School of Computing
FACULTY OF ENGINEERING & PHYSICAL SCIENCES

UNIVERSITY OF LEEDS

# COMP3911 Secure Computing

## 12: Command Injection & Input Validation

Nick Efford

https://comp3911.info

# Objectives

UNIVERSITY OF LEEDS

- To begin looking at software vulnerabilities by considering ways of injecting malicious code into applications
- To frame these issues as **input validation problems**
- To conduct a specific case study of input validation, involving URLs

# 24 Deadly Sins…

UNIVERSITY OF LEEDS

SIN 10

COMMAND INJECTION

Howard, LeBlanc & Viega, *24 Deadly Sins of Software Security*

# Command Injection

UNIVERSITY OF LEEDS

- Untrusted data is combined with trusted data, typically taking the form of text strings
- Resulting string is passed to a compiler or interpreter running on the server
- Developer expects the untrusted input to be data, failing to recognise that it can be crafted in such a way that an unexpected command can be executed on server

# Simple Example

UNIVERSITY OF LEEDS

```
import os
import sys

os.system("lpq -P " + sys.argv[1])
```

Developer expects input to be name of a printer queue

How could you make this do something else, besides listing jobs on the queue?

# Warning Signs

UNIVERSITY OF LEEDS

- Combination of input data with commands, **using simple string concatenation**
- Code that runs a shell or subprocess
  - C/C++: `system()`, `popen()`, `execlp()`, etc
  - Python: `os.system()`, `os.popen()`, use of the `subprocess` module
  - Java: `Runtime.exec()`
- Code running with high privileges

# 24 Deadly Sins…

UNIVERSITY OF LEEDS

SIN 1

SQL INJECTION

Howard, LeBlanc & Viega, *24 Deadly Sins of Software Security*

# SQL Injection

UNIVERSITY OF LEEDS

- Same principle as generic command injection; untrusted input (e.g., coming from a web form) is combined with other text to create an SQL query
- No validation of input, so attacker is free to change the meaning of the query!
- Again, one of the key warning signs is **construction of the command via simple string concatenation**…

