Secure programming

Teaching Arrangements

- Course Coordinator:
  - Yuval Yarom
  - Ingkarni Wardli 4.23
  - yval@cs.adelaide.e
  - **Do not expect me t**
- Tutor
  - Sioli O'Connell
  - a1690418@student.adelaide.edu.au
- Online resources available on Canvas
  - https://myuni-canvas.adelaide.edu.au/courses/36233
    - Not yet ready

Admin

- No lecture on week 3

Secure Programming

- Advanced course in computer security
- Covers four main topics
  - Common vulnerabilities
  - Mitigation tech
  - Cryptographic primitives
  - Side-channel attacks

Assumed knowledge

- C/C++
  - The programming language is C, but if you know C++, learning C is relatively easy.

- Computer Syst
  - Machine language, caches, me ement unit, number representation, calling .

- Operating Systems
  - Processes, threads, scheduling, virtual memory, file systems.

Submission guidelines

- Markers are instructed to not mark your assignments if you fail to follow instructions.
  - Acceptable: .pdf, .tar, .tgz.
    - Contents must match the file name extension
  - Not acceptable: .d
- Do not submit bin lly generated files. (PDF are an exception)
- Every file you submit must display your name and your student numbers
  - In some cases there are specific requirements on how these are to be displayed.

Books

- Common vulnerabilities and some mitigation techniques:
  - M. Howard and D. LeBlanc “Writing Secure Code”
  - M. Howard, D. L adly Sins of Software Security”
- Cryptographic primitives
  - Bruce Schneier “Applied Cryptography”
- Side channel attacks and other mitigation techniques
  - No books yet

What is this course about?

- Security is all about protecting assets
- Secure software protects the assets that the software uses
  - **Confidentiality**
  - **Integrity**
  - **Availability**
- The aim of this course:
  - Give you (some of) the tools for developing secure software

Where is software security required?

- Managing users passwords?

New LastPass vulnerabilities

Where is software security required?

- Managing users passwords?
- Validating Web site certificates?

Goto fail

```
if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
    goto fail;
if ((err = SSLHashSHA1.final(&ha          hOut)) != 0)
    goto fail;

err = sslRawVerify(ctx, …
```

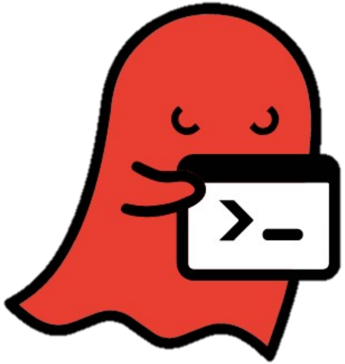
Where is software security required?

- Managing users passwords?
- Validating Web site certificates?
- Resolving host

GHOST

```
size_needed = (sizeof (*host_addr)
          + sizeof (*h_addr_ptrs) + strlen (name) + 1);

.
.
.
host_addr = (host
h_addr_ptrs = (host_addr_list_t *)
        ((char *) host_addr + sizeo       );
h_alias_ptr = (char **) ((char *) h
        sizeof (*h_addr_ptrs));
hostname = (char *) h_alias_ptr + sizeof (*h_alias_ptr);
.
.
.
   resbuf->h_name = strcpy (hostname, name);
```



Where is software security required?

- Managing users passwords?
- Validating Web site certificates?
- Resolving host
- Processing images?

ImageTragick

Source: imagetragick.com

Where is software security required?

- Managing users passwords?
- Validating Web site certificates?
- Resolving host
- Processing images?
- **Everywhere!**

Thinking like an attacker – Bike lock

- Lock has key and some kind of cable/chain to link bike to the bike rack
- Engineer: focuses on making lock unbreakable. Resistant to b t lock in the world
- Attacker / security engineer n the whole system. How can the **system** fail?
- **What does fail mean?**

What does **fail** mean?

