akamai 混淆解密 _

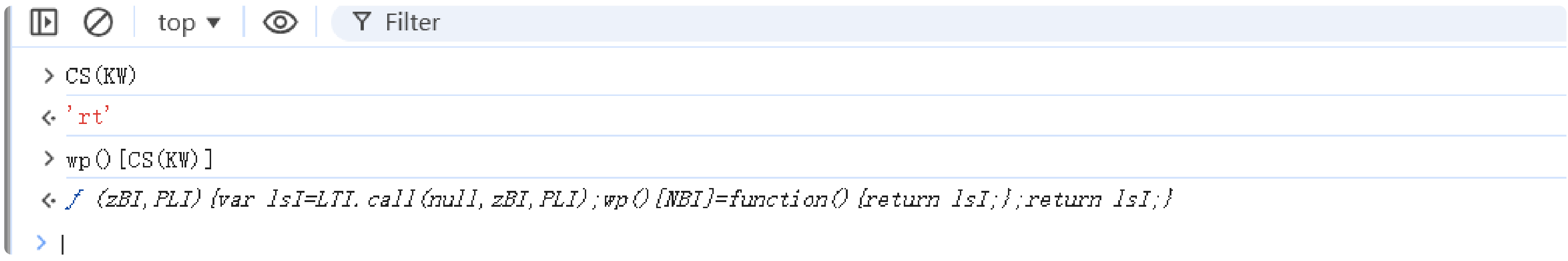
2025年1月8日 下午

9.9k 字　83 分钟

akamai 混淆是怎么运行的

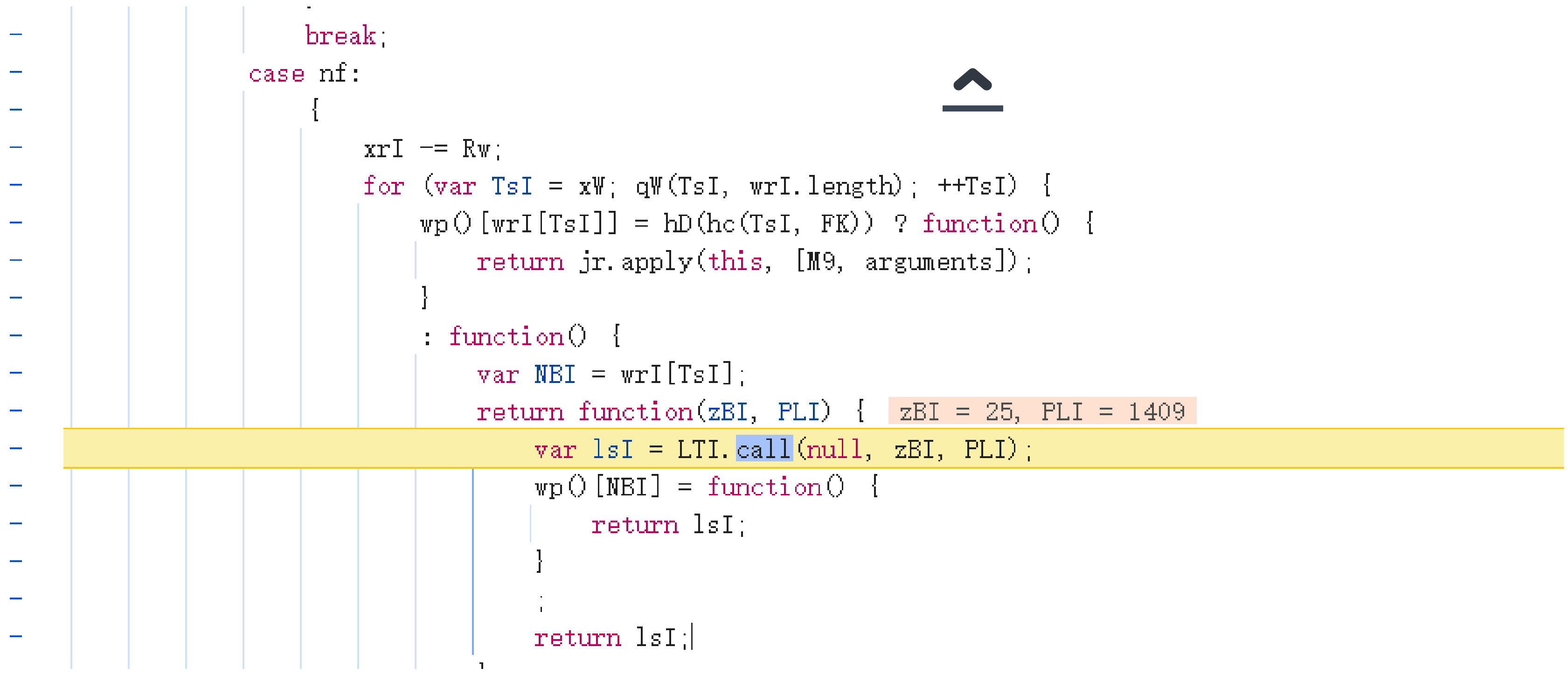
以 `'\x34': wp()[CS(KW)](SQ, zk)` 为例子， 这里 `CS(KW)` 为 `rt` `SQ` 值为 `25` ， `zk` 值为 `1409`

