Dr. Tim Blazytko
M. Sc. Philipp Koppe

Lecture Software Protection
Summer Term 2025
Ruhr-Universität Bochum

Task Sheet 4

July 4, 2025
Submission due to July 27, 2025, 11:59 PM via Moodle.
For question, feel free to ask via Discord or Moodle.

Tasks

Course assessment is based on 4-5 task sheets which will be completed within the semester. The final grade consists of the weighted average grade of all task sheets. We will report every student to the examination office who handed in at least two tasks. In other words, we allow students to drop out after handing in a single task. Task sheets have to be worked on individually; group work is not possible.

Optional tasks are voluntary and independent of the course assessment; in other words, they do not influence the final grade. Instead, they are designed for students who want to dive deeper into individual topics. These submissions will be ranked with subjective score points. The student who has the highest score at the end of the semester wins a Binary Ninja student license.

The fourth task sheet covers the design, implementation and bypassing of various anti-tamper techniques. While the first tasks focuses the design and implementation, the second as well as the optional task cover the analysis and breaking of such schemes.

Please submit your solutions as either text files, if several files have to be submitted, as zip files. Please do not hand in `.doc` or similar formats. If you have to include images or need some special document layouting, export the document as pdf file. The submitted binaries should be ELF files (Linux) and compiled with GCC for x86-64.

Task 1: Design & Implementation of Anti-tamper Techniques (15 Points)

`license_check.c` implements a basic license validation check. Your goal is to extend the source code with various anti-tamper techniques to prevent dynamic analysis and patching under Linux. We don't set any limitations to your creativity as long as the final program semantically computes the same results (a valid user/license pair is "sp@rub.de"/"dgWebu9sr"). Proceed as follows:

(a) Encrypt all strings (from the source file) in the binary. It should not be possible to statically extract the strings in their original form by opening the file in IDA or Ghidra. (2 Points)

(b) Hide all used imports/API functions. It should not be possible to see the imports by opening the file in IDA or Ghidra. (2 Points)

(c) Implement at least two different anti-debug checks, including varying reactions/responses in a case a debugger has been detected. Insert those checks/responses such that they are *reasonable* with regards to the application's context. (2 Points)

(d) Implement at least two different environmental checks to detect if the application runs in a virtual machine (VirtualBox). Describe how your checks work. (2 Points)

(e) Design and implement some kind of code checksumming which detects and circumvents code patching. In particular, it should neither be possible to patch the conditional jump in the main function to always accept the license nor to patch the conditional jumps in `validate` such that the function always returns 1. Describe how your approach and implementation works. (4 Points)

(f) Implement at least one of the aforementioned anti-tamper techniques on top of a custom virtual machine. Describe the VM's design and implementation. (3 Points)

For each anti-tamper method, explain how you designed it, how it works and why you inserted it at exactly at this location. Submit your notes, commented source code as well as a protected binary (for Linux, compiled via GCC).

Task 2: Bypassing Anti-tamper Techniques I (7 Points)

The executable `license_validation` is based on `license_check.c`. Your goal is to bypass the anti-tamper techniques and patch the binary such that it accepts every license. In particular, the binary should **not** crash. Proceed as follows:

(a) Analyze the anti-tamper techniques. Which are deployed and how do they work? Describe at least two. (3 Points)

(b) Bypass the anti-tamper techniques and patch the binary such that it accepts any license. Describe your approach & explain which code locations you patch and how/why. (4 Points)

Submit your notes as well as the patched binary.

Optional Task: Bypassing Anti-tamper Techniques II

Similar to the task before, your goal is to bypass the anti-tamper techniques in the binary `license_validation_hard` and patch the it such that every license is valid. Analyze the protection mechanisms, describe how they work and bypass them. Submit your notes as well as the patched binary.