HOGESCHOOL ROTTERDAM / CMI

Security and Advanced Development Minor

TINFUN01

Aantal studiepunten:	30 ects
Modulebeheerder:	Arne Padmos, Giuseppe Maggiore

Modulebeschrijving

Module name:	Security and Advanced Development Minor
Module code:	TINFUN01
Study points and hours of effort:	This module gives 30 ects, in correspondance with 840 hours: • 9 x 20 hours frontal lecture • the rest is self-study and project work
Examination:	Projects and tests (open questions and multiple choice)
Course structure:	Lectures, self-study, and projects
Prerequisite knowledge:	All programming courses, and fundamentals of security.
Learning tools:	For the security part: • Book: Everyday Cryptography ISBN-13: 978-0199695591 • Book: Cryptography Engineering ISBN-13: 978-0470474242 • Book: Threat Modeling Book • PDF: Security Engineering • Usable Security: History, Themes, and Challenges • Book: The art of software security assessment ISBN-13: 978-0321444424 • Book: Phishing Dark Waters: The Offensive and Defensive Sides of Malicious Emails ISBN: 978-1-118-95847-6 For the formal methods part: • Book: Friendly F# (Fun with game programming Book 1), authors: Giuseppe Maggiore, Giulia Costantini • Reader: Friendly F# (Fun with logic programming), authors: Giuseppe Maggiore, Giulia Costantini, Francesco Di Giacomo, Gerard van Kruining
Connected to competences:	• Analysis of problems with the objective of building concrete solutions • Design of solution architectures in connection with a previously performed analysis • Realization of solutions based on a previously performed design • Advice and communication in the form of documentation and clear explanations of intentions and results • Overall management of processes and team work

Learning objectives:	In the context of security, the student can: • formulate strategies to adequately evaluate security `SEC1` • dissect and construct the layers and objects of security `SEC2` • apply and engage in the red vs blue team method either in the context of implementing/configuring or in the context of sketching/architecting `SEC3` • apply and engage in the white box/laboratory study method either in the context of a code review or in the context of a user study `SEC4` • apply and engage in the black box/field study method either in the context of a *pentest* or in the context of a phishing campaign `SEC5` In the context of formal methods, the student can: • determine abstract complex properties `FM1` • encode these complex properties into languages, tools, and other libraries `FM2` • dissect the layers and objects of languages, tools, and other libraries `FM3` • architect languages, tools, and other libraries in a formal specification `FM4` • build languages, tools, and other libraries within a functional, logic, or declarative programming language `FM5`
Content:	For the security part: • Fundaments of cryptography • Computer security • Network security • Humanoid security For the formal methods part: • Functional programming, with special focus on F# • Meta-programming within F# • Meta-compilers, with special focus on the school's own implementation
Module maintainers:	Arne Padmos, Giuseppe Maggiore
Date:	1 september 2015

1 Programma

The course covers a series of topics from two different branches: security and formal methods.
Under security we understand a variety of applied concepts related to the security of multi-user, complex systems where information has varying degrees of confidentiality.
Under formal methods we understand a variety of techniques that make it possible to encode properties such as security, performance, etc. of complex systems into programming languages or similar constructs. This allows users to easily find violation of such properties reliably and without testing through advanced compilers.

Security topics

....

Formal methods topics

The minor mainly focuses on the implementation of formal methods with a meta-compiler, provided by the lecturers. This meta-compiler is accompanied by sample implementations of programming languages, of varying degrees of complexity.
The meta-compiler itself is a formal system, built in the F# programming language. This means that as an additional level of complexity it is possible for students to add highly abstract properties to the meta-compiler instead of using it to implement languages with these properties.

2 Assessment

The assessment of the course is divided into two main parts. Part I is a series of brainstorming sessions where groups are formed and project topics are chosen; in connection with the chosen topics, teachers will determine an appropriate set of preparation exercises that the students will need to complete after one month. Part II is one or more projects chosen by the students ranging from purely security to formal methods to hybrid solutions.

