Today

- Introduction
- Theoretical Part: How to Build Secure Software
- Brief Problem Overview
- Setup HowTo for Phase I
General Setup

- Group Project with groups of four or five students
- Three Phases:
  - 1st Phase:
    Each group implements software according to a specification
  - 2nd Phase:
    Each group examines the implementations of the other groups, searches for vulnerabilities and writes exploits.
  - 3rd Phase:
    Each group receives the vulnerability reports, has to fix them and write a report.
Organisation

Appointments

- 31.07.2017 10 AM s.t. - 3 PM Kickoff „Build It"
- 03.08.2017 10 AM s.t. - 1 PM Q&A session 1 (voluntary)
- 07.08.2017 10 AM s.t. - 3 PM Intermediate presentations + Kickoff „Break It"
- 10.08.2017 10 AM s.t. - 1 PM Q&A session 2 (voluntary)
- 14.08.2017 10 AM s.t. - 3 PM Intermediate presentations + Kickoff „Fix It"
- 17.08.2017 10 AM s.t. - 1 PM Q&A session 3 (voluntary)
- 21.08.2017 10 AM s.t. - 1 PM Final presentations
- 21.08.2017 11:59 PM Deadline for the final report

Examination

- Consists of your practical contest participation, the final report, the presentations and an oral exam
- Make an appointment for your oral exam with Prof. Bodden

Slides available online
Organization

Deliverables

- 2 Intermediate Presentations: 10min talk, summarise your work of the previous phase
- Final Presentation: 15min talk, summarise your overall work of all phases
- Final report: max. 10 pages, write about your implementation, your breaking approach, your bugs and their fixes and lessons learned
- Oral examination: Knowledge about theoretical parts and about your group’s implementation, breaks and fixes
General Advice

- Challenge yourself, challenge the others, expect to be challenged
- Do not start working on the Saturday before the deadline
- Raise your questions in time and use the course mailing list, so that others may benefit from the answers as well. Please participate and answer questions for others when you know the answer. Mailing List: bibifi@lists.uni-paderborn.de
- Make use of the Q/A sessions and come early, as we may leave earlier if there is no demand

Rules

- No plagiarism (zero tolerance)
- No deadline extensions
- Do not attack the infrastructure
- Allowed Programming Languages: C, Python, Java
How to Build Secure Software
Why do we care?
How risky can it be to use Adobe Flash Player?
2011 Attack on RSA and U.S. Weapon Manufacturers

1: Social engineering & phishing

• Planted “Poison Ivy” trojan horse.

2: Digital Shoulder Surfing

• Poison Ivy connects back to control server, giving full control to attacker.
• Attacker gradually moves towards higher value accounts and data.

3: Collecting SecureID secret seed records, downloading them from staging server.

• RSA issues warning on March 17
• Unusually fast (e.g., attack on Nortel went unnoticed for more then 10 years)

4: Exploiting compromised SecureID to break into the target systems at defence industry.

• June 3: Lockheed discloses a blocked attack, which exploited the breach at RSA.
• RSA announced replacement program for tokens (>40M tokens worldwide, Lockheed > 45'000).
• August 2011: RSA acknowledge immediate 66M$ for recovery.
• March 27, 2012: NSA attributes attack to Chinese hackers

What does “Security” mean for Software and Information Systems?
Security is Not a Set of Features

- Secure is an emergent property of software
  - “Being dry” in a tent in the rain
  - Being secure is the result of many, many factors, not one feature (e.g. encryption)

- ...so requirements documents should not just be a list of features
Core Security Goals
The CIA Triad

Confidentiality

The system must not disclose any information intended to be hidden, e.g., your credit card information on a website.

*Note: open source software can still be confidential.*

Integrity

The system must not allow assets to be subverted by unauthorized users, e.g., changing a prisoner’s release date.

*We must be able trust what is in the system*

- The data being stored
- The functionality being executed

Availability

The system must be able to be available and operational to users, e.g., bringing down Amazon.com

*Any system performance degradation that can be triggered by a user can be used for denial of service attacks*
Threat Classification

Spoofing

“Pretending to be something or someone other than yourself.”


- IP Spoofing
  - Set source IP address to some other IP

- E-Mail Spoofing
  - Replace sender address
  - In SMTP, “From” is not checked
Threat Classification

Tampering

“Modifying something on disk, on a network, or in memory.”


- “Web Tampering” [owasp.org]

<input type="hidden" id="1008" name="cost" value="70.00">

Threat Classification

Repudiation

“Claiming that you didn’t do something, or were not responsible. Repudiation can be honest or false, and the key question for system designers is, what evidence do you have?”


