

# H5 页面漏洞挖掘之路 - 混淆篇 - SecPulse.COM | 安全脉搏

“ 复制代码混淆过后：

## 前言

针对上次我们提交漏洞之后，我们再次查看 JS 代码，定位加密函数和解密函数的位置，发现已经不是赤裸裸没有任何防护，而是已经进行了 JS 混淆，接下来我们针对遇到 JS 混淆后，我们该如何破解 JS 混淆后的代码进行加解密，继续进行渗透测试。笔者在这里提供一个思路和方法。

## 前置知识

首先我们先了解下代码混淆的具体原理是什么？其实很简单，就是去除代码中尽可能多的有意义的信息，比如注释、换行、空格、代码负号、变量重命名、属性重命名（允许的情况下）、无用代码的移除等等。因为代码是公开的，我们必须承认没有任何一种算法可以完全不被破解，所以，我们只能尽可能增加攻击者阅读代码的成本。

我将混淆类型分为两类：

## 变量名混淆

将变量名混淆成阅读比较难阅读的字符，增加代码阅读难度，而现在大部分厂商的混淆，都会将其混淆成 16 进制变量名。

效果如下：

```
`var test = 'helloworld';`
```

混淆后：

```
`var _0x7deb = 'helloworld';`
```

## 常量提取

将 JS 中的常量提取到数组中，调用的时候用数组下标的方式调用，这样的话直接读懂基本不可能了，要么反 AST 处理下，要么一步一步调试，工作量大增。

以上面的代码为例：

```
`var test = 'helloworld';`
```

复制代码混淆过后：

```
1 var _0x9d2b = ['helloworld'];
2 var _0xb7de = function (_0x4c7513) {
3     var _0x96ade5 = _0x9d2b[_0x4c7513];
4     return _0x96ade5;
5 };
6 var test = _0xb7de(0);
```

## 常量混淆

每个文件开头会有一个很长的字符数组，然后会有一段代码对这个数组进行加工，然后还有一个函数接收一个或两个参数输出一个字符串，这个字符串更接近原始的代码。将常量进行加密处理，上面的代码中，虽然已经是混淆过后的代码了，但是 helloworld 字符串还是以明文的形式出现在代码中，例如将关键字进行 Unicode16 进制编码。如下：

```
`var test = 'helloworld';`
```

结合常量提取得到混淆结果：

```
1 var _0x9d2b = ['\x68\x65\x6c\x6c\x6f'];
2
3 var _0xb7de = function (_0x4c7513) {
4     _0x4c7513 = _0x4c7513 - 0x0;
5     var _0x96ade5 = _0x9d2b[_0x4c7513];
6     return _0x96ade5;
7 };
8
9 var test = _0xb7de('0x0');
```

## 案例

### 第一部分：变量名称存储数组

这里存储了一些在函数中用到的变量和字符串。

```
1 var _0x2ec2 = [
2   'UGtjczc=',
3   'dG9TdHJpbmc=',
4   'ZGVjcnlwdA==',
5   'c3RyaW5naWZ5',
6   'xxxx',
7   'bW9kZQ==',
8   'Q0JD',
9   'cGFk'
10];
```

### 第二部分 数组处理函数

```

1  /**
2   * params _0x167407: 上面的字符串数组
3   * params _0x353595: 计数个数
4   * 把前 _0x353595 +1 个元素放到数组末尾
5  */
6  (function (_0x167407, _0x353595) {
7      var _0x52a3ae = function (_0x3fbe47) {
8          while (--_0x3fbe47) {
9              _0x167407['push'](_0x167407['shift']());
10         }
11     };
12     _0x52a3ae(++_0x353595);
13 })(_0xec2, 312));

