Teaser Dragon CTF 2018 crypto AES-128-TSB writeup

Lilac / 2018-09-30 22:33:44 / 浏览数 3070 安全技术 CTF 顶(1) 踩(0)

---

crypto AES-128-TSB writeup

题目

```
AES-128-TSB (Cryptography, 219)
difficulty: easy (46 solvers)

Haven't you ever thought that GCM mode is overcomplicated and there must be a simpler way to achieve Authenticated Encryption?

Server: aes-128-tsb.hackable.software 1337

server.py
```

server.py 内容:

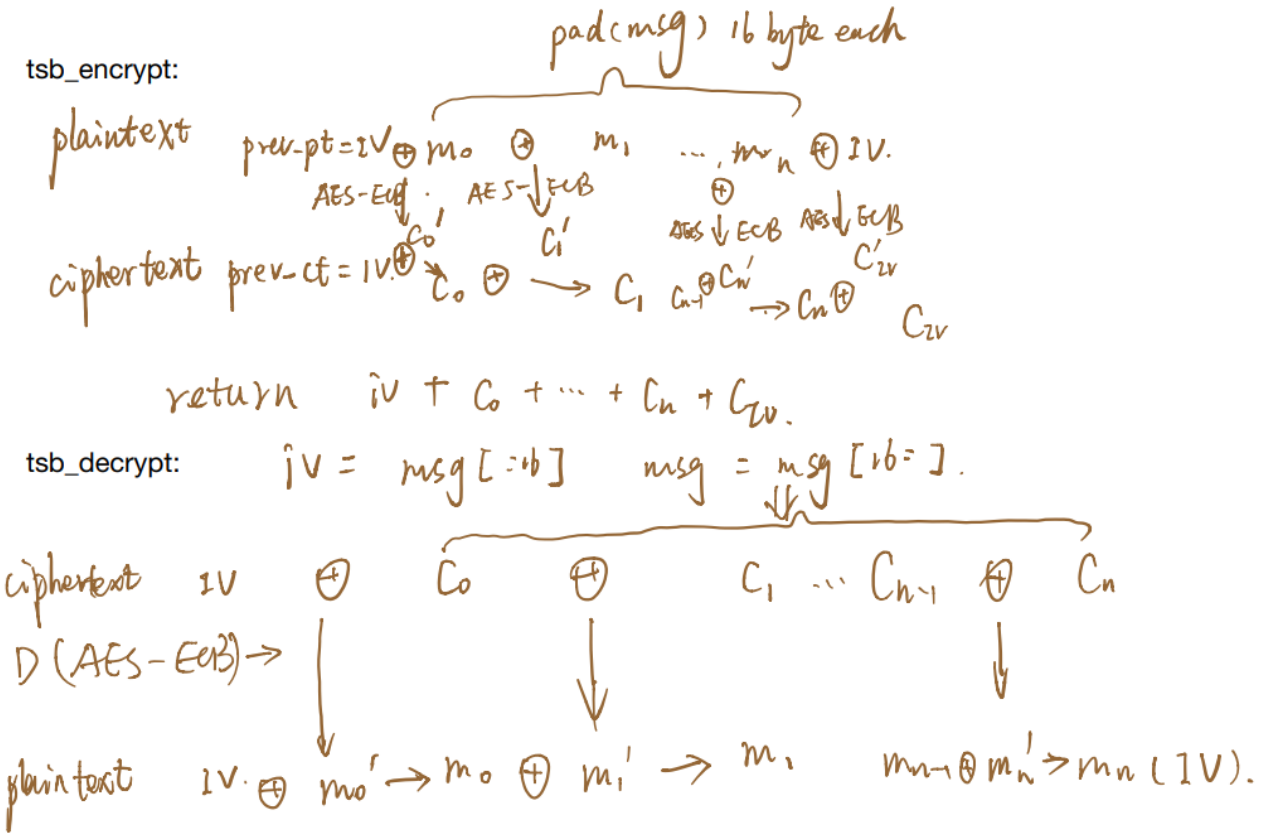
```
#!/usr/bin/env python2
import SocketServer
import socket
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
from struct import pack, unpack

from secret import AES_KEY, FLAG


class CryptoError(Exception):
    pass


def split_by(data, step):
    return [data[i : i+step] for i in xrange(0, len(data), step)]


def xor(a, b):
    assert len(a) == len(b)
    return ''.join([chr(ord(ai)^ord(bi)) for ai, bi in zip(a,b)])


def pad(msg):
    byte = 16 - len(msg) % 16
    return msg + chr(byte) * byte


def unpad(msg):
    if not msg:
        return ''
    return msg[:-ord(msg[-1])]


def tsb_encrypt(aes, msg):
    msg = pad(msg)
    iv = get_random_bytes(16)
    prev_pt = iv
    prev_ct = iv
    ct = ''
    for block in split_by(msg, 16) + [iv]:
        ct_block = xor(block, prev_pt)
        ct_block = aes.encrypt(ct_block)
        ct_block = xor(ct_block, prev_ct)
        ct += ct_block
        prev_pt = block
        prev_ct = ct_block
    return iv + ct
```

```
def tsb_decrypt(aes, msg):
   iv, msg = msg[:16], msg[16:]
   prev_pt = iv
   prev_ct = iv
   pt = ''
   for block in split_by(msg, 16):
       pt_block = xor(block, prev_ct)
       pt_block = aes.decrypt(pt_block)
       pt_block = xor(pt_block, prev_pt)
       pt += pt_block
       prev_pt = pt_block
       prev_ct = block
   pt, mac = pt[:-16], pt[-16:]
   if mac != iv:
       raise CryptoError()
   return unpad(pt)

def send_binary(s, msg):
   s.sendall(pack('<I', len(msg)))
   s.sendall(msg)

def send_enc(s, aes, msg):
   send_binary(s, tsb_encrypt(aes, msg))

def recv_exact(s, length):
   buf = ''
   while length > 0:
       data = s.recv(length)
       if data == '':
           raise EOFError()
       buf += data
       length -= len(data)
   return buf

def recv_binary(s):
   size = recv_exact(s, 4)
   size = unpack('<I', size)[0]
   return recv_exact(s, size)

def recv_enc(s, aes):
   data = recv_binary(s)
   return tsb_decrypt(aes, data)

def main(s):
   aes = AES.new(AES_KEY, AES.MODE_ECB)
   try:
       while True:
           a = recv_binary(s)
           b = recv_enc(s, aes)
           if a == b:
               if a == 'gimme_flag':
                   send_enc(s, aes, FLAG)
               else:
                   # Invalid request, send some random garbage instead of the
                   # flag :)
                   send_enc(s, aes, get_random_bytes(len(FLAG)))
           else:
               send_binary(s, 'Looks like you don\'t know the secret key? Too bad.')
   except (CryptoError, EOFError):
       pass

class TaskHandler(SocketServer.BaseRequestHandler):
   def handle(self):
       main(self.request)

if __name__ == '__main__':
   SocketServer.ThreadingTCPServer.allow_reuse_address = True
```

```
    server = SocketServer.ThreadingTCPServer(('0.0.0.0', 1337), TaskHandler)
    server.serve_forever()
```