# Exercise

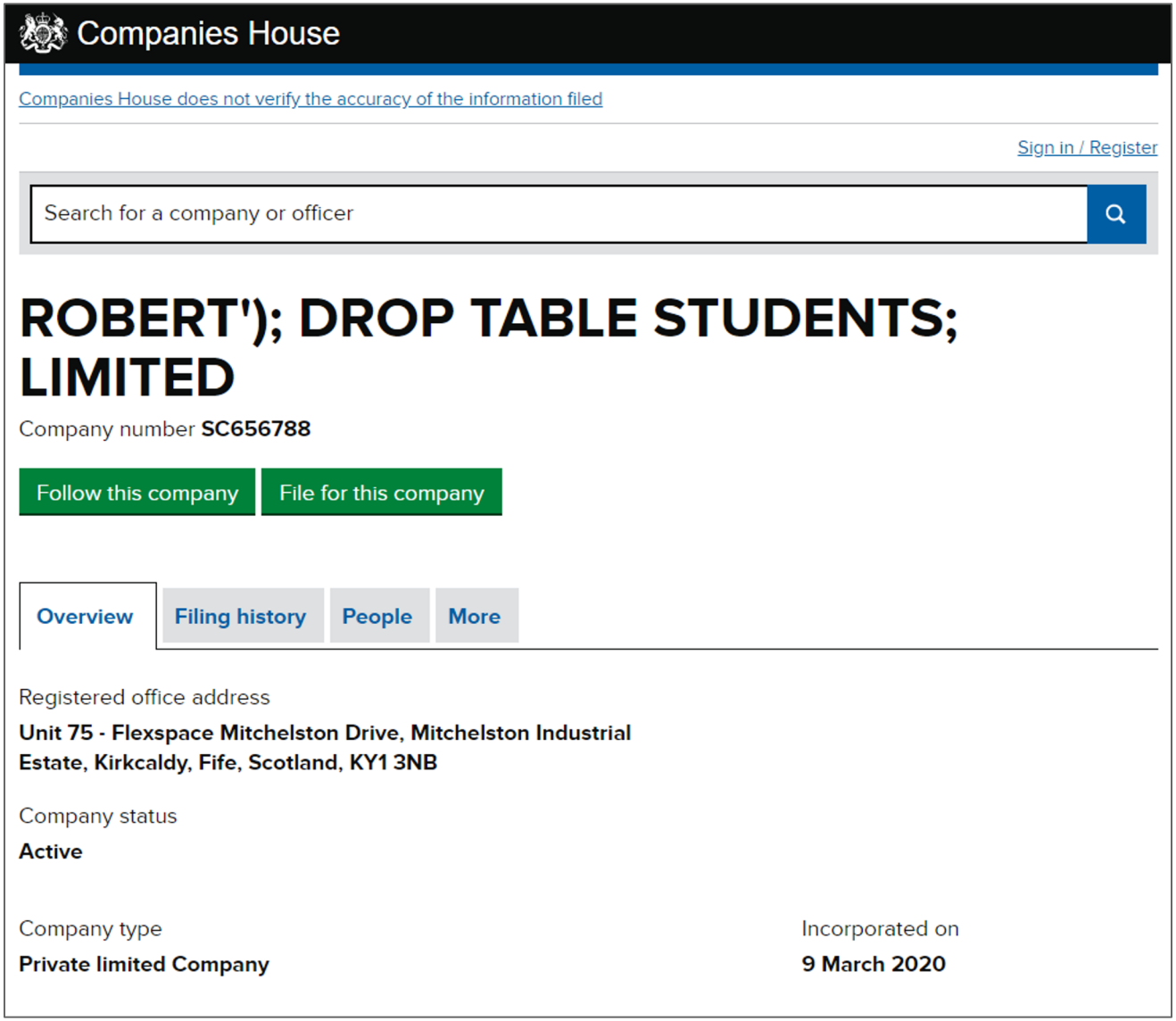
UNIVERSITY OF LEEDS

```
String q = "SELECT * from user WHERE name='" + name + "'";
Statement stmt = db.createStatement();
ResultSet results = stmt.executeQuery(q);
...
```

```
name = "joe"
```

```
name = "joe' OR 1=1 -- "
```