- Obvious: you steal the bike
- Less obvious: you steal part of the bike
- Less obvious: y
- Less obvious: you render th erable
- Less obvious: you render the lock inoperable

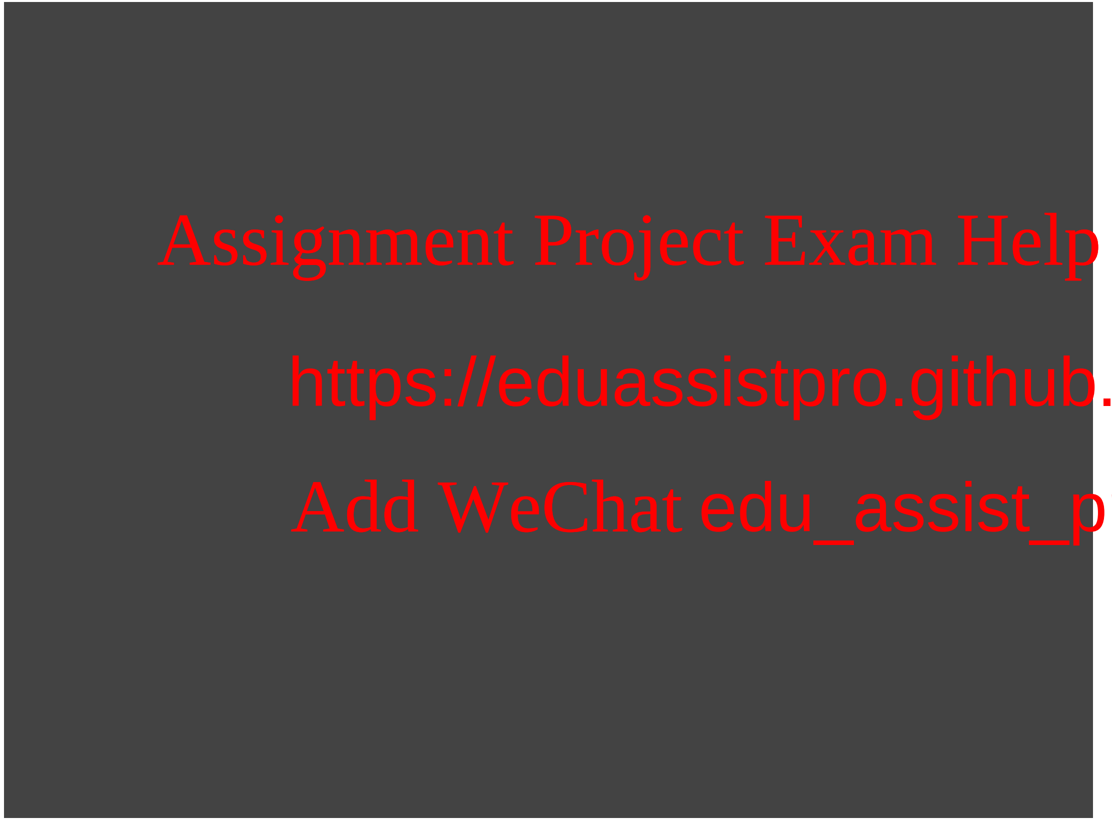
Stealing the bike

- The obvious attack – break the lock.
  - However – the lock is likely to be over-engineered.

- Lock and chain What about the bike rack?
  - Bolt cutters, angle grinders, oxy torch, shaped explosive charge, axe

Real-life examples

Another real-life example



Source: YouTube

Stealing a part of the bike

- Use a spanner
  - Or the quick release mechanism

- Leave front w
  - Walk into the local bike shop a someone stole your front wheel – slap down $50 and you have a new wheel and a whole new bike!

Real-life examples

Source: blog.priceonomics.com

Real-life example 5

Source: simplisafe.com

Real-life example

Source: www.npr.org

Real-life example 7

Source: scmp.com

Other Options

- Render the lock inoperable
  - Superglue
  - Oxy torch
  - Broken key

- Steal the lock
  - May be harder than stealing th
- Render the bike inoperable
  - Easy. Little benefit for the attacker.

Real-life example 8

Why security is hard

- Functional requirements vs. security requirements

Functional	Security
Editors c docu	*rs are able to ocuments*
What can be done	what

Assumptions

- We deal with security requirements by making assumptions
  - The only way to delete documents is by clicking 'Delete Document' on t
  - Editors do not s
  - Attackers do not have access to se
  - Attackers do not have access to se server
  - Attackers are not going to use tactical nuclear weapons to destroy documents
- All too often, the assumptions are implicit

Vulnerabilities and bugs

- Vulnerability: A flaw in a product that makes it infeasible—even when using the product properly—to maintain the required level of security

- Bug: An impleme in an unintended behaviour

- Not every vulnerability is a bug
  - But many are

- Not every bug is a vulnerability
  - It may be hard to identify "safe" bugs
  - Eliminating bugs also eliminates vulnerabilities

Abstractions and bugs

- Abstractions are the main tool we use to manage complexity
- An implementation of an abstraction provides an interface
- The consumer o [illegible] to provide higher-level abstractions
- Bugs are, usually, the result of failed abstractions