XCTF Triangle writeup(js 代码逆向)

原创

gonganDV 于 2019-07-17 11:13:08 发布 2280 收藏 1
分类专栏： 学习笔记 xctf 文章标签： xctf-wp


学习笔记 同时被 2 个专栏收录
10 篇文章 0 订阅
订阅专栏


xctf
2 篇文章 0 订阅
订阅专栏

主要参考链接：https://lim1ts.github.io/ctf/2017/10/19/hackluTriangles.html

其他参考链接：https://wu.rot26.team/CTF/Hacklu/2017/web/triangle/

https://www.vhn.vn/blog/index.php/2017/10/19/hack-lu-2017/

过程：test_pw(enc_pw(userInput), get_pw())

get_pw()返回值固定主要逆向test_pw(，)和enc_pw(userInput)得到正确的userinput

	Host	Method	URL
0	http://111.198.29.45:45054	GET	/
2	http://111.198.29.45:45054	GET	/unicorn.js
3	http://111.198.29.45:45054	GET	/secret.js
4	http://111.198.29.45:45054	GET	/util.js

输入框函数：

```
function login(){
	var input = document.getElementById('password').value;
	var enc = enc_pw(input);
	var pw = get_pw();
	if(test_pw(enc, pw) == 1){
		alert('Well done!');
	}
	else{
		alert('Try again ...');
	}
}
```

Secret.js

```
function test_pw(e,_){
    var t=stoh(atob(getBase64Image("eye"))),
        r=4096,
        m=8192,
        R=12288,
    a=new uc.Unicorn(uc.ARCH_ARM,uc.MODE_ARM);
    a.reg_write_i32(uc.ARM_REG_R9,m),a.reg_write_i32(uc.ARM_REG_R10,R),a.reg_write_i32(uc
    for(var o=0;o<o1.length;o++)a.mem_write(r+o,[t[o1[o]]]);
    a.mem_map(m,4096,uc.PROT_ALL),a.mem_write(m,stoh(_)),a.mem_map(R,4096,uc.PROT_ALL),a.
    var u=r,
        c=r+o1.length;
    return a.emu_start(u,c,0,0),
    a.reg_read_i32(uc.ARM_REG_R5)
}
function enc_pw(e){
    var _=stoh(atob(getBase64Image("frei"))),
        t=4096,
        r=8192,
        m=12288,
        R=new uc.Unicorn(uc.ARCH_ARM,uc.MODE_ARM);R.reg_write_i32(uc.ARM_REG_R8,r),R.reg_
        for(var a=0;a<o2.length;a++)R.mem_write(t+a,[_[o2[a]]]);
        R.mem_map(r,4096,uc.PROT_ALL),R.mem_write(r,stoh(e)),R.mem_map(m,4096,uc.PROT_ALL
        var o=t,
            u=t+o2.length;
        return R.emu_start(o,u,0,0),htos(R.mem_read(m,e.length))
}
function get_pw(){
    for(var e=stoh(atob(getBase64Image("templar"))),_="",t=0;t<o3.length;t++)_+=String.fr
    return _
}
```


观察get_pw()函数发现返回固定值，控制台输出

"XYzaSAAX_PBssisodjsal_sSUVWZYYYb"

观察enc_pw()函数发现写入内存指令在于_[o2[a]],与用户输入无关，需要还原写入的内存指令，了解字符串处理过程以期逆向出正确输入的字符串

为了直接在控制台以16进制的形式输出写入内存的信息，模仿enc_pw()函数构造getARM1()函数、和将10进制转换为16进制的函数toHexString（）