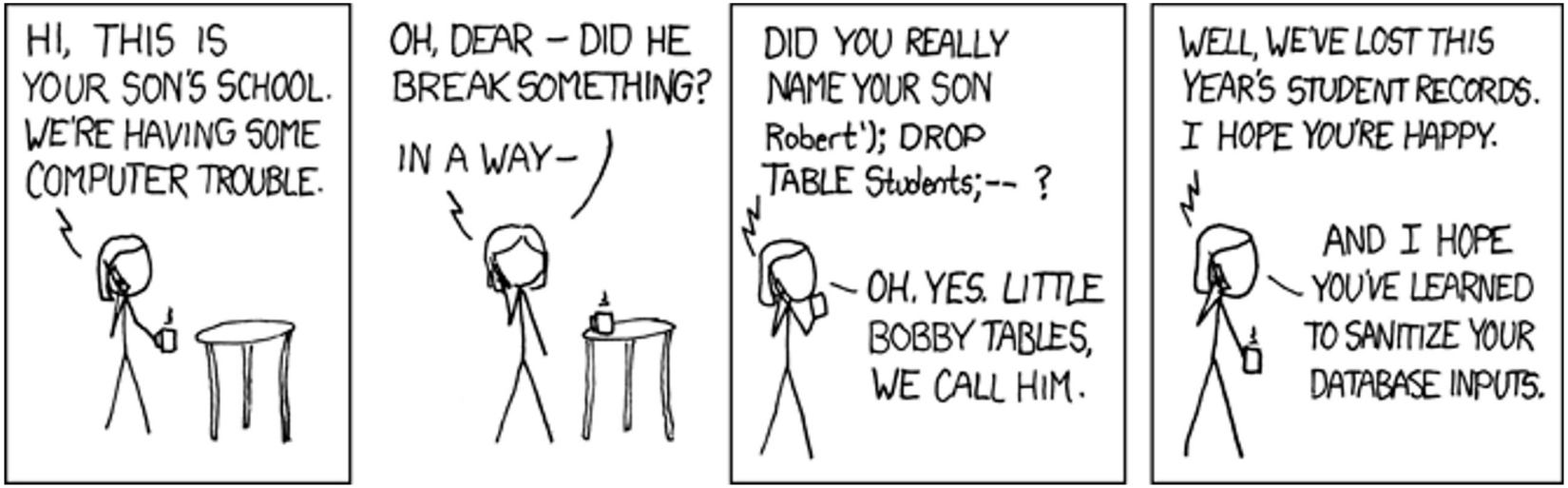

HealthCare.gov
Learn
Get Insurance
Log in
Español
Individuals & Families
Small Businesses
All Topics
;
SEARCH
;select * from users
;show tables;
;show tables; --
;premium payments
;select * from *;
; grant
; rehabilitative and habilitative
; show tables
Improving
HealthCare.gov
The Health Insurance Marketplace online application isn't a
5 a.m. EST daily while we make improvements. Additional d
