GHENT UNIVERSITY

Course Specifications

Valid as from the academic year 2024-2025

Secure Software and Systems (E017950)

Course size *(nominal values; actual values may depend on programme)*

Credits 6.0 **Study time 180 h**

Course offerings in academic year 2024-2025

A (semester 2) English Gent

Lecturers in academic year 2024-2025

Coppens, Bart	TWO6	lecturer-in-charge

Offered in the following programmes in 2024-2025	crdts	offering
Bridging Programme Master of Science in Bioinformatics(main subject Engineering)	6	A
Master of Science in Bioinformatics(main subject Engineering)	6	A
Master of Science in Computer Science	6	A
Master of Science in Computer Science Engineering	6	A
Master of Science in Computer Science Engineering	6	A
Master of Science in Information Engineering Technology	6	A
Exchange Programme in Computer Science (master's level)	6	A
Exchange Programme Information Engineering Technology	6	A

Teaching languages

English

Keywords

Security vulnerabilities, software attacks, exploits, secure code, code injection, remote code execution, defensive and offensive security techniques, security policy, legal limitations, Internet security, side channels, security analysis, security by design, secure software development

Position of the course

Software systems control a large part of our lives and society at large, and process an ever-growing amount of personal and private data. The security of these systems is thus of the utmost importance: a small design flaw or implementation bug in the code of such a system can lead to a loss of confidentiality (for example, leaking private data), a loss of integrity (for example, an attackers being able to take over control of the system), and availability (such that legitimate users can no longer access the system). Unfortunately, most software is riddled with such bugs. Thus, it is of the utmost importance to prevent the exploitation of bugs or flaws. To do so, we need to understand how exactly an attacker can leverage which bugs into working exploits. This will provide insights into how we need to mitigate the impact of such exploits, and how to prevent these bugs from being exploitable. By appropriately limiting the programming constructs used when developing software, some classes of exploitable bugs can even be eliminated entirely.
In this course, we will exploit, mitigate, and prevent vulnerabilities in software systems. We focus on (remotely) exploitable flaws and bugs in software designs and implementations through which an attacker can gain (more) control over a system (escalation of privilege, remote code execution, ...), or can gain unauthorized access to information. We study the root causes of design flaws and implementation bugs, and study how these can be exploited by an attacker to violate the confidentiality, integrity, and availability of a system. We will pay particular attention to bugs caused by the use of lower-level programming languages such as C and C++. We furthermore study how we can find vulnerable constructs in code during testing. We study and apply methods that allow us to

create code that inherently contain less vulnerabilities. We compare different ways to mitigate the impact of attacks (potentially aided by extensions of the operating system), which includes preventing entire classes of attack.
This course complements the courses Software Hacking and Protection, which focuses on man-at-the-end attack techniques, which are deployed by parties that have (almost) complete control over the end devices on which they attack and analyze the software, such that they don't need to rely on exploitable vulnerabilities for their attacks and analyses; and Network Security, which focuses on attacks with which network infrastructures are penetrated.
This course builds on the knowledge gained in earlier courses on computer architecture (in particular on x86 assembly), operating systems, and programming in low-level languages (C, C++).

Contents

- vulnerabilities: classifications, bugs vs flaws, tracking, disclosure processes, undefined behavior
- vulnerability exploitation techniques: control data attacks, non-control data attacks, including HW-based vulnerabilities and exploitation (covert channels, timing-based side channels, Spectre, Meltdown, ...)
- exploit mitigation techniques (diversification policies, integrity policies, sandboxing, hardware-based mitigations, ...)
- secure software architecture, development and management, principle of least privilege
- unsafe and safe programming languages, secure coding patterns, secure coding guidelines
- tools and techniques for detecting vulnerabilities: fuzzing techniques, sanitizers, testing, model checking, symbolic and concolic techniques, ...
- tools and techniques for maintaining and attestating system integrity
- security and privacy by design applied to software and systems
- secure software development life cycle aspects, software maturity models, DevSecOps, Shift Left, ...
- system threat modeling, security requirements
- system security economics and legal aspects of hacking and security issues, bug bounties, ...

