Ty2y.com 混淆加密配置说明

局部变量名混淆

例程:

Before

```
function demo(){
  var age=99;
}
```

After

```
function demo(){
  var _0xk$r=99;
}
```

全局变量名混淆

例程:

Before

```
var age=99;
function demo(){
  var age = 99;
}
```

After

```
var _0xk$r=99;
function demo(){
  var age = 99;
}
```

局部函数名混淆

例程:

Before

```
function demo(){
  var age = 99;
  function demo_sub(){
  }
}
```

After

```
function demo(){
  var age = 99;
```

```
    function _0x62a87c() {
    }
}
```

全局函数名混淆

例程：

Before

```
function demo() {
  var age = 99;
  function demo_sub() {
  }
}
```

After

```
function _0x62ab7d() {
  var age = 99;
  function demo_sub() {
  }
}
```

成员函数加密

如对 console.log()的 log 函数加密。

例程：

Before

```
console.log("demo");
```

After

```
console['\x6c\x6f\x67']("demo");
```

数值常量加密

将数值常量变为运算表达式。

例程：

Before

```
var num = 123;
```

After

```
var num = 683517 ^ 683398;
```

二进制表达式混淆

将二进制表达式变形为函数调用表达式。

例程：

Before

var num = 683517 ^ 683398;

After

var num = function (s, h) {
  return s ^ h;
}(683517, 683398);

布尔型数值加密

例程：

Before

var done = true;

After

var done = !![];

JSON 数据加密

注意：需同时启用“字符串阵列化”和“阵列化加密”。

例程：

Before

var man = {"name":"tim","age":18};

After

var
_0xeb6d9b=["114.3.41.41.43.103.104.100.108.43.51.41.43.125.96.100.43.37.3.41.41.43.104.110.108.43.51.41.56.49.3.116."];function _0xf72b(str,dy_key){dy_key=9;var i,k,str2="";k=str.split(".");for(i=0;i<k.length-1;i++){str2+=string.fromcharcode(k[i]^dy_key);}return str2;}var="" man="<span" style="text-shadow: 1px 0px 1px #666666; font-weight:600; opacity:0.8; font-size:10px;">JSON.parse(_0xf72b(_0xeb6d9b[0]));

正则表达式加密

注意：需同时启用“字符串阵列化”和“阵列化加密”。

例程：

Before

var r =/regexp test/g;

After

var
_0x796d=["123.108.110.108.113.121.41.125.108.122.125.","110."];function _0xcca(str,dy_key){dy_key=9;var i,k,str2="";k=str.split(".");for(i=0;i<k.length-1;i++){str2+=Stri

```
ng.fromCharCode(k[i]^dy_key);}return str2;}var r=new
RegExp(_0xcca(_0x796d[0]),_0xcca(_0x796d[1]));
```

字符串 Unicode 化加密

例程:

Before

```
var obf = "JShaman JavaScrpt Obfuscator";
```

After

```
var obf =
"\u004a\u0053\u0068\u0061\u006d\u0061\u006e\u0020\u004a\u0061\u0076\u0061\u0053\u0063\u0072\u0070\u0074\u0020\u004f\u0062\u0066\u0075\u0073\u0063\u0061\u007
4\u006f\u0072";
```

赋值花指令

对赋值语句右侧的内容，如字符串、数值等，进行花指令处理。

例程:

Before

```
var name;
name = "jack";
```

After

```
var name;
name = function () {
  return "jack";
}();
```

僵尸代码植入

在代码中随机插入僵尸代码，增加代码理解难度。

例程:

Before

```
var a=1;
var b=2;
```

After

```
var _0x;
var a = 1;
_0x = "jfci";
var b = 2;
```

Eval 加密

对特定的语句进行 Eval 加密

Before

```
var a = 1 + 2;
```

After

```
var a = eval(String.fromCharCode(49, 32, 43, 32, 50));
```

平展控制流

将函数中代码平坦化，并打乱代码显示顺序。

例程：

Before

```
function demo(){
  var name = "tom";
  var age = "18";
  return name + age;
}
```

After

```
function demo() {
  var _array = "1|0|2".split("|"),
  _index = 0;
  while (!![]) {
    switch (+_array[_index++]) {
    case 0:
      var age = "18";
      continue;
    case 1:
      var name = "tom";
      continue;
    case 2:      return name + age;
      continue;
    }
    break;
  }
}
```

