何为Js混淆

JavaScript 混淆是一种将 JavaScript 代码进行某种处理的方式，目的是使代码难以被阅读和理解。why do that ?

JavaScript大都是运行在浏览器端,这就导致任何人都可以直接对网站的代码进行查看，如果代码没有进行任何处理就会导致直接暴露源码，他人便可轻而易举的复制你的劳动成果,所以可以做的就是让代码变得更加的难以阅读,他人难以复制你的成果,实现"加密"的目的。

混淆的方法可以包括改变变量名、添加无意义的代码、删除空格和注释等。混淆后的代码仍然可以正常运行，但是对于未经授权的人来说难以理解和维护。

- 变量重命名
  - 变量重命名是 JavaScript 代码混淆中最简单且最常用的方法之一。这种方法的基本思想是将所有可能包含敏感信息的变量重命名为无意义的字符串。例如，将 `username` 替换为 `a`，将 `password` 替换为 `b`。将变量名替换为无意义的字符串或者短字符，使得人类阅读难度大大增加。例如：
  - ```
    // 没有代码混淆
    var username = "user123";
    var password = "pass456";
    function login(user, pass) {
        if (user === username && pass === password) {
        console.log("Login successful!");
      } else {
        console.log("Login failed!");
      }}
    login(username, password);

    // 使用变量名混淆进行代码混淆
    var a = "user123";
    var b = "pass456";
    function c(d, e) {
        if (d === a && e === b) {
        console.log("Login successful!");
      } else {
        console.log("Login failed!");
      }}
    c(a, b);
    ```

- 函数名混淆
  - 函数名混淆是另一种常用的 JavaScript 代码混淆技术。它的基本思想是将所有函数名替换为随机的、无意义的字符串，从而使代码更难被理解和调试。
    ```

    - ```
      // 没有代码混淆
      function add(x, y) {
        return x + y;
      }
      console.log(add(2, 3));

      // 使用函数名混淆进行代码混淆
      var a = function(b, c) {
                  return b + c;
                  }
      console.log(a(2, 3));

      //这种本质还是和上面介绍的变量重命名一样。
      ```

- 压缩代码
    - 压缩是另一种常见的JavaScript代码混淆技术。通过使用各种压缩算法，可以将JavaScript代码文件缩小到原始大小的一半以下。虽然这样的代码难以阅读，但对于需要快速加载和运行的Web应用程序来说非常有用。例如：
    - ```
      //原始代码
      function calculateSum(num1, num2) {
          var sum = num1 + num2;
        return sum;
      }
      var result = calculateSum(3, 4);
      console.log(result);
      ```
    - ```
      //压缩代码
      function calculateSum(a,b){return a+b}console.log(calculateSum(3,4));
      ```

- 代码打乱 / JS 控制流混淆
    - JS 控制流混淆是一种消除JavaScript代码的可预测性的技术，通过使用控制流混淆算法，改变程序结构来防止代码被轻易地理解和分析的技术。简单来说，就是通过修改代码中的控制流语句（如 if、while 等）的顺序、嵌套、去除等方式来增加代码的复杂性，从而使得代码难以被反编译、破解或者逆向工程攻击。以下是一个简短的示例：
    - ```
      var a = 1;
      var b = 0;

      if (a === 1) {
          b = 3;
        } else {
          b = 4;
        }

      //结果：a=1,b=3
      ```
    - 这段代码可以通过控制流混淆的方式进行优化。例如，可以将 `if` 块内部的语句交错、嵌套，增加运算符的数量，达到混淆的效果：

- ```
  var a = 1;
  var b = 0;
  if (!(a !== 1)) {
    if (!(![])) b = 3;
  } else {
    if (!(!{})) b = 4;
  }
  //结果：a=1,b=3
  ```

OB混淆

OB 混淆是指将 JavaScript 代码中的变量名、函数名、字符串等替换为无意义的字符串，从而增加代码的保护性和防止代码的逆向分析。此外，它还可以在代码中添加死代码、无用的函数等，增加代码的复杂度和难以理解性，从而增加代码的保密性。

官网：https://obfuscator.io/

```
var _0x30bb = ['log', 'Hello\x20World!']; //列表元素(被混淆的真实代码部分)也有可能会被base64等机制编码