```

## 第三部分 数组字符串处理函数

```

// 这个是数组内容解码的函数，实际上第二个参数是没有用到的
var _0x523d = function (_0x4c10d0, _0x393bf7) {
    _0x4c10d0 = _0x4c10d0 - 0; // 这里第一个参数是通过字符串
    var _0x70d87b = _0x2ec2[_0x4c10d0]; // 这里 _0x70d87b
    // 接下来判断有没有进行过初始化操作，如果没有的话，先初始化
    if (_0x523d['CuFQcU'] === undefined) {
        (function () {
            var _0x5b57a4 = function () {
                var _0x29e588;
                try {
                    _0x29e588 = Function('return (function() ' + '{}');
                } catch (_0x4956c9) {
                    _0x29e588 = window;
                }
                return _0x29e588;
            };
            var _0x2b121a = _0x5b57a4(); // 这里实际上返回的是 Window
            var _0x6c99b9 = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz';
            // 下面这个是判断Window有没有atob这个函数，如果没有的话生成
            _0x2b121a['atob'] || (_0x2b121a['atob'] = function (
                var _0x901f5e = String(_0x13f6f4) ['replace'](/=+/g,
                for (var _0x240979 = 0, _0x43e3e8, _0x42ec25, _0x6c99b9 =
                    _0x42ec25 = _0x6c99b9['indexOf'](_0x42ec25);
                )
                return _0x1c0a86;
            });
        })();
    }
}

```

```
});  
})();  
  
_0x523d['ZEesoG'] = function (_0x1de802) {  
    var _0x216ff1 = atob(_0x1de802);  
    var _0x42331f = [  
    ];  
    for (var _0x3a392f = 0, _0x2319db = _0x216ff1['length'];  
        _0x42331f += '%' + ('00' + _0x216ff1['charCodeAt'](0x3a392f + _0x2319db)));  
    return decodeURIComponent(_0x42331f);  
};  
// 到这里完成初始化操作，置CuFQcU为true，添加VgXLDn属性  
_0x523d['VgXLDn'] = {};  
_0x523d['CuFQcU'] = !![];  
}  
  
// 后面这段是先判断之前有没有对传入的参数进行解密过，如果解密过  
var _0x22ee7f = _0x523d['VgXLDn'][_0x4c10d0];  
if (_0x22ee7f === undefined) {  
    _0x70d87b = _0x523d['ZEesoG'](_0x70d87b);  
    _0x523d['VgXLDn'][_0x4c10d0] = _0x70d87b;  
} else {  
    _0x70d87b = _0x22ee7f;  
}  
return _0x70d87b;  
};
```

## 第四部分 加解密函数

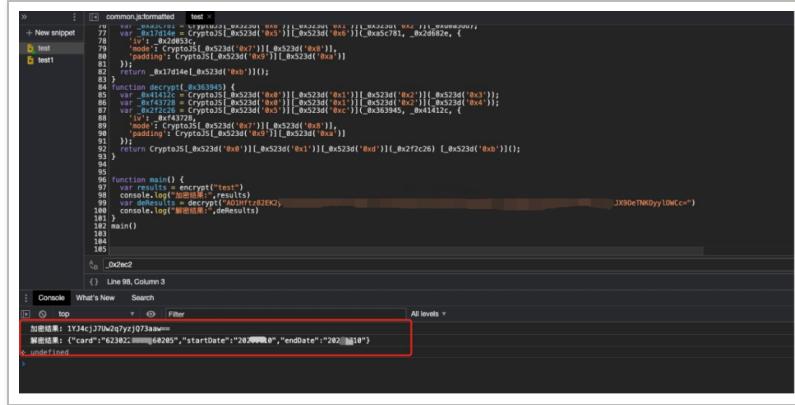
```
function encrypt(_0xd0a5dd) {
    var _0x2d682e = CryptoJS[_0x523d('0x0')][_0x523d('0x1')]();
    var _0x2d053c = CryptoJS[_0x523d('0x0')][_0x523d('0x1')]();
    var _0xa5c781 = CryptoJS[_0x523d('0x0')][_0x523d('0x1')]();
    var _0x17d14e = CryptoJS[_0x523d('0x5')][_0x523d('0x6')]();
    return {
        'iv': _0x2d053c,
        'mode': CryptoJS[_0x523d('0x7')][_0x523d('0x8')](),
        'padding': CryptoJS[_0x523d('0x9')][_0x523d('0xa')]()
    });
    return _0x17d14e[_0x523d('0xb')]();
}
function decrypt(_0x363945) {
    var _0x41412c = CryptoJS[_0x523d('0x0')][_0x523d('0x1')]();
    var _0xf43728 = CryptoJS[_0x523d('0x0')][_0x523d('0x1')]();
```

```

var _0x2f2c26 = CryptoJS[_0x523d('0x5')][_0x523d('0x
    'iv': _0xf43728,
    'mode': CryptoJS[_0x523d('0x7')][_0x523d('0x8')],
    'padding': CryptoJS[_0x523d('0x9')][_0x523d('0xa')]
  });
  return CryptoJS[_0x523d('0x0')][_0x523d('0x1')][_0x5
}

