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，而是已经进行的了 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';`
```

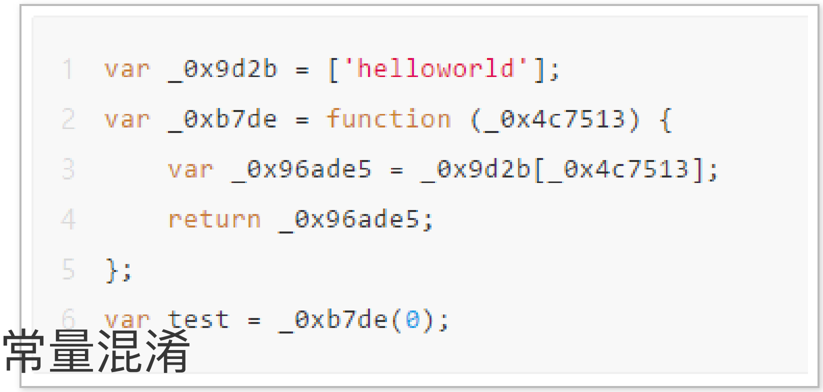
常量提取

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：



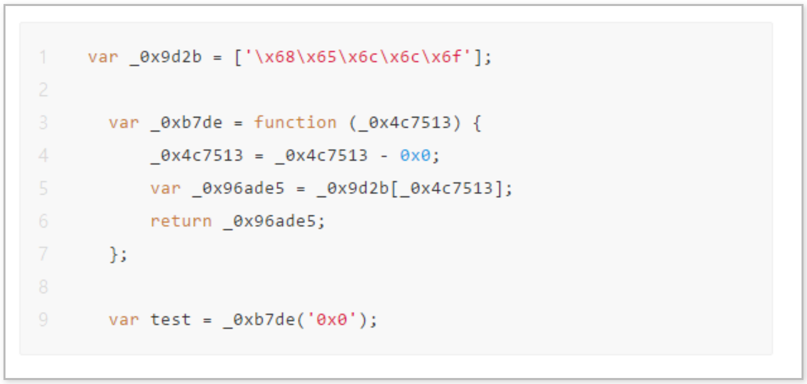
```
var _0x9d2b = ['helloworld'];
var _0xb7de = function (_0x4c7513) {
    var _0x96ade5 = _0x9d2b[_0x4c7513];
    return _0x96ade5;
};
var test = _0xb7de(0);
```

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：

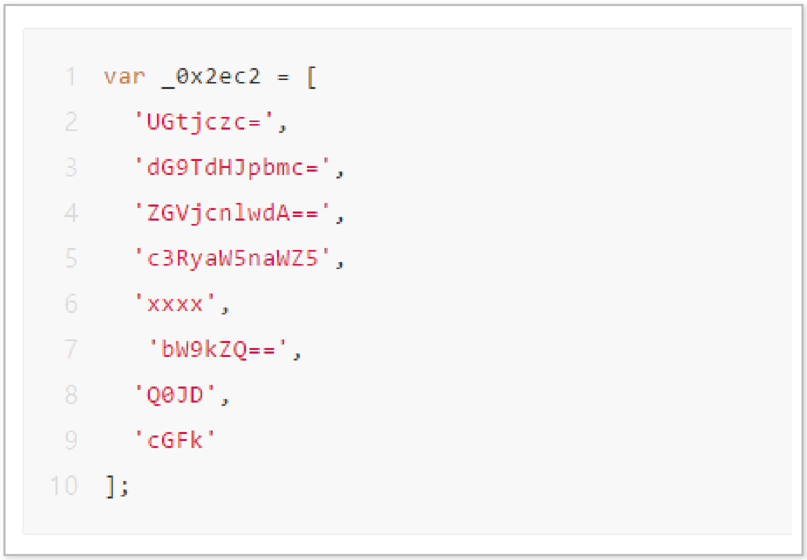


