babycrack

发表于 2021-01-21 分类于 Challenge ， 2017 ， HCTF ， Web
Challenge | 2017 | HCTF | Web | babycrack

点击此处获得更好的阅读体验

WriteUp 来源

https://xz.aliyun.com/t/1589

题目考点

解题思路

```
babycrack

Description
just babycrack
1.flag.substr(-5,3)=="333"
2.flag.substr(-8,1)=="3"
3.Every word makes sence.
4.sha256(flag)=="d3f154b641251e319855a73b010309a168a12927f3873c97d2e5163ea5cbb443"

Now Score 302.93
Team solved 45
```

还是很抱歉题目的验证逻辑还是出现了不可逆推的问题，被迫在比赛中途加入4个hint来修复问题，下面我们来慢慢看看代码。

整个题目由反调试+代码混淆+逻辑混淆3部分组成，你可以说题目毫无意义完全为了出题而出题，但是这种代码确实最最真实的前端代码，现在许多站点都会选择使用反调试+混淆+一定程度的代码混淆来混淆部分前端代码。

出题思路主要有两篇文章：

http://www.jianshu.com/p/9148d215c119

https://zhuanlan.zhihu.com/p/29214928

整个题目主要是在我分析chrome拓展后门时候构思的，代码同样经过了很多重的混淆，让我们来一步步解释。

反调试

第一部分是反调试，当在页面内使用F12来调试代码时，会卡死在debugger代码处。

这里举个例子就是蘑菇街的登陆验证代码。

具体代码是这样的

```
eval(function(p,a,c,k,e,r){e=function(c){return c.toString(a)};if(!''.replace(/^/,String)){while(c--)r[e(c)]=k[c]||e(c);k=[function(e){return r[e]}];e=function(){return'\\
```

美化一下

```
(function () {
    (function a() {
        try {
            (function b(i) {
                if (('' + (i / i)).length !== 1 || i % 20 === 0) {
                    (function () {}).constructor('debugger')()
                } else {
                    debugger
                }
                b(++i)
            })(0)
        } catch (e) {
            setTimeout(a, 5000)
        }
    })()
})();
```

这就是比较常见的反调试。我这里提供3种办法来解决这步。

1、使用node做代码调试。 由于这里的debugger检测的是浏览器的调试，如果直接对代码调试就不会触发这样的问题。

2、静态分析 因为题目中代码较少，我没办法把代码混入深层逻辑，导致代码可以纯静态分析。

3、patch debugger函数 由于debugger本身智慧触发一次，不会无限制的卡死调试器，这里会出现这种情况，主要是每5s轮询检查一次。那么我们就可以通过patch settimeout函数来绕过。

```
window._setTimeout = window.setTimeout;
window.setTimeout = function () {};
```

这里可以用浏览器插件TamperMonkey解决问题。

除了卡死debug以外，我还加入了轮询刷新console的代码。

```
setInterval("window.console.log('Welcome to HCTF :>')", 50);
```

同样的办法可以解决，就不多说了。

代码混淆

在去除掉这部分无用代码之后，我们接着想办法去除代码混淆。

这里最外层的代码混淆，我是通过https://github.com/javascript-obfuscator/javascript-obfuscator做了混淆。