function getARM1(){

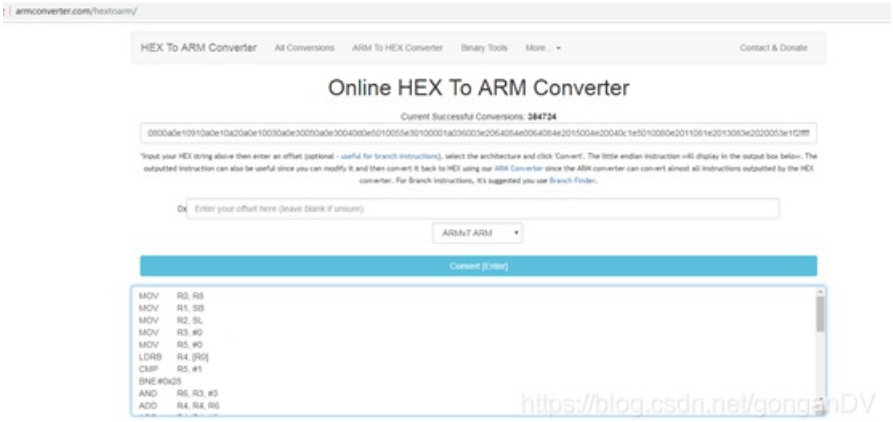
```
  var x = stoh(atob(getBase64Image("frei")));
  var output = new Array();
  for(var i = 0; i < o2.length ; i++){
    output[i] = x[o2[i]];
  }
  return output;
}

//返回值为整数需要转化为16进制

function toHexString(byteArray) {
  return Array.from(byteArray, function(byte) {
    return ('0' + (byte & 0xFF).toString(16)).slice(-2);
  }).join('')
}
```

在控制台运行toHexString(getARM1())

得到：

0800a0e10910a0e10a20a0e10030a0e30050a0e30040d0e5010055e30100001a036003e2064084e0064084e2

将得到的16进制数据转换为arm指令，在线转换网址：http://armconverter.com/hextoarm/



MOV R0, R8 ； R0 = 8192，这是输入密码的地址

MOV R1, SB ；SB是静态基址寄存器，R1=R9=m(从这获取最后的输出结果，即输出结果的地址)

MOV R2, SL ；SL是堆栈限制寄存器，R2=R10=e(输入密码的长度)