`wp()[CS(KW)](SQ, zk) => wp()['rt'](25,1409)`



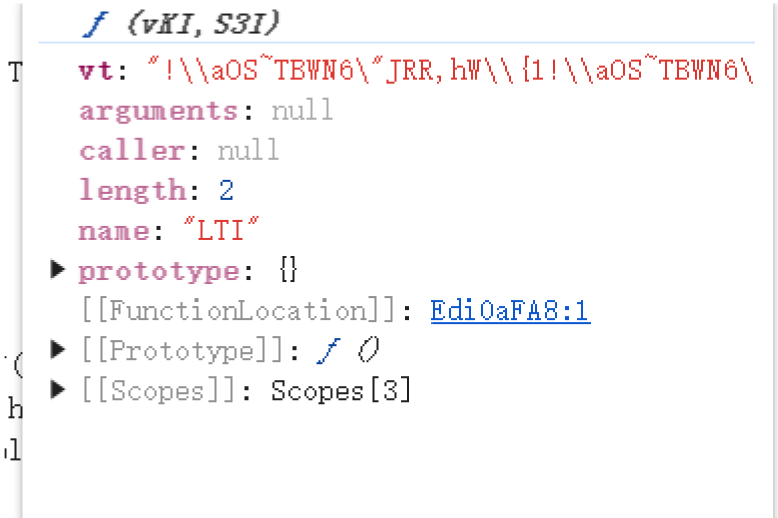
```
> CS(KW)
<· 'rt'
> wp()[CS(KW)]
<· ƒ (zBI,PLI){var IsI=LTI.call(null,zBI,PLI);wp()[NBI]=function(){return IsI;};return IsI;}
> |
```

image-20250102134823850



```
                    break;
                case nf:
                    {
                        xrI -= Rw;
                        for (var TsI = xW; qW(TsI, wrI.length); ++TsI) {
                            wp()[wrI[TsI]] = hD(hc(TsI, FK)) ? function() {
                                return jr.apply(this, [M9, arguments]);
                            }
                            : function() {
                                var NBI = wrI[TsI];
                                return function(zBI, PLI) {  zBI = 25, PLI = 1409
                                    var lsI = LTI.call(null, zBI, PLI);
                                    wp()[NBI] = function() {
                                        return lsI;
                                    }
                                    ;
                                    return lsI;
```

image-20250102134841352

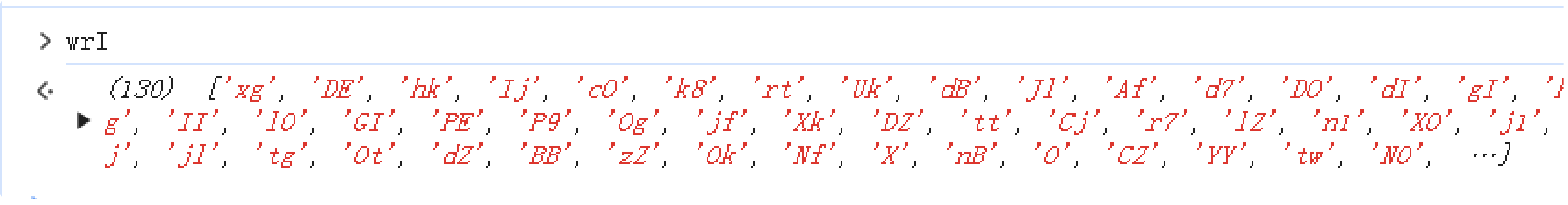


```
ƒ (vKI, S3I)
vt: "!\\aOS~TBWN6\"JRR,hW\\{1!\\aOS~TBWN6\
arguments: null
caller: null
length: 2
name: "LTI"
▶ prototype: {}
[[FunctionLocation]]: Edi0aFA8:1
▶ [[Prototype]]: ƒ ()
▶ [[Scopes]]: Scopes[3]
```

image-20250108160716740

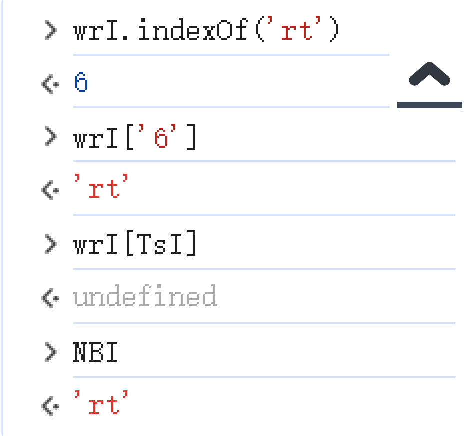
这里LTI有个 vt 属性，后面会用到

```javascript
xrI -= Rw;
for (var TsI = xW; qW(TsI, wrI.length); ++TsI) {
    wp()[wrI[TsI]] = hD(hc(TsI, FK)) ? function() {
        return jr.apply(this, [M9, arguments]);
    }
    : function() {
        var NBI = wrI[TsI];
        return function(zBI, PLI) {
            var lsI = LTI.call(null, zBI, PLI);
            wp()[NBI] = function() {
                return lsI;
            }
            ;
            return lsI;
        }
        ;
    }();
}
```



```
> wrI
<· (130) ['xg', 'DE', 'hk', 'Ij', 'cO', 'k8', 'rt', 'Uk', 'dB', 'Jl', 'Af', 'd7', 'DO', 'dI', 'gI', 'l
 ▶ g', 'II', 'lO', 'GI', 'PE', 'P9', 'Og', 'jf', 'Xk', 'DZ', 'tt', 'Cj', 'r7', 'lZ', 'nl', 'XO', 'jl',
   j', 'jl', 'tg', 'Ot', 'dZ', 'BB', 'zZ', 'Ok', 'Nf', 'X', 'nB', 'O', 'CZ', 'YY', 'tw', 'NO', …]
```

