Secure Programming Lecture 1: Introduction

David Aspinall, Informatics @ Edinburgh

13th January 2014

Outline

Orientation

This course is **Secure Programming**

It is taught by **David Aspinall**

In the **School of Informatics** at the **University of Edinburgh**.

What is this course about?

- Building software that's more secure
  - finding security flaws in existing software
  - avoiding flaws in new software
  - techniques, tools and understanding to do this

What is this course about?

- Building software that's more secure
  - finding security flaws in existing software
  - avoiding flaws in new software
  - techniques, tools and understanding to do this

A better title might be ***Software Security***

It's not only about the programming. . .

- Infrastructure *around* software
  - language, libraries, run-time; other programs
  - data storage, distributed software
- Security *policies*:
  - what should be protected
  - what is trusted
- Risk assessment

Why should you take this course?

- Want to work in the **cyber security industry**?
  - security appraisal, system and code reviewing
  - ethical hacking
  - malware analysis
  - cyber defence (or attack, espionage, . . .)
- Want to work in **security research**?
  - *academic* (conceptual advances, fixing, breaking)
  - *commercial* (mainly breaking and fixing)
- (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)
- You don’t want to risk taking a **new course**
  - running earlier than expected
  - materials are still in development
  - will have teething troubles
  - please bear with us
  - **honest, constructive feedback is very welcome**

Target audience

- Aimed at 4th year UGs
- Should have passed 3rd year *Computer Security*
- **Programming practice**
  - should be confident in programming
  - necessarily will use a range of languages
  - . . . including some C
  - but don't have be "master hacker"
- **Programming theory**
  - interest in PL concepts and design
  - knowledge of *compilers* useful
  - also software engineering, esp *testing*
  - theory courses helpful, *semantics*

Learning outcomes

Here is the list from the Course Catalogue Entry:

1. Know how to respond to security alerts (concerning software)

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

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

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.

Outline

Safety versus security

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

Security is concerned with 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 challenge of software security

Software artefacts are among the most complex built.

- **Design flaws** are likely

The challenge of software security

Software artefacts are among the most complex built.

- **Design flaws** are likely
- **Bugs** are near inevitable

The challenge of software security

Software artefacts are among the most complex built.

- **Design flaws** are likely
- **Bugs** are near inevitable

The challenge of software security

Software artefacts are among the most complex built.

- **Design flaws** are likely
- **Bugs** are near inevitable

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

Usually: to learn secrets, obtain money.

Cost estimates are difficult

But it’s agreed they’re increasing. . .

Cyber warfare is real

And privacy is disappearing

Why isn’t software security better?

What if Microsoft breaches its warranty? If Microsoft breaches its limited warranty, your only remedy is the repair or replacement of the software. We also have the option to refund to you the price you paid for the software (if any) instead of repairing or replacing it. Prior to refund, **you must uninstall the software and return it to Microsoft, with proof of purchase.**

What if Microsoft breaches any part of this agreement? If you have any basis for recovering damages from Microsoft, you can recover only direct damages up to the amount that you paid for the software (or up to $50 USD if you acquired the software for no charge). **You may not recover any other damages, including consequential, lost profits, special, indirect, or incidental damages.** The damage exclusions and limitations in this agreement apply even if repair, replacement or a refund for the software does not fully compensate you for any losses or if Microsoft knew or should have known about the possibility of the damages. Some states and countries do not allow the exclusion or limitation of incidental, consequential, or other damages, so those limitations or exclusions may not apply to you. **If your local law allows you to recover other damages from Microsoft even though this agreement does not, you cannot recover more than you paid for the software (or up to $50 USD if you acquired the software for no charge.)**

Why (else) 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 security defences vastly improved
- Current *penetrate-and-patch* approach is broken
  - understood by managers and developers (“show me the problem!”)
  - but no substitute for secure design

What’s the outlook?