- Deleting Logs

- Using symmetric keys for signing
  - Both parties can dispute the signing
Threat Classification

Information Disclosure

“Providing information to someone not authorized to see it.”

Threat Classification
Denial of Service

“Absorbing resources needed to provide service.”


- Any system performance degradation that can be triggered by an user can be used for denial of service attacks, e.g., too many server requests
  - Blog of security blogger Brian Krebs was taken down in September 2016
  - DNS services of Dyn were attacked in October 2016
    - Many prominent websites were not reachable
  - Botnet of thousands of IoT devices, e.g., IP-cameras
Threat Classification

Elevation of privilege

“Allowing someone to do something they’re not authorized to do.”


- Library for image processing
  - Used by many websites
- “Image Tragic”
  - Upload a compromised SVG
  - Execute code with the privileges of the calling server process, e.g., deleting all images
Threat Classification

Summary

Spoofing

Tampering

Repudiation

Information Disclosure

Denial of Service

Elevation of Privilege
Security is usually **not** preserved under composition!
At which point in the development can we apply security?

At which point in the development can we introduce insecurity?
Software Development Lifecycle – Potentials for Insecurity

- Incomplete Requirements and Use Cases
- Flawed Architecture And Design
- Incomplete Test Plans
- Overlooking vuln. Code
- Bad Configuration Test and Test Results
- Feedback from the Field

Incomplete
Flawed
Incomplete
Overlooking vuln.
Bad Configuration
Security Touchpoints

- Security Requirements
- External Review
- Code Review
- Penetration Testing

Abuse Cases
- Requirements and Use Cases
- Architecture And Design
- Test Plans
- Code
- Test and Test Results
- Feedback from the Field

Risk Analysis
- Risk-Based Security Tests
- Risk Analysis
- Security Operations

Architecture And Design
- Test Plans
- Code
- Test and Test Results
- Feedback from the Field
Think Evil: Consider Attacker’s Point of View

- What could be interesting for attackers?
- What needs to be protected?
- Which of the security goals need to be achieved for
- What are the capabilities of attackers?
- What shall attackers with those capabilities be able to achieve at most?

This is the fundamental aspect to consider for security.
Defence in depth
- If they break into this, they can’t get any farther
- Think Middle-Age castles
- Original meaning of “firewall”, not today’s firewall

Least privilege
- Every user or module is given the least amount of privilege they need
- Evil: `sudo chmod -R a+rwx /

```php
<?php
$host = 'localhost';
$userID = 'root';
$password = 'password';
$db = mysql_connect($host, $userID, $password) or die ('Error connecting to mysql');
$name = 'testdatabase';
mysql_select_db($name);
$sql="SELECT * FROM theTable";
$result=mysql_query($sql);
?>

Source: OWASP.org
```
More Misc. Philosophies & Proverbs

- Fail securely
  - Exceptions put the system into weird states
  - Error message information leak
  - Take care of those exceptions properly!

- No Security by Obscurity
  - You can’t rely upon being obscure to be secure
    - Crowds are good at guessing
    - Insiders are corruptible

"If the Rebels have obtained a complete technical readout of this station it is possible, however unlikely, that they might find a weakness, and exploit it."
"The plans you refer to will soon be back in our hands."
—General Cassio Tagge and Darth Vader (Star Wars, 1977)

- Exceptions?
  - Passwords, encryption keys, ASLR

```php
try {
  openDbConnection();
}
catch (Exception $e) {
  echo 'Caught exception: ',
  $e->getMessage(), '
';
  echo 'Check credentials in config file at: ', $Mysql_config_location,
  '
';
}
Source: cwe.mitre.org
```
Even More Misc. Philosophies & Proverbs

- Detect and record
  - Even if you can’t always sift through that data ahead of time
  - Post-mortem analysis

- Don’t trust [input | environment | dependencies | *]
  - Know what to trust
  - Know whom to trust
  - Know how to trust

- Keep it simple and stupid (KISS)
  - You aren’t gonna need it (YAGNI)
  - Speculative generality can be risky
  - Minimise the attack surface
  - Write readable, clear code! Unreadable code is broken code.
Validating Input

- Input validation is blocking bad inputs

- Black list, also known as sanitization
  - Enumerate the bad stuff
  - Don’t allow anything on the blacklist
  - Drawback: infinite, easy to get around
  - Straightforward approach

- White list
  - Only accept known good input
  - Often done with regular expressions
  - Sometimes not possible to block certain characters