image-20250102135157188

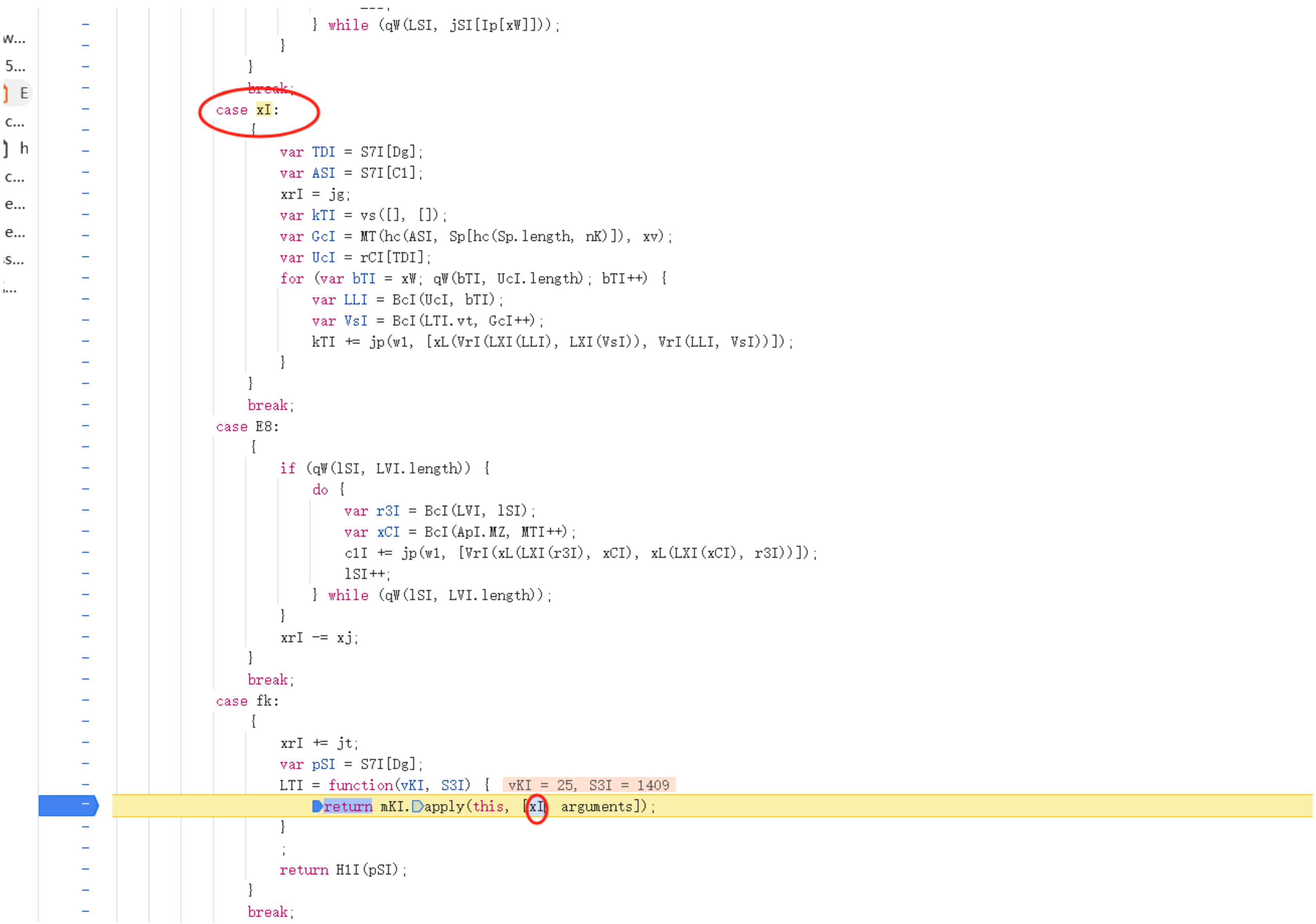


```
> wrI.indexOf('rt')
<· 6
> wrI['6']
<· 'rt'
> wrI[TsI]
<· undefined
> NBI
<· 'rt'
```

