Recep Tayyip Erdoğan University

Faculty of Engineering and Architecture — Computer Engineering

CEN429 Secure Programming — Syllabus

Fall Semester, 2026-2027

Course Information

Instructor	Asst. Prof. Dr. Uğur CORUH
Contact	ugur.coruh@erdogan.edu.tr — subject line must start with **[CEN429]**
Office	F-301
Office hours	By appointment by e-mail; meetings in the office or online with the university account
Lecture day, time, room	Friday 09:00–12:00 · İİBF & Faculty of Law Building, D-402 (ED-K4-2)
Course website	https://ucoruh.github.io/cen429-secure-programming/
Course class	A new class is opened every term; the class code is announced in week 1
Language	Turkish
Type / semester	Elective · 7th semester
Weekly hours / credit / ECTS	Theory 3 h · Credit 3 · ECTS 5
Prerequisite	CEN107 Algorithms and Programming I (former code CE103)

A. Course Description

This course offers a comprehensive approach to understanding secure software development techniques. Students learn software protection methods to address common vulnerabilities such as buffer overflows, memory leaks and injection attacks: data security and cryptography, code hardening for C/C++ and Java, runtime application self-protection (RASP),

code obfuscation and diversification, white-box cryptography, security standards and penetration test planning. The course focuses on real-world applications and best practices in secure software development. Each topic follows the order vulnerable code → attack → fix and is reinforced by in-class exercises and by a term project run as if it were going through a certification process.

B. Learning Outcomes

A student who completes this course successfully:

Code	Learning outcome
LO.1	Identifies and classifies common software vulnerabilities (buffer overflow, injection attacks, memory leaks, etc.).
LO.2	Explains basic encryption methods (symmetric/asymmetric, hash functions) and secure communication principles (SSL/TLS) for protecting sensitive data.
LO.3	Explains code hardening techniques (input validation, secure memory management, RASP, code obfuscation) and applies them to different languages (C/C++, Java).
LO.4	Explains the principles of building secure communication channels using encryption and authentication mechanisms.
LO.5	Creates a software plan using secure software design principles (least privilege, defence in depth, etc.) and defence strategies.
LO.6	Knows basic security review and vulnerability assessment methods for detecting software vulnerabilities.
LO.7	Knows secure programming standards (e.g. ETSI, EMV, FIPS) and the principles of penetration test planning.

Contribution of learning outcomes to program outcomes (0–5)

	PO.1	PO.2	PO.3	PO.4	PO.5	PO.6
LO.1	–	3	–	5	–	–
LO.2	–	–	–	3	–	–
LO.3	–	3	3	3	–	–
LO.4	–	–	3	3	–	–
LO.5	–	3	3	3	–	–
LO.6	–	3	–	3	3	–
LO.7	–	–	–	3	–	–

PO.2 Problem solving · PO.3 Design · PO.4 Modern tools and techniques · PO.5 Research and experimentation · PO.8 Communication · PO.9 Social awareness · PO.11 Ethics and standards · PO.12 Project and risk management.

C. Weekly Schedule

Rule for all assessments: **project demonstrations take place in the week just before the midterm and final exam weeks; quizzes take place inside the midterm and final exam weeks** so that every student can attend. Topics follow the order defined in ritim.

Week	Date	Topics	LO
1	18.09.2026 (make-up 30.09)	Course plan and communication. Introduction to secure programming: security goals (confidentiality, integrity, availability), attacker model, threat modelling (attack trees, STRIDE). Overview of application protection; buffer overflows and prevention techniques; memory management and security; secure processing with protected code partitioning and encryption. Preparing an application protection plan.	1, 5
2	25.09.2026	Computer viruses and malware: virus types (program, macro, boot sector), worms and trojans; countermeasures against viruses. Attack trees and security models (Bell–LaPadula, Biba, Clark–Wilson). Classifying software vulnerabilities: CWE and the CWE Top 25, OWASP Top 10, CVE and CVSS.	1
3	02.10.2026	Data security: in transit, at rest and in use. Encryption basics (symmetric/asymmetric, hash functions, authenticated encryption); using SSL/TLS and the TLS 1.3 handshake; certificate pinning; introduction to white-box cryptography applications; dynamic key management and session keys; data masking techniques; secure erasure of sensitive data in memory.	2, 4
4	09.10.2026	Code hardening (C/C++): secure memory management and secure coding rules (SEI CERT C/C++); buffer overflow, use-after-free and integer overflow examples; sanitizers (AddressSanitizer, UndefinedBehaviorSanitizer) and an introduction to fuzzing; compiler and operating system protections (stack canaries, ASLR, DEP/NX, CFI, SafeStack); control flow flattening; function name obfuscation; memory allocation obfuscation; dynamic encryption techniques.	3
5	16.10.2026	Code hardening (Java/interpreted languages): input validation and defence against injection attacks (SQL, command, path traversal; parameterised queries; SEI CERT Oracle Java); code obfuscation with ProGuard and R8; dynamic method obfuscation; static string obfuscation and protection; advanced ProGuard rules; dependency security and the software bill of materials (SBOM).	3
6	23.10.2026	Runtime application self-protection (RASP) (C/C++): root and emulator detection; APK signature verification; runtime code block integrity (checksum) checks; debugger detection and attach prevention; hook attack detection; dynamic memory protection and memory monitoring detection; protection against dynamic analysis tools; responses when tampering is detected.	3
7	30.10.2026	**Midterm project demonstrations** and midterm project report submission.	1, 2, 3, 5, 7
8	31.10–08.11.2026	**Midterm exam week — Quiz-1** (weeks 1–6).	1, 2, 3
9	13.11.2026	Advanced code obfuscation and diversification: dynamic control flow obfuscation; opaque predicates, bogus control flow and dead code	3