work to make things better. The rest of the site and the Ma
during these hours.
Find health coverage that works for you
Get quality coverage at a price you can afford.
Open enrollment in the Health Insurance Marketplace continues until March 31, 2014.
4 Ways to Get Marketplace Coverage




https://find-and-update.company-information.service.gov.uk/company/SC656788

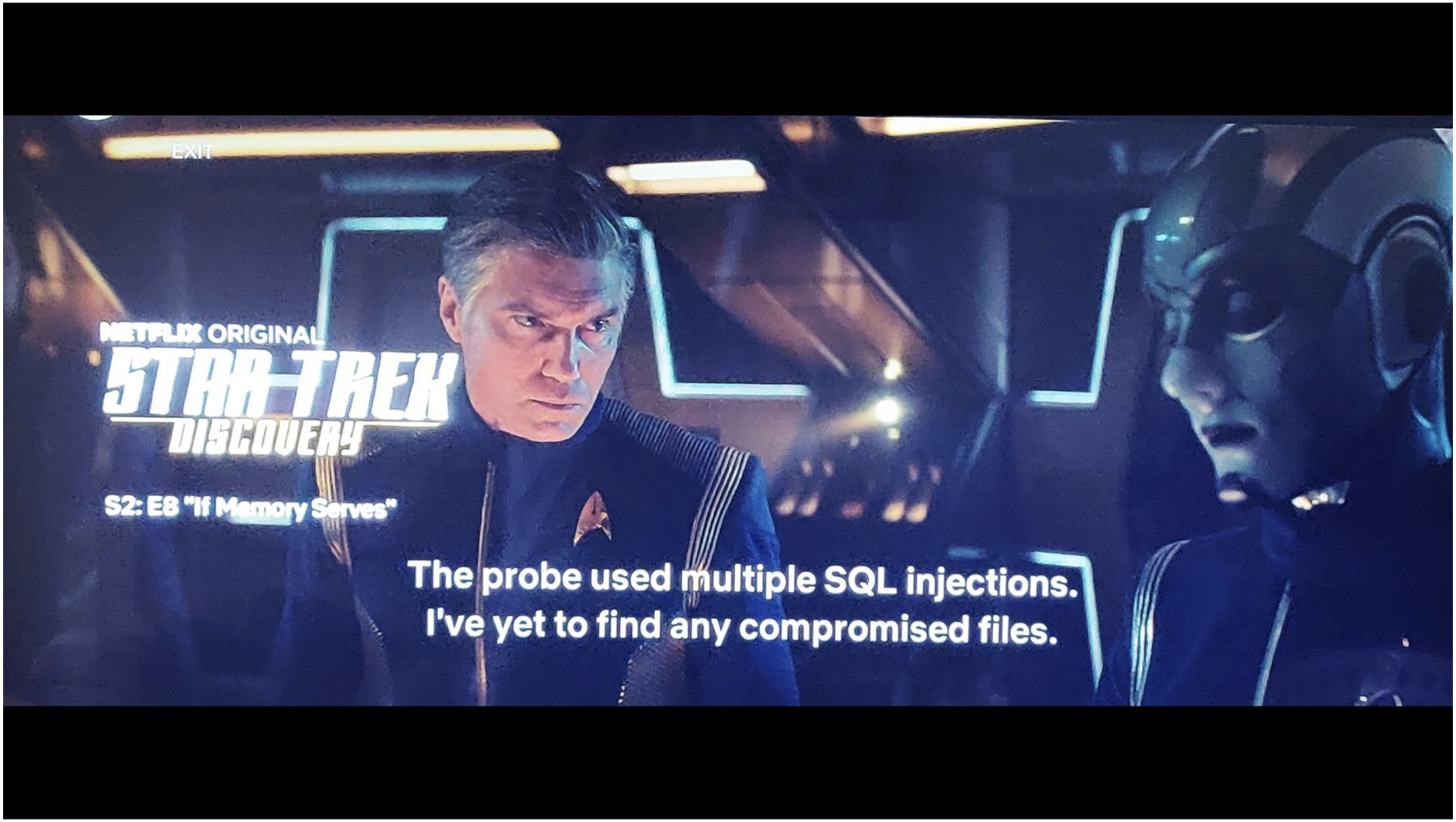
# ‘Little Bobby Tables’