ps:因为我在代码里加入了es6语法，市面上的很多工具都不支持es6语法，会导致去混淆的代码语法错误！

更有趣的是，这种混淆是不可逆的，所以我们只能通过逐渐去混淆的方式来美化代码。

我们可以先简单美化一下代码格式

```
(function (_0xd4b7d6, _0xad25ab) {
    var _0x5e3956 = function (_0x1661d3) {
        while (--_0x1661d3) {
            _0xd4b7d6['push'](_0xd4b7d6['shift']());
        }
    };
    _0x5e3956(++_0xad25ab);
}(_0x180a, 0x1a2));
var _0xa180 = function (_0x5c351c, _0x2046d8) {
    _0x5c351c = _0x5c351c - 0x0;
    var _0x26f3b3 = _0x180a[_0x5c351c];
    return _0x26f3b3;
};

function check(_0x5b7c0c) {
    try {
        var _0x2e2f8d = ['code', _0xa180('0x0'), _0xa180('0x1'), _0xa180('0x2'), 'invalidMonetizationCode', _0xa180('0x3'), _0xa180('0x4'), _0xa180('0x5'), _0xa180('0x6'),
        var _0x50559f = _0x5b7c0c[_0x2e2f8d[0x5]](0x0, 0x4);
        var _0x5cea12 = parseInt(btoa(_0x50559f), 0x20);
        eval(function (_0x200db2, _0x177f13, _0x46da6f, _0x802d91, _0x2d59cf, _0x2829f2) {
            _0x2d59cf = function (_0x4be75f) {
                return _0x4be75f['toString'](_0x177f13);
            };
            if (!'' ['replace'](/^/, String)) {
                while (_0x46da6f--) _0x2829f2[_0x2d59cf(_0x46da6f)] = _0x802d91[_0x46da6f] || _0x2d59cf(_0x46da6f);
                _0x802d91 = [function (_0x5e8f1a) {
                    return _0x2829f2[_0x5e8f1a];
                }];
                _0x2d59cf = function () {
                    return _0xa180('0x2b');
                };
                _0x46da6f = 0x1;
            };
            while (_0x46da6f--)
                if (_0x802d91[_0x46da6f]) _0x200db2 = _0x200db2[_0xa180('0x2c')](new RegExp('\x5cb' + _0x2d59cf(_0x46da6f) + '\x5cb', 'g'), _0x802d91[_0x46da6f]);
            return _0x200db2;
        }(_0xa180('0x2d'), 0x11, 0x11, _0xa180('0x2e')['split']('|'), 0x0, {}));
        (function (_0x3291b7, _0xced890) {
            var _0xaed809 = function (_0x3aba26) {
                while (--_0x3aba26) {
                    _0x3291b7[_0xa180('0x4')](_0x3291b7['shift']());
                }
            };
            _0xaed809(++_0xced890);
        }(_0x2e2f8d, _0x5cea12 % 0x7b));
        var _0x43c8d1 = function (_0x3120e0) {
            var _0x3120e0 = parseInt(_0x3120e0, 0x10);
            var _0x3a882f = _0x2e2f8d[_0x3120e0];
            return _0x3a882f;
        };
        var _0x1c3854 = function (_0x52ba71) {
            var _0x52b956 = '0x';
            for (var _0x59c050 = 0x0; _0x59c050 < _0x52ba71[_0x43c8d1(0x8)]; _0x59c050++) {
                _0x52b956 += _0x52ba71[_0x43c8d1('f')](_0x59c050)[_0x43c8d1(0xc)](0x10);
            }
            return _0x52b956;
        };
        var _0x76e1e8 = _0x5b7c0c[_0x43c8d1(0xe)]('_');