Week	Date	Topics	LO
		insertion; program obfuscation techniques (data encoding, virtualisation-based obfuscation); dynamic function calls and runtime optimisations; measuring obfuscation (potency, resilience, cost).	
10	20.11.2026	Certificates and cryptographic methods (AES, RSA, PKI): AES and RSA encryption, modes of operation and padding; data integrity with HMAC; creating and verifying digital signatures; PKI components (CA, RA, certificate chain); creating X.509 certificates (OpenSSL); certificate revocation (CRL, OCSP); the key distribution problem.	2, 4
11	27.11.2026	White-box cryptography: white-box and black-box attacker models; white-box AES and DES implementations (table-based); key protection with white-box cryptography; known attacks (differential computation and fault analysis) and countermeasures; software-based security solutions (e.g. a software security module with SoftHSM).	2, 3
12	04.12.2026	Security certifications and penetration test planning: ETSI and EMV security standards; security testing under PCI DSS and ISO/IEC 27001; security review and vulnerability assessment (code review, static and dynamic analysis, fuzzing); penetration test plan: scope, rules of engagement, methodology (OWASP WSTG and MASTG, PTES) and reporting.	5, 6, 7
13	11.12.2026	Security requirements: ETSI, GSMA and EMV security requirements; Common Criteria (ISO/IEC 15408) and EAL levels; FIPS 140-3 requirements; turning requirements into the software plan and asset management.	5, 7
14	18.12.2026	Tigress and diversification: C source transformations with Tigress (control flow flattening, virtualisation, literal and arithmetic encoding, opaque predicates, function split and merge); combining obfuscation methods; diversification to produce a different binary for each copy; defending against attacks and evaluating how well obfuscated code resists analysis.	3
15	25.12.2026	**Final project demonstrations** and final project report submission.	1–7
16	04–17.01.2027	**Final exam period — Quiz-2** (weeks 9–14).	2–7

Enrichment topics covered in the course notes as optional reading: memory-safe languages (Rust) and their use alongside C/C++, the secure software development life cycle (NIST SSDF), an introduction to side-channel attacks, and examining obfuscated code with reverse engineering tools (Ghidra).

D. Textbooks, Software and Equipment

The course notes on the course website are the main resource and are self-contained. The notes are built on the recipes of the textbook below and on the code protection and security methods the instructor developed in software protection and security certification; these methods are taught within the related weeks.

Textbook:

- J. Viega, M. Messier. *Secure Programming Cookbook for C and C++*. O'Reilly, 2003. — The book's concepts and error patterns still hold; up-to-date replacements for its 2003 APIs and algorithms (OpenSSL 3, TLS 1.3, AES-GCM, Argon2id, etc.) are given in the course notes.

Other resources defined in ritim:

- Deitel & Deitel. *C How to Program*, 7th ed. Prentice Hall, 2013.
- T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein. *Introduction to Algorithms*, 3rd ed. MIT Press.
- J. R. Hanly, E. B. Koffman. *Problem Solving and Program Design in C*.

Further reading:

- R. C. Seacord. *Secure Coding in C and C++*, 2nd ed. Addison-Wesley, 2013.
- C. Collberg, J. Nagra. *Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection*. Addison-Wesley, 2009.
- M. Dowd, J. McDonald, J. Schuh. *The Art of Software Security Assessment*. Addison-Wesley, 2006.
- R. Anderson. *Security Engineering*, 3rd ed. Wiley, 2020.
- J.-P. Aumasson. *Serious Cryptography*, 2nd ed. No Starch Press, 2024.
- Open standards: SEI CERT C/C++ and Oracle Java secure coding standards, OWASP Top 10, ASVS, MASVS and MASTG, MITRE CWE, NIST FIPS 140-3.

Laptop required. You will use your own development environment in class and in the project: a C/C++ compiler (GCC, Clang or MSVC), CMake, GoogleTest, OpenSSL 3, SQLite, SoftHSM2, JDK 21 with ProGuard/R8, Tigress (Linux; on Windows through WSL2 or Docker), and Git with a GitHub account. For the Android topics, Android Studio (SDK and NDK) with an emulator is recommended. Installation steps are given in week 1 and in the course notes; project templates are provided.