```
var _0x9d2b = ['\x68\x65\x6c\x6c\x6f'];

  var _0xb7de = function (_0x4c7513) {
      _0x4c7513 = _0x4c7513 - 0x0;
      var _0x96ade5 = _0x9d2b[_0x4c7513];
      return _0x96ade5;
  };

  var test = _0xb7de('0x0');
```

案例

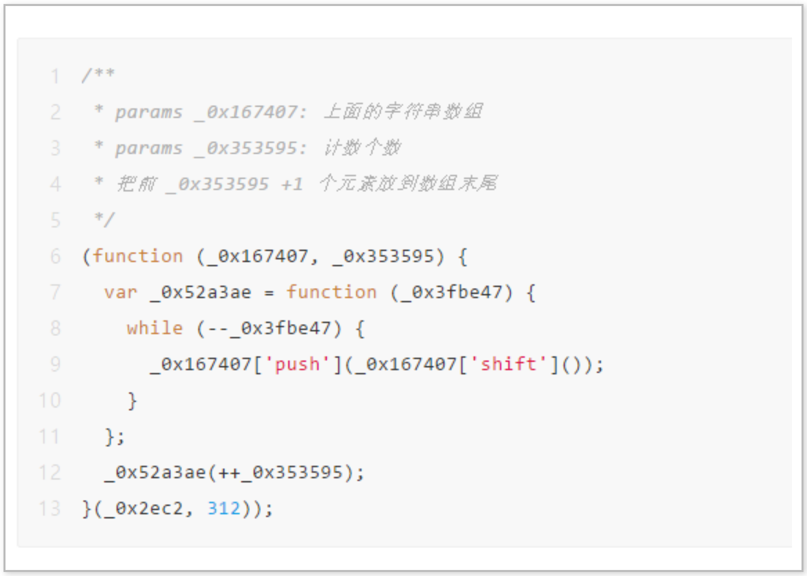
第一部分: 变量名称存储数组

这里存储了一些在函数中用到的变量和字符串。



```
var _0x2ec2 = [
  'UGtjczc=',
  'dG9TdHJpbmc=',
  'ZGVjcnlwdA==',
  'c3RyaW5naWZ5',
  'xxxx',
   'bW9kZQ==',
  'Q0JD',
  'cGFk'
];
```

第二部分 数组处理函数



```
/**
 * params _0x167407: 上面的字符串数组
 * params _0x353595: 计数个数
 * 把前 _0x353595 +1 个元素放到数组末尾
 */
(function (_0x167407, _0x353595) {
  var _0x52a3ae = function (_0x3fbe47) {
    while (--_0x3fbe47) {
      _0x167407['push'](_0x167407['shift']());
    }
  };
  _0x52a3ae(++_0x353595);
}(_0x2ec2, 312));
```

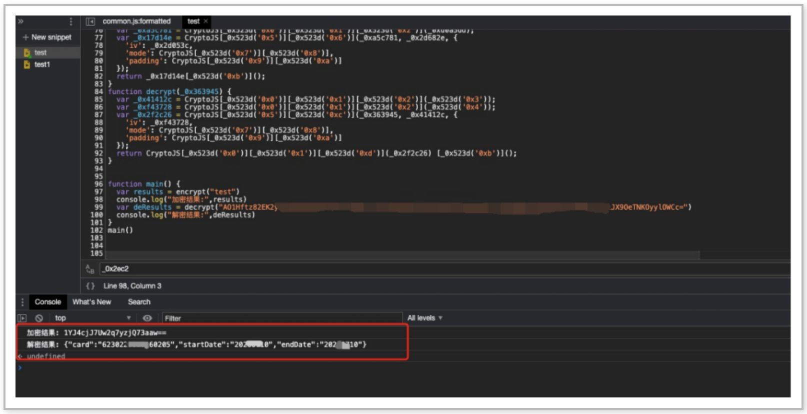
第三部分 数组字符串处理函数

```
// 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 _0x70d87
  // 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 Win
  var _0x6c99b9 = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghi
  // 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'](/=+
   for (var _0x240979 = 0, _0x43e3e8, _0x42ec25, _0x6
      _0x42ec25 = _0x6c99b9['indexOf'](_0x42ec25);
    }
    return _0x1c0a86;
    });
   }());