//该函数是将_0x30bb列表进行了元素进行了移位，将列表变为了['Hello\x20World!','log']
(function (_0x38d89d, _0x30bbb2) {
    var _0xae0a32 = function (_0x2e4e9d) {
        while (--_0x2e4e9d) {
            _0x38d89d['push'](_0x38d89d['shift']());
        }
    };
    _0xae0a32(++_0x30bbb2);
}(_0x30bb, 0x153)); //在浏览器Console中加载执行该自运行函数后，打印_0x30bb列表查看移位结果

//该函数为解密函数（用来还原真实的代码）
var _0xae0a = function (_0x38d89d, _0x30bbb2) {
    _0x38d89d = _0x38d89d - 0x0;
    var _0xae0a32 = _0x30bb[_0x38d89d];
    return _0xae0a32;
};
//ob混淆的js代码
function hi() {
    console[_0xae0a('0x1')](_0xae0a('0x0'));
    //使用_0xae0a函数还原代码：_0xae0a('0x1')还原为了log，因此在OB混淆中该函数使用的频率最高。
}

hi();
```

实战

爬取不同年份对应的票房信息：

https://www.endata.com.cn/BoxOffice/BO/Year/index.html
```

分析

- 1、根据抓包工具可以抓取到票房数据的数据包（切换不同年份，获取对应年份的票房数据）
  - url：https://www.endata.com.cn/API/GetData.ashx
  - 请求方式：POST （Ajax的请求模式）
  - 请求参数：year(变化的)和MethodName(固定形式)
  - 响应数据：经过加密的密文数据
    - 观察密文数据，发现全部为16进制的数据。
- 2、数据解密分析：
  - 获取不同年份的票房数据的请求为ajax请求，请求到的是密文数据，但是在网页中显示却是明文数据，因此解密的操作肯定会出现在ajax请求的代码中：
    - ```
      $("#btn").click(function() {
            $.ajax({
                type: "POST",
                url: "yourUrl", // 请求地址
                data: { // 请求参数
                    param1: "value1",
                    param2: "value2"
                },
                dataType: "json", // 服务器返回的数据类型
                success: function(data) { // 成功回调函数
                    //data就是请求到的密文数据
                     //请求成功后对data的处理肯定就是解密的操作代码
                },
                error: function(error) { // 失败回调函数
                    console.log(error); // 打印错误信息
                }
            });
        });
      ```
- 因此只需要定位到该ajax的代码即可，如何定位呢？
  - 在ajax中需要指定请求的url，则根据步骤1种的ajax请求的url进行定位
    - 全局搜索：GetData.ashx，定位到Common.js数据包，在数据包内部看到了Ajax对应的代码。
      - ```
        $.ajax({
                        url: n + "/API/GetData.ashx",
                        data: t + "&MethodName=" + e,
                        type: "POST",
                        dataType: "text",
                        timeout: 3e4,
                        success: function(e, t, n) {
                            try { //核心解密代码！！！
                                1 == (e = "{" == e[0] ? JSON.parse(e) :
        JSON.parse(webInstace.shell(e))).Status || 200 == e.Code ?
        r(e.Data) : 200 == e.code ? r(e.data) : o(e.Msg)
                            } catch (e) {
                                o(e)
                            }
                        },
        ```

```
                error: function(e, t, n) {
                    console.log(t),
                    o(t)
                }
            })
        }
```