```

当我们分析整个混淆后的代码后，我们可以手动断点调试，来看看具体的解密之后每参数是什么。我们首先将整个混淆后的 js 代码 copy 下来，定义 main() 函数，调用加密 encrypt 和 decrypt 解密这两个函数，在浏览器下调试运行。



代码完美运行，在第三部分数组字符串处理函数的位置我们手动断点 F10 进行调试。

密钥 key 成功拿到：

```

test x VM737 test
47     }
48     return _0x1c8a86;
49   };
50 }
51 _0x23d['ZeesoG'] = function (_0x1de802) {
52   var _0x216ff1 = atob(_0x1de802);
53   var _0x42331f = '';
54   for (var i = 0; i < 0x3a392f; i++) {
55     if (_0x22ee7f == '%' + ('0' + '_0x216ff1['CharCodeAt'](0x3a392f)')['toString'](16)) ['slice'](-2);
56   }
57   return decodeURIComponent(_0x42331f);
58 }
59 _0x23d['Vg0L0n'] = {
60   _0x523d['CuP0tU'] = !![];
61 }
62 _0x23d['CuP0tU'] = {
63   var _0x22ee7f = _0x523d['Vg0L0n'][_0x4c10d0]; _0x22ee7f = undefined, _0x4c10d0 = 3;
64   if (_0x22ee7f === undefined) {
65     _0x7d887b = _0x523d['ZeesoG'].(_0x7d887b); _0x7d887b = "73PZ7...|47CK";
66     _0x523d['Vg0L0n'][!_0x22ee7f] = _0x7d887b; _0x4c10d0 = 3;
67   } else {
68     _0x7d887b = _0x22ee7f; _0x7d887b = "73PZ7...|47CK"; _0x22ee7f = undefined;
69   }
70 }
71 return _0x7d887b;
72 };
73 function encrypt(_0x0a5dd) {
74   var _0x23d['ZeesoG'] = CryptoJS._0x523d('0x0')[_0x523d('0x1')][_0x523d('0x2')][_0x523d('0x3')];
75   var _0x2d053c = CryptoJS._0x523d('0x0')[_0x523d('0x1')][_0x523d('0x2')][_0x523d('0x4')];
76   var _0xa5c781 = CryptoJS._0x523d('0x0')[_0x523d('0x1')][_0x523d('0x2')][_0x523d('0x3')];
77   var _0x42331f = CryptoJS._0x523d('0x0')[_0x523d('0x1')][_0x523d('0x2')][_0x523d('0x3')];
78   var _0x22ee7f = _0x523d['Vg0L0n'][_0x4c10d0];
79   var _0x7d887b = _0x22ee7f + '%0' + '_0x216ff1['CharCodeAt'](0x3a392f)['toString'](16) ['slice'](-2);
80   return decodeURIComponent(_0x42331f);
81 }
82 _0x23d['Vg0L0n'] = {
83   _0x523d['CuP0tU'] = !![];
84 }
85 _0x23d['CuP0tU'] = {
86   var _0x22ee7f = _0x523d['Vg0L0n'][_0x4c10d0]; _0x22ee7f = undefined, _0x4c10d0 = 4;
87   if (_0x22ee7f === undefined) {
88     _0x7d887b = _0x523d['ZeesoG'].(_0x7d887b); _0x7d887b = "5u1d...|u12";
89     _0x523d['Vg0L0n'][!_0x22ee7f] = _0x7d887b; _0x4c10d0 = 4;
90   } else {