   _0x523d['ZEesoG'] = function (_0x1de802) {
     var _0x216ff1 = atob(_0x1de802);
     var _0x42331f = [
     ];
     for (var _0x3a392f = 0, _0x2319db = _0x216ff1['le
       _0x42331f += '%' + ('00' + _0x216ff1['charCodeA
      }
      return decodeURIComponent(_0x42331f);
     };
     // 到这里完成初始化操作，置CuFQcU为true，添加VgXLDn属
     _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
   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
   '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('0x
  var _0xf43728 = CryptoJS[_0x523d('0x0')][_0x523d('0x
  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。



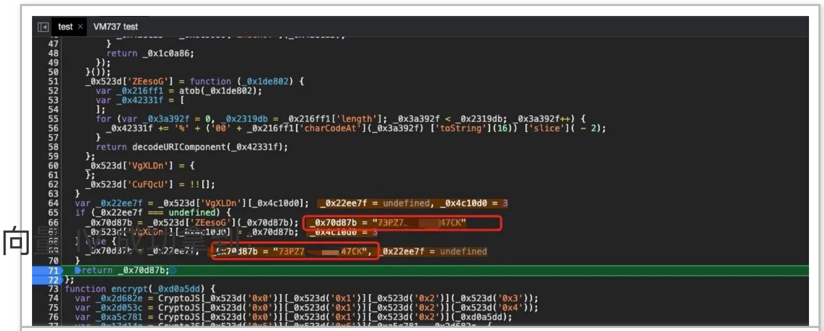
```
  var _0x17d14e = CryptoJS[_0x523d('0x5')][_0x523d('0x6')](_0xa5c781, _0x2d682e, {
    'iv': _0x2d053c,
    'mode': CryptoJS[_0x523d('0x7')][_0x523d('0x8')],
    'padding': CryptoJS[_0x523d('0x9')][_0x523d('0xa')]
  });
  return _0x17d14e[_0x523d('0xb')]();
}
function decrypt(_0x363945) {
  var _0x41412c = CryptoJS[_0x523d('0x0')][_0x523d('0x1')][_0x523d('0x2')](_0x523d('0x3'));
  var _0xf43728 = CryptoJS[_0x523d('0x0')][_0x523d('0x1')][_0x523d('0x2')](_0x523d('0x4'));
  var _0x2f2c26 = CryptoJS[_0x523d('0x5')][_0x523d('0xc')](_0x363945, _0x41412c, {
    'iv': _0xf43728,
    'mode': CryptoJS[_0x523d('0x7')][_0x523d('0x8')],
    'padding': CryptoJS[_0x523d('0x9')][_0x523d('0xa')]
  });
  return CryptoJS[_0x523d('0x0')][_0x523d('0x1')][_0x523d('0xd')](_0x2f2c26) [_0x523d('0xb')]();
}


function main() {
  var results = encrypt("test")
  console.log("加密结果:",results)
  var deResults = decrypt("A01Hftz82EK2y[redacted]JX90eTNKOyylOWCc=")
  console.log("解密结果:",deResults)
}
main()
```

```
加密结果: 1YJ4cjJ7Uw2q7yzjQ73aaw==
解密结果: {"card":"623022[redacted]60205","startDate":"202[redacted]0","endDate":"202[redacted]10"}
< undefined
>
```

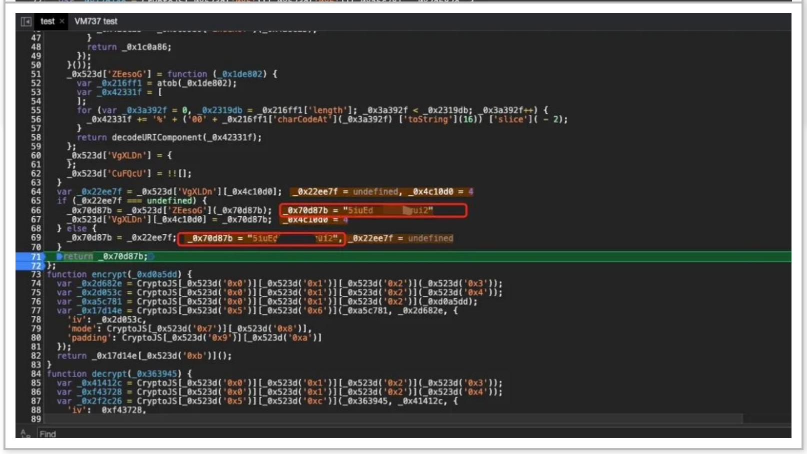
代码完美运行，在第三部分数组字符串处理函数的位置我们手动断点 F10 进行调试。