- 核心解密代码分析
  - ```
    //对其进行拆解
    1 == (e = "{" == e[0] ? JSON.parse(e) :
    JSON.parse(webInstace.shell(e))).Status || 200 == e.Code ?
    r(e.Data) : 200 == e.code ? r(e.data) : o(e.Msg)

    //1.三目运算符：其中e是密文数据
    1 == (e = "{" == e[0] ? JSON.parse(e) :
    JSON.parse(webInstace.shell(e)))
    ```
    e是一个16进制数据因此e[0]不会等于{,因此三目运算返回的是：JSON.parse(webInstace.shell(e))，然后三目运算返回的结果赋值给了e。则现在简化成了：e=JSON.parse(webInstace.shell(e))。这个简化结果就暴露了数据解密的接口，因为：JSON.parse()使用来进行反序列化的（字符串转换为对象），那么16进制的密文数据无法被直接parse反序列化，因此webInstace.shell(e)肯定就是对密文e进行解密，将解密后的数据被parse进行反序列化。因此webInstace.shell(e)就是解密接口。

    知道了解密接口后，后面的操作就无需再看了。

- 3、断点调试
  - 断点打在步骤2的核心代码处。单点进入到解密函数shell(e)的函数实现中进行分析查看：
    - 函数名应该是shell，但是进去看到的函数名为：0x2246('0x257', 'nArV')，可能是ob混淆对函数名进行了编码，可以选中0x2246('0x257', 'nArV')发现其值就是为shell，因此这个_0x2246函数就是用来在ob混淆中还原编码代码的（**代码还原函数**）。并且发现后续代码中该函数使用的频率非常大。
      - 在shell函数实现的第一行应该并没有看到shell函数名，而是_0x2246('0x257', 'nArV')这个东西，则在console里运行0x2246('0x257', 'nArV')，返回的就是shell这个函数名。
  - 分析函数_0x2246()的实现：在代码文件的最上面看到了该函数的定义，定义中看到了atob和btoa等函数，这些函数的功能等同于python中的base64。atob函数是将一组字符被base64进行编码，btoa是反编码。
    - 在代码还原函数0x2246()中有base64的编码操作，因此OB混淆的大数组中的元素肯定是被经过了Base64编码的结果。
  - 并且在在函数_0x2246()上面的代码中也看到了OB混淆主要结构（大数组、移位函数、代码还原函数），因此0x2246()这个函数就作为了OB混淆中的代码还原函数，因此它被用的次数最多。
  - 分析改写shell函数的实现：
    - ```
      //还原内部的_0x2246
      function shell(_0xa0c834) {
              //一个字典对象，内部定义了一些键值对
              var _0x51eedc = {
                  'pKENi': function _0x2f627(_0x5b6f5a, _0x440924) {
      ```

```
                return _0x5b6f5a === _0x440924;
            },
            'wnfPa': 'ZGz',
            'VMmle': '7|1|8|9|5|2|3|6|0|4',
            'GKWFf': function _0x1a4e13(_0x40cfde, _0x16f3c2) {
                return _0x40cfde == _0x16f3c2;
            },
            'MUPgQ': function _0x342f0d(_0x19038b, _0x4004d6) {
                return _0x19038b >= _0x4004d6;
            },
            'hLXma': function _0x55adaf(_0x45a871, _0x161bdf) {
                return _0x45a871 + _0x161bdf;
            },
            'JdOlO': function _0x13e00a(_0x5899a9, _0x4bb34d) {
                return _0x5899a9 + _0x4bb34d;
            },
            'qrTpg': function _0x1198fb(_0x55b317, _0x22e1db,
_0x1b091a) {
                return _0x55b317(_0x22e1db, _0x1b091a);
            },
            'pdmMk': function _0xe2b022(_0x4af286, _0x4c2fd4) {
                return _0x4af286 - _0x4c2fd4;
            },
            'xVKWW': function _0x1094a3(_0x5f3627, _0x2a0ac5,
_0x3ad2e5) {
                return _0x5f3627(_0x2a0ac5, _0x3ad2e5);
            }
        };

        var _0x492a62 = [7,1,8,9,5,2,3,6,0,4]
          , _0x356b01 = 0;
        //将switch内部代码按照指定顺序执行改写，还原内部的_0x2246
        if (!navigator || !navigator['userAgent'])
                        return '';
        if (_0x51eedc['GKWFf'](null, _0xa0c834) ||
_0x51eedc['MUPgQ'](0x10, _0xa0c834['length']))
                        return _0xa0c834;
        var _0x554c90 = _0x51eedc['JdOlO'](_0x51eedc['qrTpg']
(parseInt, _0xa0c834[_0x51eedc['pdmMk'](_0xa0c834['length'], 0x1)],
0x10), 0x9)
                      , _0x2cf8ae = _0x51eedc['xVKWW'](parseInt,
_0xa0c834[_0x554c90], 0x10);
        _0xa0c834 = _0x9843d3(_0xa0c834, _0x554c90, 0x1);
        _0x554c90 = _0xa0c834['substr'](_0x2cf8ae, 0x8);
        _0xa0c834 = _0x9843d3(_0xa0c834, _0x2cf8ae, 0x8);
        _0x2cf8ae = _grsa_JS['enc']['Utf8']['parse'](_0x554c90);
        _0x554c90 = _grsa_JS['enc']['Utf8']['parse'](_0x554c90);
        _0x554c90 = _grsa_JS['DES']['decrypt']({
                        'ciphertext': _grsa_JS['enc']['Hex']
['parse'](_0xa0c834)
                    }, _0x2cf8ae, {
                        'iv': _0x554c90,
                        'mode': _grsa_JS['mode']['ECB'],
                        'padding': _grsa_JS['pad']['Pkcs7']
                    })['toString'](_grsa_JS['enc']['Utf8']);
```

