Secure Programming Lecture 1: Introduction

David Aspinall, Informatics Edinburgh

15th January 2018

Orientation

▶ This course is Secure Programming.
▶ More accurately: it is about Software Security.
▶ Aimed at Informatics MSc and 4th/5th year students
▶ Primarily: those anticipating a career in software
  ▶ programming: architects, developers, testers, …
  ▶ security: pentesters, malware/reverse engineers
  ▶ researchers: verification, compilers, languages, …
▶ It is taught by David Aspinall.

Web home: http://www.inf.ed.ac.uk/teaching/courses/sp

Ubiquitous software is broken (2014)

Ubiquitous hardware is flawed (2018)

Attacks can cause physical damage (2014)

Nobody can keep online records safe (2015)
Known good practice ignored (2015)

TalkTalk discloses possible breach, admits some data not encrypted

IoT: easily raise a DDoS botnet army (2016)

Why does this happen?
Ostensibly, many security failures are due to software vulnerabilities. Are they inevitable?

Many surrounding questions. Can we:
▶ find vulnerabilities (before attacks)?
▶ detect exploits in-the-wild?
▶ repair vulnerabilities (routinely/automatically)?
▶ program better to avoid vulnerabilities?
▶ measure risk associated with software?
▶ design or verify to prevent them?
▶ develop new technology to help the above?

Questions beyond the technical, too. Can we:
▶ insure against cyber incidents?
▶ regulate for better security?

What is this course about?
Building software that's more secure
▶ finding flaws in existing software
▶ avoiding flaws in new software (design and code)
▶ techniques, tools and understanding to do this

The infrastructure around secure software:
▶ language, libraries, run-time; other programs
▶ data storage, distribution, protocols and APIs
▶ development and deployment methods

And first of all, setting policies for security
▶ what should be protected
▶ who/what is trusted
▶ risk assessment: cost of defences.

Target audience
▶ Aimed at MSc, 4th/5th year UGs
▶ Have passed Computer Security or similar
▶ Basic notions, crypto, protocols
▶ Programming practice
▶ must be confident in programming
▶ necessarily will use a range of languages
▶ ...including assembler, C, Java
▶ but don’t have to be “master hacker”
▶ grounded in software engineering
▶ Programming theory
▶ interest in PL concepts and design
▶ knowledge of compilers useful
▶ also software engineering, esp, testing
▶ theory courses helpful, semantics

Why should you take this course?
Want to work in the cyber security industry?
▶ security appraisal, system and code reviewing
▶ pen-testing, ethical hacking
▶ malware analysis, reverse engineering
▶ operations and response (SOCs)
▶ cyber defence, attack, espionage
▶ innovation: found a cyber start-up

Want to work in security research?
▶ academic (conceptual advances, fixing, breaking)
▶ commercial (breaking, fixing, defending)

(Hopefully): you think it’s fun and interesting!
Why should you not take this course?

- None of the previous points apply.
- You don’t have the right background (see next slide)

Or perhaps, you don’t want to risk a relatively new course. This course is still “bedding in”. Frank and constructive feedback is very welcome.

Expected background

Please see Guide to Background Needed on course homepage.

1. Security properties: C, I, A, non-repudiation, privacy
2. Attacks against each of these
3. Defences: Au x 2, access control, crypto, networks
4. Coding skills and problem solving
5. Practicals: command line Linux

Learning outcomes

Here is the list from the Course Catalogue Entry:

1. Know how to respond to security alerts (concerning software)
2. Identify possible security programming errors when conducting code reviews in languages such as Java, C or Python
3. Define a methodology for security testing and use appropriate tools in its implementation
4. Apply new security-enhanced programming models and tools which help ensure security goals, e.g., with access control, information flow tracking, protocol implementation, or atomicity enforcement.

Safety versus security

Safety is concerned with ensuring bad things don’t happen accidentally. For example, aeroplanes don’t fall out of the sky because maintenance checks are forgotten.

Security is concerned with ensuring that bad things don’t happen because of malicious actions by others. For example, terrorists cannot drive bombs into airport departure halls.

The distinction is sometimes blurred, and the two interact in intriguing ways. For example (Q.) {why?}

The challenge of software security

Software artefacts are among the most complex built.

- Design flaws are likely
- Bugs seem inevitable

Flaws and bugs lead to vulnerabilities which are exploited by attackers.

Often to learn secrets, obtain money. But many other reasons: a security risk assessment for a system should consider different attackers and their motives.

Cost estimates are difficult

THE COST OF CYBER CRIME.

A DETICA REPORT IN PARTNERSHIP WITH THE OFFICE OF CYBER SECURITY AND INFORMATION ASSURANCE IN THE CABINET OFFICE.
But it's agreed they're increasing...

Cyber warfare is real

Privacy is being eroded

Why isn't software security better?

- Asymmetry: attackers have the advantage
  - just need to find one viable attack route
  - defenders have to anticipate all
- Attackers focus on weakest links:
  - since 1990s, network defences vastly improved
  - rise of insider threats
- Current penetrate-and-patch approach is broken
  - understandable by managers ("show me the problem!")
  - but no substitute for secure design

Why (else) isn't software security better?