密钥 key 成功拿到：

向量 IV 成功拿到



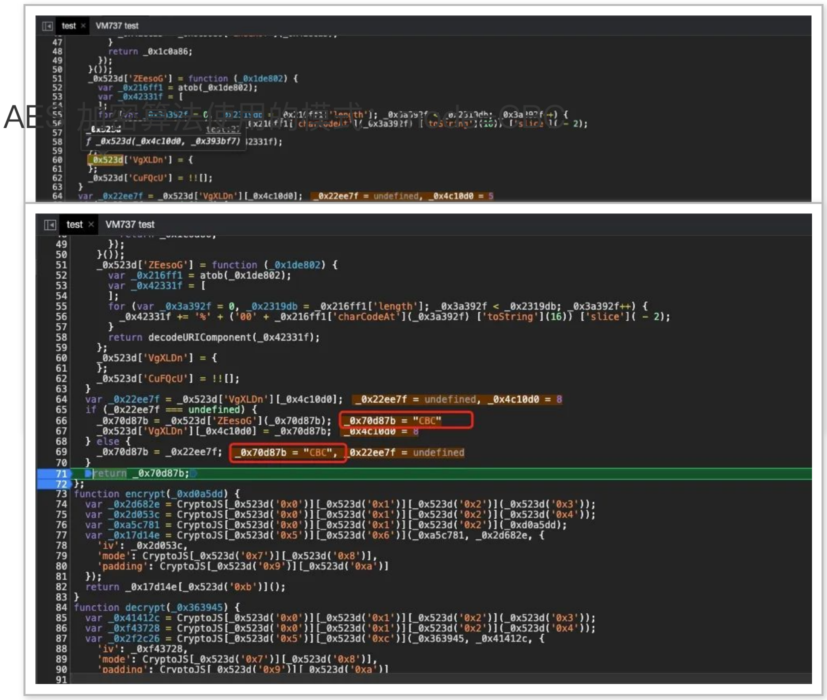
```
        }
        return _0x1c0a86;
      });
    }());
    _0x523d['ZEesoG'] = function (_0x1de802) {
      var _0x216ff1 = atob(_0x1de802);
      var _0x42331f = [
      ];
      for (var _0x3a392f = 0, _0x2319db = _0x216ff1['length']; _0x3a392f < _0x2319db; _0x3a392f++) {
        _0x42331f += '%' + ('00' + _0x216ff1['charCodeAt'](_0x3a392f) ['toString'](16)) ['slice']( - 2);
      }
      return decodeURIComponent(_0x42331f);
    };
    _0x523d['VgXLDn'] = {
    };
    _0x523d['CuFQcU'] = !![];
  }
  var _0x22ee7f = _0x523d['VgXLDn'][_0x4c10d0];  _0x22ee7f = undefined, _0x4c10d0 = 3
  if (_0x22ee7f === undefined) {
    _0x70d87b = _0x523d['ZEesoG'](_0x70d87b);  _0x70d87b = "73PZ7[redacted]47CK"
    _0x523d['VgXLDn'][_0x4c10d0] = _0x70d87b;  _0x4c10d0 = 3
  } else {
    _0x70d87b = _0x22ee7f;  _0x70d87b = "73PZ7[redacted]47CK", _0x22ee7f = undefined
  }
  return _0x70d87b;
};
function encrypt(_0xd0a5dd) {
  var _0x2d682e = CryptoJS[_0x523d('0x0')][_0x523d('0x1')][_0x523d('0x2')](_0x523d('0x3'));
  var _0x2d053c = CryptoJS[_0x523d('0x0')][_0x523d('0x1')][_0x523d('0x2')](_0x523d('0x4'));
  var _0xa5c781 = CryptoJS[_0x523d('0x0')][_0x523d('0x1')][_0x523d('0x2')](_0xd0a5dd);
```



```
        }
        return _0x1c0a86;
      });
    }());
    _0x523d['ZEesoG'] = function (_0x1de802) {
      var _0x216ff1 = atob(_0x1de802);