Initial competences

- Programming in C and C++
- Knowledge of computer architecture
- Knowledge of assembler (x86)
- Knowledge of operating system internals
- Basic knowledge of software engineering practices and paradigms
- Basic knowledge of databases, programming in SQL

Final competences

1 Have an understanding of, and be able to use the terminology of basic aspects of systems and software security, of risks, security requirements, handling of security issues, policy options, legal limitations (ethical hacking, white hat vs. black hat, responsible handling and disclosing of sensitive information, ...).
2 Deep understanding of vulnerabilities in software, being able to find them, analysing their impact, and constructing exploits for them.
3 Deep understanding of preventive counter-measures and detection methods, being able to choose and apply and combine appropriate protections.
4 Creating software systems that contain fewer vulnerabilities, and are able to resist exploits better.
5 Knowledge of security by design software development principles.
6 Have insight in attacks based on systems being implemented on specific hardware platforms (side channel attacks) and in protections to mitigate those attacks.
7 Communicating and presenting domain-specific knowledge in a correct and clear manner, with the appropriate language skills, including the correct use of terminology.

Conditions for credit contract

Access to this course unit via a credit contract is determined after successful competences assessment

Conditions for exam contract

This course unit cannot be taken via an exam contract

Teaching methods

Group work, Lecture, Practical, Independent work

Extra information on the teaching methods

- We will create exploits for x86-64 code in some of the labs. This means that students should be able to run x86-64 virtual machines.

Study material

Type: Slides

Name: Slides with lecture notes provided by the lecturer
Indicative price: Free or paid by faculty
Optional: no
Language : English
Available on Ufora : Yes

Type: Other

Name: Papers for reading assignments
Indicative price: Free or paid by faculty
Optional: no
Language : English
Available on Ufora : Yes

References

- Security Engineering: A Guide To Building Dependable Distributed Systems, Third Edition, by Ross Anderson, 2020. ISBN 978-1-119-64278-7
- Threat Modeling: Designing for Security, by Adam Shostack, 2014. ISBN 978-1-118-80999-0
- The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities, by Mark Dowd, John McDonald, and Justin Schuh, 2007. ISBN 978-0-321-44442-4
- Computer Security and the Internet: Tools and Jewels from Malware to Bitcoin, by Paul C. Van Oorschot, 2021. ISBN 978-3-030-83410-4

Course content-related study coaching

- Interactive support and coaching through the electronic learning platform appointments
- Teachers and assistants are available for extra help before and after contact hours, and via chat and conf calls
- Use of flipping the classroom and blended learning techniques

Assessment moments

end-of-term and continuous assessment

Examination methods in case of periodic assessment during the first examination period

Written assessment

Examination methods in case of periodic assessment during the second examination period

Written assessment

Examination methods in case of permanent assessment

Oral assessment, Participation, Presentation, Peer and/or self assessment, Written assessment, Assignment

Possibilities of retake in case of permanent assessment

examination during the second examination period is possible in modified form

Extra information on the examination methods

- Periodic evaluation: written examination with open and closed questions on theory and practice (exercises), with closed-book
- During semester: graded lab sessions (including written reports and possibly oral evaluation); project; participation 'flipping the classroom' activities

Calculation of the examination mark

- 50% on permanent evaluation, 50% on periodic exam.
- Lack of participation in permanent evaluation for no valid reason results in a zero for that part.

- In the case of group assignments, the students in a group get the same score by default. Only when there is a clear difference in contribution, the students will be given different scores.
- The student must pass (>=10/20) both parts to pass the whole course. If they fail for one part while still scoring >=10/20 on average, the final score becomes 9/20.

Facilities for Working Students

Option to be freed from presence in labs with alternative assignment after consultation with the responsible teacher. Option for oral exam with written preparation at another time in the academic year. Option for feedback by appointment during and after business hours.