image-20250102140010380

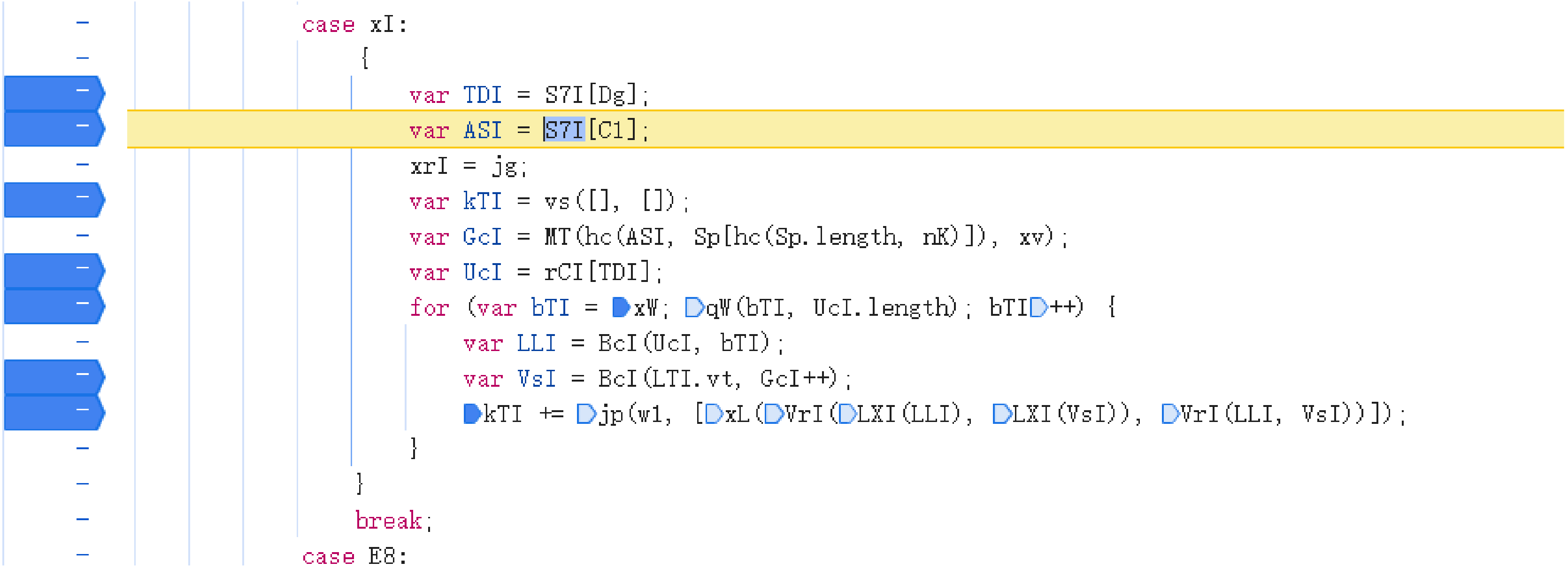
上面可以看到 `LTI.call(null, zBI, PLI);` 返回了结果之后赋值给了 lsI， `wp['rt']` 这个函数又被重新赋值成了 返回lsI 值的函数，也就是 `wp['rt']()` 运行的时候可以直接返回这个函数本来要返回值的，不用再传入实参。

往下跟栈，可以看到 `mKI` 这个函数 这里的XI 是一个控制数，



```
                        } while (qW(LSI, jSI[Ip[xW]]));
                    }
                }
                break;
            case xI:
                {
                    var TDI = S7I[Dg];
                    var ASI = S7I[C1];
                    xrI = jg;
                    var kTI = vs([], []);
                    var GcI = MT(hc(ASI, Sp[hc(Sp.length, nK)]), xv);
                    var UcI = rCI[TDI];
                    for (var bTI = xW; qW(bTI, UcI.length); bTI++) {
                        var LLI = BcI(UcI, bTI);
                        var VsI = BcI(LTI.vt, GcI++);
                        kTI += jp(w1, [xL(VrI(LXI(LLI), LXI(VsI)), VrI(LLI, VsI))]);
                    }
                }
                break;
            case E8:
                {
                    if (qW(lSI, LVI.length)) {
                        do {
                            var r3I = BcI(LVI, lSI);
                            var xCI = BcI(ApI.MZ, MTI++);
                            c1I += jp(w1, [VrI(xL(LXI(r3I), xCI), xL(LXI(xCI), r3I))]);
                            lSI++;
                        } while (qW(lSI, LVI.length));
                    }
                    xrI -= xj;
                }
                break;
            case fk:
                {
                    xrI += jt;
                    var pSI = S7I[Dg];
                    LTI = function(vKI, S3I) {  vKI = 25, S3I = 1409
                        return mKI.apply(this, [xI, arguments]);
                    }
                    ;
                    return H1I(pSI);
                }
                break;
```

image-20250108161141103

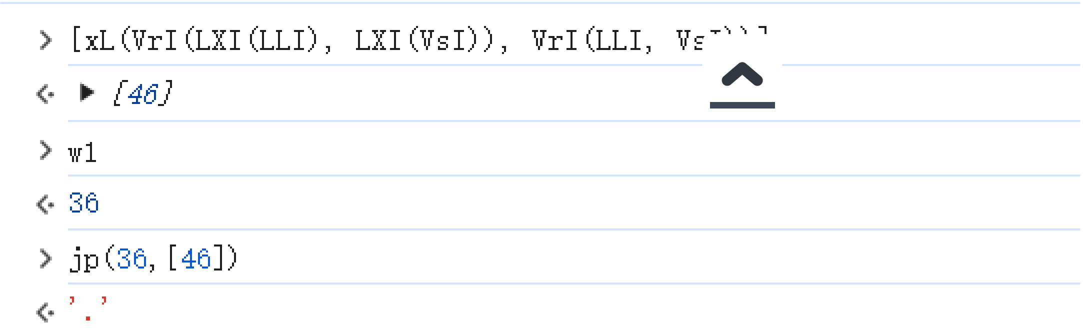


```
case xI:
    {
        var TDI = S7I[Dg];
        var ASI = S7I[C1];
        xrI = jg;
        var kTI = vs([], []);
        var GcI = MT(hc(ASI, Sp[hc(Sp.length, nK)]), xv);
        var UcI = rCI[TDI];
        for (var bTI = xW; qW(bTI, UcI.length); bTI++) {
            var LLI = BcI(UcI, bTI);
            var VsI = BcI(LTI.vt, GcI++);
            kTI += jp(w1, [xL(VrI(LXI(LLI), LXI(VsI)), VrI(LLI, VsI))]);
        }
    }
    break;
case E8:
```