```
        return _0x554c90['substring'](0x0, _0x51eedc['hLXma']
(_0x554c90['lastIndexOf']('}'), 0x1));
    }
```

- ```
//还原_0x51eedc()函数
function shell(_0xa0c834) {

        //将switch内部代码按照指定顺序执行改写，还原内部的_0x2246,然后在处理
_0x51eedc
        if (!navigator || !navigator['userAgent'])
                        return '';
        if (null == _0xa0c834 || 0x10 >= _0xa0c834['length'])
                        return _0xa0c834;
        var _0x554c90 =
parseInt(_0xa0c834[_0xa0c834['length']-0x1],0x10)+0x9
                        , _0x2cf8ae =
parseInt(_0xa0c834[_0x554c90],0x10)

        _0xa0c834 = _0x9843d3(_0xa0c834, _0x554c90, 0x1);
        _0x554c90 = _0xa0c834['substr'](_0x2cf8ae, 0x8);
        _0xa0c834 = _0x9843d3(_0xa0c834, _0x2cf8ae, 0x8);
        _0x2cf8ae = _grsa_JS['enc']['Utf8']['parse'](_0x554c90);
        _0x554c90 = _grsa_JS['enc']['Utf8']['parse'](_0x554c90);
        _0x554c90 = _grsa_JS['DES']['decrypt']({
                        'ciphertext': _grsa_JS['enc']['Hex']
['parse'](_0xa0c834)
                    }, _0x2cf8ae, {
                        'iv': _0x554c90,
                        'mode': _grsa_JS['mode']['ECB'],
                        'padding': _grsa_JS['pad']['Pkcs7']
                    })['toString'](_grsa_JS['enc']['Utf8']);
        return _0x554c90['substring'](0x0, _0x554c90['lastIndexOf']
('}')+ 0x1);
    }

//上述代码中_0x9843d3函数被多次调用，但是没有该函数的实现，补充该函数的实现，
shell函数定义中可以找到该函数，将其添加到改写文件中
```

- ```
//处理、添加_0x9843d3()函数，函数内部的_0x2246进行还原即可

 // var _0x9843d3 = function(_0x29d556, _0xcc6df, _0x3d7020) {
 //        if (0x0 == _0xcc6df)
 //            return _0x29d556[_0x2246('0x254', '4vZ$')]
(_0x3d7020);
 //        var _0x48914b;
 //        _0x48914b = '' + _0x29d556[_0x2246('0x255', 'GL3Q')](0x0,
_0xcc6df);
 //        return _0x48914b += _0x29d556['substr']
(_0x4da59e[_0x2246('0x256', 'DK[&')](_0xcc6df, _0x3d7020));
 //    };

//还原了_0x9843d3()函数的_0x2246后，发现该函数内部还使用了_0x4da59e函数，因
此将该函数的定义添加上。
 var _0x4da59e = {
        'bUIIa': function _0x2a2af9(_0x779387, _0x4a4fec) {
```

