Secure Programming Lecture 1: Introduction

David Aspinall, Informatics @ Edinburgh

12th January 2016

Orientation

- This course is **Secure Programming**.
- Aimed at Informatics **4th year and MSc** students
- Primarily: those anticipating a career in software
    - **programming**: architects, developers, testers, dots
    - **security**: pentesters, malware/reverse engineers
    - **researchers**: verification, compilers, language design, dots
- It is taught by **David Aspinall**.
- I am a Reader in the **School of Informatics** at the **University of Edinburgh**.

Outline

Recent motivations

Course syllabus

Software security overview

Practicalities

Structure of course

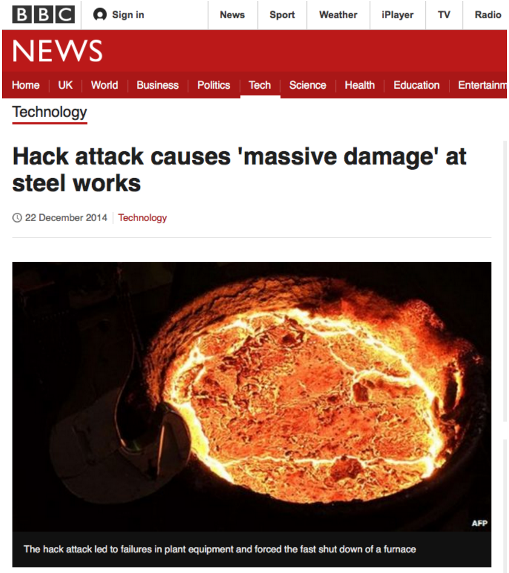
An example: overflow in X server

Summary

Heartbleed (2014)

Shellshock (2014)

Attacks can cause physical damage (2014)



Technology

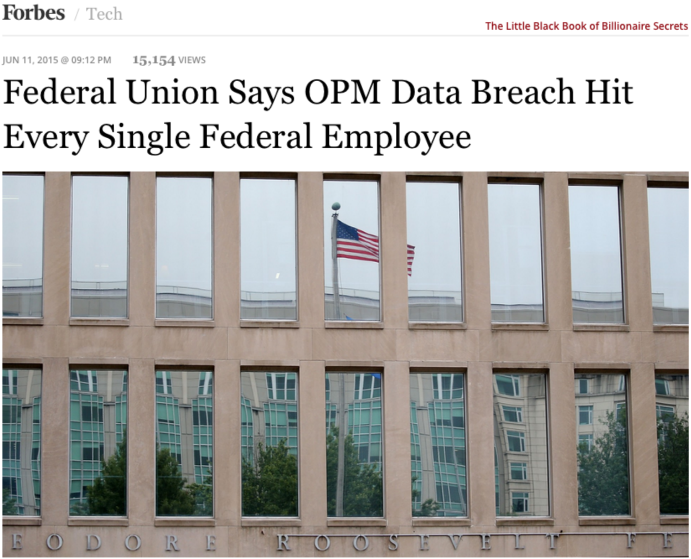
Hack attack causes 'massive damage' at steel works

22 December 2014 | Technology

AFP

The hack attack led to failures in plant equipment and forced the fast shut down of a furnace

Nobody can keep online records safe (2015)

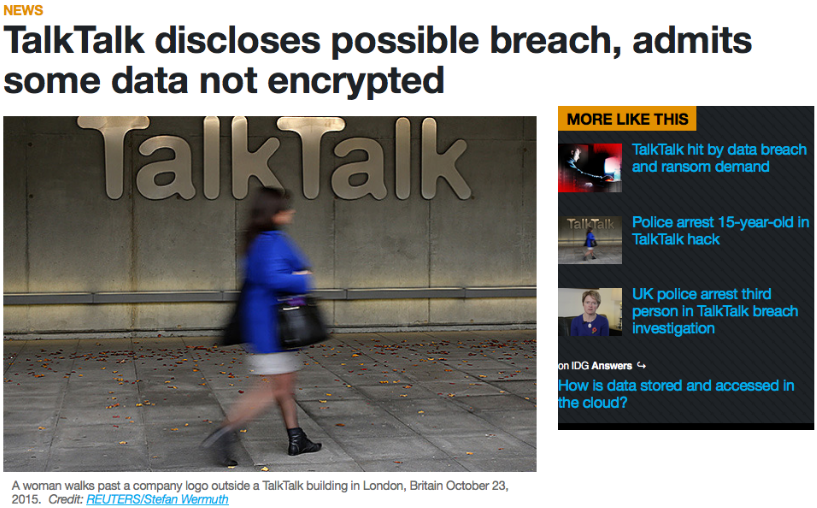


Forbes / Tech

JUN 11, 2015 @ 09:12 PM 15,154 VIEWS

Federal Union Says OPM Data Breach Hit Every Single Federal Employee

Known good practice ignored (2015)



NEWS

TalkTalk discloses possible breach, admits some data not encrypted

A woman walks past a company logo outside a TalkTalk building in London, Britain October 23, 2015. *Credit:* *REUTERS/Stefan Wermuth*

MORE LIKE THIS

- TalkTalk hit by data breach and ransom demand
- Police arrest 15-year-old in TalkTalk hack
- UK police arrest third person in TalkTalk breach investigation

on IDG **Answers** ↪

How is data stored and accessed in the cloud?