image-20250102140549191

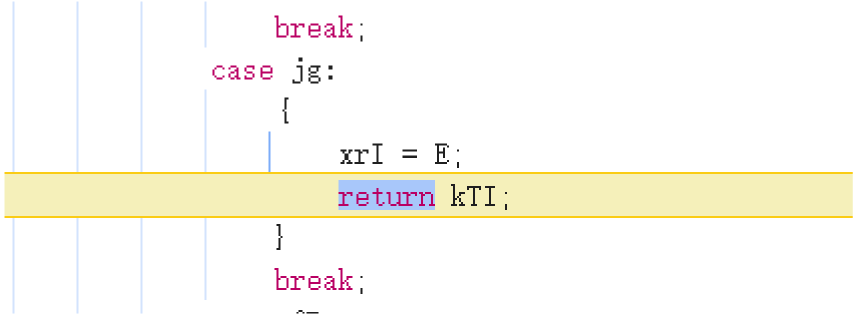
这里 S71 是mKi 的第二个参数，也就是 上面函数的的arguments 参数

往下面跟栈，可以看到这个参数值和这个最终运算出来的数据



```
> [xL(VrI(LXI(LLI), LXI(VsI)), VrI(LLI, Vs
<· ▶ [46]
> w1
<· 36
> jp(36,[46])
<· '.'
```

image-20250102141722155



```
            break;
        case jg:
            {
                xrI = E;
                return kTI;
            }
            break;
```