```
            return _0x779387 + _0x4a4fec;
        }
    };
//还原了_0x9843d3()函数的_0x2246
 var _0x9843d3 = function(_0x29d556, _0xcc6df, _0x3d7020) {
        if (0x0 == _0xcc6df)
            return _0x29d556['substr'](_0x3d7020);
        var _0x48914b;
        _0x48914b = '' + _0x29d556['substr'](0x0, _0xcc6df);
        return _0x48914b += _0x29d556['substr'](_0x4da59e['bUIIa']
(_0xcc6df, _0x3d7020));
    };

function shell(_0xa0c834) {

        //将switch内部代码按照指定顺序执行改写，还原内部的_0x2246,然后在处理
_0x51eedc
        if (!navigator || !navigator['userAgent'])
                        return '';
        if (null == _0xa0c834 || 0x10 >= _0xa0c834['length'])
                        return _0xa0c834;
        var _0x554c90 =
parseInt(_0xa0c834[_0xa0c834['length']-0x1],0x10)+0x9
                      , _0x2cf8ae =
parseInt(_0xa0c834[_0x554c90],0x10)

        _0xa0c834 = _0x9843d3(_0xa0c834, _0x554c90, 0x1);
        _0x554c90 = _0xa0c834['substr'](_0x2cf8ae, 0x8);
        _0xa0c834 = _0x9843d3(_0xa0c834, _0x2cf8ae, 0x8);
        _0x2cf8ae = _grsa_JS['enc']['Utf8']['parse'](_0x554c90);
        _0x554c90 = _grsa_JS['enc']['Utf8']['parse'](_0x554c90);
        _0x554c90 = _grsa_JS['DES']['decrypt']({
                        'ciphertext': _grsa_JS['enc']['Hex']
['parse'](_0xa0c834)
                    }, _0x2cf8ae, {
                        'iv': _0x554c90,
                        'mode': _grsa_JS['mode']['ECB'],
                        'padding': _grsa_JS['pad']['Pkcs7']
                    })['toString'](_grsa_JS['enc']['Utf8']);
        return _0x554c90['substring'](0x0, _0x554c90['lastIndexOf']
('}')+ 0x1);
    }
```

- ```
  var CryptoJS = require('crypto-js') //需要事先安装crypto这个库:npm
  install crypto-js,然后使用CryptoJS替换所有的_grsa_JS
  var _0x4da59e = {
          'bUIIa': function _0x2a2af9(_0x779387, _0x4a4fec) {
              return _0x779387 + _0x4a4fec;
          }
      };

  var _0x9843d3 = function(_0x29d556, _0xcc6df, _0x3d7020) {
          if (0x0 == _0xcc6df)
              return _0x29d556['substr'](_0x3d7020);
          var _0x48914b;
          _0x48914b = '' + _0x29d556['substr'](0x0, _0xcc6df);
  ```
  ```

```
        return _0x48914b += _0x29d556['substr'](_0x4da59e['bUIIa']
(_0xcc6df, _0x3d7020));
    };

function shell(_0xa0c834) {

        //将switch内部代码按照指定顺序执行改写，还原内部的_0x2246,然后在处理
_0x51eedc
        // if (!navigator || !navigator['userAgent'])
        //                 return '';
        if (null == _0xa0c834 || 0x10 >= _0xa0c834['length'])
                        return _0xa0c834;
        var _0x554c90 =
parseInt(_0xa0c834[_0xa0c834['length']-0x1],0x10)+0x9
                      , _0x2cf8ae =
parseInt(_0xa0c834[_0x554c90],0x10)

        _0xa0c834 = _0x9843d3(_0xa0c834, _0x554c90, 0x1);
        _0x554c90 = _0xa0c834['substr'](_0x2cf8ae, 0x8);
        _0xa0c834 = _0x9843d3(_0xa0c834, _0x2cf8ae, 0x8);
        _0x2cf8ae = CryptoJS['enc']['Utf8']['parse'](_0x554c90);
        _0x554c90 = CryptoJS['enc']['Utf8']['parse'](_0x554c90);
        _0x554c90 = CryptoJS['DES']['decrypt']({
                        'ciphertext': CryptoJS['enc']['Hex']
['parse'](_0xa0c834)
                    }, _0x2cf8ae, {
                        'iv': _0x554c90,
                        'mode': CryptoJS['mode']['ECB'],
                        'padding': CryptoJS['pad']['Pkcs7']
                    })['toString'](CryptoJS['enc']['Utf8']);
        return _0x554c90['substring'](0x0, _0x554c90['lastIndexOf']
('}')+ 0x1);
    }

console.log(shell('密文数据')) //密文数据从response中复制

//在终端里：node xxx.js进行代码执行,报错后将包含navigato的if进行删除即可
（nodejs中没有navigator）
```