收缩控制流

将函数中符合条件的多行代码收缩为单行，形成逗号运算符语法。

Before

```
function demo(){
  var name = "tom";
  var age = "18";
  return name + age;
}
```

After

```
function demo(name, age) {
    return age = (name = "tom", "18"), name + age;
}
```

字符串阵列化

将代码中包含的字字符串集中放置到数组。

例程：

Before

```
function demo(){
  var name = "tom";
  var age = "18";
  return name + age;
}
```

After

```
var _0x312g = ["tom", "18"];
function demo() {
  var name = _0x312g[0];
  var age = _0x312g[1];
  return name + age;
}
```

阵列字符串加密

将阵列中的字符串内容进行加密，使用此选项时，会强制启用字符串阵列化。

例程：

Before

```
function demo(){
  var name = "tom";
  var age = "18";
  return name + age;
}
```

After

```
var _0x=['125.102.100.','56.49.'];
function _0xa5bdc(str,dy_key){dy_key=9;var i,k,str2='';k=str.split('.');for(i=0;i<k.length-1;i++){str2+=String.fromCharCode(k[i]^dy_key);}return str2;}
function demo() {
  var name = _0xa5bdc(_0x[0]);
  var age = _0xa5bdc(_0x[1]);
  return name + age;
}
```

虚拟机执行保护

将某些代码转为虚拟机 OP 指令，在虚拟机中执行。

例程：

Before

```
var num = 1+2;
```

After

```
function _0xbd18dc(vm_opcode){var op={push:32,add:33,sub:34,mul:35,div:36,pop:37,xor:38};var stack=[];var ip=-1;var
sp=-1;while(ip<vm_opcode.length){ip++;switch(vm_opcode[ip]){case op.push:{ip++;stack.push(vm_opcode[ip]);sp++;break;}case op.add:{var op_1=stack[sp-1];var
op_2=stack[sp];var value=op_1+op_2;stack.push(value);sp++;break;}case op.sub:{var op_1=stack[sp-1];var op_2=stack[sp];var
value=op_1-op_2;stack.push(value);sp++;break;}case op.mul:{var op_1=stack[sp-1];var op_2=stack[sp];var value=op_1*op_2;stack.push(value);sp++;break;}case
op.div:{var op_1=stack[sp-1];var op_2=stack[sp];var value=op_1/op_2;stack.push(value);sp++;break;}case op.xor:{var op_1=stack[sp-1];var op_2=stack[sp];var
value=op_1^op_2;stack.push(value);sp++;break;}case op.pop:{return stack[sp];}}}}var num=_0xbd18dc([32,1,32,2,33,37]);
```

AST 执行保护

将某些代码转为 AST，即：抽象语法树，代码运行时，直接执行此 AST。

例程：

Before

```
console.log("hello");
```

After

```
var
visitors={File(node,scope){ast_excute(node.program,scope);},Program(program,scope){for(i=0;i<program.body.length;i++){ast_excute(program.body[i],scope);}},E
xpressionStatement(node,scope){return ast_excute(node.expression,scope);},CallExpression(node,scope){var func=ast_excute(node.callee,scope);var
args=node.arguments.map(function(arg){return ast_excute(arg,scope);});var
value;if(node.callee.type==='MemberExpression'){value=ast_excute(node.callee.object,scope);}return func.apply(value,args);},MemberExpression(node,scope){var
obj=ast_excute(node.object,scope);var name=node.property.name;return obj[name];},Identifier(node,scope){return scope[node.name];},StringLiteral(node){return
node.value;},NumericLiteral(node){return node.value;}};function ast_excute(node,scope){var evalute=visitors[node.type];if(!evalute){throw new Error("Unknown
AST type:",node.type);}return
evalute(node,scope);}ast_excute({"type":"CallExpression","callee":{"type":"MemberExpression","object":{"type":"Identifier","name":"console"},"property":{"ty
pe":"Identifier","name":"log"}},"arguments":[{"type":"StringLiteral","value":"hello"}]},{console:console});;
```

保留注释： 保留代码中的注释。

代码压缩： 去除回车换行、空格，压缩代码体积。

保留关键字： 对指定的变量、变量名、函数名不进行加密。