image-20250102142317383

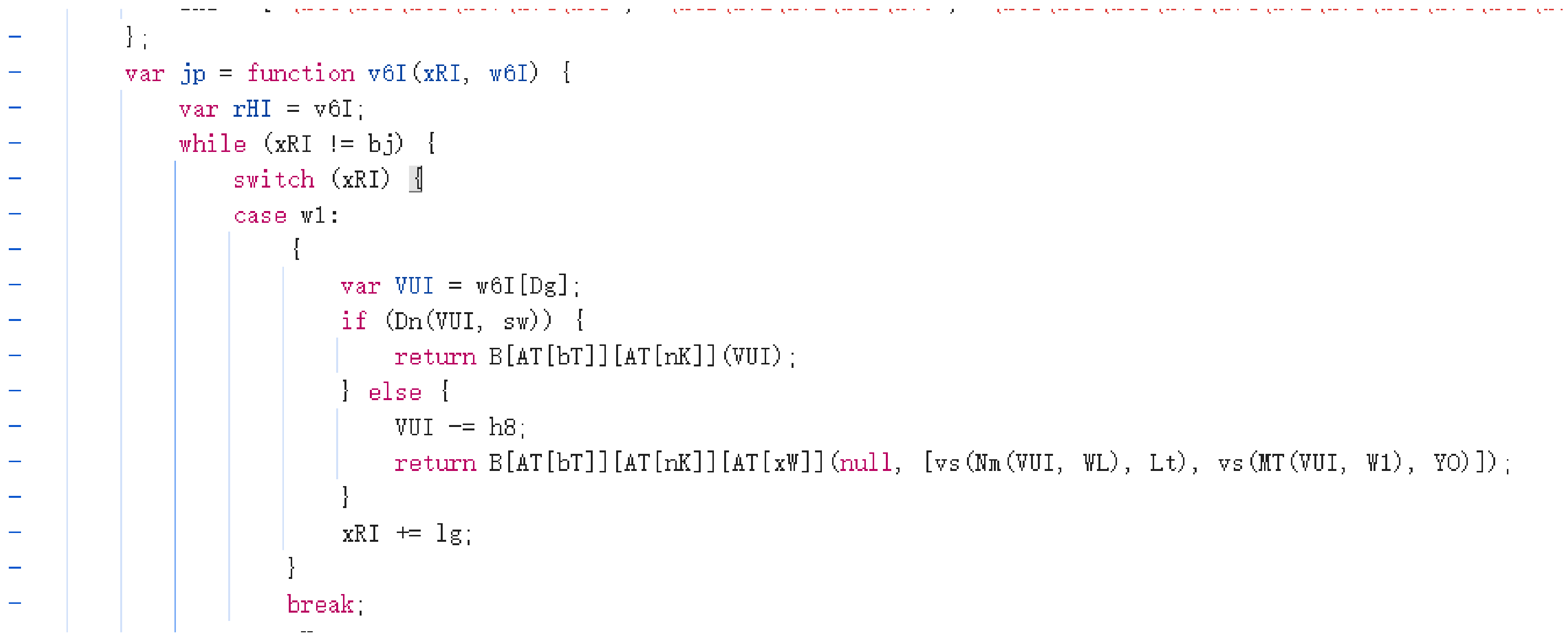
再往下跟栈可以看到返回了 `kTI` 这个值

到这里整理下这个运行逻辑

ZEPHIR

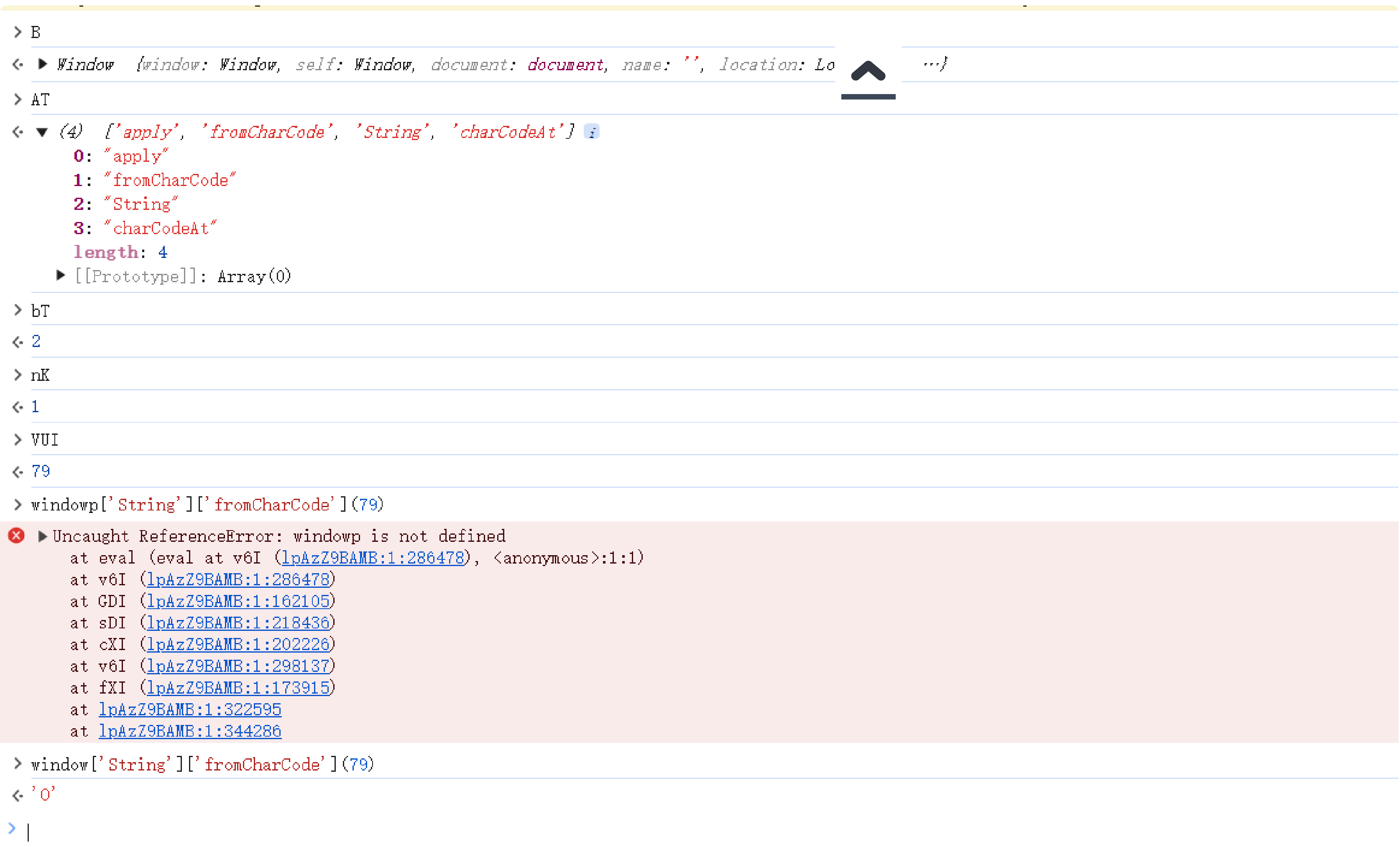
```
wp()['rt'](25,1409) =>
LTI.call(null, zBI, PLI) =>
LTI = function(vKI, S3I) {
       return mKI.apply(this, [xI, arguments]);
                   } =>
function mKI (){
  case xI:
     计算出kTI 的值
  case jp:
    return kTI
}
```

计算kTI 的值，最后用到了jp 函数



```
    };
    var jp = function v6I(xRI, w6I) {
        var rHI = v6I;
        while (xRI != bj) {
            switch (xRI) {
            case w1:
                {
                    var VUI = w6I[Dg];
                    if (Dn(VUI, sw)) {
                        return B[AT[bT]][AT[nK]](VUI);
                    } else {
                        VUI -= h8;
                        return B[AT[bT]][AT[nK]][AT[xW]](null, [vs(Nm(VUI, WL), Lt), vs(MT(VUI, W1), YO)]);
                    }
                    xRI += lg;
                }
                break;
```

image-20250102142620025



```
> B
<· ▶ Window  {window: Window, self: Window, document: document, name: '', location: Lo  …}
> AT
<· ▼ (4)  ['apply', 'fromCharCode', 'String', 'charCodeAt']
      0: "apply"
      1: "fromCharCode"
      2: "String"
      3: "charCodeAt"
      length: 4
    ▶ [[Prototype]]: Array(0)
> bT
<· 2
> nK
<· 1
> VUI
<· 79
> windowp['String']['fromCharCode'](79)
✖ ▶Uncaught ReferenceError: windowp is not defined
      at eval (eval at v6I (lpAzZ9BAMB:1:286478), <anonymous>:1:1)
      at v6I (lpAzZ9BAMB:1:286478)
      at GDI (lpAzZ9BAMB:1:162105)
      at sDI (lpAzZ9BAMB:1:218436)
      at cXI (lpAzZ9BAMB:1:202226)
      at v6I (lpAzZ9BAMB:1:298137)
      at fXI (lpAzZ9BAMB:1:173915)
      at lpAzZ9BAMB:1:322595
      at lpAzZ9BAMB:1:344286
> window['String']['fromCharCode'](79)
<· 'O'
> |
```

image-20250102171208026

函数jp 控制数w1 的函数逻辑为以下代码

```js
function jp(x) {
    return String.fromCharCode(x[0])
}
```