UNIVERSITY OF LEEDS




“Exploits of a Mom”

https://xkcd.com/327/


EXIT
NETFLIX ORIGINAL
STAR TREK
DISCOVERY
S2: E8 "If Memory Serves"
The probe used multiple SQL injections.
I've yet to find any compromised files.

# Fixing It

UNIVERSITY OF LEEDS

- Manual filtering of input is a poor solution
- Real (and easy) defence is to use **prepared statements**

```
String q = "SELECT * from user WHERE name='" + name + "'";
Statement stmt = db.createStatement();
ResultSet results = stmt.executeQuery(q);
...
```

```
String q = "SELECT * from user WHERE name=?";
PreparedStatement stmt = db.prepareStatement(q);
stmt.setString(1, name);
ResultSet results = stmt.executeQuery();
...
```

# A Generic Problem

UNIVERSITY OF LEEDS

- Why do these all attacks succeed?
  - Data come from an untrusted source
  - **Too much trust** is placed in format & content – e.g., assuming it won't exceed buffer capacity, doesn't contain malicious code, etc
  - Data are written into memory, or used to construct a command of some sort
- Only the second of these is under our control
- Hence the problem we need to solve is **input validation**

# Handling Malicious Input

UNIVERSITY OF LEEDS

**Fundamental rule:**

**All input is evil until proven otherwise**

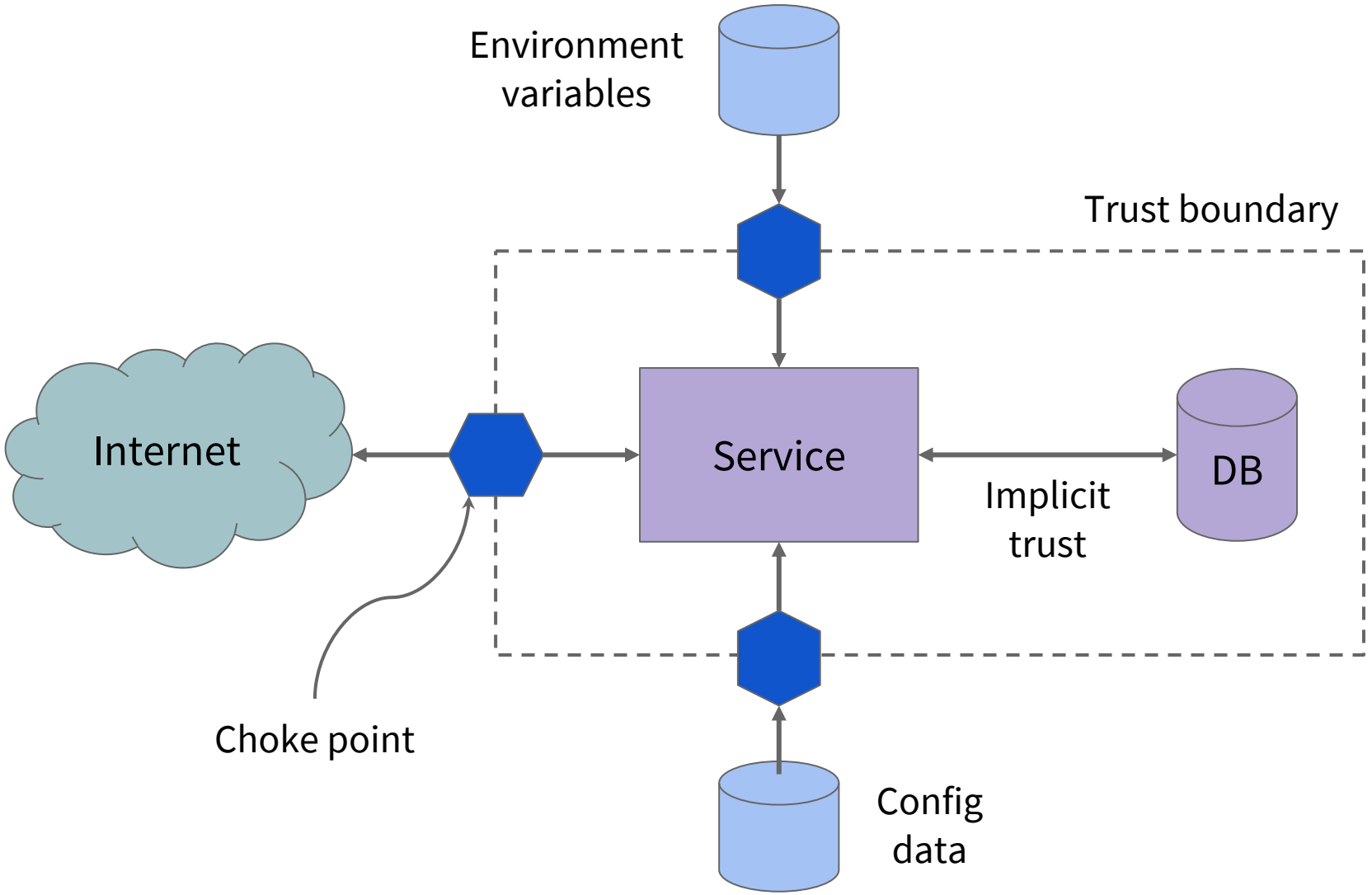
All input must be validated, but where do we do it?