        var _0x34f55b = (_0x1c3854(_0x76e1e8[0x0][_0x43c8d1(0xd)](-0x2, 0x2)) ^ _0x1c3854(_0x76e1e8[0x0][_0x43c8d1(0xd)](0x4, 0x1))) % _0x76e1e8[0x0][_0x43c8d1(0x8)] == (
        if (!_0x34f55b) {
            return ![];
        }
        b2c = function (_0x3f9bc5) {
            var _0x3c3bd8 = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ234567';
            var _0x4dc510 = [];
            var _0x4a199f = Math[_0xa180('0x25')](_0x3f9bc5[_0x43c8d1(0x8)] / 0x5);
            var _0x4ee491 = _0x3f9bc5[_0x43c8d1(0x8)] % 0x5;
            if (_0x4ee491 != 0x0) {
                for (var _0x1e1753 = 0x0; _0x1e1753 < 0x5 - _0x4ee491; _0x1e1753++) {
                    _0x3f9bc5 += '';
                }
                _0x4a199f += 0x1;
            }
            for (_0x1e1753 = 0x0; _0x1e1753 < _0x4a199f; _0x1e1753++) {
                _0x4dc510[_0x43c8d1('1b')](_0x3c3bd8[_0x43c8d1('1d')](_0x3f9bc5[_0x43c8d1('f')](_0x1e1753 * 0x5) >> 0x3));
                _0x4dc510[_0x43c8d1('1b')](_0x3c3bd8[_0x43c8d1('1d')]((_0x3f9bc5[_0x43c8d1('f')](_0x1e1753 * 0x5) & 0x7) << 0x2 | _0x3f9bc5[_0x43c8d1('f')](_0x1e1753 * 0x
                _0x4dc510[_0x43c8d1('1b')](_0x3c3bd8[_0x43c8d1('1d')]((_0x3f9bc5[_0x43c8d1('f')](_0x1e1753 * 0x5 + 0x1) & 0x3f) >> 0x1));
                _0x4dc510[_0x43c8d1('1b')](_0x3c3bd8[_0x43c8d1('1d')]((_0x3f9bc5[_0x43c8d1('f')](_0x1e1753 * 0x5 + 0x1) & 0x1) << 0x4 | _0x3f9bc5[_0x43c8d1('f')](_0x1e17
                _0x4dc510[_0x43c8d1('1b')](_0x3c3bd8[_0x43c8d1('1d')]((_0x3f9bc5[_0x43c8d1('f')](_0x1e1753 * 0x5 + 0x2) & 0xf) << 0x1 | _0x3f9bc5[_0x43c8d1('f')](_0x1e17
                _0x4dc510[_0x43c8d1('1b')](_0x3c3bd8[_0x43c8d1('1d')]((_0x3f9bc5[_0x43c8d1('f')](_0x1e1753 * 0x5 + 0x3) & 0x7f) >> 0x2));
                _0x4dc510[_0x43c8d1('1b')](_0x3c3bd8[_0x43c8d1('1d')]((_0x3f9bc5[_0x43c8d1('f')](_0x1e1753 * 0x5 + 0x3) & 0x3) << 0x3 | _0x3f9bc5[_0x43c8d1('f')](_0x1e17
                _0x4dc510[_0x43c8d1('1b')](_0x3c3bd8[_0x43c8d1('1d')](_0x3f9bc5[_0x43c8d1('f')](_0x1e1753 * 0x5 + 0x4) & 0x1f));
            }
            var _0x545c12 = 0x0;
            if (_0x4ee491 == 0x1) _0x545c12 = 0x6;
            else if (_0x4ee491 == 0x2) _0x545c12 = 0x4;
            else if (_0x4ee491 == 0x3) _0x545c12 = 0x3;
            else if (_0x4ee491 == 0x4) _0x545c12 = 0x1;
            for (_0x1e1753 = 0x0; _0x1e1753 < _0x545c12; _0x1e1753++) _0x4dc510[_0xa180('0x2f')]();
            for (_0x1e1753 = 0x0; _0x1e1753 < _0x545c12; _0x1e1753++) _0x4dc510[_0x43c8d1('1b')]('=');
            (function () {
                (function _0x3c3bd8() {
                    try {