New frontiers:

- PCs in decline, but connected devices increasing
- Mobile new target point (convergence, mobility)
- Cloud storage: storage providers, protocols
- Cyber resilience: need for automatic response, patching
- Secure programming practice must extend to new domains, modes

Outline

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:

- **Threats**
- **Vulnerabilities**
- **Defences**
- **Processes**
- **Emerging Methods**

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

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

Vulnerabilities

- Overflows – *example next*
- Injections
- Race conditions
- Information leaks

Overflow example

```
338  char        charName[100];
339  int         ignore;
340
341  if (sscanf((char *) line, "STARTCHAR %s", charName) != 1) {
342  bdfError("bad character name in BDF file\n");
343      goto BAILOUT; /* bottom of function, free and return error */
344  }
```

Overflow example

```
338  char        charName[100];
339  int         ignore;
340
341  if (sscanf((char *) line, "STARTCHAR %s", charName) != 1) {
342  bdfError("bad character name in BDF file\n");
343      goto BAILOUT; /* bottom of function, free and return error */
344  }
```

SYNOPSIS

```
#include <stdio.h>

int sscanf(const char *str, const char
*format, ...);
```

DESCRIPTION

`sscanf()` scans input from the character string pointed to by `str`, according to `format` string. This may contain conversions; results are stored in locations pointed to by the pointer arguments that follow `format`.

Overflow example

```
338  char        charName[100];
339  int         ignore;
340
341  if (sscanf((char *) line, "STARTCHAR %s", charName) != 1) {
342      bdfError("bad character name in BDF file\n");
343      goto BAILOUT; /* bottom of function, free and return error */
344  }
```

Jan. 7, 2014 - Stack buffer overflow in parsing of BDF font files in libXfont

CVE-2013-6462: An authenticated X client can cause an X server to read a font file that overflows a buffer on the stack in the X server, potentially leading to crash and/or privilege escalation in setuid servers. The fix is included in libXfont 1.4.7. See the advisory for more details.

What is a BDF file?

```
STARTFONT 2.1
COMMENT
COMMENT  Copyright (c) 1999, Thomas A. Fine
COMMENT
...
FONT -atari-small
SIZE 11 75 75
FONTBOUNDINGBOX 4 8 0 -1
STARTCHAR C000
ENCODING 0
SWIDTH 1 0
DWIDTH 4 0
BBX 4 8 0 -1
BITMAP
00
00
...
```

- BDF = **B**itmap **D**istribution **F**ormat
- A (mostly) obsolete font format by Adobe

Advisory: **Description**

Scanning of the libXfont sources with the *cppcheck* static analyzer included a report:

```
[lib/libXfont/src/bitmap/bdfread.c:341]: (warning)
  scanf without field width limits can crash...
```

Evaluation of this report by X.Org developers concluded that a BDF font file containing a longer than expected string could **overflow the buffer on the stack**. Testing in X servers built with Stack Protector resulted in an immediate crash when reading a user-provided specially crafted font.

As libXfont is used to read user-specified font files in all X servers distributed by X.Org, including the Xorg server which is often run with root privileges or as setuid-root in order to access hardware, this bug may lead to an **unprivileged user acquiring root privileges** in some systems.

Advisory: **Affected Versions**

This bug appears to have been introduced in the initial RCS version 1.1 **checked in on 1991/05/10, and is thus believed to be present in every X11 release starting with X11R5** up to the current libXfont 1.4.6. (Manual inspection shows it is present in the sources from the X11R5 tarballs, but not in those from the X11R4 tarballs.)