- Recommendation: do both, but prefer a whitelist
Input in Many Forms

- Not always strings and numbers

- Consider: images with metadata
  - PHP had many issues with EXIF JPEG metadata
  - Adobe Acrobat & embedded fonts

- Videos, music, radio signals...!
Sanitising Input

- Instead of blocking input, *sanitise* it
  - All input comes in, but it’s manipulated
  - Convert it to something that won’t be interpreted as code
  - Usually utilises escape characters

- e.g. Java
  - “ is \”

- Drawback: need to know everything to escape
  - Very blacklist-like
  - False positives are also annoying
  - Need to remember to do it… everywhere
  - Field of active research: New compilers and other tools may come to the rescue

```java

// ...
String escaped = escapeHtml4(source);
```
Error Handling

- “This won’t ever fail” – source for evil bugs!
- Most Methods/Functions may fail!
  - If you don’t know, whether or how, look it up!
  - Assess the criticality of the possible failure!
  - Handle failures properly! Do not “work around” critical failures!
- Error handling can become a noticeable amount of code!
Exception Handling

- Most weird, unexpected behavior results in an exception
  - Handle the exceptions you know about
  - Know that sometimes some get away

- Design your system to handle exceptions at the top-level
  - E.g. Java → catch Throwable, not Exception

- For maintainability & complexity:
  - Avoid promoting unnecessarily
    e.g. “throws Exception”
  - Deal with related exceptions in one place, near the problem
    e.g. wrapper around private methods in a class

- Sheer laziness: try{something();} catch{}  
  If you do this in this course, you will fail.
Just re-throwing exceptions may leak information

- CWE-209: Information Exposure Through an Error Message

```php
try {
    openDbConnection();
}
//print exception message that includes
//exception message and configuration file location
catch (Exception $e) {
    echo 'Caught exception: ', $e->getMessage(), '\n';
    echo 'Check credentials in config file at: ', $Mysql_config_location, '\n';
}
```
Debugging and Development Constructs

- Temporarily needed code during development can become a big problem if it is forgotten to be removed
- Be conscious about such code
- Mark it explicitly in comments
- Search for and fix it before releasing the program
finally

- Don’t forget about the **finally** clause!
  - Anything in the **finally** clause gets executed no matter what happens
  - Good for cleanup of resources

```java
public void something() {
    Connection conn = null;
    try {
        conn = getConnection();
        /* do db stuff */
    } catch (SQLException e) {
        /* handle it */
    } finally {
        DBUtil.closeConnection(conn);
    }
}
```
Tidy up after yourself

- Release resources you created and don’t need anymore!
  - Yes, Java has a GC.
  - Yes, it does clean up after you.
  - No, it does not clean up everything.
  - Other languages may not have a GC. For example, C has none and will never have one.
- Actively think about object’s lifetime when you create it
- Keep in mind: memory exhaustion leads to Denial of Service!
Think of the Children

- Subclassing overrides methods
  - In untrusted API situations, make sure you can’t be extended and have a sensitive method overridden
  - Use the `final` keyword:
    ```java
    public final class Countdown{}
    public final class String{}
    ```

- Malicious subclasses can override the `finalize()` method to resurrect objects.
Cloning is Insecure

- Every Java object has a `clone()` method
  - Often error-prone
  - Doesn’t do what you think it does
  - Most people don’t abide by the contract

- Even the Java architects don’t like it
  - The Java Language Secure Coding Guidelines from Oracle recommend not using `java.lang.Cloneable` entirely
  - Recommendation: Override the clone method to make classes unclonable unless required

- Problem: Cloning allows an attacker to instantiate a class without running any of the class constructors.
- Use your own copy mechanism if needed
Immutability in OO

- Setters are evil (except the Irish kind)
  - What if we construct, run, set, then run again?
  - Unnecessarily increases complexity
  - Violates encapsulation
  - Don’t just throw setters in if you don’t have a reason

- Beans are one exception to this rule
  - Functionality is *only* get & set
  - Little other functionality
    - *Mapping* to validation
    - *Mapping* to relations

- Generally prefer immutability where possible
Concurrency is Always a Risk

- Treat anything concurrent with initial distrust
  - Race conditions $\rightarrow$ Denial of Service
  - Shared memory $\rightarrow$ Potential Leakage
  - Weird circumstances $\rightarrow$ Potential Tampering

- Concurrency is ubiquitous: webapps, databases, GUIs, games, etc.