题目分析

题目实现了如下图的密码系统,定义为TSB:



here is the problem with the crypto system.

we can change the last byte of IV, C0, ..., Cn, to keep IV == mn. the same time, the last byte of pt changes. so len(unpad(pt)) = 1. and makes a == b much easier by brute fore first byte of IV (remember to change every block first byte to make IV =

程序逻辑如下:

```
def main(s):
   aes = AES.new(AES_KEY, AES.MODE_ECB)
   try:
       while True:
           a = recv_binary(s)
           b = recv_enc(s, aes)
           if a == b:
               if a == 'gimme_flag':
                   send_enc(s, aes, FLAG)
               else:
                   # Invalid request, send some random garbage instead of the
                   # flag :)
                   send_enc(s, aes, get_random_bytes(len(FLAG)))
           else:
               send_binary(s, 'Looks like you don\'t know the secret key? Too bad.')
   except (CryptoError, EOFError):
       pass
```

解题思路

我们可以控制a,b,a为明文,b为密文,并且有个解密oracle,通过判断a==b,和服务器的返回结果,判断密文的解密情况.
step1是拿到一串服务器加密的密文,修改某些位置的byte后,让a=='gimme_flag',得到flag加密后的密文.通过构造 a="",b="",则可以得到`send_enc(s, aes, get_random_bytes(len(FLAG)))`密文,则我们知道padding后的flag长度为64,通过unpad的漏洞,可以知道padding值是13并且能通过oracle得到b="gimme_flagxxxxx
step2:
通过一位位的增加a,修改flag密文的pading值,从而很轻松得到flag(注意当到16byte时,check通过时,爆破的是padding值,而不是flag值)

