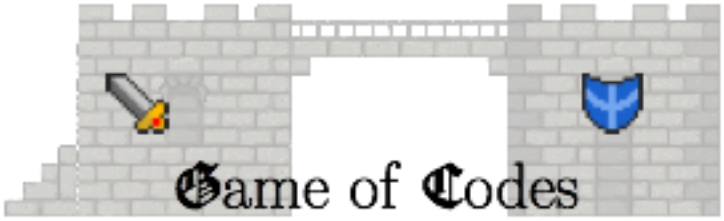
Secure Coding Project



Phase 4: White-Box Analysis &
Reverse Engineering

Phase 4: White-Box Analysis & Reverse Engineering

- The bank for which you are developing the Internet banking web application is still not convinced that your solution is secure
- Therefore, it hired another team of engineers who will test your application's PHP sources, the C/C++ batch processing binary and the Smart-Card Simulator (SCS) JAR for vulnerabilities
- You are also given one application to test, developed by one of your competitors, but this time you have access to their PHP code, but not to the C/C++ or Java code
- **Your goal:** perform systematic **white-box testing and reverse engineering** of the given application and your own app
- **Deadline:** 14:00 AM, 8th January 2016

Phase 4: White-Box Analysis & Reverse Engineering

- **Target:** PHP source code and C/C++ binary from the USB stick. The SCS JAR file can be obtained by registering a new user in their new web-app and choosing the SCS option.
- **NOTE:** Use the VM as a black-box, don't try to break the password of the OS user (no points will be awarded for this)
- Confirm whether vulnerabilities found in white-box fashion are **true positives**!

- **Documentation:** slides with use-case description from the other team's USB stick

- **Testing tools:**
  - any tool installed on your own Samurai WTF VM
  - white-box / static analysis tools mentioned during the tutorials / lectures
  - use at least 2 tools per team member
  - tools used by different members must be different

- **Requirements:**
  - use a systematic approach, e.g. via checklists such as OWASP testing guide
  - use the advanced features of tools where possible
  - creating your own tools/extensions/scripts is encouraged

Phase 4: White-Box Analysis & Reverse Engineering

- **Static analysis of PHP source code:**
  - PHP source code must be statically analyzed either manually or using tools, scripts
  - Useful list of tools: https://www.owasp.org/index.php/Source_Code_Analysis_Tools
  - Tools generally find a large number of false positives. You must check if the results given to you by tools are false positives or not. Write in the report what the percentage of false positives are for each tool and for each type of vulnerability that the tool finds.

- **Reverse engineering for C/C++ and Java:**
  - **The goals of reverse engineering are**:
    - To understand what the algorithm of the program. You must write an equivalent program in C/C++ code for the given binary. **Equivalent means** that it must have the same input-output behavior, not the same exact syntax or variable names.
    - Extract any credentials and hidden secrets (keys) from the binary and JAR and write them in the report.
  - **C/C++ binary must be reverse engineered using:**
    - an (interactive) disassembler, e.g. IDA Free/Pro, objdump, Boomerang
    - a debugger, e.g. gdb, IDA Free/Pro, ddd
    - **NOTE:** This is a warm-up exercise for the obfuscation challenge in phase 5, which will be performed by every student individually. Everyone is invited to reverse engineer this un-obfuscated binary to get familiar with the concept/technique.
  - **Java SCS must be reverse engineered using:**
    - a decompiler and/or a debugger

Deliverables for Phase 4

- Deadline is (Friday) 8th of January 2016 at 14:00

  → You can hand in via **e-mail** or USB stick before this date at the office MI 01.11.040 or on that date in the classroom

  → There will be a 1% grade penalty for every minute after the deadline

- E-mail / USB stick must contain the following files:
  1. **Presentation-TeamX-Phase4.pdf** (NO .PPT or .pages files)
  2. **OWASP-Checklist-TeamX-Phase4.xls (**Excel 97-2003**)**
  3. **WBTestingReport-TeamX-Phase4.pdf** (where X is your team number)
  4. **Binary-equivalent-TeamX-Phase4.c** (C source code equivalent to the binary from other team)

  The structure of the presentation, checklist and report are presented on next slides

- Teams and students who present are picked randomly on the presentation date

  → If the picked student is not present s/he gets zero points for this phase (except if s/he can provide written evidence that the absence was justified)

Structure of Phase 4 Presentation

Presented in **MAXIMUM 7 MINUTES**

NOTE: In the presentation you should **talk mainly about the competitor's app**, not your own app

1. First slide: team number and team members (full-names)

2. Second slide: enumeration of most important vulnerabilities in competitor's app
   - **Mention which vulnerabilities were already found in Phase 2 and which only in Phase 4**
   - Mention name, impact and likelihood of each vulnerability
   - Impact and likelihood are measured as: low, medium or high
   - For each vulnerability give the CVSS 3.0 template **and** the score (next slide)
   - Don't mention vulnerabilities that were **not** found

3. At this point in the presentation you either give a live-demo or video-demo of the vulnerabilities or present them in more detail on the following slides

CVSS v3.0 template

	Enter your data below
Access Vector	
Access Complexity	
Privileges Required	
User Interaction	
Scope	
Confidentiality	
Integrity	
Availability	
Score	

Structure of Phase 4 Checklist

The checklist will not be presented during the lecture

- Custom version of the OWASP testing checklist https://docs.google.com/spreadsheets/d/1RZYzw7OXjjmiG0U1F8gYCV_SGrZY3guSncRwluwmiGM/edit?usp=sharing
- Fill out columns G and H in the spreadsheet from previous URL

- **Testing columns** (G and H) should be filled out with 2 possible color-coded values. Secure (green), Vulnerable (red), Not Tested (yellow), NA for Not Applicable (gray)
  - Insert note/comment for each cell that is Secure (green) or Vulnerable (red) with reference to the page in the WB testing report document where testing for this vulnerability is performed (example notes are provided for a few cells of the spreadsheet linked above)

- See the description of tests in the OWASP Testing Guide https://www.owasp.org/index.php/OWASP_Testing_Project

Structure of Phase 4 White-Box Testing Report

The report will not be presented during the lecture

NOTE: In the report you MUST talk about both your app and the competitor's app

1. First Page: Course title, Phase title, team number and team members (full names)

2. Second Page: executive summary (maximum 1 page). Must contain the most critical vulnerabilities and their associated consequences.

3. Third Page: table of contents

4. Fourth Page: time tracking table (info / student / task, max 2 hours per task)
Tasks should be evenly distributed – project management part (this will be graded)

5. Fifth Page: overview of most important observations with impacts, likelihood and risk estimation. Each vulnerability should also have the CVSS 3.0 score. If necessary add a table with a legend for special symbols or abbreviations used in the document.

6. Next few pages: **compare** the 8 different tools used by your team. Which vulnerabilities can each tool find? Which vulnerabilities can they not find?

Structure of Phase 4 White-Box Testing Report (continued)

Remaining pages: describe identified vulnerabilities mentioning the following for each one

- **Observation:** which part of the application is vulnerable and why?
- **Discovery:** how was this vulnerability discovered? Which tools were used and which steps were performed?
- **Likelihood:** what is the likelihood that this vulnerability is exploited? Which assumptions must hold and which skills must an attacker have?
- **Impact:** what is the potential impact of an exploit of this vulnerability? What could happen?
- **Recommendation:** how exactly would you fix this problem?
- **Instantiate the CVSS v3.0 template** for this particular vulnerability.
- **Comparison with your own application:** how and why is your app better/worse

NOTE: Each vulnerability should be described on a new page!