HASH扩展攻击

Da7ura_N0ir / 2017-11-13 16:49:00 / 浏览数 2704 安全技术 CTF 顶(0) 踩(0)

起因

一道ctf题

```
$flag = "XXXXXXXXXXXXXXXXXXXXXXX";
$secret = "XXXXXXXXXXXXXXX"; // This secret is 15 characters long for security!

$username = $_POST["username"];
$password = $_POST["password"];

if (!empty($_COOKIE["getmein"])) {
   if (urldecode($username) === "admin" && urldecode($password) != "admin") {
       if ($COOKIE["getmein"] === md5($secret . urldecode($username . $password))) {
           echo "Congratulations! You are a registered user.\n";
           die ("The flag is ". $flag);
       }
       else {
           die ("Your cookies don't match up! STOP HACKING THIS SITE.");
       }
   }
   else {
       die ("You are not an admin! LEAVE.");
   }
}

setcookie("sample-hash", md5($secret . urldecode("admin" . "admin")), time() + (60 * 60 * 24 * 7));

if (empty($_COOKIE["source"])) {
   setcookie("source", 0, time() + (60 * 60 * 24 * 7));
}
else {
   if ($_COOKIE["source"] != 0) {
       echo ""; // This source code is outputted here
   }
}
```

这里的关键绕过是这一句:
`if ($COOKIE["getmein"] === md5($secret . urldecode($username . $password)))`
要
`cookie['getmein']===$secret . urldecode($username . $password)`
的md5加密，而这里的secret是不可知的，但却知道他的长度，这里我们就涉及到hash扩展攻击。

MD5加密原理

MD5会把原数据分成512为一块的许多块，最后一块加上64字节来表示他的长度，一共构成512*n个字节然后再对这N个512数据块进行N次加密计算(因为过程较复杂，此处

加密过程

现在我们知道的是
`secretusernamepassword`这个数据，那么我们怎么进行攻击呢，我们看一下这个数据的16进制

算一下，22个字符,512/8=64,64/16=4，我们需要4排数据然后最后给一个整个数据长度，22=0x14,然后md5计算是小端存储，所以我们修改如下图

secretusernamepassword转16进制
`0x7365637265747457365726e616d657617373776f7264`
然后填充成
`0x7365637265747457365726e616d6570617373776f72648000000000000000000000000000000000000000000000000000000000000000000000000000000000000000b0000000000000`
md5('secretusernamepassword')==3105ff5f8723abe628d54387f2de5641
可以倒推出这个时候的ABCD1:
A=5fff0531
B=e6ab2387
C=8743d528

D=4156def2
现在如果我们继续加数据
0x736563726574757365726e616d6570617373776f726480000000000000000000000000000000000000000000000000000000000000000000000000000000b00000000000
现在我们已知前面512位计算出来的ABCD1,现在我们去掉前面直接用运算出来的ABCD1运算后面0x72747576得到的结果应该和加密全部的结果是一样的
0X72747576会被自动填充为
0x72747576800000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000200200000000
'secretusernamepassword'+'\x80'+'\x00'*33+'\xb0'+'\x00'*7+'\x72\x74\x75\x76'
直接md5加密结果为8e847c325fb05c60d437b23dc38ea6da
使用ABCD1手动加密0X72747576
A=327c848e,B=605cb05f,C=3db237d4,D=daa68ec3
md5:8e847c325fb05c60d437b23dc38ea6da
可以看到相同

攻击流程

既然如初，我们只要知道一个hash值，知道原来数据的数据长度，那么我们就可以算出
■■■+■■■■■512+■■■■的hash值
那么我们来看代码
他是直接用secret+username+password输入的是username和password，那么我们直接得出cookie里面的hash值，拿出这个hash值，倒推出这个ABCD1,然后用这个A
secret+username+password+■■■■+■■■的hash
like this :

加密代码(引用现成代码)

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# @Author■DshtAnger
# theory reference:
#   blog■
#       http://blog.csdn.net/adidala/article/details/28677393
#       http://blog.csdn.net/forgotaboutgirl/article/details/7258109
#       http://blog.sina.com.cn/s/blog_6fe0eb1901014cpl.html
#   RFC1321■
#       https://www.rfc-editor.org/rfc/pdfrfc/rfc1321.txt.pdf
################################################################################
import sys
def genMsgLengthDescriptor(msg_bitsLenth):
   '''
   ---args:
           msg_bitsLenth : the bits length of raw message
   --return:
           16 hex-encoded string , i.e.64bits,8bytes which used to describe the bits length of raw message added after padding
   '''
   return __import__("struct").pack(">Q", msg_bitsLenth).encode("hex")


def reverse_hex_8bytes(hex_str):
   '''
   --args:
           hex_str: a hex-encoded string with length 16 , i.e.8bytes
   --return:
           transform raw message descriptor to little-endian
   '''
   hex_str = "%016x" % int(hex_str, 16)
   assert len(hex_str) == 16
   return __import__("struct").pack("<Q", int(hex_str, 16)).encode("hex")


def reverse_hex_4bytes(hex_str):
   '''
   --args:
           hex_str: a hex-encoded string with length 8 , i.e.4bytes
   --return:
           transform 4 bytes message block to little-endian
   '''
   hex_str = "%08x" % int(hex_str, 16)
   assert len(hex_str) == 8
   return __import__("struct").pack("<L", int(hex_str, 16)).encode("hex")
```

