Protecting Web Applications from Universal PDF XSS:

A discussion of how weird the web application security world has become

Ivan Ristic
Chief Evangelist
Breach Security
ivanr@modsecurity.org

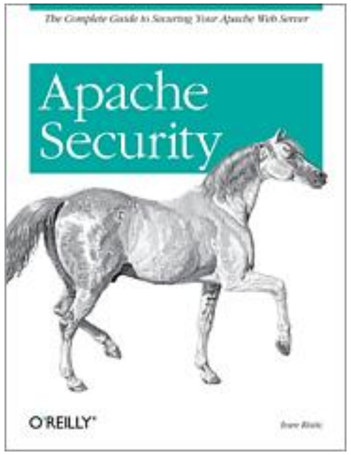
6th OWASP AppSec Conference
Milan - May 2007

The OWASP Foundation
http://www.owasp.org/

Table of Contents

About Ivan Ristic

- Software developer/technical architect/security analyst/whatever.
- Web application security and web application firewall specialist.
- Author of **Apache Security**.
- Author of **ModSecurity**.
- Employed by **Breach Security** to work on ModSecurity.



Introduction

DOM-based Cross-Site Scripting (1)

- It all started back in 2005 when **Amit Klein** published **DOM Based Cross Site Scripting or XSS of the Third Kind**.
- Amit observed that XSS does not necessarily need a vulnerable server-side program to manifest itself. Everything can take place in the browser itself.
- He also observed how the **#** character can be used to, very conveniently, avoid sending attack payload to the server.

DOM-based Cross-Site Scripting (2)

- DOM-based XSS typically uses JavaScript. Example (taken from Amit’s paper):

```
<HTML><TITLE>Welcome!</TITLE>
Hi <SCRIPT>
var pos = document.URL.indexOf("name=") + 5;
document.write(document.URL.substring(pos,document.URL.length));
</SCRIPT>
</HTML>
```

- Normally invoked with:

```
http://www.example.com/welcome.html?name=Joe
```

- Does not work equally well when invoked with:

```
http://www.example.com/welcome.html?name=
<script>alert(document.cookie)</script>
```

Enter Acrobat Reader
Universal PDF XSS (1)

- In December 2006 **Stefano Di Paola** and friends speak about the universal XSS flaw in the Acrobat Reader plug-in on Windows.
- The world found out when the advisory went out on January 3rd, 2007. (*The flaw was already fixed in Reader v8 in early December 2006.*)
- **The word spread like fire** among security bloggers (**pdp**) and on the mailing lists.
- **RSnake** discovered the attack can be used against PDF files hosted on the local filesystem.

Enter Acrobat Reader
Universal PDF XSS (2)

For many people this was the last straw. They acknowledged that the end of the World is near.

So What Was the Problem?

- It turns out the Reader plug-in loved JavaScript so much it would execute it when a link in the following format is encountered:

```
http://www.example.com/file.pdf#a=
javascript:alert('Alert')
```

- **Uh-oh**
  - Notice the **#** character!

Threat Assessment (1)

- Discoverability - **10**
- Reproducibility - **10**
- Exploitability - **7**
  - Attack code not trivial but not very difficult to write.
  - Victim must click a link (email) or visit a malicious web site. *Both attack vectors are examples of CSRF.*
- Affected users - **10**
  - PDF is a standard for printable documentation.
  - Most computers have Adobe Reader installed.
  - Most sites carry PDF files.

