4 (White-Box testing/RE)*
14. Android / Browser security / Recap
15. *Presentations Phase 5 (Fixing + final report)*

Development project



- There will be an ongoing development/testing project in parallel to the course.
  - Groups of 3 persons.
  - 5 Intermediate presentations.

- **Phase I**: Development of basic functionality
- **Phase II**: Black-box testing of 2 applications
- **Phase III**: Fix application + additional functionality
- **Phase IV**: White-box testing/ Reverse Engineering
- **Phase V**: Fix and final report

You will be able to vote for your favourite presentation in each phase. The best groups will be awarded a prize!

Project Timeline

- No written exam. Only project consisting of 5 phases
- Remember: 6 ECTS = 180 hours of work = 2x15 lectures + 3x15 lab + 105 HW

Phase	Name	Start Date	End Date	Grade %	Minimum Time / Student
1	Development	07.10.14	28.10.14	3/14	22.5 hours
2	Black-box Testing	28.10.14	18.11.14	3/14	22.5 hours
3	Fixing + Development	18.11.14	02.12.14	1/7	15 hours
4	White-box Testing	02.12.14	06.01.15	2/7	30 hours
5	Fixing	06.01.15	20.01.15	1/7	15 hours
					105 hours

Project Rules

- Project must be developed on Samurai WTF VM provided by us
- You are allowed to use only:
  - PHP, C/C++, MySQL for Phase 1
  - Also Java for Phase 3
  - You are allowed to use only GUI frameworks/libraries
  - Not allowed to use CMSs (e.g. Wordpress, Drupal, Joomla, etc.)

- VM must be exported and provided via USB stick at the beginning of the lecture on the end date of each phase as indicated in the table on slide 2

- If a team cannot provide the VM at the beginning of the lecture on the end date of a phase they get 0 points for that phase (NO deadline extensions)

Project Rules (continued)

- Every project phase has deliverables:
  - 10 minute oral presentation in front of class
  - User guide for implemented features or attacks
  - VM containing source code (mandatory for phases 1, 3 and 5)

Phase	Deliverable 1 (30 % of grade)	Deliverable 2 (70% of grade)
1	Presentation + User guide	VM with source code
2	Presentation	Technical guide of attacks
3	Presentation + User guide	VM with source code
4	Presentation	Technical guide of attacks
5	Presentation + User guide	VM with source code

Project Rules (continued)

- Every team member **must present** at least once
  (otherwise s/he fails the course, not the other team members)
- Presentation slides + user guide must be delivered in the same PDF document
- **Time tracking**:
  - Every team member must keep track of how much time s/he spent on which activity
  - Time spent on an activity should be proportional to effort and outcome
  - Back-up slides in presentation should also include table with time-tracking

Development project: Phase I

A bank hires you to implement a modern internet-banking system.

- The system must consist of: a web GUI and a back-end for server-side processing.
- The client-side of this web-application must be implemented in HTML and JavaScript.
- The server-side must be implemented using PHP, MySQL and C.

The Internet-banking GUI must allow clients to:

- register via a registration form
- log-into their account and see their transaction history
- make online payments via securely generated transaction numbers
- transactions can be performed by:
  - filling in an HTML form
  - uploading a text file containing the same information as provided in the HTML form; (parsing text files must be implemented in C)

Development project

The GUI of the internet-banking system must allow bank-employees to:

- log-into their accounts
- approve client registration requests
- approve transfers larger than 10.000 EUR

The following information must be stored in a relational database on the bank-server side:

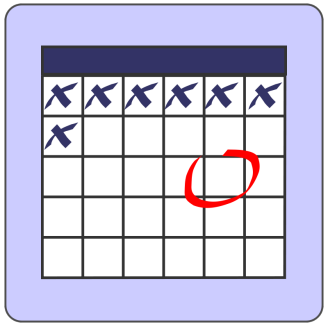
- user accounts (of clients and employees)
- bank accounts (of clients)
- a history of all transfer details

Development project

- When a client registers on the Internet-banking web-app, he must get an e-mail with 100 unique transaction codes of (15 printable characters), which he can use when transferring money.
- It is important to note that all clients must have different transaction codes and one code must only work one-time.
- Entering the same transaction code for another transaction must be encountered with an error message by the web-application.
- The transaction codes must either be entered via the HTML GUI or in the uploaded text-file.

Development project

- Your task is to implement this functionality and present your solution on Tuesday the 28th of October.
- Use the official lecture's VM
- Don't worry about security, we will get there. Focus on functionality!
- Presentations will max. 10 mins. each, go to the point.
- Have a backup slide including a summary of how much time did each member of the group doing what.
- Have fun!



Groups and VM

- Please make groups of 3 persons and send the names of the members to

banescu@in.tum.de

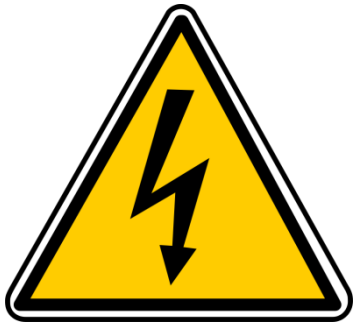
By **today**!

If you don’t have a group, please send us an e-mail as well, we will assign you to a group.

You can download the VM from:

https://drive.google.com/file/d/0B_mxtTZhDRPCVlNQLXJwUTRGZVE/

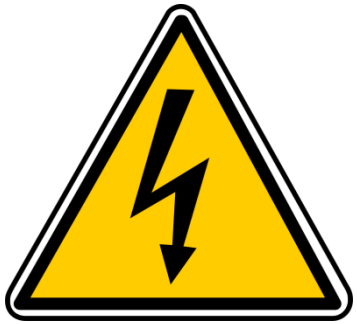
Disclaimer



- In this lecture we will learn security testing techniques for didactical purposes. However these techniques are realistic and could be used to test real-life software. Be aware that testing proprietary software without permission is illegal, so be sure to ask first.

- In the internet you will find plenty of perfectly legal, didactical applications to test! (We will provide you with a few to play around!)

Disclaimer



- In particular for black-box testing, the EU law is: **EU Directive 2013/40/EU**
  - Illegal for unlawfully accessing a vulnerable information system
  - Illegal for interception of non-public transmissions of computer data
  - Illegal to build tools with the intention of using them form the previous purposes

- For reverse-engineering the EU law is: **EU Directive 2009/24/EC**
  - Illegal for copying intellectual property from software
  - Legal for interoperability, government investigation and protection of privacy

!!! Violation of these laws is considered a criminal offense punishable with 2 years imprisonment !!!