**Option 1**: validate on input

**Option 2**: validate before first use

# Trust Boundary & Choke Points

UNIVERSITY OF LEEDS



# Case Study: URL Validation

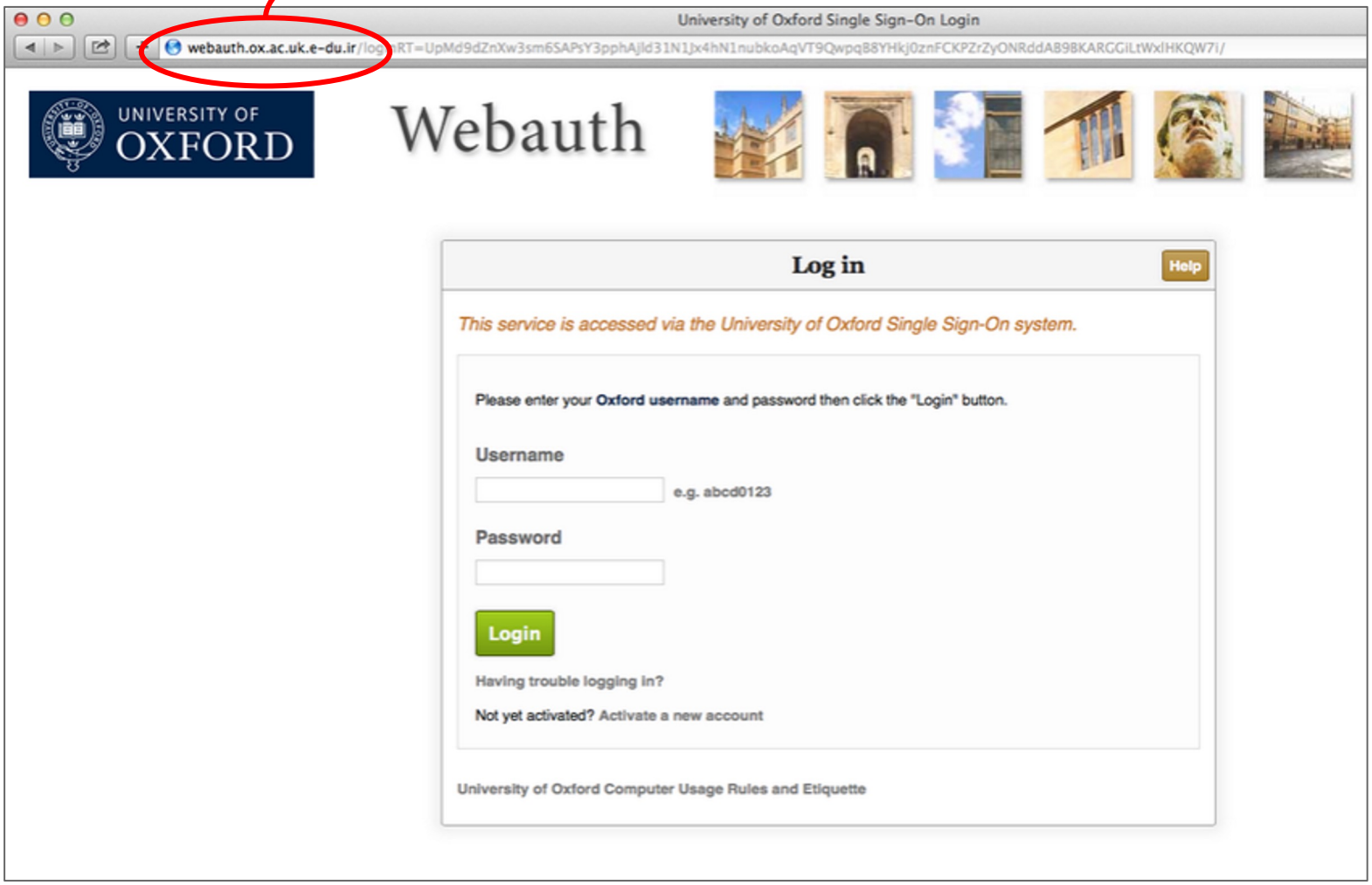
UNIVERSITY OF LEEDS

- Who is responsible for validation?
  - User
  - Browser
  - Server
- What can go wrong if no validation is done?

# Example: Phishing

UNIVERSITY OF LEEDS

web.auth.ox.ac.uk.e-du.ir



# Homograph Attacks

UNIVERSITY OF LEEDS

Which of these show identical domain names?