MOV R3, #0 ；R3是计数器

MOV R5, #0 ；R5储存输入密码的上一个地址位数据的奇偶性

LDRB R4, [R0] ；此处是0x14，将寄存器数值传入R4

CMP R5, #1 ；将R5与1相减，结果存在标志位中

BNE #0x28 ；根据标志位的结果，判断R5与1是否相等，若不相等则跳转到0x28处

AND R6, R3, #3 ；将R3和3相与结果传入R6，相当于截取R3二进制最后两位传入R6

ADD R4, R4, R6 ；将R4 与R6相加的结果传入R4

ADD R4, R4, #6 ；此处是0x28，R4加6

AND R5, R4, #1 ；将R4和1相与的结果传入R5，若R4为偶数则R5=0反之R5=1

STRB R4, [R1] ；将R4的低8位传入以R1为基址的存储器地址中

ADD R0, R0, #1 ；R0加一

ADD R1, R1, #1 ；R1加一

ADD R3, R3, #1 ；R3加一，R3是计数器

CMP R3, R2 ；将R3与R2相减，结果存在标志位中

BLT #0x14 ；根据标志位的结果判断R3是否小于R2若小于则跳转到0x14处，即若计数器小于输入密码长度则继续循环

MOV R0, #0

MOV R1, #0

MOV R2, #0

MOV R3, #0

MOV R4, #0

MOV R5, #0

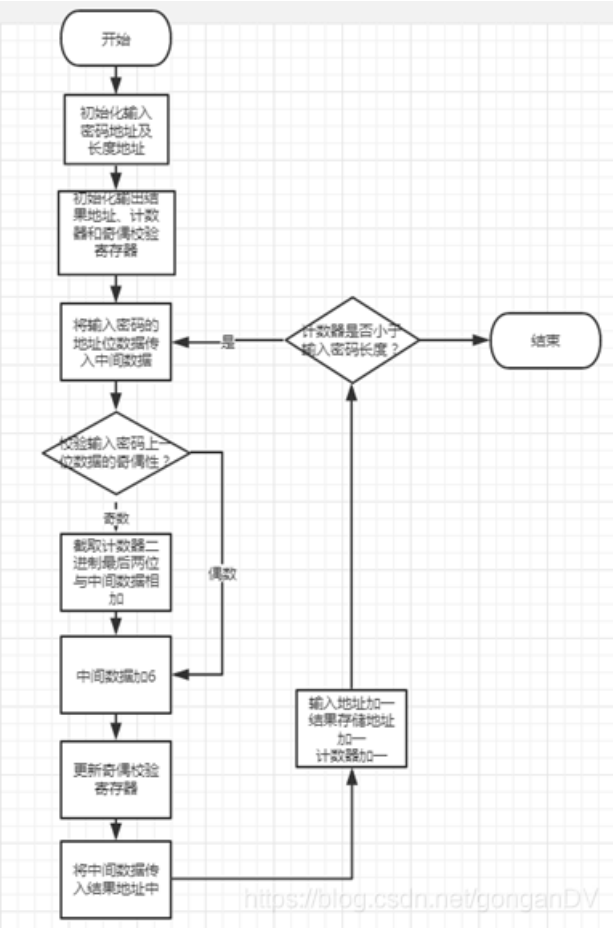
MOV R6, #0

MOV R7, #0

MOV SB, #0

MOV SL, #0

流程图如下（核心流程类似于移位密码加密的过程）：



为将test_pw()函数写入内存的指令输出，类似getARM1()函数编写getARM2()，用toHexString(getARM2())，再将16进制结果转换成arm指令

```
» function getARM2(){
    var x = stoh(atob(getBase64Image("eye")));
    var output = new Array();
    for(var i = 0; i < 01.length ; i++){
      output[i] = x[01[i]];
    }
    return output;
  }
← undefined
» toHexString(getARM2())
← "0900a0e10a10a0e10830a0e10040a0e30050a0e300c0a0e30020d0e50060d1e5056086e201c004e200005ce30000000a036046e2060052e10500001a010080e2011081e2
0140B4e2030054e1f1fffba0150a0e30000a0e30010a0e30020a0e30030a0e30040a0e30060a0e30070a0e30080a0e30090a0e300a0a0e300c0a0e3"
```

0900a0e10a10a0e10830a0e10040a0e30050a0e300c0a0e30020d0e50060d1e5056086e201c004e200005ce3(

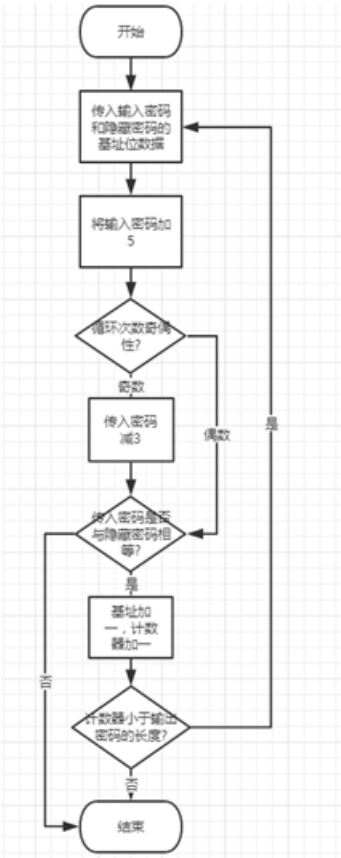

```
MOV     R0, SB          ；SB是静态基址寄存器,R0=R9=m,这是隐藏的密码（get_pw()的返回值）的头地址
MOV     R1, SL          ；SL是堆栈限制寄存器,R1=R10=R,这是输入的密码的头地址
MOV     R3, R8          ; R8是输入密码的长度
MOV     R4, #0
MOV     R5, #0
MOV     IP, #0
LDRB    R2, [R0]        ；此处是0x18 传入隐藏密码
LDRB    R6, [R1]        ;传入输入密码
ADD     R6, R6, #5      ；将R6加5的结果传入R6
AND     IP, R4, #1      ;将R4与1相与的结果传入IP
CMP     IP, #0          ；判断IP与0是否相等
BEQ #0x34               ；如果IP==0或者说R4是偶数将会跳转到0x34
SUB R6, R6, #3          ；如果IP!=0 将R6减3的结果传入R6
CMP     R2, R6          ; 此处是0x34，判断R2与R6是否相等
BNE     #0x54               ; 如果R2与R6不相等则跳转到0x54
ADD     R0, R0, #1      ；   R0加一
ADD     R1, R1, #1      ；   R1加一
ADD     R4, R4, #1          ；R4加一,R4是一个计数器
CMP     R4, R3              ；比较R4与R3的大小
BLT  #0x18                  ；如果R4小于R3则跳转到0x18
MOV     R5, #1
MOV     R0, #0              ；此处是0x54
MOV     R1, #0
MOV     R2, #0
```

```
MOV     R3, #0
MOV     R4, #0
MOV     R6, #0
MOV     R7, #0
MOV     R8, #0
MOV     SB, #0
MOV     SL, #0
MOV     IP, #0
```

