

CA647 Secure Programming

Darragh O'Brien
dobrien@computing.dcu.ie

Office L2.35
School of Computing
Dublin City University

September 29, 2009



Course Home Page

- <http://www.computing.dcu.ie/~dobrien/ca647>
- Announcements
- Lecture notes
- Videos and recordings
- Lab exercises
- Assignments
- Results
- Further reading
- Practicum ideas
- Useful links



Lectures

Three 1-hour lectures per week

- Tuesday, 1000-1100, Q1.22
- Wednesday, 1400-1500, QG.02
- Thursday, 1400-1500, Q2.18



Labs

One 2-hour lab per week for 10 weeks (weeks 2-11)

- Tuesday, 1100-1300, LG.25
- A postgraduate student **may** be available to help you
- Do not expect to complete each lab exercise within the allotted time: expect to work on lab exercises throughout the week
- For best results, work in pairs



Marking

- Continuous assessment 30%
 - One assignment
 - One lab exam
- Examination 70%
 - 3 hours
 - A mixture of short and long questions



How To Pass

- Your final mark is calculated by combining your continuous assessment and exam results
- To pass the module your final mark must be 40+
- Failed components must be repeated
- Repeat examinations take place in August
- Repeat lab exam takes place in August
- Repeat continuous assessment is due in August



History

The course is an amalgam of two modules:

- Two thirds CA500 Secure Coding (practical)
- One third CA548 Secure Software Development (theoretical)



Course Objectives

1. To instill in you a healthy paranoia, one that will serve you in questioning the security of the applications you design and the code you write
2. To equip you with the skills required to assess the security of a design or piece of code and to apply an appropriate solution where necessary
3. To provide you with an understanding of processes and models underpinning secure software development



Achieving Our Objectives

- We will study a range of common software vulnerabilities
- We will examine in some detail how hackers exploit such vulnerabilities: this serves two purposes. . .
 1. It boosts your forensic analysis skills
 2. It raises your healthy paranoia levels by highlighting the possible harm caused when vulnerable software is exploited
- We will look at some of the libraries and tools available to help developers avoid introducing exploitable flaws into their code
- We will also examine a selection of the processes and theoretical models proposed to guide secure software development



Constraints

- We cannot cover every vulnerability, exploit and solution for every general purpose programming language on every operating system in a 12-week course
- Thus it makes sense to start off by examining in some depth a programming language that permits the showcasing of as many errors as possible: we choose C
- Given the interaction between exploits and the underlying operating system, it makes sense to choose an open source operating system: we choose Linux
- Of course, the same principles apply across other languages and operating systems



Constraints

- We are presented with a similar problem when it comes to studying web security: how do we choose from the myriad of available web development technologies?
- It makes sense to choose one we can get to grips with reasonably quickly: we choose LAMP (Linux, Apache, MySQL, PHP/Perl) because it is flexible, lightweight and popular

A tentative course outline. . .



Section 1: Computer Systems Review

- To understand many vulnerabilities and exploits we need to know how an executable is built from a program and run on a machine
- The notion of process and process organisation in memory
- The stack, procedure call and return, local variable allocation
- Linking object files to build an executable



Section 2: Secure Coding in C

- Buffer overflows, off-by-one errors, format string attacks, heap overflows, return-into-libc attacks etc.
- Integer programming errors
- Unix file permissions, setuid programming, time-of-check/time-of-use errors
- Defences and tools



Section 3: Web Application Security

- The internet and the world wide web
- Web application architecture
- Cookies, session management, client-side tampering, injection attacks, cross-site scripting attacks, phishing etc.
- Tools for web application security testing



Section 4: Developing Secure Software

- Secure design principles
- Risk analysis and threat modelling
- Security testing
- Security code reviews



Section 5: Policies and Models

- Policies, models and mechanisms for secure software development
- Access control mechanisms
- Models for confidentiality, integrity and hybrid security policies



Section 6: Security Assurance

- The role of auditing in secure software development
- Software assurance methodologies and their relation to international software assurance standards



Sample Reading List

- *Computer Systems: A Programmer's Perspective* by Randal Bryant and David O'Hallaron
- *Secure Coding in C and C++* by Robert Seacord
- *CERT C Secure Coding Standard* by Robert Seacord
- *Secure Programming with Static Analysis* by Brian Chess and Jacob West
- *Innocent Code* by Sverre Huseby



Sample Reading List

- *Secure Programming for Linux and Unix HOWTO* by David Wheeler [online and free to download]
- *Introduction to Computer Security* by Matt Bishop
- *Computer Security, Principles and Practice* by William Stallings and Lawrie Brown
- See also "further reading" off the home page



Approach

There will be no traditional lectures. . .

- Screen and audio capture software will be used to record lecture material to be made available on the course homepage
- You watch the videos and do some reading before attending tutorial and labs where we will try to answer your questions
- You must bring along questions with you
- So we have:
 - Tuesday, 1000-1100, Q1.22, tutorial
 - Tuesday, 1100-1300, LG.25, labs



Warning

- There may be occasions when “real” lectures and/or tutorials are required and these will take place during the timetabled lecture slots
- Check the CA647 Google calendar linked off the course homepage if in doubt



Starting off

- Some computer systems revision: process organisation in memory and the stack
- We will be writing some simple C programs, running them on Linux, studying some basic assembly and making use of `gdb`, the GNU debugger



Common Reactions

- “I do not know any C!”
- “Assembly?! Please no!”
- “I have never used Linux!”
- The C and assembly you will write is basic (we will not be developing any serious programs)
- There are hundreds of relevant tutorials on the web (see “further reading” on the course home page)
- You have labs and a tutorial in which to ask questions



Process Organisation

- **Text:** instructions
- **Data:** data
- **DLLs:** shared libraries
- **Heap:** `malloc`, `free`
- **Stack:** return addresses, parameters, local variables
- **Command line parameters:**
`argc`, `argv`
- **Environment:**
`PATH`, `HOME` etc.