                        (function _0x4dc510(_0x460a91) {
                            if (('' + _0x460a91 / _0x460a91)[_0xa180('0x30')] !== 0x1 || _0x460a91 % 0x14 === 0x0) {
                                (function () {}['constructor']('debugger')());
                            } else {
                                debugger;
                            }
                            _0x4dc510(++_0x460a91);
                        }(0x0));
                    } catch (_0x30f185) {
                        setTimeout(_0x3c3bd8, 0x1388);
                    }
                }());
            }());
            return _0x4dc510[_0xa180('0x31')]('');
        };
        e = _0x1c3854(b2c(_0x76e1e8[0x2])[_0x43c8d1(0xe)]('=')[0x0]) ^ 0x53a3f32;
        if (e != 0x4b7c0a73) {
            return ![];
        }
        f = _0x1c3854(b2c(_0x76e1e8[0x3])[_0x43c8d1(0xe)]('=')[0x0]) ^ e;
        if (f != 0x4315332) {
            return ![];
        }
        n = f * e * _0x76e1e8[0x0][_0x43c8d1(0x8)];
        h = function (_0x4c466e, _0x28871) {
            var _0x3ea581 = '';
            for (var _0x2fbf7a = 0x0; _0x2fbf7a < _0x4c466e[_0x43c8d1(0x8)]; _0x2fbf7a++) {
                _0x3ea581 += _0x28871(_0x4c466e[_0x2fbf7a]);
            }
```

```
            return _0x3ea581;
        };
        j = _0x76ele8[0x1][_0x43c8d1(0xe)]('3');
        if (j[0x0][_0x43c8d1(0x8)] != j[0x1][_0x43c8d1(0x8)] || (_0x1c3854(j[0x0]) ^ _0x1c3854(j[0x1])) != 0x1613) {
            return ![];
        }
        k = _0xffcc52 => _0xffcc52[_0x43c8d1('f')]() * _0x76ele8[0x1][_0x43c8d1(0x8)];
        l = h(j[0x0], k);
        if (l != 0x2f9b5072) {
            return ![];
        }
        m = _0x1c3854(_0x76ele8[0x4][_0x43c8d1(0xd)](0x0, 0x4)) - 0x48a05362 == n % l;

        function _0x5a6d56(_0x5a25ab, _0x4a4483) {
            var _0x55b09f = '';
            for (var _0x508ace = 0x0; _0x508ace < _0x4a4483; _0x508ace++) {
                _0x55b09f += _0x5a25ab;
            }
            return _0x55b09f;
        }
        if (!m || _0x5a6d56(_0x76ele8[0x4][_0x43c8d1(0xd)](0x5, 0x1), 0x2) == _0x76ele8[0x4][_0x43c8d1(0xd)](-0x5, 0x4) || _0x76ele8[0x4][_0x43c8d1(0xd)](-0x2, 0x1) - _0x
            return ![];
        }
        o = _0x1c3854(_0x76ele8[0x4][_0x43c8d1(0xd)](0x6, 0x2))[_0x43c8d1(0xd)](0x2) == _0x76ele8[0x4][_0x43c8d1(0xd)](0x6, 0x1)[_0x43c8d1('f')]() * _0x76ele8[0x4][_0x43
        return o && _0x76ele8[0x4][_0x43c8d1(0xd)](0x4, 0x1) == 0x2 && _0x76ele8[0x4][_0x43c8d1(0xd)](0x6, 0x2) == _0x5a6d56(_0x76ele8[0x4][_0x43c8d1(0xd)](0x7, 0x1), 0x
    } catch (_0x4cbb89) {
        console['log']('gg');
        return ![];
    }
}
```