91     _0x7d887b = _0x22ee7f; _0x7d887b = "5u1d...|u12"; _0x22ee7f = undefined;
92   }
93 }
94 return _0x7d887b;
95 };
96 function encrypt(_0x0a5dd) {
97   var _0x23d['ZeesoG'] = CryptoJS._0x523d('0x0')[_0x523d('0x1')][_0x523d('0x2')][_0x523d('0x3')];
98   var _0x2d053c = CryptoJS._0x523d('0x0')[_0x523d('0x1')][_0x523d('0x2')][_0x523d('0x4')];
99   var _0xa5c781 = CryptoJS._0x523d('0x0')[_0x523d('0x1')][_0x523d('0x2')][_0x523d('0x3')];
100  var _0x42331f = CryptoJS._0x523d('0x0')[_0x523d('0x1')][_0x523d('0x2')][_0x523d('0x3')];
101  var _0x22ee7f = _0x523d['Vg0L0n'][_0x4c10d0];
102  var _0x7d887b = _0x22ee7f + '%0' + '_0x216ff1['CharCodeAt'](0x3a392f)['toString'](16) ['slice'](-2);
103  var _0x2d053c3c = CryptoJS._0x523d('0x0')[_0x523d('0x1')][_0x523d('0x2')][_0x523d('0x3')];
104  var _0x43728 = CryptoJS._0x523d('0x0')[_0x523d('0x1')][_0x523d('0x2')][_0x523d('0x4')];
105  var _0x2f2c26 = CryptoJS._0x523d('0x5')[_0x523d('0xc')][_0x363945, _0x41412c, {
106    'IV': _0x43728,
107    'padding': CryptoJS._0x523d('0x9')[_0x523d('0xa')]
108  }];
109  return _0x1d14e1._0x523d('0xb')();
110 }
111 function decrypt(_0x363945) {
112   var _0x23d['ZeesoG'] = CryptoJS._0x523d('0x0')[_0x523d('0x1')][_0x523d('0x2')][_0x523d('0x3')];
113   var _0x2d053c = CryptoJS._0x523d('0x0')[_0x523d('0x1')][_0x523d('0x2')][_0x523d('0x4')];
114   var _0xa5c781 = CryptoJS._0x523d('0x0')[_0x523d('0x1')][_0x523d('0x2')][_0x523d('0x3')];
115   var _0x42331f = CryptoJS._0x523d('0x0')[_0x523d('0x1')][_0x523d('0x2')][_0x523d('0x3')];
116   var _0x22ee7f = _0x523d['Vg0L0n'][_0x4c10d0];
117   var _0x7d887b = _0x22ee7f + '%0' + '_0x216ff1['CharCodeAt'](0x3a392f)['toString'](16) ['slice'](-2);
118   var _0x2d053c3c = CryptoJS._0x523d('0x0')[_0x523d('0x1')][_0x523d('0x2')][_0x523d('0x3')];
119   var _0x43728 = CryptoJS._0x523d('0x0')[_0x523d('0x1')][_0x523d('0x2')][_0x523d('0x4')];
120   var _0x2f2c26 = CryptoJS._0x523d('0x5')[_0x523d('0xc')][_0x363945, _0x41412c, {
121     'IV': _0x43728,
122     'padding': CryptoJS._0x523d('0x9')[_0x523d('0xa')]
123   }];
124   return _0x1d14e1._0x523d('0xb')();
125 }
126 
```