Threat Assessment (2)

- Damage potential - **8**
  - After a successful attack the code is executed in the context of the site that hosts the PDF file.
  - The attacker is in full control of the victim’s browser (think *session hijacking*, *request forgery*, etc.).
  - Individual users are fully compromised.
  - System compromise is possible through escalation.
  - When a locally-hosted PDF file is targeted attackers can gain access to the workstation (*requires further tricks to be used, e.g. the QTL hack, but doable*).
  - **Damage potential depends on site content.**

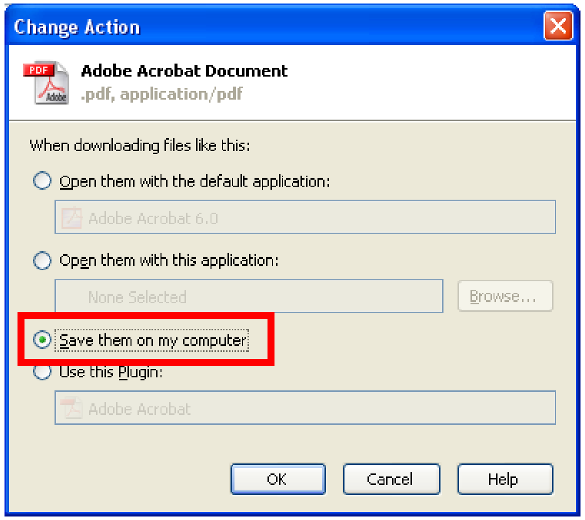
Threat Assessment (3)

- The potential for damage is there, all right, but where are the exploits?
  - Many have expected doom and gloom.
  - But no major scale attacks reported.
  - Why?
- Where do we stand today?
  - The excitement is gone.
  - Security-aware people have fixed the problems.
  - But how many vulnerable people and sites remain?
- This problem is as dangerous as it was a few months ago.

Fixing Universal PDF XSS

Fixing The Problem - Users

- In many ways this is a simple problem to solve. ***Just upgrade*** **the client-side software**:
  - Adobe Reader 8 not vulnerable.
  - Internet Explorer 7 not vulnerable.
  - Other PDF viewers (e.g. Foxit Reader) not vulnerable.
- Alternatively, you can configure the browser not to open PDF files at all.
- But we know **many users will not upgrade**.


Change Action

Adobe Acrobat Document
.pdf, application/pdf

When downloading files like this:

- ( ) Open them with the default application: Adobe Acrobat 6.0
- ( ) Open them with this application: None Selected
- [✓ Save them on my computer]
- ( ) Use this Plugin: Adobe Acrobat

Fixing The Problem – Sites (1)

- **Not possible to detect attack on the server**.
- Therefore our only option is to **"protect"** all PDF files no matter if they are being attacked or not.

- Proposed mitigation revolves around three ideas:
  - Moving PDF files to some other domain name.
  - Preventing browsers from recognising PDF files. (*Some are very stubborn in this regard.*)
  - Forcing browsers to download PDF files.
- This can be done via header modification in *web server configuration (all files)* or *application (dynamic files only)*.

Fixing The Problem – Sites (2)

- Key headers:

```
Content-Type: application/octet-stream
Content-Disposition: attachment; filename=x.pdf
```

- Apache fix:

```
AddType application/octet-stream .pdf
<FileMatch "\.pdf$">
    Header set Content-Disposition \
    "attachment; filename=document.pdf"
</FileMatch>
```

- Detailed instructions available from Adobe:
  http://www.adobe.com/support/security/advisories/apsa07-02.html

Analysis of the Solution So Far

- **Advantages:**
  - The web server configuration-based approach is very easy to implement.
    - But it may not possible to use this approach with all environments.
- **Weaknesses:**
  - Changing application code can be time consuming.
  - Forcing downloads of PDF files is not very user friendly (*many* users *will* get confused).
  - **Dynamically-generated PDF files are easy to forget (and thus miss).**

Sidebar: Approaches That Do Not Work

- Trying to detect attack from the server.
  - Not possible to see the attack from the server.
- Relying on the Referer request header.
  - It's not always there.
  - Can be forged.
- Changing Content-Type only.
  - IE will sniff the content to determine the C-T.
- URI Encryption & Requiring sessions:
  - Defied using session fixation.
  - Not usable on public sites anyway.

