Dr. Tim Blazytko
M. Sc. Philipp Koppe

Lecture Software Protection
Summer Term 2025
Ruhr-Universität Bochum

Task Sheet 3

June 5, 2025
Submission due to July 6, 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 third task sheet covers the generation and analysis of various (real-world) MBA constructs. The optional exercise includes a deep dive in to a heavily obfuscated challenge.

For some of the practical implementation tasks, you will use the Miasm reverse engineering framework. While an older, outdated stable version can be installed via PIP, it is recommended to choose (one of the) latest GitHub commits.

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.

Task 1: MBA Construction (8 Points)

One way to construct MBAs is to insert non-trivial identities (such as $(x \oplus y) + 2 \cdot (x \wedge y)$ for $x + y$) into invertible functions. An invertible function is a function f for which exists an inverse f^{-1} such that the following holds for all arguments x:

$$f(f^{-1}(x)) = x$$

For example, $f : x \mapsto 39x + 23$ and $f^{-1} : x \mapsto 151x + 111$ are invertible functions of degree 1 on the byte level:

$$\forall x \in \{0, \ldots, 255\} : f(f^{-1}(x)) = f(151x+111) = 39(151x+111)+23 = 39 \cdot 151x+39 \cdot 111+23 \equiv x$$

Your task is to build a Python program which automatically builds MBAs based on synthesized invertible byte level functions of degree 1. Proceed as follows:

(a) Design an algorithm which generates random invertible functions on the byte level. You can limit the invertible functions to functions of degree 1. Describe how and why the algorithm works. (3 Points)

(b) Implement the algorithm in Python and construct the invertible functions as expressions in Miasm IR. (2 Points)

(c) Extend your implementation to automatically insert non-trivial identities into randomly generated invertible functions and construct at least 10 different MBAs in Miasm IR. For the sake of simplicity, you can use the non-trivial identity for the addition from above. (2 Points)

(d) For each generated MBA expression, prove with an SMT solver that it is semantically equivalent to $x + y$. (1 Point)

Submit your notes on algorithm design, your commented Python implementation and the output of an example run (in a text file).

Task 2: MBA Deobfuscation (8 Points)

In this task you will analyze and break the obfuscation of a malware sample which uses custom MBAs. Proceed as follows:

(a) Analyze the obfuscation in the sample `mba_malware`. How does it work and what is it used for? (2 Points)

(b) Come up with an idea to break the obfuscaton. Describe it in a text file. (2 Points)

(c) Use Miasm to implement your deobfuscation approach. Detect all control-flow edges which cannot be taken in the function at address `0x1800873d0`. You can additionally use msynth with a small pre-computed database, but there are also other ways to approach it. Optionally, you can also patch the binary. (4 Points)

Submit your notes, commented Python code, the output of an example run (in a text file) and, if existing, your patched binary.

Task 3: Obfuscation Analysis I (9 Points)

In 2021, the Grand Reverse Engineering Challenge[1] included a binary `challenge-4-x86_64-Linux.exe` which remains unbroken until today. Your goal is to analyze how the obfuscation works and figure out why it is that hard to analyze. Proceed as follows:

(a) Describe how the call obfuscation works and resolve the call in basic block `0x407566`. (2 Points)

(b) Analyze the control-flow flattening obfuscation in function `0x400660`. How does it work? Why is it hard to predict the next state? (2 Points)

(c) Take a closer look at the MBAs. How do they work? (2 Points)

(d) Locate one basic block in function `0x400660` which encodes a conditional jump. Describe your analysis process. (3 Points)

In this task, you are free to choose whatever tooling and approach you like. For the individual questions, describe what you did and how you came to your conclusions. Submit your answers in a text file. If you write some analysis tooling, include it in the submission.

[1] https://grand-re-challenge.org/

Optional Task: Obfuscation Analysis II

In this task, you will dive deeper into the challenge from above. Your goal is to learn as much about the obfuscation as possible and come up with some deobfuscation ideas. This can be realized in various ways:

(a) Perform a global analysis. Write a script to resolve all calls and reconstruct the call tree.

(b) Analyze how MBAs are applied on a larger scale. Do your recognize any patterns?

(c) Try out different deobfuscation approaches or come up with some ideas on how you would approach it.

While it is not impossible to remove large parts of the obfuscation, it is unlikely. However, this does not mean you can't achieve any partial results. Whatever direction you go, describe what you did, your thought process, and analysis results. This can be some notes in text files, annotated IDA databases or whatever works. If you write any analysis tooling, feel free to submit it.