      var _0x42331f = [
      ];
      for (var _0x3a392f = 0, _0x2319db = _0x216ff1['length']; _0x3a392f < _0x2319db; _0x3a392f++) {
        _0x42331f += '%' + ('00' + _0x216ff1['charCodeAt'](_0x3a392f) ['toString'](16)) ['slice']( - 2);
      }
      return decodeURIComponent(_0x42331f);
    };
    _0x523d['VgXLDn'] = {
    };
    _0x523d['CuFQcU'] = !![];
  }
  var _0x22ee7f = _0x523d['VgXLDn'][_0x4c10d0];  _0x22ee7f = undefined, _0x4c10d0 = 4
  if (_0x22ee7f === undefined) {
    _0x70d87b = _0x523d['ZEesoG'](_0x70d87b);  _0x70d87b = "5iuEd[redacted]ui2"
    _0x523d['VgXLDn'][_0x4c10d0] = _0x70d87b;  _0x4c10d0 = 4
  } else {
    _0x70d87b = _0x22ee7f;  _0x70d87b = "5iuEd[redacted]ui2", _0x22ee7f = undefined
  }
  return _0x70d87b;
};
function encrypt(_0xd0a5dd) {
  var _0x2d682e = CryptoJS[_0x523d('0x0')][_0x523d('0x1')][_0x523d('0x2')](_0x523d('0x3'));
  var _0x2d053c = CryptoJS[_0x523d('0x0')][_0x523d('0x1')][_0x523d('0x2')](_0x523d('0x4'));
  var _0xa5c781 = CryptoJS[_0x523d('0x0')][_0x523d('0x1')][_0x523d('0x2')](_0xd0a5dd);
  var _0x17d14e = CryptoJS[_0x523d('0x5')][_0x523d('0x6')](_0xa5c781, _0x2d682e, {
    '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')][_0x523d('0x2')](_0x523d('0x3'));
  var _0xf43728 = CryptoJS[_0x523d('0x0')][_0x523d('0x1')][_0x523d('0x2')](_0x523d('0x4'));
  var _0x2f2c26 = CryptoJS[_0x523d('0x5')][_0x523d('0xc')](_0x363945, _0x41412c, {
    'iv': _0xf43728,
```