1: micr0s0ft.com / microsoft.com

2: MICR0S0FT.COM / MICROSOFT.COM

3: microsoft.com / microsoft.com

4: microsoft.com / microsoft.com

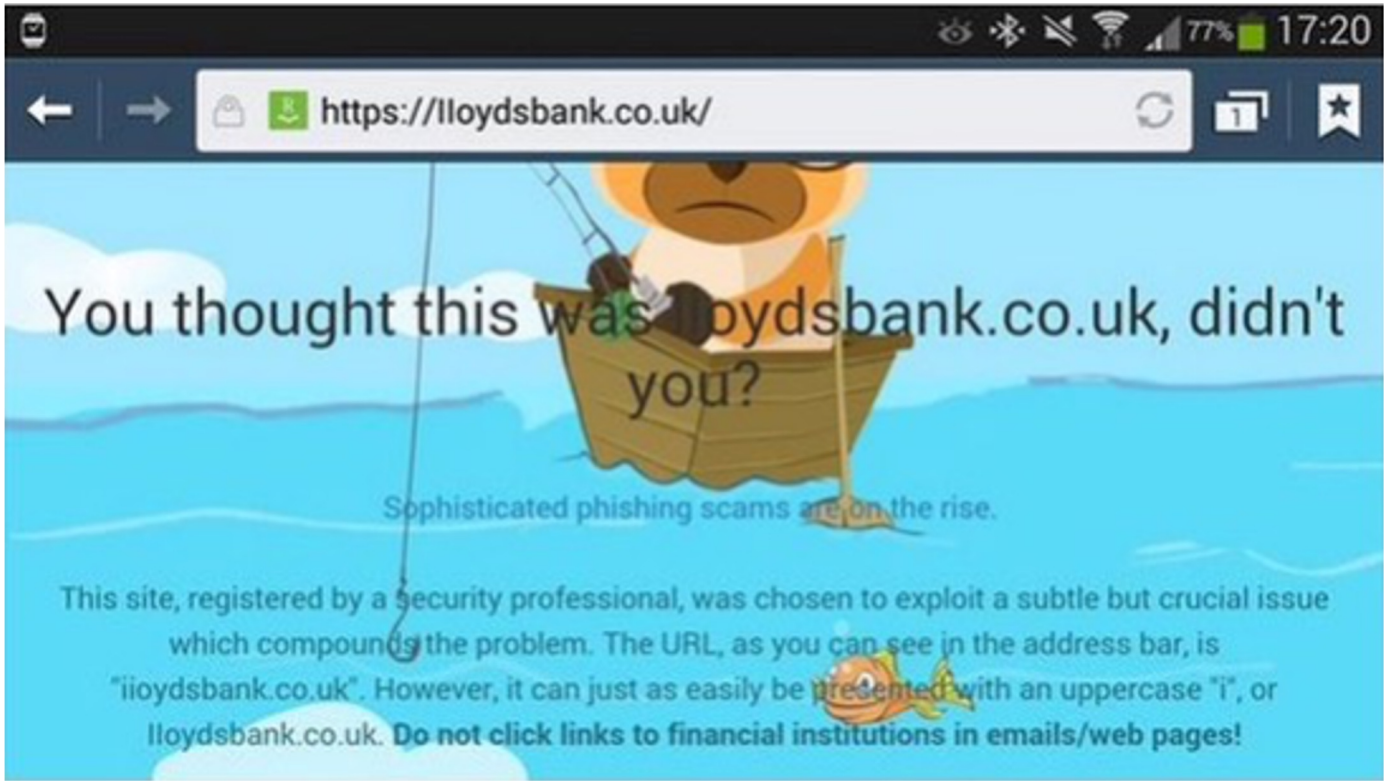
Further details in *Comm. ACM* 45(2), page 128

https://cacm.acm.org/magazines/2002/2/7143-the-homograph-attack

# 2015 Example

UNIVERSITY OF LEEDS

IIoydsbank.co.uk domain registered by a security researcher, who even managed to obtain a TLS certificate for it!