脚本

step1.py

```
#!/usr/bin/env python2
#flag format DrgnS{...}
import SocketServer
import socket
from pwn import *
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
from struct import pack, unpack
from sys import argv, stdout
import time
import copy

io = remote("aes-128-tsb.hackable.software",1337)
# io = remote("127.0.0.1",1337)
# context.log_level = "debug"

#useful function
def split_by(data, step):
   return [data[i : i+step] for i in xrange(0, len(data), step)]


def xor(a, b):
   assert len(a) == len(b)
   return ''.join([chr(ord(ai)^ord(bi)) for ai, bi in zip(a,b)])


def pad(msg):
   byte = 16 - len(msg) % 16
   return msg + chr(byte) * byte


def unpad(msg):
   if not msg:
       return ''
   return msg[:-ord(msg[-1])]


def once(size_a,a,size_b,b):
   io.send(size_a)
   io.send(a)
```

```
   io.send(size_b)
   io.send(b)
once(p32(0),"",p32(0),"")
size = unpack('<I', io.recv(4))[0]
data = io.recv(size)
data = bytearray(data)

#test get the last char of the padded plaintext

collect_xor = []
block = split_by(data,16)
print repr("".join(str(i) for i in block))
modify_bk = split_by(data,16)
target = "gimme_flag"
for loop in range(0,len(target)):
   print "working on \033[31m%s\033[0m"%(target[loop])
   for xx in range(0,256):
       # print xx
       flag = 0
       for a in [13]:
           stdout.write("[+] Test [Byte %03i/256 - Byte %03i/256] \r\n\r" % (xx,a))
           # stdout.flush()
           for index in range(len(modify_bk)):
               if index==0:
                   modify_bk[index][-1] = chr(a^(63-loop)^block[index][-1])
                   modify_bk[index][loop] = chr(xx^block[index][loop])
               else:
                   modify_bk[index][-1]=chr(block[index-1][-1]^block[index][-1]^modify_bk[index-1][-1])
                   modify_bk[index][loop]=chr(block[index-1][loop]^block[index][loop]^modify_bk[index-1][loop])
           modify = "".join(str(i) for i in modify_bk)
           once(p32(loop+1),target[:loop+1],p32(size),modify)
           check_size = unpack('<I', io.recv(4))[0]
           check_data = io.recv(check_size)
           if 'Looks like you don\'t know the secret key? Too bad.' not in check_data:
               print repr(modify)
               print repr("".join(str(i) for i in block))
               for index in range(len(block)):
                   if index==0:
                       block[index][loop] = chr(xx^block[index][loop])
                   else:
                       block[index][loop]=chr(block[index-1][loop]^block[index][loop]^modify_bk[index-1][loop])
               print repr("".join(str(i) for i in block))
               flag = 1
               break

       if flag:
           break
print repr("".join(str(i) for i in block))
```