得知加密算法为 AES:

```

test < VM737 test
47     }
48     return _0x1c0a66;
49   });
50   })();
51   _0x523d['_EesoG'] = function (_0x1de802) {
52     var _0x216ff1 = atob(_0x1de802);
53     var _0x2319db = _0x216ff1['length'];
54     var _0x2319dbf = _0x216ff1['charCodeAt'](0x3a392f) | _0x3a392f | _0x3a392f | _0x3a392f;
55     var _0x2319dbff = _0x216ff1['charCodeAt'](0x3a392f) | _0x3a392f | _0x3a392f | _0x3a392f;
56     var _0x2319dbfff = _0x216ff1['charCodeAt'](0x3a392f) | _0x3a392f | _0x3a392f | _0x3a392f;
57     var _0x2319dbffff = _0x216ff1['charCodeAt'](0x3a392f) | _0x3a392f | _0x3a392f | _0x3a392f;
58     _0x523d['VgxDm'] = {
59       '_2': _0x2319dbffff,
60     };
61   };
62   _0x22ee7f = _0x523d['VgxDm'][0x4c10d0]; _0x22ee7f = undefined; _0x4c10d0 = 5
63   _0x523d['_CuFcU'] = ![];
64   _0x523d['_CuFcU'] = ![];
65   if (_0x22ee7f === undefined) {
66     _0x70d87b = _0x523d['_EesoG'](_0x70d87b); _0x70d87b = "CBC"
67   } else {
68     _0x70d87b = _0x22ee7f; _0x70d87b = "CBC"; _0x22ee7f = undefined
69   }
70 } _0x70d87b = _0x70d87b; _0x70d87b = "CBC"; _0x22ee7f = undefined
71 } _0x70d87b;
72 }
73 function encrypt(_0x0055d) {
74   var _0x2d682e = CryptoJS[_0x523d('0x0')][0x523d('0x1')][0x523d('0x2')][0x523d('0x3'));
75   var _0x2d053c = CryptoJS[_0x523d('0x0')][0x523d('0x1')][0x523d('0x2')][0x523d('0x4');
76   var _0x17d14e = CryptoJS[_0x523d('0x5')][0x523d('0x6')][0x523d('0x7')][0x523d('0x8')];
77   var _0x17d14e[_0x523d('0x5')] = _0x2d682e;
78   var _0x17d14e[_0x523d('0x6')] = _0x2d053c;
79   var _0x17d14e[_0x523d('0x7')] = _0x17d14e;
80   var _0x17d14e[_0x523d('0x8')] = _0x17d14e;
81   _0x17d14e[_0x523d('0x9')] = _0x17d14e;
82   return _0x17d14e[_0x523d('0x10')];
83 }
84 function decrypt(_0x363945) {
85   var _0x41412c = CryptoJS[_0x523d('0x0')][0x523d('0x1')][0x523d('0x2')][0x523d('0x3');
86   var _0x4f3728 = CryptoJS[_0x523d('0x0')][0x523d('0x1')][0x523d('0x2')][0x523d('0x4');
87   var _0x4f3728 = CryptoJS[_0x523d('0x5')][0x523d('0x6')][0x523d('0x7')][0x363945];
88   var _0x4f3728[_0x523d('0x5')] = _0x41412c;
89   var _0x4f3728[_0x523d('0x6')] = _0x4f3728;
90   var _0x4f3728[_0x523d('0x7')] = _0x4f3728;
91   var _0x4f3728[_0x523d('0x8')] = _0x4f3728;
92   var _0x4f3728[_0x523d('0x9')] = _0x4f3728;
93   var _0x4f3728[_0x523d('0xa')] = _0x4f3728;