What’s the outlook?

- New frontiers:
  - PCs in decline, but connected devices increasing
  - Mobile new target point (convergence, mobility)
  - Internet of Things: repeating same mistakes!
  - Cloud: XaaS, storage
  - Cyber resilience: speedy, automatic recovery
  - Data sharing and its limits: privacy
- Emerging new solutions:
  - Build Security In, Secure By Design
  - Defensive technologies continuing to evolve
  - New cryptographic, verification techniques
  - Old ideas re-appear: MLS, containment, isolation
  - Updates: automatic, pushed patching
**Delivery and assessment**

We will have

- 16 lectures covering core course topics
- 4 lab sessions
- 1 coursework contributing 30% of final mark
- 1 written exam contributing 70% of final mark

Lecture **slides** will be made available in several formats. They have numerous embedded links to useful resources (the links are more noticeable in the online versions).

Lecture **recordings** will be available, subject to systems. These are *backup*

**Lab sessions**

Three 3hrs lab sessions, 10am-1pm Tuesdays.

- Week 2
- Week 4
- Week 6
- Week 9

Each session will examine software vulnerabilities: why they exist, how they can be *discovered*, *exploited*, and *repaired*.

Labs may start with a *short guided introduction*.

**Working together is encouraged.** We want to foster a supportive learning environment. Students who have prior knowledge or expertise are especially welcome.

**Formative feedback during Labs**

One reason to introduce labs in this course is to allow us to give face-to-face *formative feedback* on your learning.

We will do this by reviewing the results from one lab session at the next lab session. To do this effectively we will ask that you *submit your work* and/or *discuss it with us* during the lab sessions.

Lab sessions will be run by me together with the course demonstrators, Arthur Chan, Connie Crowe and Margus Lind.

**Coursework**

The coursework will be an assignment following a similar pattern to the lab exercises: *discover*, *exploit* then *repair*.

1. as usual: **your work should be your own**
2. **no publication**, please do not publish solutions even after the deadline

(at least two reasons for last point).

The coursework deadline is scheduled for Week 8.

**An ethical point (reminder)**

*Nothing in this course is intended as incitement to crack into running systems!*

- Breaking into systems to “demonstrate” security problems at best causes a headache to overworked sysadmins, at worst compromises systems for many users and could lead to *prosecution*
- If you spot a security hole in a running system, *don’t exploit it*, instead contact the relevant administrators or developers confidentially.
- To experiment with security holes, play with your own machine, or better, your own private network of machines.

**Communications**

- Fast moving, evolving course:
  - *honest, constructive feedback is very welcome*
- As with any course, I welcome
  - *questions after lectures*
  - *questions by email*

Shall we have a course-wide online facility? Open to class opinion:

1. University forum (private in UoE)
2. University VLE tool (*Learn*)
3. Piazza for questions, discussion (signup)
4. None, but FAQs sent to class list sp-students by email
Exam

Will follow the common format:
- Choose 2 questions to answer from 3
- Two hours allowed

Towards the end of the course I will provide:
- A list of topics and concepts that may be examined
- A hint about the format of the questions

There is some guidance on the web along with a sample question.

Dimensions: practice and theory

Practice
- Programming securely, identifying security issues
- Mistakes in language, APIs, crypto, comms...
- Ultimately: detailed, highly specific knowledge

Theory
- Understand reasons for failure, ways to mitigate
- Understand advanced techniques, automated tools
- In general: transferable concepts and methods.

This is not really a “vocational” course. I hope it will give you the foundation to allow you to rapidly develop detailed specific knowledge needed later. There are a number of certification schemes for building practical knowledge.

Overview of topics

General organisation:
1. Threats
2. Vulnerabilities
3. Defences
4. Processes
5. Emerging Methods

We’ll look at details under each of these headings (in various orders).

1. Threats
- What attackers want, can do
- Types of bad code: malware, spyware, PUPs
- How bad code gets in
- Classification of vulnerabilities and weaknesses, CVE/CWEs

2. Vulnerabilities
- Overflows
- Injections
- Information leaks
- Race conditions
- Side channels and covert channels

3. Defences
- Protection mechanisms: validation, diversification, monitoring
- Trade-offs in adding protection mechanisms
- Provision for recovery
4. Processes

- Secure design principles
- Testing and reviewing to find vulnerabilities
- Assessing/measuring security of code

5. Emerging methods

- Methods and tools to find problems
- Detecting buggy patterns automatically
- Building security in, methodology and technology

Review questions

- Safety versus Security
  - Explain the difference between these two, and why ensuring security may be harder.

- Security flaws and their impact on society.
  - Explain some recent secure programming flaws that made the news and explain what the underlying problems were.
  - Discuss the fundamental reasons that software security fails and the wider questions around cyber security.

References and reading

The slides contain links which you can click on to find referenced or connected material.

References and reading will also be given for each lecture in a separate web page for that lecture. For this lecture, see here.

There is no single recommended course textbook, although a few books will be mentioned. See the page above for pointers.