Using Redirection (1)

- Amit Klein proposed a defence mechanism, which was subsequently discussed and refined on the mailing lists:
  - http://www.webappsec.org/lists/websecurity/archive/2007-01/msg00058.html
- While searching for a better solution many people noticed that it is possible to *overwrite the attack payload* using **redirection** and a **harmless fragment identifier**.
- If we get:

  ```
  http://example.com/test.pdf#x=ATTACK
  ```

  We redirect to:

  ```
  http://example.com/test.pdf#neutralise
  ```

Using Redirection (2): Preventing Loops

- But how do we tell we've already redirected the user?
  - If we don't we'll just end up with an endless loop.
- We can use **one-time tokens as flags**.
- So this:

  `http://example.com/test.pdf#x=ATTACK`

  Is now redirected to:

  `http://example.com/test.pdf?TOKEN=XXXXXXX#neutralise`

Using Redirection (3):
Token Generation

- If we generate a completely random token then we'd have to start keeping state on the server (i.e. token repository, garbage collection of expired tokens).
  - It's a fine approach.
  - But it can have non-negligible impact on the performance and maintenance of non-trivial sites.
  - It can also affect cacheability.
- Alternatively, we can store state on the client.
  - Use cryptography to validate tokens.
  - Embed the expiry time.

Using Redirection (4):
Token Hijacking?

- Unfortunately, our solution is not foolproof yet.
- The attacker can simply generate a number of tokens to use against his victims.
  - We have to associate tokens with clients somehow.
- It would be nice to use the application session but not all sites have them.
  - Exploitation possible through session fixation.
  - Thus we have no choice but *use the IP address*.
- But what happens if the IP address changes (user behind a proxy)?
  - We fall back to forced download.

Using Redirection (5):
It's Not Foolproof!

- There are still holes in our solution!
- If the attacker shares the same IP address as the victim (proxy, NAT) he will be able to obtain tokens to use in attacks.
  - The timeout feature does not help much.
  - If the attacker can get the victim to browse a malicious web site he can:
    - Generate responses dynamically while…
    - …obtaining valid tokens behind the scenes.
- At best, we can prevent mass-exploitation.
  - Focused attacks remain an issue.

A Foolproof Protection Mechanism Would…

- A foolproof protection mechanism would:
  - Associate tokens with *client* SSL certificates. (Or to session IDs where sessions have already been associated with client SSL certificates.)
  - This would prevent session fixation.
- And it would only work on:
  - Sites that have sessions and
  - We would have to know where the session ID resides.
- Not usable as a general purpose protection method.

Implementation Details

- Most protection mechanisms rely on detecting the PDF extension in the request URI.
- Let’s have a look at some request types:
  - `GET /innocent.pdf`
  - `GET /download.php/innocent.pdf`
  - `GET /download.php?file=innocent.pdf`
  - `GET /download.php?fileid=619`
  - `POST /generateReport.php`
    (with a bunch of parameters in the request body)
- To catch the last three cases we have to inspect the outgoing headers:

```
Content-Type: application/pdf
```

Potential Performance Issue

- There is a potential performance issue if we redirect a GET request based on what we see in the response headers.
  - The PDF is going to have to be generated twice.
  - Think long-running reports… *not good*.
- There is a way to solve this but it's a bit of a stretch:
  - Store the response (PDF) into a temporary file.
  - Redirect request, serving the PDF (from the temporary file, without invoking the backend) when we see the corresponding token again.

Can we deal with POST requests?

- No; all redirections are to a GET.
  - We lose POST parameters.
- Well, strictly speaking, there is a way:
  - We could respond with a page that contains a self-submitting form with original parameters.
  - Or, as we did on the previous slide, store the response and issue a GET with a token to fetch it.
- But that's would be bit too much.
  - It could break applications in subtle ways.
  - It's probably "cheaper" to simply force PDF download in such cases.

Redirection Defence Implementations

- ModSecurity implements it as of 2.2.0-dev1: http://www.modsecurity.org
- Java Servlet filter:

  http://www.owasp.org/index.php/PDF_Attack_Filter_for_Java_EE
- .Net filter: http://www.techplay.net/pdfxssfilter.zip
- Using mod_rewrite: http://www.owasp.org/index.php/PDF_Attack_Filter_for_Apache_mod_rewrite
- F5 Solution using iRules: http://devcentral.f5.com
- There may be others...
  - Let me know if you find any.