代码里主要有几点混淆：

- *1、变量名替换，a --> _0xd4b7d6，这种东西最烦，但是也最简单，批量替换，在我看来即使abcd这种变量也比这个容易读*
- *2、提取了所有的方法到一个数组，这种也简单，只要在chrome中逐步调试替换就可以了。*

还有一些小的细节，很常见，没什么可说的"s".length() --> "s"['length']最终代码可以优化到这个地步，基本已经可读了，下一步就是分析代码了。

```
function check(flag){
    var _ = ['\x63\x6f\x64\x65', '\x76\x65\x72\x73\x69\x6f\x6e', '\x65\x72\x72\x6f\x72', '\x64\x6f\x77\x6e\x6c\x6f\x61\x64', '\x69\x6e\x76\x61\x6c\x69\x64\x4d\x6f\x6e\x6
    var head = flag['substring'](0, 4);
    var base = parseInt(btoa(head), 0x20); //344800


    (function (b, c) {
        var d = function (a) {
                while (--a) {
                    b['push'](b['shift']())
                }
            };
        d(++c);
    }(_, base%123));

    var g = function (a) {
            var a = parseInt(a, 0x10);
            var c = _[a];
            return c;
        };

    var s2h = function(str){
        var result = "0x";
        for(var i=0;i<str['length'];i++){
            result += str['charCodeAt'](i)['toString'](16)
        }
        return result;
    }

    var b = flag['split']("_");
    var c = (s2h(b[0]['substr'](-2,2)) ^ s2h(b[0]['substr'](4,1))) % b[0]['length'] == 5;
    if(!c){
        return false;
    }

    b2c = function(s) {
    var alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ234567";

    var parts = [];
    var quanta = Math.floor((s['length'] / 5));
    var leftover = s['length'] % 5;

    if (leftover != 0) {
        for (var i = 0; i < (5 - leftover); i++) {
            s += '\x00';
        }
        quanta += 1;
    }

    for (i = 0; i < quanta; i++) {
        parts.push(alphabet.charAt(s['charCodeAt'](i * 5) >> 3));
        parts.push(alphabet.charAt(((s['charCodeAt'](i * 5) & 0x07) << 2) | (s['charCodeAt'](i * 5 + 1) >> 6)));
        parts.push(alphabet.charAt(((s['charCodeAt'](i * 5 + 1) & 0x3F) >> 1)));
        parts.push(alphabet.charAt(((s['charCodeAt'](i * 5 + 1) & 0x01) << 4) | (s['charCodeAt'](i * 5 + 2) >> 4)));
        parts.push(alphabet.charAt(((s['charCodeAt'](i * 5 + 2) & 0x0F) << 1) | (s['charCodeAt'](i * 5 + 3) >> 7)));
        parts.push(alphabet.charAt(((s['charCodeAt'](i * 5 + 3) & 0x7F) >> 2)));
        parts.push(alphabet.charAt(((s['charCodeAt'](i * 5 + 3) & 0x03) << 3) | (s['charCodeAt'](i * 5 + 4) >> 5)));
        parts.push(alphabet.charAt(((s['charCodeAt'](i * 5 + 4) & 0x1F))));
    }

    var replace = 0;
    if (leftover == 1)
        replace = 6;
    else if (leftover == 2)
        replace = 4;
    else if (leftover == 3)
        replace = 3;
    else if (leftover == 4)
        replace = 1;

    for (i = 0; i < replace; i++)
        parts.pop();
    for (i = 0; i < replace; i++)
        parts.push("=");

    return parts.join("");
    }
```

```

    e = s2h(b2c(b[2])['split']("=")[0])^0x53a3f32
    if(e != 0x4b7c0a73){
        return false;
    }

    f = s2h(b2c(b[3])['split']("=")[0]) ^ e;
    if(f != 0x4315332){
        return false;
    }

    n = f*e*b[0]['length'];

    h = function(str, func){
        var result = "";
        for(var i=0;i<str['length'];i++){
            result += func(str[i])
        }
        return result;
    }

    j = b[1]['split']("3");
    if(j[0]['length'] != j[1]['length'] || (s2h(j[0])^s2h(j[1])) != 0x1613){
        return false;
    }

    k = str => str['charCodeAt']()*b[1]['length'];

    l = h(j[0],k);
    if(l!=0x2f9b5072){
        return false;
    }

    m = s2h(b[4]['substr'](0,4))-0x48a05362 == n%l;

    function u(str, j){
        var result = "";
        for(var i=0;i<j;i++){
            result += str;
        }
        return result;
    }

    if(!m || u(b[4]['substr'](5,1),2) == b[4]['substr'](-5,4) || (b[4]['substr'](-2,1) - b[4]['substr'](4,1)) != 1){
        return false
    }

    o = s2h(b[4]['substr'](6,2))['substr'](2) == b[4]['substr'](6,1)['charCodeAt']()*b[4]['length']*5;

    return o && b[4]['substr'](4,1) == 2 && b[4]['substr'](6,2) == u(b[4]['substr'](7,1),2);
}
```

思路分析

剩下的代码已经没什么可说的了。

1、首先是确认flag前缀，然后按照_分割为5部分。 2、g函数对基础数组做了一些处理，已经没什么懂了。 3、s2h是字符串到hex的转化函数 4、第一部分的验证不完整，导致严重的多解，只能通过爆破是否符合sha256来解决。 5、后面引入的b2c函数很简单，测试就能发现是一个base32函数。 6、第三部分和第四部分最简单，异或可得 7、h函数会对输入的字符串每位做func函数处理，然后拼接起来。 8、第二部分由3分割，左右两边长度相等，同样可以推算出结果。 9、k是我专门加入的es6语法的箭头语法，对传入的每个字母做乘7操作。 10、最后一题通过简单的判断，可以确定最后一部分的前四位。 11、u函数返回指定字符串的指定前几位 12、剩下的就是一连串的条件: 13、首先是一些很关键的的重复位，由于我写错了一些东西，导致这里永远是false，后被迫给出这几位.`!m || u(b[4]'substr',2) == b[4]'substr' || (b[4]'substr' - b[4]'substr') != 1` 14、最后一部分是集合长度、以及部分条件完成的，看上去存在多解，但事实上是能逆向出来结果的。 当我们都完成这部分的时候，flag就会被我们解出来了。

Flag

```
1 无
```


Challenge # Web # 2017 # HCTF
boring website
poker