得知加密算法为 AES：

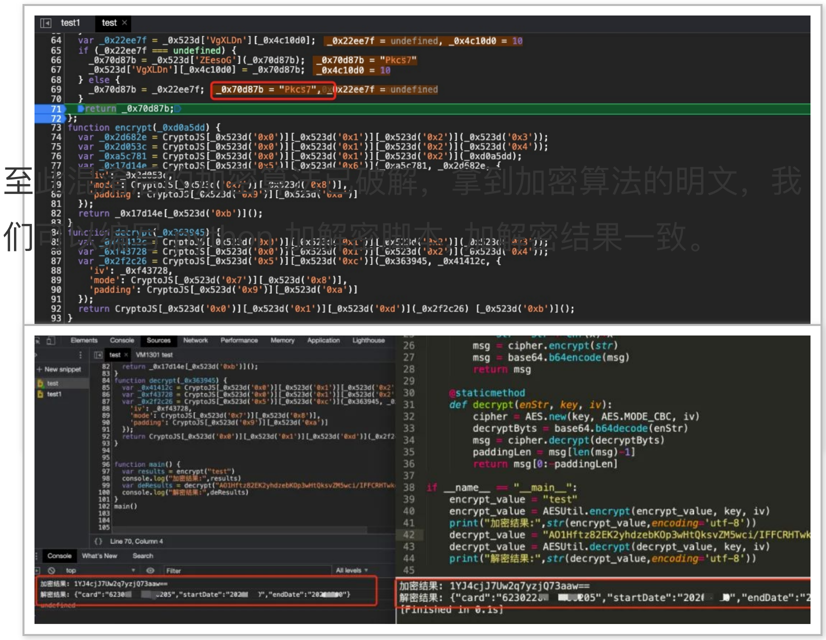
AES 加密算法使用的模式：mode CBC



```
      }
      return _0x1c0a86;
    });
  }());
  _0x523d['ZEesoG'] = function (_0x1de802) {
    var _0x216ff1 = atob(_0x1de802);
    var _0x42331f = [
    ];
    for (var _0x3a392f = 0, _0x2319db = _0x216ff1['length']; _0x3a392f < _0x2319db; _0x3a392f++) {
                                  _0x216ff1['charCodeAt'](_0x3a392f) ['toString'](16)) ['slice']( - 2);
  _0x523d                   test:27
  f _0x523d(_0x4c10d0, _0x393bf7)  2331f);

  _0x523d['VgXLDn'] = {
  };
  _0x523d['CuFQcU'] = !![];
}
var _0x22ee7f = _0x523d['VgXLDn'][_0x4c10d0];  _0x22ee7f = undefined, _0x4c10d0 = 5
```

```
    });
  }());
  _0x523d['ZEesoG'] = function (_0x1de802) {
    var _0x216ff1 = atob(_0x1de802);
    var _0x42331f = [
    ];
    for (var _0x3a392f = 0, _0x2319db = _0x216ff1['length']; _0x3a392f < _0x2319db; _0x3a392f++) {
      _0x42331f += '%' + ('00' + _0x216ff1['charCodeAt'](_0x3a392f) ['toString'](16)) ['slice']( - 2);
    }
    return decodeURIComponent(_0x42331f);
  };
  _0x523d['VgXLDn'] = {
  };
  _0x523d['CuFQcU'] = !![];
}
var _0x22ee7f = _0x523d['VgXLDn'][_0x4c10d0];  _0x22ee7f = undefined, _0x4c10d0 = 8
if (_0x22ee7f === undefined) {
  _0x70d87b = _0x523d['ZEesoG'](_0x70d87b);  _0x70d87b = "CBC"
  _0x523d['VgXLDn'][_0x4c10d0] = _0x70d87b;  _0x4c10d0 = 8
} else {
  _0x70d87b = _0x22ee7f;  _0x70d87b = "CBC", _0x22ee7f = undefined
}
return _0x70d87b;
};
function encrypt(_0xd0a5dd) {
  var _0x2d682e = CryptoJS[_0x523d('0x0')][_0x523d('0x1')][_0x523d('0x2')](_0x523d('0x3'));
  var _0x2d053c = CryptoJS[_0x523d('0x0')][_0x523d('0x1')][_0x523d('0x2')](_0x523d('0x4'));
  var _0xa5c781 = CryptoJS[_0x523d('0x0')][_0x523d('0x1')][_0x523d('0x2')](_0xd0a5dd);
  var _0x17d14e = CryptoJS[_0x523d('0x5')][_0x523d('0x6')](_0xa5c781, _0x2d682e, {
    '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')][_0x523d('0x2')](_0x523d('0x3'));
  var _0xf43728 = CryptoJS[_0x523d('0x0')][_0x523d('0x1')][_0x523d('0x2')](_0x523d('0x4'));
  var _0x2f2c26 = CryptoJS[_0x523d('0x5')][_0x523d('0xc')](_0x363945, _0x41412c, {
    'iv': _0xf43728,
    'mode': CryptoJS[_0x523d('0x7')][_0x523d('0x8')],
    'padding': CryptoJS[_0x523d('0x9')][_0x523d('0xa')]
```