```

AES 加密算法使用的填充方式：Pkcs7

至  
们  
此  
源  
码  
被  
解  
密  
后  
的  
结  
果  
一  
致。

```

test1 test x
64 var _0x2ee7f = _0x523d['VgXLdn'][_0x4c10d0], _0x2ee7f = undefined, _0x4c10d0 = 10
65 if (_0x2ee7f === undefined) {
66   _0x2ee7f = CryptoJS[0x523d['mode']](_0x708b7b, "PKCS7");
67   _0x523d['VgXLdn'][_0x4c10d0] = _0x708b7b; _0x4c10d0 = 10
68 } else {
69   _0x708b7b = _0x2ee7f; _0x708b7b = "PKCS7"; _0x2ee7f = undefined
70 }
71 return _0x708b7b;
72 }

73 function encrypt(_0x0a5dd) {
74   var _0xd682e = CryptoJS[0x523d('0x0')][_0x523d('0x1')][_0x523d('0x2')][_0x523d('0x3')];
75   var _0x2ee7f = CryptoJS[0x523d('0x0')][_0x523d('0x1')][_0x523d('0x2')][_0x523d('0x4')];
76   var _0x5c781 = CryptoJS[0x523d('0x5')][_0x523d('0x6')][_0x523d('0x7')][_0x523d('0x8')];
77   var _0x17d14e = CryptoJS[0x523d('0x5')][_0x523d('0x6')][_0x523d('0x7')][_0x523d('0x8')];
78   _0xa01c = CryptoJS[0x523d('0x1')][_0x523d('0x2')][_0x523d('0x3')][_0x523d('0x4')];
79   _0xpudding : CryptoJS[0x523d('0x1')][_0x523d('0x2')][_0x523d('0x3')][_0x523d('0x4')];
80 }
81 return _0x17d14e[_0x523d('0xb')];
82 }

83 function decrypt(_0x0a5dd) {
84   var _0x4112c = CryptoJS[0x523d('0x0')][_0x523d('0x1')][_0x523d('0x2')][_0x523d('0x3')];
85   var _0x143728 = CryptoJS[0x523d('0x0')][_0x523d('0x1')][_0x523d('0x2')][_0x523d('0x4')];
86   var _0x17d14e = CryptoJS[0x523d('0x5')][_0x523d('0x6')][_0x523d('0x7')][_0x523d('0x8')];
87   var _0x17d14e = CryptoJS[0x523d('0x5')][_0x523d('0x6')][_0x523d('0x7')][_0x523d('0x8')];
88   var _0x17d14e = CryptoJS[0x523d('0x5')][_0x523d('0x6')][_0x523d('0x7')][_0x523d('0x8')];
89   var _0x17d14e = CryptoJS[0x523d('0x7')][_0x523d('0x8')];
90   var _0xpadding : CryptoJS[0x523d('0x9')][_0x523d('0xa')];
91 }
92 return CryptoJS[0x523d('0x0')][_0x523d('0x1')][_0x523d('0x2')][_0x2f2c26][_0x523d('0xb')]);
93 }

I test - VM1301.net
+ New snippet
  1 test
  2 test1 test x
  3 test
  4 test1 test x
  5 test
  6 test1 test x
  7 test
  8 test1 test x
  9 test
  10 test1 test x
  11 test
  12 test1 test x
  13 test
  14 test1 test x
  15 test
  16 test1 test x
  17 test
  18 test1 test x
  19 test
  20 test1 test x
  21 test
  22 test1 test x
  23 test
  24 test1 test x
  25 test
  26 test1 test x
  27 test
  28 test1 test x
  29 test
  30 test1 test x
  31 test
  32 test1 test x
  33 test
  34 test1 test x
  35 test
  36 test1 test x
  37 test
  38 if __name__ == "__main__":
  39     encrypt_value = AESUtil.encrypt(encrypt_value, key, iv)
  40     encrypt_value = AESUtil.decrypt(decrypt_value, key, iv)
  41     print("加密结果:", str(encrypt_value, encoding='utf-8'))
  42     decrypt_value = AESUtil.decrypt(decrypt_value, key, iv)
  43     decrypt_value = AESUtil.decrypt(decrypt_value, key, iv)
  44     print("解密结果:", str(decrypt_value, encoding='utf-8'))
  45
  46 加密结果: 1Y34cj7Uw2q7yzjQ73aaew=
  47 解密结果: {"card": "62302200 12345605", "startDate": "2024-01-01", "endDate": "2026-01-01"}
  48 [Finished in 0.1s]

```

## 总结

JS 混淆在安全对抗中必不可少，一是对保护前端页面的代码逻辑，二是对前端登陆的算法密钥和向量 IV 进行保护。而我们通过反混淆还原代码或者直接调用混淆后的 JS 代码进行调试，获取密钥和向量 IV，从而达到解密密文，篡改数据包继续进行漏洞挖掘。

# 参考

<https://www.52pojie.cn/thread-1104122-1-1.html#2985657>

---

全文完

---

本文由 简悦 SimpRead 优化，用以提升阅读体验

使用了 全新的简悦词法分析引擎 beta，[点击查看详细说明](#)