主要循环判断过程如下：

1. 传入输入密码和隐藏密码的基址位数据
2. 将输入密码加5
3. 判断循环次数奇偶性，若为奇数将传入密码减3并比较传入密码与隐藏密码是否相等，若为偶数直接比较传入密码与隐藏密码是否相等
4. 如果传入密码与隐藏密码不相等则相当于直接退出，如果相等，基址加一，计数器加一
5. 判断计数器是否小于输出密码的长度，若小于直接回到1若不小于则相当于直接退出

流程图如下（核心类似于凯撒密码的核心）：



得到test_pw(enc_pw(userInput), get_pw())和enc_pw(userInput)函数的基本流程后开始着手编写这两个函数的逆向函数

1.test_pw()的逆向函数：

```
function findReqR6(){
  var pw = stoh("XYzaSAAX_PBssisodjsal_sSUVWZYYYb"); //从get_pw()的到的返回值
  var required = new Array();
  for(var i = 0 ; i < pw.length; i ++ ){
     var a = pw[i];
     a = a - 5;           //原流程加5
     if(i & 1 == 1){
       a = a + 3;         // 原流程减3
     }
     required[i] = a;
  }
  return required;
}
```

htos(findRqR6())返回enc_pw(user_input)的字符串转换结果返回值：

```
» htos(findReqR6())
← "SWu_N?<VZN=qngnm_hn_g]nQPTRXTWT`"
```

SWu_N?<VZN=qngnm_hn_g]nQPTRXTWT`

2.构造enc_pw()函数的逆向函数获取正确的UserInput

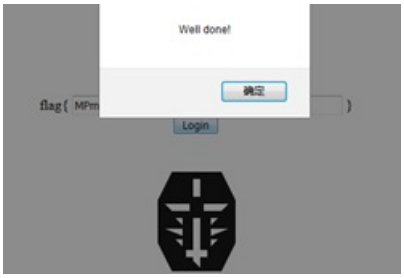
```
function reverseEnc(argarray){
  var test = 0;
  var output = new Array();

  for(var i = 0 ; i < argarray.length ; i++){
    var x = argarray[i];
    if(test == 1){
      var sub = (i & 3);
      x = x - sub;      //原流程加上相与值.
    }
    x = x - 6;              //原流程加6
    test = (argarray[i] & 1);
    output[i] = x;
```

```
  }
  return output;
}
```

htos(reverseEnc(findReqR6()))返回正确的userinput

```
» htos(reverseEnc(findReqR6()))
← "MPmVH94PTH7hhafgYahYaVfKJNLRNQLZ"
```

MPmVH94PTH7hhafgYahYaVfKJNLRNQLZ



flag:{ MPmVH94PTH7hhafgYahYaVfKJNLRNQLZ}