Advisory: **Fix**

```
diff --git a/src/bitmap/bdfread.c b/src/bitmap/bdfread.c
index e2770dc..e11c5d2 100644
--- a/src/bitmap/bdfread.c
+++ b/src/bitmap/bdfread.c
@@ -338,7 +338,7 @@ bdfReadCharacters(FontFilePtr file, FontPtr pFont,
    char        charName[100];
    int         ignore;

-   if (sscanf((char *) line, "STARTCHAR %s", charName) != 1) {
+   if (sscanf((char *) line, "STARTCHAR %99s", charName) != 1) {
        bdfError("bad character name in BDF file\n");
        goto BAILOUT; /* bottom of function, free and return error */
    }
```

Defences

- Protection mechanisms
- Avoidance by secure coding
- Trade-offs in adding protection mechanisms

Processes

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

Emerging methods

- Methods and tools to find problems
- Detecting buggy patterns automatically
- Building security in: *language based security*

Outline

Delivery and assessment

We will have

- **16** lectures covering core course topics

Delivery and assessment

We will have

- **16** lectures covering core course topics
- **4** lab sessions

Delivery and assessment

We will have

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

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

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

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).

Lab sessions

Scheduled on Fridays at 2pm-5.30pm:

- Week 3, 31st January
- Week 5, 14th February
- Week 7, 7th March
- Week 9, 21st March

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

Labs will start with a **guided introduction** (up to 30 mins).

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 plan to 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 TA, who is **Joseph Hallett**.

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.

Ethics

A repeat of the advice given in the Computer Security course:

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

- New 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. Nota Bene for asking questions, annotating documents (public)
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

A new course; there are *no past papers available*.

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
- one sample question

Outline

Older books

J. Viega and G. McGraw. *Building Secure Software: How to Avoid Security Problems the Right Way*. Addison-Wesley, 2001.

One of the first books on the topic of Secure Programming. Still useful to understand some of the principles, although details are not current.

M. Howard and D. LeBlanc. *Writing Secure Code*. Microsoft Press, second edition, 2003.

Another early book; this one focuses on Windows. Again highly influential and useful for reference, but not up-to-date for current use. More recent titles are available from the Microsoft Press.

Newer books (1)

B. Chess and J. West. *Secure Programming with Static Analysis*. Addison-Wesley, 2007.

This book introduces ideas behind static analysis tools for detecting security flaws. Written by the founders of Fortify, now a part of HP.

M. Dowd, J. McDonald and J. Schuh. *The Art of Software Security Assessment*. Addison-Wesley 2007.

A lengthy book with deailed guidance on code reviewing for secure programming.

Newer books (2)

David Basin, Patrick Schaller, Michael Schlapfer. *Applied Information Security: A Hands-on Approach*. Springer, 2011.

> *A short practical introduction using Linux VMs to demonstrate some attacks and defences.*

Fred Long et al. The CERT Oracle Secure Coding Standard for Java, Addison-Wesley, 2012.

> *A set of guidelines for Java. Some need to be enforced by design and code reviews; others might be enforced automatically by tools.*

CERT also provide a shorter book *Java Coding Guidelines: 75 Recommendations . . .* as well as books giving coding standards for C and C++.

Online resources

- OWASP, the web application security project is one of the best places to find out about software security.
- The CERT secure coding website provides online versions of the CERT coding standards, which are developed in a Wiki.
- SANS, a security training organisation, provides some useful resource including some material on secure programming.

Outline

End notes (links to references)

Cost of cyber crime

- The 2007 US Government report estimated a cost of $105bn pa to US
- The 2011 UK Government report, which suggested a cost of £27bn pa to UK.
- It was criticised for being overblown The 2013 evidence-based review estimated “several billions”.

Ransomware

- The CryptoLocker ransomware and copies of it infected PCs at the end of 2013. It uses vulnerabilities in Java and PDF plugins.

Stuxnet

- Stuxnet (2010) was the first example of cyber warfare made public. Designed to thwart Iran's nuclear enrichment programme, in 2013, it was reported that Stuxnet also infected Russian facilities

Responsibility for insecure software

- See Microsoft's licensing terms for their EULAs. Most manufacturers (and open source vendors) have similar terms, of course.
- *Bad Code: Should Software Makers Pay?* in New Republic magazine, October 2013.
- David Rice. *Geekonomics: The Real Cost of Insecure Software*, Addison-Wesley, 2007. Read an interview with the author on SANS.
- *The Rugged Software Manifesto*, an attempt to start a grassroots movement among developers.