```
def deal_rawInputMsg(input_msg):
   '''
   --args:
           input_msg : inputed a ascii-encoded string
   --return:
           a hex-encoded string which can be inputed to mathematical transformation function.
   '''
   ascii_list = [x.encode("hex") for x in input_msg]
   length_msg_bytes = len(ascii_list)
   length_msg_bits = len(ascii_list) * 8
   # padding
   ascii_list.append('80')
   while (len(ascii_list) * 8 + 64) % 512 != 0:
       ascii_list.append('00')
   # add Descriptor
   ascii_list.append(reverse_hex_8bytes(genMsgLengthDescriptor(length_msg_bits)))
   return "".join(ascii_list)


def getM16(hex_str, operatingBlockNum):
   '''
   --args:
           hex_str : a hex-encoded string with length in integral multiple of 512bits
           operatingBlockNum : message block number which is being operated , greater than 1
   --return:
           M : result of splited 64bytes into 4*16 message blocks with little-endian

   '''
   M = [int(reverse_hex_4bytes(hex_str[i:(i + 8)]), 16) for i in
        xrange(128 * (operatingBlockNum - 1), 128 * operatingBlockNum, 8)]
   return M


# ■■■■■■■■■■■■T[i]■■■■■■■■■32■■■■■■■&0xffffffff■■■■■■■■■■■■■■■
def T(i):
   result = (int(4294967296 * abs(__import__("math").sin(i)))) & 0xffffffff
   return result


# ■■■■■■■■■■■
# RL■■■■■■■■■■■■■■■■■■■32■■■■■■■0xffffffff■■■■■■■■■■■32■
F = lambda x, y, z: ((x & y) | ((~x) & z))
G = lambda x, y, z: ((x & z) | (y & (~z)))
H = lambda x, y, z: (x ^ y ^ z)
I = lambda x, y, z: (y ^ (x | (~z)))
RL = L = lambda x, n: (((x << n) | (x >> (32 - n))) & (0xffffffff))


def FF(a, b, c, d, x, s, ac):
   a = (a + F((b), (c), (d)) + (x) + (ac) & 0xffffffff) & 0xffffffff;
   a = RL((a), (s)) & 0xffffffff;
   a = (a + b) & 0xffffffff
   return a


def GG(a, b, c, d, x, s, ac):
   a = (a + G((b), (c), (d)) + (x) + (ac) & 0xffffffff) & 0xffffffff;
   a = RL((a), (s)) & 0xffffffff;
   a = (a + b) & 0xffffffff
   return a


def HH(a, b, c, d, x, s, ac):
   a = (a + H((b), (c), (d)) + (x) + (ac) & 0xffffffff) & 0xffffffff;
   a = RL((a), (s)) & 0xffffffff;
   a = (a + b) & 0xffffffff
   return a
```

```
def II(a, b, c, d, x, s, ac):
   a = (a + I((b), (c), (d)) + (x) + (ac) & 0xffffffff) & 0xffffffff;
   a = RL((a), (s)) & 0xffffffff;
   a = (a + b) & 0xffffffff
   return a


def show_md5(A, B, C, D):
   return "".join(["".join(__import__("re").findall(r"..", "%08x" % i)[::-1]) for i in (A, B, C, D)])


def run_md5(A=0x67452301, B=0xefcdab89, C=0x98badcfe, D=0x10325476, readyMsg=""):
   a = A
   b = B
   c = C
   d = D