setp2_1.py (flag位置不是16byte整数倍时)

```
#!/usr/bin/env python2
#flag format DrgnS{...}
import SocketServer
import socket
from pwn import *
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
from struct import pack, unpack
from sys import argv, stdout
import time
import copy

io = remote("aes-128-tsb.hackable.software",1337)
# io = remote("127.0.0.1",1337)
# context.log_level = "debug"

#useful function
def split_by(data, step):
```

```
   return [data[i : i+step] for i in xrange(0, len(data), step)]


def xor(a, b):
   assert len(a) == len(b)
   return ''.join([chr(ord(ai)^ord(bi)) for ai, bi in zip(a,b)])


def pad(msg):
   byte = 16 - len(msg) % 16
   return msg + chr(byte) * byte


def unpad(msg):
   if not msg:
       return ''
   return msg[:-ord(msg[-1])]


def once(size_a,a,size_b,b):
   io.send(size_a)
   io.send(a)
   io.send(size_b)
   io.send(b)
target = 'gimme_flag'
pre_padding = 13
mac = '\xdb\x1bC\xa0\x9c\x90\xda2\x10\xb0$^\xa6\xf2\xdd\x9e\xbd\xe1:\xca\xb9Wy\xf8\x13!C69\xf2>\xaf\xc7\x8ba?U\x13/ZS\xb1\x00S
# print repr(mac)
# length = len(mac)
# print length
# mac = bytearray(mac)
# mac_block = split_by(mac,16)
# after_mac = split_by(mac,16)
# for index in range(len(mac_block)):
#     if index==0:
#         after_mac[index][-1] = chr(pre_padding^(63-9)^mac_block[index][-1])
#     else:
#         after_mac[index][-1]=chr(mac_block[index-1][-1]^mac_block[index][-1]^after_mac[index-1][-1])
# fake_mac = "".join(str(i) for i in after_mac)
fake_mac = '"\xc7\xb6\x18|\xce3\x88@jV\x9fy\xc4\xae\xb6P\x91\xbe\xc2\xc2v2\x128\r\x08\xb0\xc0\xebLq\xc62\xe7\xccVg/\x8e\x1a\xe
print repr(fake_mac)
once(p32(len(target)),target,p32(len(fake_mac)),fake_mac)
size = unpack('<I', io.recv(4))[0]
data = io.recv(size)
data = bytearray(data)

#test get the last char of the padded plaintext

data = bytearray("XfMj\x03\x0c^[6\xda@\xd6c\x81av\x8b\xf0\x03\xdb\x06YK%\x1a\x9d\x81q\x8a^<\x9d\xb4\x0e\xbd\n#VTg\xa7l\xad{s\x

block = split_by(data,16)
modify_bk = split_by(data,16)
print repr("".join(str(i) for i in block))

# flag="DrgnS{"
true_flag = "DrgnS{Thank_god_no_one_deployed_this_on_producti"
flag_array = bytearray(true_flag)
flag_copy  = bytearray(true_flag)
for loop in range(48,51):
   print "working on \033[31m%s\033[0m"%(str(loop))
   for byte in range(126,31,-1):
       # stdout.write("guessing %d"%(byte)+"\r\n")
       # stdout.write("\r")
       # stdout.flush()
       block = split_by(data,16)
       modify_bk = split_by(data,16)
       for index in range(len(modify_bk)):
           if index==0:
               modify_bk[index][-1] = chr(pre_padding^(63-loop)^block[index][-1])
```

```
                modify_bk[index][loop%16] = chr(flag_array[loop%16]^byte^block[index][loop%16])
            else:
                modify_bk[index][-1]=chr(block[index-1][-1]^block[index][-1]^modify_bk[index-1][-1])
                modify_bk[index][loop%16]=chr(flag_array[loop%16]^byte^block[index][loop%16])
        modify = "".join(str(i) for i in modify_bk)
        # print flag_copy
        flag_copy[loop%16]=byte
        flag_copy[16+loop%16]=byte^flag_array[16+loop%16]^flag_array[loop%16]
        flag_copy[32+loop%16]=byte^flag_array[32+loop%16]^flag_array[loop%16]
        flag_copy[15] = ord("_")^(63-loop)^13
        flag_copy[31] = ord("_")^(63-loop)^13
        flag_copy[47] = ord("i")^(63-loop)^13
        # print flag_copy
        once(p32(loop+1),str(flag_copy)+true_flag[loop%16],p32(size),modify)
        check_size = unpack('<I', io.recv(4))[0]
        check_data = io.recv(check_size)
        if 'Looks like you don\'t know the secret key? Too bad.' not in check_data:
            flag_array = str(flag_array)+chr(byte)
            print type(flag_array)
            flag_array = bytearray(flag_array)
            true_flag+=chr(byte)
            flag_copy = copy.deepcopy(flag_array)
            print "[+]flag is \033[31m%s\033[0m"%(repr(true_flag))
            break
print true_flag
```

output:

```
working on 48
<type 'str'>
[+]flag is 'DrgnS{Thank_god_no_one_deployed_this_on_productio'
working on 49
<type 'str'>
[+]flag is 'DrgnS{Thank_god_no_one_deployed_this_on_production'
working on 50
<type 'str'>
[+]flag is 'DrgnS{Thank_god_no_one_deployed_this_on_production}'
DrgnS{Thank_god_no_one_deployed_this_on_production}
```