- Common poor assumptions
  - “There will be only one copy of this thread”
  - “There will only be X threads”
  - “Nobody knows about my mutability”
Advice for Concurrency

- Create a concurrency concept and keep it as simple as possible, but as complex as needed.
- Document the concurrency concept!
- Make use of synchronization primitives!
- Synchronization needs to protect *data*, not code!
- Challenge: minimizing serial execution time while doing synchronization right
Serial Killer

- Serialization is often unnecessary, difficult to get right
- Deserializing is essentially constructing an object without executing the constructor
  - If your system uses it, don’t assume the constructor will be executed
  - Can reverse-engineer to violate constructor post-conditions
- Also, serialized != encrypted
  - Confidentiality disclosure
  - Use `transient` for variables that don’t need serialization
    e.g. environment info, timestamps, keys
  - OWASP recommends:

```java
private final void readObject(ObjectInputStream in) throws java.io.IOException {
    throw new java.io.IOException("Class cannot be deserialized");
}
```
Summary

- Think about security from the beginning
- Keep your code simple
- Handle failures properly
- Don’t trust input
- Be very careful with error output
Brief Problem Overview

Disclaimer: Only the specification document is binding! The information provided here is meant as an introduction.

- An art gallery wants to log, which persons entered the gallery and its rooms and when they did so
- … in a “secure” way.
- The software to be implemented consists of two programs “logappend” and “logread”
- logappend: add information to the log file in a controlled way
- logread: read information from the log file and perform various statistics
- Obfuscation is not allowed!
  - No bogus code (method calls, classes, unnecessary variables, etc..)
  - No identifier renaming
  - No encrypted source code
  - …
- Obfuscated submissions will be disqualified!
Built-it Phase

- Register at the course platform: https://bibifi.cs.upb.de/
  - Please fill out the registration form!

- Build groups of four people and register as such at the course platform (/profile/Teams)
  - Register as team to the „S2017“ contest (announcements/sign up)
  - Come to us if you cannot find a group or your group is too small!

- Create a git repository for your group on the University GitLab: https://git.cs.upb.de
  - Grant user „Birkhard Bifi“ reporter access to that repo
  - Grant everyone in your group at least developer access
  - Register the team repository at the course platform (announcements/participants)
Built-it Phase

- Your code goes into the **build** directory in your repository root!

- Further information on the Gallery problem and usage of the infrastructure can be found in the specification: [https://bibifi.cs.upb.de/static/doc/s2017/spec/index.html](https://bibifi.cs.upb.de/static/doc/s2017/spec/index.html)
  - Should by now be linked in the announcements
  - Read carefully, everything you need is there! Ask if unsure after **second** read through.

- Your implementations will be tested against our oracle implementation
  - We have test cases for correctness, performance and optional features that are executed on each commit to your repository

- If you are unsure about the expected behaviour of your application, you can always ask the oracle for the correct answer (explained in specification)
Built-it Phase

- Please use the VM to make sure your Makefile and executables work before complaining about the infrastructure
  - The infrastructure will provide you only with the information printed to stdout
  - If nothing is printed, chances are high that your Makefile is broken!
  - Again, read the specification carefully to reveal possible mistakes of your implementation or check-in process!

- A lot of contest points can be achieved during the Build-it phase, make sure you use the time and implement as much as you can!
  - Every break of your implementation will loose you points

- Last but not least:

  HAVE FUN!
2nd Phase

- How to submit your exploits?
  - Bibif….upb.de
  - Demo on how to do that.

- ...

...
3rd Phase

- How to submit your reports?
  - Bibif….upb.de or email?
## Lecture Schedule (preliminary)

<table>
<thead>
<tr>
<th>Date</th>
<th>Topic</th>
<th>Lecturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>31.07.</td>
<td>Build-it Lecture</td>
<td>Martin, Manuel</td>
</tr>
<tr>
<td>03.08.</td>
<td>Q&amp;A with supervisors (optional)</td>
<td>All three of us</td>
</tr>
<tr>
<td>07.08.</td>
<td>Build-it Presentations &amp; Break-it Lecture</td>
<td>Stefan</td>
</tr>
<tr>
<td>10.08.</td>
<td>Q&amp;A with supervisors (optional)</td>
<td>All three of us</td>
</tr>
<tr>
<td>14.08.</td>
<td>Break-it Presentations &amp; Fix-it Lecture</td>
<td>Martin</td>
</tr>
<tr>
<td>17.08.</td>
<td>Q&amp;A with supervisors (optional)</td>
<td>Martin &amp; Manuel</td>
</tr>
<tr>
<td>19.08.</td>
<td>Submission of final reports</td>
<td></td>
</tr>
<tr>
<td>20.08.</td>
<td>Final Presentations</td>
<td></td>
</tr>
</tbody>
</table>