E. Assessment

You carry out **one term project**: a C/C++ application on a chosen topic that meets security requirements and is designed as if it were going through a certification process. Projects are done individually or in teams of at most 4 students; a project plan is prepared on GitHub and approved before development starts. The project has two checkpoints, each evaluated with its rubric: a midterm checkpoint and a final checkpoint. You also take one quiz in the midterm exam week and one quiz in the final exam period. The project topics, requirements, deliverables and the **detailed midterm and final rubrics** (criteria, points, related learning outcomes and performance levels) are given in the course's project guide.

Assessment	Code	Weight	When
Project checkpoint 1 — midterm report and demonstration (rubric)	RAP1	60% of midterm	Week 7 (30.10.2026)
Quiz-1 (weeks 1–6)	QUIZ1	40% of midterm	Week 8, midterm exam week (31.10–08.11.2026)
Project checkpoint 2 — final report and demonstration (rubric)	RAP2	70% of final	Week 15 (25.12.2026)
Quiz-2 (weeks 9–14)	QUIZ2	30% of final	Week 16, final exam period (04–17.01.2027)

Rubric criteria (ritim): **midterm checkpoint** — security analysis (LO.1), data security (LO.2), C/C++ code hardening and RASP techniques (LO.3), project management (LO.5), midterm report (LO.7); **final checkpoint** — cryptography implementation (LO.2), secure communication (LO.4), asset management (LO.5), binary application protections (LO.3), security testing and unit tests (LO.6), security standards, final report and presentation (LO.7).

$$Grade_{Midterm} = 0.6\,RAP1 + 0.4\,QUIZ1 \qquad Grade_{Final} = 0.7\,RAP2 + 0.3\,QUIZ2$$

$$Passing\ Grade = 0.4\,Grade_{Midterm} + 0.6\,Grade_{Final}$$

Workload (ECTS 5 = 125 hours)

Activity	Count	Hours	Total
Class attendance	14	3	42
Individual study (weekly notes and exercises)	14	1	14
Quiz (midterm exam week and final exam period)	2	2	4
Individual study for quizzes	2	10	20
Project preparation (midterm and final checkpoints)	2	16	32
Report preparation	2	5	10
Project presentation (demonstration and questions)	2	1.5	3
Total			**125**

F. Instructional Strategies and Methods

Lectures are face-to-face in the classroom and combine explanation, question–answer and hands-on programming. Each content week comes with course notes, slides, worked examples and self-check questions; exercises follow the order vulnerable code → attack → fix and run in the isolated environments provided in class (containers or virtual machines). Attack techniques learned in class are tried only in these environments and on the student's own systems. Announcements, resources and submissions are handled in the course class. Attendance is taken.

G. Late Homework

Assignments and projects must be submitted by the announced deadlines. Late submissions will not be accepted. Unexpected situations must be reported to the instructor as soon as possible.

H. Course Platform and Communication

All announcements, resources and submissions are shared in the course class, which is opened anew every term; the class code is announced in week 1. Course notes, slides and downloadable documents are on the course website. Check the class and your university e-mail every day.

I. Academic Integrity, Plagiarism & Cheating

Academic integrity is one of the most important principles of RTEÜ. Anyone who breaches the principles of academic honesty is severely punished.

It is natural to interact with classmates and others to "study together". It may also be the case where a student asks for help from someone else, paid or unpaid, to better understand a difficult topic or a whole course. However, what is the borderline between "studying together" or "taking private lessons" and "academic dishonesty"? When is it plagiarism, when is it cheating?

Looking at another student's paper or any source other than what is allowed during the exam is cheating and will be punished. However, many students come to university with very little experience of what is acceptable and what counts as "copying", especially for assignments. The following guidelines highlight the philosophy of academic honesty for graded assignments. If a situation arises that is not described below, ask the instructor whether what you intend to do stays within academic honesty.

a. What is acceptable when preparing an assignment?

- Communicating with classmates about the assignment to understand it better.
- Putting ideas, quotes, paragraphs or small pieces of code (snippets) found online or elsewhere into your assignment, provided that they are not themselves the whole solution and you cite their origin.
- Asking for help with the language of your assignment.
- Sharing small pieces of your assignment in class to start a discussion.
- Turning to the web or elsewhere for instructions, references and solutions to technical difficulties, but not for direct answers to the assignment.

- Discussing solutions with others using diagrams or summarised statements, but not actual text or code.
- Working with (even paying) a tutor, provided the tutor does not do your assignment for you.

b. What is not acceptable?

- Asking a classmate to see their solution before submitting your own.
- Failing to cite the origin of any text or code that you found outside the course and used in your work.
- Giving or showing your solution to a classmate who is struggling to solve the problem.

J. Expectations

You are expected to attend classes on time and complete the weekly requirements (readings and project milestones). The main communication channel between the instructor and students is e-mail. Send your questions from your university e-mail address; **include the course code in the subject line and your name in the message**. The instructor will also contact you by e-mail when necessary, so check your e-mail every day.

K. Lecture Content and Syllabus Updates

If deemed necessary, the lecture content or course schedule may change. Any change within the scope of this document will be announced by the instructor.