setp2_2.py (flag位置是16byte整数倍时)

```
#!/usr/bin/env python2
#flag format DrgnS{...}
import SocketServer
import socket
from pwn import *
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
from struct import pack, unpack
from sys import argv, stdout
import time
import copy

io = remote("aes-128-tsb.hackable.software",1337)
# io = remote("127.0.0.1",1337)
# context.log_level = "debug"

#useful function
def split_by(data, step):
   return [data[i : i+step] for i in xrange(0, len(data), step)]


def xor(a, b):
   assert len(a) == len(b)
   return ''.join([chr(ord(ai)^ord(bi)) for ai, bi in zip(a,b)])


def pad(msg):
   byte = 16 - len(msg) % 16
   return msg + chr(byte) * byte
```

```
def unpad(msg):
   if not msg:
       return ''
   return msg[:-ord(msg[-1])]


def once(size_a,a,size_b,b):
   io.send(size_a)
   io.send(a)
   io.send(size_b)
   io.send(b)
target = 'gimme_flag'
pre_padding = 13
mac = '\xdb\x1bC\xa0\x9c\x90\xda2\x10\xb0$^\xa6\xf2\xdd\x9e\xbd\xe1:\xca\xb9Wy\xf8\x13!C69\xf2>\xaf\xc7\x8ba?U\x13/ZS\xb1\x00S
# print repr(mac)
# length = len(mac)
# print length
# mac = bytearray(mac)
# mac_block = split_by(mac,16)
# after_mac = split_by(mac,16)
# for index in range(len(mac_block)):
#     if index==0:
#         after_mac[index][-1] = chr(pre_padding^(63-9)^mac_block[index][-1])
#     else:
#         after_mac[index][-1]=chr(mac_block[index-1][-1]^mac_block[index][-1]^after_mac[index-1][-1])
# fake_mac = "".join(str(i) for i in after_mac)
fake_mac = '"\xc7\xb6\x18|\xce3\x88@jV\x9fy\xc4\xae\xb6P\x91\xbe\xc2\xc2v2\x128\r\x08\xb0\xc0\xebLq\xc62\xe7\xccVg/\x8e\x1a\xe
print repr(fake_mac)
once(p32(len(target)),target,p32(len(fake_mac)),fake_mac)
size = unpack('<I', io.recv(4))[0]
data = io.recv(size)
data = bytearray(data)

#test get the last char of the padded plaintext

data = bytearray("XfMj\x03\x0c^[6\xda@\xd6c\x81av\x8b\xf0\x03\xdb\x06YK%\x1a\x9d\x81q\x8a^<\x9d\xb4\x0e\xbd\n#VTg\xa7l\xad{s\x

block = split_by(data,16)
modify_bk = split_by(data,16)
print repr("".join(str(i) for i in block))

# flag="DrgnS{"
true_flag = "DrgnS{Thank_god_no_one_deployed_this_on_product"
flag_array = bytearray(true_flag)
flag_copy  = bytearray(true_flag)
for loop in range(47,51):
   print "working on \033[31m%s\033[0m"%(str(loop))
   for byte in range(126,31,-1):
       # stdout.write("guessing %d"%(byte)+"\r\n")
       # stdout.write("\r")
       # stdout.flush()
       block = split_by(data,16)
       modify_bk = split_by(data,16)
       for index in range(len(modify_bk)):
           if index==0:
               modify_bk[index][-1] = chr(pre_padding^(63-loop)^block[index][-1])
               # modify_bk[index][loop%16] = chr(flag_array[loop%16]^byte^block[index][loop%16])
           else:
               modify_bk[index][-1]=chr(block[index-1][-1]^block[index][-1]^modify_bk[index-1][-1])
               # modify_bk[index][loop%16]=chr(flag_array[loop%16]^byte^block[index][loop%16])
       modify = "".join(str(i) for i in modify_bk)
       # print flag_copy
       # flag_copy[loop%16]=byte
       # flag_copy[16+loop%16]=byte^flag_array[16+loop%16]^flag_array[loop%16]
       # flag_copy[32+loop%16]=byte^flag_array[32+loop%16]^flag_array[loop%16]
       flag_copy[15] = ord("_")^(63-loop)^13
       flag_copy[31] = ord("_")^(63-loop)^13
```

```
        # print flag_copy
        once(p32(loop+1),str(flag_copy)+chr(byte^(63-loop)^13),p32(size),modify)
        check_size = unpack('<I', io.recv(4))[0]
        check_data = io.recv(check_size)
        if 'Looks like you don\'t know the secret key? Too bad.' not in check_data:
            flag_array = str(flag_array)+chr(byte)
            print type(flag_array)
            flag_array = bytearray(flag_array)
            true_flag+=chr(byte)
            flag_copy = copy.deepcopy(flag_array)
            print "[+]flag is \033[31m%s\033[0m"%(repr(true_flag))
            break
print true_flag
```

得到位置为16byte整数倍的flag.

总结

思路很清楚,代码能力太差了,希望跟各位多交流.

tsb_crypto.zip (3.273 MB) 下载附件
点击收藏 | 0 关注 | 1
上一篇：[红日安全]PHP-Audit-L... 下一篇：Teaser Dragon CTF...
1. 0 条回复
- 动动手指，沙发就是你的了！

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