AES 加密算法使用的填充方式：Pkcs7



```
  var _0x22ee7f = _0x523d['VgXLDn'][_0x4c10d0];  _0x22ee7f = undefined, _0x4c10d0 = 10
  if (_0x22ee7f === undefined) {
    _0x70d87b = _0x523d['ZEesoG'](_0x70d87b);  _0x70d87b = "Pkcs7"
    _0x523d['VgXLDn'][_0x4c10d0] = _0x70d87b;  _0x4c10d0 = 10
  } else {
    _0x70d87b = _0x22ee7f;  _0x70d87b = "Pkcs7", _0x22ee7f = undefined
  }
  return _0x70d87b;
};
function encrypt(_0xd0a5dd) {
  var _0x2d682e = CryptoJS[_0x523d('0x0')][_0x523d('0x1')][_0x523d('0x2')](_0x523d('0x3'));
  var _0x2d053c = CryptoJS[_0x523d('0x0')][_0x523d('0x1')][_0x523d('0x2')](_0x523d('0x4'));
  var _0xa5c781 = CryptoJS[_0x523d('0x0')][_0x523d('0x1')][_0x523d('0x2')](_0xd0a5dd);
  var _0x17d14e = CryptoJS[_0x523d('0x5')][_0x523d('0x6')](_0xa5c781, _0x2d682e, {
    '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')][_0x523d('0x2')](_0x523d('0x3'));
  var _0xf43728 = CryptoJS[_0x523d('0x0')][_0x523d('0x1')][_0x523d('0x2')](_0x523d('0x4'));
  var _0x2f2c26 = CryptoJS[_0x523d('0x5')][_0x523d('0xc')](_0x363945, _0x41412c, {
    'iv': _0xf43728,
    'mode': CryptoJS[_0x523d('0x7')][_0x523d('0x8')],
    'padding': CryptoJS[_0x523d('0x9')][_0x523d('0xa')]
  });
  return CryptoJS[_0x523d('0x0')][_0x523d('0x1')][_0x523d('0xd')](_0x2f2c26) [_0x523d('0xb')]();
}
```

```
  return _0x17d14e[_0x523d('0xb')]();
}
function decrypt(_0x363945) {
  var _0x41412c = CryptoJS[_0x523d('0x0')][_0x523d('0x1')][_0x523d('0x2'
  var _0xf43728 = CryptoJS[_0x523d('0x0')][_0x523d('0x1')][_0x523d('0x2'
  var _0x2f2c26 = CryptoJS[_0x523d('0x5')][_0x523d('0xc')](_0x363945, _0
    'iv': _0xf43728,
    'mode': CryptoJS[_0x523d('0x7')][_0x523d('0x8')],
    'padding': CryptoJS[_0x523d('0x9')][_0x523d('0xa')]
  });
  return CryptoJS[_0x523d('0x0')][_0x523d('0x1')][_0x523d('0xd')](_0x2f2


function main() {
  var results = encrypt("test")
  console.log("加密结果:",results)
  var deResults = decrypt("A01Hftz82EK2yhdzebKOp3wHtQksvZM5wci/IFFCRHTwk
  console.log("解密结果:",deResults)
}
main()
```

```
加密结果: 1YJ4cjJ7Uw2q7yzjQ73aaw==
解密结果: {"card":"6230[redacted]205","startDate":"202[redacted]","endDate":"202[redacted]0"}
undefined
```

```
        msg = cipher.encrypt(str)
        msg = base64.b64encode(msg)
        return msg

    @staticmethod
    def decrypt(enStr, key, iv):
        cipher = AES.new(key, AES.MODE_CBC, iv)
        decryptByts = base64.b64decode(enStr)
        msg = cipher.decrypt(decryptByts)
        paddingLen = msg[len(msg)-1]
        return msg[0:-paddingLen]

if __name__ == "__main__":
    encrypt_value = "test"
    encrypt_value = AESUtil.encrypt(encrypt_value, key, iv)
    print("加密结果:",str(encrypt_value,encoding='utf-8'))
    decrypt_value = "A01Hftz82EK2yhdzebKOp3wHtQksvZM5wci/IFFCRHTwk
    decrypt_value = AESUtil.decrypt(decrypt_value, key, iv)
    print("解密结果:",str(decrypt_value,encoding='utf-8'))
```

```
加密结果: 1YJ4cjJ7Uw2q7yzjQ73aaw==
解密结果: {"card":"623022[redacted]205","startDate":"202[redacted]","endDate":"2
[Finished in 0.1s]
```

至此混淆后的加密算法已破解，拿到加密算法的明文，我们可以编写 python 加解密脚本，加解密结果一致。

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
```