2.1 Grading

Grading of Part I is based on correctness of the assignments. The questions are either multiple choice, or are technical exercises with an either fully correct or fully incorrect answer. Each question adds one point to the total, so that the final grade each week is:

$$\frac{P}{N} \times 10$$

where P is the number of correct answers and N is the total number of answers.
Grading of Part II is based on adherence to the students *plan van aanpak*, and evaluation of the expected satisfaction of the client. **Handing-in** is completely based on GitHub, and the scale of grading is as follows:

Grade range	Process	Results
0 - 2	The students cannot document any significant work done on the project. There are few to no check-ins in the shared repository, and the repository itself is substantially empty.	There are no usable results to speak of.
2 - 4	The students have done very little work on the project. There are few check-ins in the shared repository, and the repository itself is poorly organized and has little useful content.	A few barely usable results are visible.
4 - 5,4	The students have done little work on the project. There are few check-ins in the shared repository, and the repository itself has some glimpses of organization and some useful content.	Limited and barely usable results are visible.
5,5 - 7	The students have done some work on the project. There are daily or weekly check-ins in the shared repository, and the repository itself is organized and contains useful content.	Primitive usable results are visible and clearly identified in a release accessible through the wiki pages of the repository.
7 - 10	The students have done significant amounts of work on the project. There are daily or weekly check-ins in the shared repository, and the repository itself is organized, efficient, and contains only useful content.	Concretely and highly usable results are visible and clearly identified in a release accessible through the wiki pages of the repository.

At the end of the minor all evidence must also be uploaded by students to N@tschool.

Appendix: examination matrix

Leerdoelen	Dublin descriptoren
SEC1	...
SEC2	...
SEC3	...
SEC4	...
SEC5	...
FM1	1, 2
FM2	4, 5
FM3	1, 2
FM4	3, 4
FM5	4, 5

Dublin-descriptoren:

1. Knowledge and understanding
2. Application
3. Analysis
4. Evaluation
5. Creation

Formal methods project - guidelines

An acceptable project for the formal methods aspect of the SAD minor covers the design of a programming language or comparable formalism with clearly specified useful properties.
The properties of interest for the minor include, but are not limited to:

- security aspects; for example, build a programming language which type system makes it impossible to send classified data without encryption over a network
- safety aspects; for example, build a programming language which type system makes it impossible to duplicate a unique value
- performance aspects; for example, build a programming language which uses code-generation to speed up some decisions by making them compile-time instead of run-time.

For a longer list of proposals, see `https://github.com/vs-team/metacompiler/issues/`. You may add your own proposal.

Structure of a programming language

An important word of warning concerns the use of tools and languages. It is, in theory, possible to build your own system from scratch, using a programming language made for other tasks (such as Java or C/C++). It is the teacher's estimation that doing so will almost always result in your failure, which for a whole minor has dire consequences. The reason why this will happen is simple. A project such as the ones described above has (very roughly) the following structure:

TESTING	SAMPLES*
BACKEND	• CODE-GEN • OPTIMIZER* • ANALYZER*
FRONTEND	• PARSER • LEXER

The only parts relevant and graded during the minor are those marked with an asterisk, that is the *analyzer* and the *optimizer* of the language. These are the parts where "real intelligence" can be put into a programming language, through knowledge of formal systems. Everything else is irrelevant for formal systems, and therefore not graded. The reason is simple: grading all of it would quickly put us out of scope and require way too much work on your behalf. Building a reliable front-end is a lot of work. In addition, building a well-working, reliable code generator that outputs useable code is a daunting task. Finally, whatever you choose to do, you will need to provide a series of basic samples that show that all aspects of your language work correctly.

Allowed tools and languages

To shield you all from this work, which would be significantly more than a single minor, we provide you with a ready-made meta-compiler that allows the definition of programming languages by focusing directly on the analyzer and the optimizer. The meta-compiler comes equipped with an extensive suite of tests that will help you understand if you broke something, and also how the program works.
You are very strongly encouraged to make use of the meta-compiler because you will get more done, and you will also get more help as the teachers are very expert in it. The meta-compiler is accessible as a GitHub project at `https://github.com/vs-team/metacompiler`.
Should you be interested in actually doing something within the meta-compiler itself, then it is also possible. The meta-compiler is written in F#, and therefore you may add features to it in the same language.

You may also suggest your own project (which should still be inspired from the existing open issues). In this case, use of F# is a still a strict requirement, and previous acceptance from the teacher is required.

To sum it up

If you are looking for a relatively easy task, then learn to use the meta-compiler and build your project in it. If you are looking for a challenging task, then learn to use both the meta-compiler and F#, and build a feature in the meta-compiler itself. If you are looking for a learning experience through very likely failure, then build the whole structure of your system in F# from scratch.