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?

Outline

- Recent motivations
- Course syllabus
- Software security overview
- Practicalities
- Structure of course
- An example: overflow in X server
- Summary

What is this course about?

Mainly: building software that's more secure

- finding security flaws in existing software
- avoiding flaws in new software
- techniques, tools and understanding to do this

also *infrastructure* around software:

- language, libraries, run-time; other programs
- data storage, distribution, protocols and APIs
- development methodologies

and in the first place, *policies* for security

- what should be protected
- who/what is trusted
- risk assessment: cost of defences.

Target audience

- Aimed at 4th year UGs, MSc by permission
- Should have passed 3rd year *Computer Security*
    - Basic crypto, protocols, secure prog ideas
- **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*

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)
- innovation: start-ups, spin-outs
- cyber defence, attack, espionage
- 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)
- You don’t want to risk a **relatively new course**
    - second time to run (2013/14 first time)
    - still “bedding in”
    - **honest, constructive feedback is very welcome**

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

Recent motivations

Course syllabus

Software security overview

Practicalities

Structure of course

An example: overflow in X server

Summary

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 distinction is sometimes blurred, and the two interact in intriguing ways. (**Q.** why?)

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** seem inevitable

The challenge of software security

Software artefacts are among the most complex built.

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

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*.

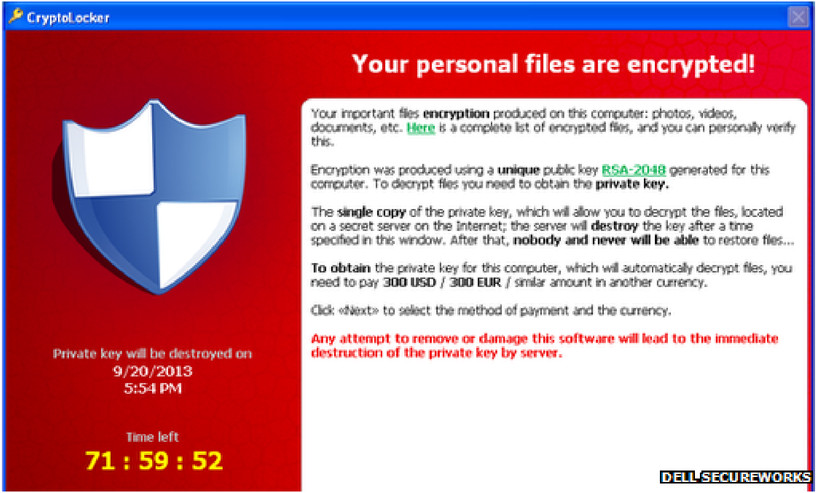
Often to learn secrets, obtain money. But many other reasons: a security risk assessment for a system should consider different attackers and their motives.

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. . .



Your personal files are encrypted!

Your important files **encryption** produced on this computer: photos, videos, documents, etc. Here is a complete list of encrypted files, and you can personally verify this.

Encryption was produced using a **unique** public key RSA-2048 generated for this computer. To decrypt files you need to obtain the **private key.**

The **single copy** of the private key, which will allow you to decrypt the files, located on a secret server on the Internet; the server will **destroy** the key after a time specified in this window. After that, **nobody and never will be able** to restore files...

To obtain the private key for this computer, which will automatically decrypt files, you need to pay **300 USD** / **300 EUR** / similar amount in another currency.

Click «Next» to select the method of payment and the currency.

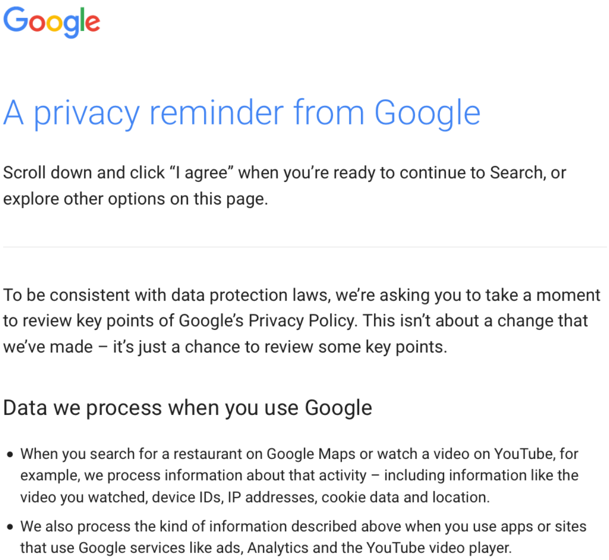
Any attempt to remove or damage this software will lead to the immediate destruction of the private key by server.