   for i in xrange(0, len(readyMsg) / 128):
       M = getM16(readyMsg, i + 1)
       for i in xrange(16):
           exec "M" + str(i) + "=M[" + str(i) + "]"
       # First round
       a = FF(a, b, c, d, M0, 7, 0xd76aa478L)
       d = FF(d, a, b, c, M1, 12, 0xe8c7b756L)
       c = FF(c, d, a, b, M2, 17, 0x242070dbL)
       b = FF(b, c, d, a, M3, 22, 0xc1bdceeeL)
       a = FF(a, b, c, d, M4, 7, 0xf57c0fafL)
       d = FF(d, a, b, c, M5, 12, 0x4787c62aL)
       c = FF(c, d, a, b, M6, 17, 0xa8304613L)
       b = FF(b, c, d, a, M7, 22, 0xfd469501L)
       a = FF(a, b, c, d, M8, 7, 0x698098d8L)
       d = FF(d, a, b, c, M9, 12, 0x8b44f7afL)
       c = FF(c, d, a, b, M10, 17, 0xffff5bb1L)
       b = FF(b, c, d, a, M11, 22, 0x895cd7beL)
       a = FF(a, b, c, d, M12, 7, 0x6b901122L)
       d = FF(d, a, b, c, M13, 12, 0xfd987193L)
       c = FF(c, d, a, b, M14, 17, 0xa679438eL)
       b = FF(b, c, d, a, M15, 22, 0x49b40821L)
       # Second round
       a = GG(a, b, c, d, M1, 5, 0xf61e2562L)
       d = GG(d, a, b, c, M6, 9, 0xc040b340L)
       c = GG(c, d, a, b, M11, 14, 0x265e5a51L)
       b = GG(b, c, d, a, M0, 20, 0xe9b6c7aaL)
       a = GG(a, b, c, d, M5, 5, 0xd62f105dL)
       d = GG(d, a, b, c, M10, 9, 0x02441453L)
       c = GG(c, d, a, b, M15, 14, 0xd8a1e681L)
       b = GG(b, c, d, a, M4, 20, 0xe7d3fbc8L)
       a = GG(a, b, c, d, M9, 5, 0x21e1cde6L)
       d = GG(d, a, b, c, M14, 9, 0xc33707d6L)
       c = GG(c, d, a, b, M3, 14, 0xf4d50d87L)
       b = GG(b, c, d, a, M8, 20, 0x455a14edL)
       a = GG(a, b, c, d, M13, 5, 0xa9e3e905L)
       d = GG(d, a, b, c, M2, 9, 0xfcefa3f8L)
       c = GG(c, d, a, b, M7, 14, 0x676f02d9L)
       b = GG(b, c, d, a, M12, 20, 0x8d2a4c8aL)
       # Third round
       a = HH(a, b, c, d, M5, 4, 0xfffa3942L)
       d = HH(d, a, b, c, M8, 11, 0x8771f681L)
       c = HH(c, d, a, b, M11, 16, 0x6d9d6122L)
       b = HH(b, c, d, a, M14, 23, 0xfde5380c)
       a = HH(a, b, c, d, M1, 4, 0xa4beea44L)
       d = HH(d, a, b, c, M4, 11, 0x4bdecfa9L)
       c = HH(c, d, a, b, M7, 16, 0xf6bb4b60L)
       b = HH(b, c, d, a, M10, 23, 0xbebfbc70L)
       a = HH(a, b, c, d, M13, 4, 0x289b7ec6L)
       d = HH(d, a, b, c, M0, 11, 0xeaa127faL)
       c = HH(c, d, a, b, M3, 16, 0xd4ef3085L)
       b = HH(b, c, d, a, M6, 23, 0x04881d05L)
```

```
        a = HH(a, b, c, d, M9, 4, 0xd9d4d039L)
        d = HH(d, a, b, c, M12, 11, 0xe6db99e5L)
        c = HH(c, d, a, b, M15, 16, 0x1fa27cf8L)
        b = HH(b, c, d, a, M2, 23, 0xc4ac5665L)
        # Fourth round
        a = II(a, b, c, d, M0, 6, 0xf4292244L)
        d = II(d, a, b, c, M7, 10, 0x432aff97L)
        c = II(c, d, a, b, M14, 15, 0xab9423a7L)
        b = II(b, c, d, a, M5, 21, 0xfc93a039L)
        a = II(a, b, c, d, M12, 6, 0x655b59c3L)
        d = II(d, a, b, c, M3, 10, 0x8f0ccc92L)
        c = II(c, d, a, b, M10, 15, 0xffeff47dL)
        b = II(b, c, d, a, M1, 21, 0x85845dd1L)
        a = II(a, b, c, d, M8, 6, 0x6fa87e4fL)
        d = II(d, a, b, c, M15, 10, 0xfe2ce6e0L)
        c = II(c, d, a, b, M6, 15, 0xa3014314L)
        b = II(b, c, d, a, M13, 21, 0x4e0811a1L)
        a = II(a, b, c, d, M4, 6, 0xf7537e82L)
        d = II(d, a, b, c, M11, 10, 0xbd3af235L)
        c = II(c, d, a, b, M2, 15, 0x2ad7d2bbL)
        b = II(b, c, d, a, M9, 21, 0xeb86d391L)

        A += a
        B += b
        C += c
        D += d

        A = A & 0xffffffff
        B = B & 0xffffffff
        C = C & 0xffffffff
        D = D & 0xffffffff

        a = A
        b = B
        c = C
        d = D
        print "%x,%x,%x,%x" % (a, b, c, d)

   return show_md5(a, b, c, d)
samplehash="571580b26c65f306376d4f64e53cb5c7"

s1=0x5fff0531
s2=0xe6ab2387
s3=0x8743d528
s4=0x4156def2
secret = 'secretusernamepassword'

test=secret+'\x80'+'\x00'*33+'\xb0'+'\x00'*7+'\x72\x74\x75\x76'
s = deal_rawInputMsg(test)
inp = s[len(s)/2:]
print test+'\n'
print '------------------------------------------------------------'
print s
print '------------------------------------------------------------'
print inp
print '------------------------------------------------------------'
print "md5:"+run_md5(s1,s2,s3,s4,inp)
```


上一篇：如何自行搭建一个威胁感知大脑 SIEM 下一篇：渗透技巧——Windows系统的帐户隐藏

1. 1 条回复


Da7ura_N0ir 2017-11-23 10:34:02

第二张图那里填充有点问题应该是22*8的16进制，不好意思

0 回复Ta

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板