以下为整个加密的代码整理 `var rs = function mKI(xrI, S7I) {}` 第一个值为switch 控制数，不同的xrI 实际值会有不同的分支 S7I，为实际传入的参数

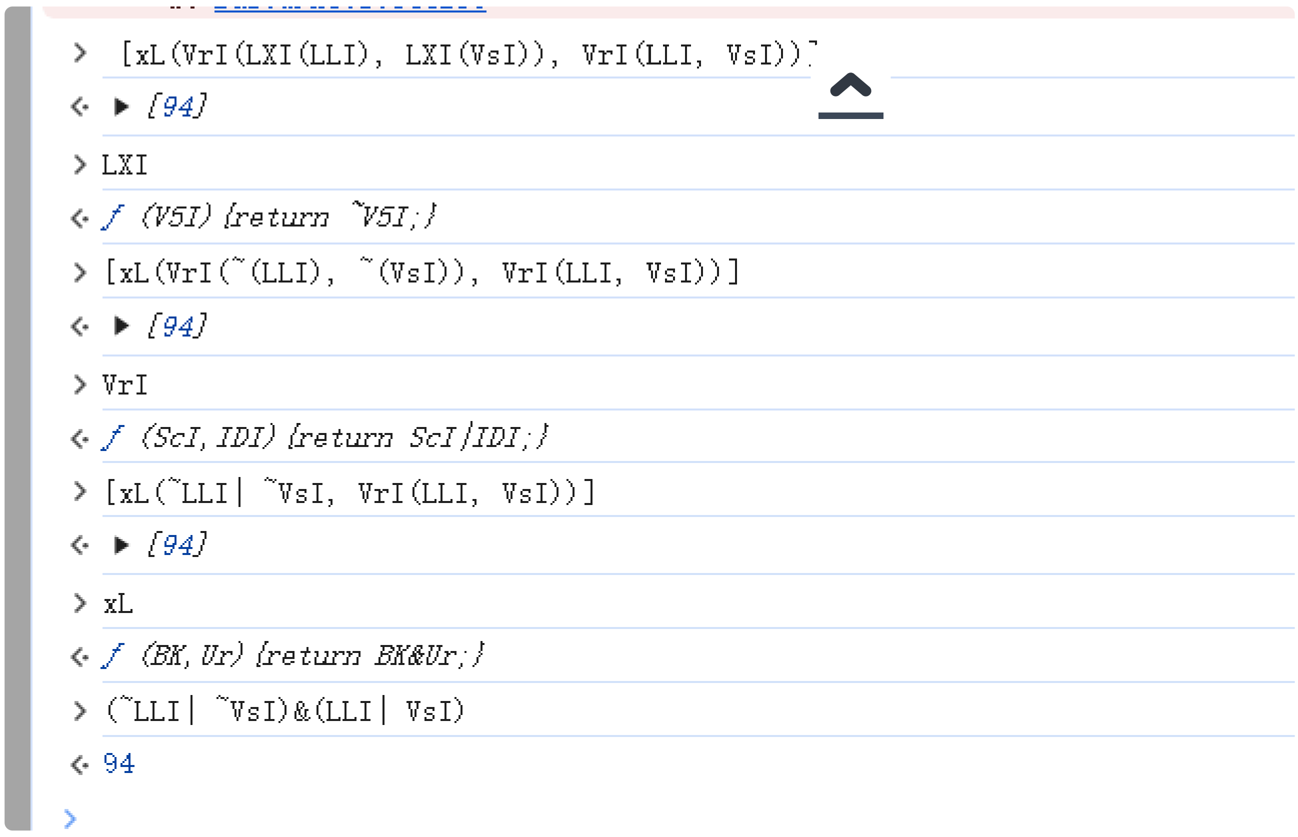
```js
var rs = function mKI(xrI, S7I) {
    switch (xrI) {
        case xI: {
            var TDI = S7I[Dg];
            var ASI = S7I[C1];
            xrI = jg;
            var kTI = vs([], []);
            var GcI = MT(hc(ASI, Sp[hc(Sp.length, nK)]), xv);
            var UcI = rCI[TDI];
            for (var bTI = xW; qW(bTI, UcI.length); bTI++) {
                var LLI = BcI(UcI, bTI);
                var VsI = BcI(LTI.vt, GcI++);
                kTI += jp(w1, [xL(VrI(LXI(LLI), LXI(VsI)), VrI(LLI, VsI))]);
            }
        }
            break;
        case jg: {
            xrI = E;
            return kTI;
        }
            break;

    }

}
```

精简 运算

`xL(VrI(LXI(LLI), LXI(VsI)), VrI(LLI, VsI))  =>  (~LLI| ~VsI)&(LLI| VsI)`



```
> [xL(VrI(LXI(LLI), LXI(VsI)), VrI(LLI, VsI))]
<· ▶ [94]
> LXI
<· ƒ (V5I) {return ~V5I;}
> [xL(VrI(~(LLI), ~(VsI)), VrI(LLI, VsI))]
<· ▶ [94]
> VrI
<· ƒ (ScI, IDI) {return ScI|IDI;}
> [xL(~LLI| ~VsI, VrI(LLI, VsI))]
<· ▶ [94]
> xL
<· ƒ (BK, Ur) {return BK&Ur;}
> (~LLI| ~VsI)&(LLI| VsI)
<· 94
>
```