# Use of ‘Punycode’

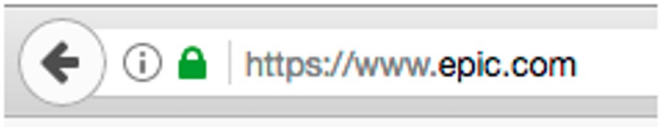
UNIVERSITY OF LEEDS

Here is what the real epic.com looks like in Chrome:

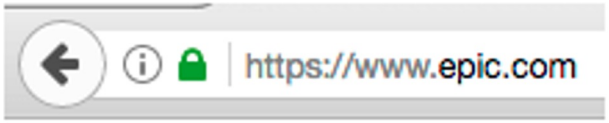

Here is our fake epic.com in Chrome:


Domain registered as xn--e1awd7f.com

And the real epic.com in Firefox:




And here is our fake epic.com in Firefox:



# URL Obfuscation in Browsers

UNIVERSITY OF LEEDS

URLs can have a *user@location* format:

```
http://nick@www.foo.com
```

Malicious URL could have a null byte between the *user* part and the @ symbol:

```
http://www.trusted.com%00@www.evil.com
```

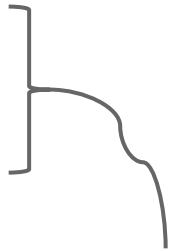
Older browsers would display only the *user* part in status bar when cursor hovered over the link!

New variations of this appeared in 2017 and 2018, affecting email clients and iOS 11

# Directory Traversal

UNIVERSITY OF LEEDS

- Can potentially affect both conventional static websites and dynamic web applications
- Allows an attacker to step outside the web server's root directory and access other parts of the filesystem
- URL contains repetitions of
  - `../` or `..\`
  - `%2e%2e%2f` or `%2e%2e%5c`
  - `..%c1%1c` or `..%c0%af` or `..%c1%9c`



Why did attackers try these?

# Example 1

UNIVERSITY OF LEEDS

vulnerable.php

```
<?php
$template = 'blue.php';
if (is_set($_COOKIE['TEMPLATE']))
    $template = $_COOKIE['TEMPLATE'];
include("/home/users/php/templates/" . $template);
?>
```

**Request**

```
GET /vulnerable.php HTTP/1.0
Cookie: TEMPLATE=../../../../etc/passwd
```

**Response**

```
HTTP/1.0 200 OK
Content-Type: text/html
Server: Apache

root:fi3sED95ibqR6:0:1:System Operator:/:/bin/ksh
daemon:*:1:1::/tmp:
php:f8fk3j1OIf31.:182:100:dev:/home/users/php:/bin/csh
```

Security

192

# Dishwasher has directory traversal bug

## Thanks a Miele-on for making everything dangerous, Internet of Things firmware slackers

By Richard Chirgwin 26 Mar 2017 at 23:08

SHARE ▼

https://www.theregister.co.uk/2017/03/26/miele_joins_internetofst_hall_of_shame/

# Canonicalisation

UNIVERSITY OF LEEDS

- Canonical = “In its simplest or standard form”
- Canonicalisation = process of converting the various equivalent forms of a name into a single standard name
- Applications often make wrong decisions based on non-canonical representation of a name
- Solution: convert all user input to canonical form before making any security decisions

# Summary

UNIVERSITY OF LEEDS

We have

- Explored the general idea of command injection and seen how it is frequently used in SQL injection attacks
- Framed these issues as problems of **input validation**
- Seen that phishing can bypass casual user validation of URLs (e.g., via a homograph attack)
- Noted that browsers can mishandle malicious URLs, as can servers (e.g., directory traversal)

# Follow-Up / Further Reading

UNIVERSITY OF LEEDS

- Relevant exercises: 17, 18 and 22
- Sins 1 & 10 of *24 Deadly Sins of Software Security*
- sqlmap: penetration testing tool for databases
- The inception bar: a new phishing method
  (visit in Chrome on a mobile device!)
- Creative usernames and Spotify account hijacking
  (example of a homograph attack from 2013)
- Mailsploit: null injection in email headers, etc (2017)
- Obfuscation of URLs from QR codes in iOS 11