Private key will be destroyed on
9/20/2013
5:54 PM

Time left
71 : 59 : 52

Cyber warfare is real

Privacy is being eroded



Google

A privacy reminder from Google

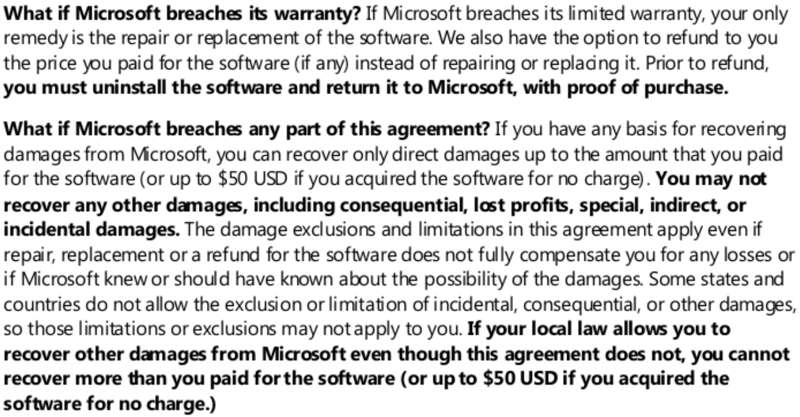
Scroll down and click “I agree” when you’re ready to continue to Search, or explore other options on this page.

To be consistent with data protection laws, we’re asking you to take a moment to review key points of Google’s Privacy Policy. This isn’t about a change that we’ve made – it’s just a chance to review some key points.

Data we process when you use Google

- When you search for a restaurant on Google Maps or watch a video on YouTube, for example, we process information about that activity – including information like the video you watched, device IDs, IP addresses, cookie data and location.
- We also process the kind of information described above when you use apps or sites that use Google services like ads, Analytics and the YouTube video player.

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 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

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 storage: storage providers, protocols
- Cyber resilience: speedy, automatic recovery
- Data sharing and its limits: **privacy**

New solutions:

- More and easier-to-use secure programming
- Defensive technologies continuing to evolve
- New cryptographic, verification techniques
- Old ideas re-appear: MLS, containment, isolation
- Updates: automatic, pushed patching

Outline

Recent motivations

Course syllabus

Software security overview

Practicalities

Structure of course

An example: overflow in X server

Summary

Delivery and assessment

We will have

- **16** lectures covering core course topics

Delivery and assessment

We will have

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

Delivery and assessment

We will have

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

Delivery and assessment

We will have

- **16** lectures covering core course topics
- **3** 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
- **3** 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
- **3** 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

Three 3hrs lab sessions.

Scheduled TBC.

- Week 4
- Week 6
- Week 8

Each session will examine some 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. medskip

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. bigskip

Lab sessions will be run by me together with the course TAs, who are **Joseph Hallett** and **Arthur Chan**.

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

- New, 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

Piazza is ready to use.

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.

Outline

Recent motivations

Course syllabus

Software security overview

Practicalities

Structure of course

An example: overflow in X server

Summary

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

3. Defences

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

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

Outline

Recent motivations

Course syllabus

Software security overview

Practicalities

Structure of course

An example: overflow in X server

Summary

1. Threats

General aim: *services running on Unix systems should be robust against local and remote attackers*.

Otherwise: attackers may exploit a service to cause a DoS attack, gain access to a system, etc.

For a specific system, a threat analysis would consider the kinds of attackers and their motives (local? remote? what is being protected?) and then all the services running on the system.

Question. What’s the easiest form of defence?

2. Vulnerability

A security review should first discover (and then monitor) relevant published **security advisories**.

For high value situations (and application code), dedicated review may be needed.

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.)

The vulnerability in the code

```
char        charName[100];
int         ignore;

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

The vulnerability in the code

```c
char          charName[100];
int           ignore;

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

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`.

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 filen");
        goto BAILOUT; /* bottom of function, free and return error */
    }
```

The text above is an example of a *context diff* which shows the difference between two file versions. The **patch** command can be used to update the older file given this text. You need to know how to make and apply patches for this course.

Defences

Options:

- Disable service
- Repair service: *downstream* updates
- Mitigate impact of attack

In running systems:

- Have there been past attacks?
- Can we check for future ones?

Outline

Recent motivations

Course syllabus

Software security overview

Practicalities

Structure of course

An example: overflow in X server

Summary

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.