image-20250108164453808

JAVASCRIPT

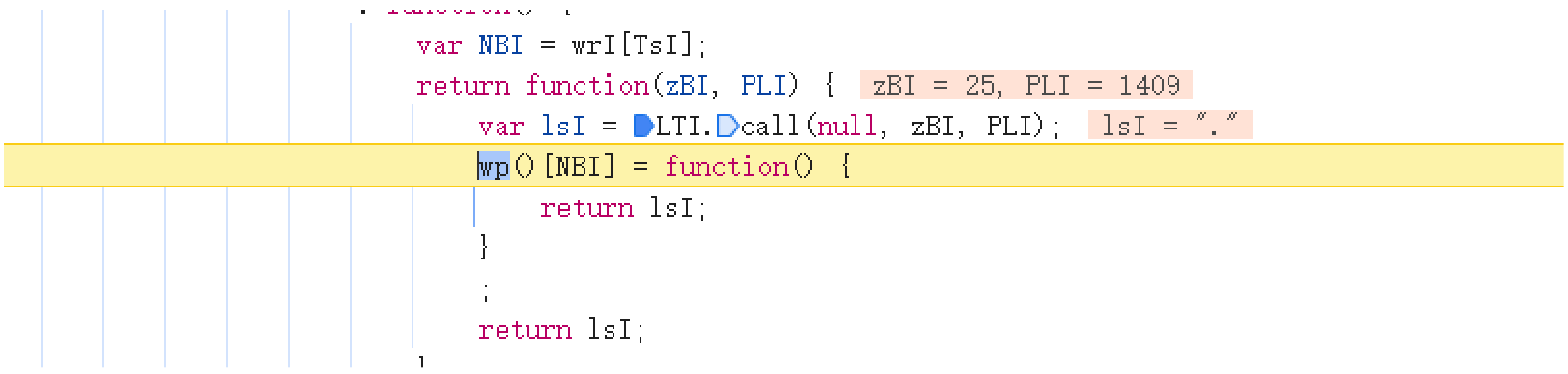
```javascript

function jp(x) {
    return String.fromCharCode(x[0])
}

var DcI = function() {
    return ["< \n\'", ",68_E+&=^82.XR/\b =\r", "[M\x3f!7A\x07!9", ">&lt;61,>;[}/$3\x406(", "-7&k\t:9\vPE/", "F
};
rCI = DcI()
let LTI = {}
LTI.vt = '!\\aOS~TBWN6"JRR,hW\\{1!\\aOS~TBWN6"JRR,hW\\{1!\\aOS~TBWN6"JRR,hW\\{1!\\aOS~TBWN6"JRR,hW\\{1'
var BcI = function (KVI, hVI) {
    return KVI['charCodeAt'](hVI);
};


function dec(S7I) {
    var TDI = S7I[0];
    var ASI = S7I[1];
    var kTI = ''
    var GcI = (ASI - 992) % 21
    var UcI = rCI[TDI];
    for (var bTI = 0; bTI <UcI.length; bTI++) {
        var LLI = BcI(UcI, bTI);
        var VsI = BcI(LTI.vt, GcI++);
        kTI += jp( [(~LLI| ~VsI)&(LLI| VsI)]);
    }
    return kTI;
}


console.log(dec([25, 1409]));
// 输出结果为 "."
```



```
            var NBI = wrI[TsI];
            return function(zBI, PLI) {  zBI = 25, PLI = 1409
                var lsI = LTI.call(null, zBI, PLI);  lsI = "."
                wp()[NBI] = function() {
                    return lsI;
                }
                ;
                return lsI;
```

image-20250108164958314

只要带入 参数的实际值，就可以计算出所有 `wp()` 开头的混淆，

以下代码 `wp()[CS(KW)](25, 1409)` 和 `LTI.call(null, 25, 1409)` 实际值相等。

其他如 `Yr()` 等函数的解密类似，不过会有点不一样，除了列表和解密的key 之外，还可能会有类似的这种结构里 `var GcI = (ASI - 992) % 21` %之后的数字不一样，或者是jp 函数传入参数前的运算逻辑不一

以下是一个函数的解密

```js
let vCI = {U7: "3YTotqYT{ |C6G3$JC3YTotqYT{ |C6G3$JC3YTotqYT{ |C6G3$JC3YTotqYT{ |C6G3$JC3YTotqYT{ |C6G3$JC3YTot

function BcI(KVI, hVI) {
    return KVI.charCodeAt(hVI);
}

function jp(x) {
    return String.fromCharCode(x);
}

let RCI = ["\t&\x00��*1", "8�A�*X ", "7VV,,A45���", ",%\x40", "P6:�\x00�,7�O�", "�O\b", "G*0", "4GV#3c+;�\r7+;�


function meth(WVI) {
    let hBI = WVI[1], G1I = WVI[2], dDI = '', VcI = (hBI - 379) % 18;
    let MVI = RCI[G1I], gBI = 0;
    while (gBI < MVI.length) {
        let mCI = BcI(MVI, gBI), RTI = BcI(vCI.U7, VcI++);
        dDI += jp(~(mCI & RTI) & (mCI | RTI));
        gBI++;
    }
    return dDI;
}

console.log(meth([53, 990, 100])); // un
console.log(meth([53, 637, 43])); // pass
console.log(meth([53, 680, 81])); // secret
console.log(meth([0, 680, 81])); // secret

```

#js逆向 #akamai

akamai 混淆解密
https://kingjem.github.io/2025/01/08/逆向/akamai 混淆解密/

作者	发布于	许可协议
Ruhai	2025年1月8日	ⓘ

NC 发送HTTP 报文请求 ❯