Universal PDF XSS Defence Conclusion

- There is no perfect solution - only a trade-off between security, usability, and performance.
  - Isn't everything?
- Flaws to be aware of:
  - Does not protect from attackers sharing IP address with you.
  - Must fall back to forced download for dynamic requests.
- In general:
  - Carefully examine your chosen defence method to understand exactly when you are protected!

Experimenting with Content Injection

Client-side Defence Using Content Injection

- Why don't we inject a JavaScript fragment at the top of all outgoing HTML pages?
  - The JavaScript fragment will run in the browser.
  - It can get to the fragment identifier.
  - It can talk back to the server if anything suspicious is detected.
    - But it's trivial for someone (i.e. adversaries) to willingly produce too many to cause false positives.
      - Come to think of it, the same goes for any attack type.
  - Even prevention might work!

Content Injection Example

- Starting with 2.2.0-dev1 ModSecurity supports content injection (*prepend* & *append* features).
  - We are likely add features to inject content at arbitrary places in HTML at a later date.
- Example code:

```
SecRule RESPONSE_CONTENT_TYPE ^text/html \
"phase:3,nolog,pass,prepend:'PAGE_HEADER<hr>'"
```

- With JavaScript:

```
SecRule RESPONSE_CONTENT_TYPE ^text/html \
"phase:3,nolog,pass,prepend:\
'<script>document.write(\'Hello World\')</script>'
```

Content Injection Use Cases

- Possible uses of content injection:
  - Detect & prevent DOM-based Cross-Site Scripting attacks.
  - Detect anomalies (attacks) in DOM.
  - Perform DOM hardening at run-time.
  - Install code to intercept JavaScript events.
  - Perform implicit authentication to use to prevent session hijacking.
  - Even non-HTML responses can be replaced with an intermediate self-refreshing HTML page.

Conclusions, lessons, etc...

Conclusions

- The PDF XSS issue goes to the checklist of security professionals as a new problem all web applications must deal with.
- It's practically impossible to design and deploy a web application securely.
  - It's possible to get very close in a small number of cases – but at what cost?
- There is no hope for the current web application security model.
  - And we are sick from having to deal with it!

Collaborative Security Research

- Individually we are not smart enough to deal with the web application security issues.
  - Too many environments and moving parts.
  - Takes too long.
- Exciting things happen when a discussion is sparked in the community.
- Collaborative security research as the only viable option.
  - But it needs formalising – lacks structure.
  - Each issue needs a comprehensive summary.
  - We also need to address bad advice (in documentation).

Links and Resources

- **Vulnerability information:**
  - http://www.wisec.it/vulns.php?page=9#
  - http://events.ccc.de/congress/2006/Fahrplan/attachments/1158-Subverting_Ajax.pdf
  - http://www.adobe.com/support/security/bulletins/apsb07-01.html
- **Blogs:**
  - http://www.gnucitizen.org/blog/danger-danger-danger/
  - http://ha.ckers.org/blog/20070103/universal-xss-in-pdfs/
  - http://jeremiahgrossman.blogspot.com/2007/01/what-you-need-to-know-about-uxss-in.html
  - http://www.gnucitizen.org/blog/universal-pdf-xss-after-party/
- **Mailing lists:**
  - http://www.webappsec.org/lists/websecurity/archive/2007-01/msg00005.html

The End!

- **Do you have any questions?**
- Credits (in chronological order):

Amit Klein	Anonymous Slashdot user
Stefano Di Paola	Robert Auger
Giorgio Fedon	Martin O'Neal
Elia Florio	Tom Spector
Petko D. Petkov (pdp)	Ofer Shezaf
Robert Hansen (RSnake)	Ivan Ristic
James Landis	

- ...and others from the community.
  - You know who you are!