

Forum:[Hack Forums](#)

Thread: [TCP Hijacking](#)

Clumsy - October 24, 2024, 2:43 am

This Python script is a TCP connection-breaking tool that uses the Scapy library to manipulate network packets. It defines custom exceptions for connection handling and provides functions to retrieve endpoints based on process IDs (PIDs) or TCP ports by reading from /proc/net/tcp, which lists current TCP connections on a Linux system. The TCPBreaker class is central to the functionality, allowing it to filter and intercept packets associated with specified local and remote TCP ports, capturing a TCP session's sequence number to send a reset (RST) packet. This effectively terminates the connection. The script can also interact with ipset, adding or removing IP pairs to manage firewall rules. It includes a command-line interface for user interaction, enabling options such as specifying a PID, local/remote ports, and choosing the packet capture method (raw or nflog). Overall, the script aims to provide a means to control and terminate unwanted TCP connections programmatically.

Usage

```
python tcp_breaker.py [OPTIONS] [ipset_name]
```

Options

`ipset_name`: (Optional) The name of the ipset to temporarily insert the local IP into. This can be used in an iptables filter rule to reject outgoing packets matching it with a TCP reset packet.

`--pid <PID>`: (Optional) The Process ID (PID) of the process whose connection you want to terminate.

`--port <PORT>`: (Optional) The local TCP port of the connection to close.

`-d, --remote-port`: (Optional) Treat the port argument as a remote port rather than a local port.

`-c <driver>, --capture <driver>`: (Optional) Specify the capture method. Options are "raw" or "nflog" (default is "nflog" if `ipset_name` is specified, otherwise "raw").

`--debug`: (Optional) Enable verbose operation mode for debugging purposes.

Example Commands

Terminate a connection by PID

```
python tcp_breaker.py --pid 1234 --port 8080
```

Terminate a connection using an ipset

```
python tcp_breaker.py my_ipset --pid 1234
```

Terminate a connection by specifying local and remote ports

```
python tcp_breaker.py my_ipset --port 80 -d --remote-port
```

Using raw capture method without ipset

```
python tcp_breaker.py --pid 5678 --port 443 --capture raw
```

Note

Ensure you have the necessary permissions to run the script, as it requires access to `/proc/net/tcp` and may need elevated privileges to modify ipset rules or terminate connections.

Made By ClumsyLulz aka SleepTheGod

```
Code #!/usr/bin/env python
```

```
from __future__ import print_function
```

```
from scapy.all import *
import itertools as it
import operator as op
import functools as ft
from subprocess import Popen, PIPE
from os.path import join
import os
import sys
import signal
import re
```

```
class ConnMultipleMatches(Exception):
    pass
```

```
class ConnNotFound(Exception):
    pass
```

```
def get_endpoints(pid=None, port_local=None, port_remote=None):
    if pid:
        matches = list(get_endpoints_by_pid(pid))
    else:
        matches = list(get_endpoints_by_port(port_local, port_remote))

    if not matches:
        raise ConnNotFound()
    if len(matches) > 1:
        raise ConnMultipleMatches(matches)

    local, remote = matches[0]
    return (
        (inet_ntoa(struct.pack('<I', local[0] & 0xffffffff)), local[1]),
        (inet_ntoa(struct.pack('<I', remote[0] & 0xffffffff)), remote[1])
    )
```

```

def get_endpoints_by_port(port_local=None, port_remote=None):
    'Lookup connection for a specified port(s) in /proc/net/tcp table.'
    assert port_local or port_remote, (port_local, port_remote)
    with open('/proc/net/tcp') as src:
        src = iter(src)
        next(src) # skip header line
        for line in src:
            local, remote = (
                tuple(int(v, 16) for v in ep.split(':'))
                for ep in op.itemgetter(1, 2)(line.split())
            )
            for ep, port in [(local, port_local), (remote, port_remote)]:
                if ep[1] == port:
                    break
            else:
                continue
            yield local, remote

def get_endpoints_by_pid(pid):
    'Lookup connection(s) for a specified pid.'
    assert isinstance(pid, int), pid
    conns = {}
    with open('/proc/net/tcp') as src:
        src = iter(src)
        next(src) # skip header line
        for line in src:
            line = line.split()
            conns[line[9]] = line[1], line[2]

    pid_fds = f'/proc/{pid}/fd'
    for fd in os.listdir(pid_fds):
        match = re.search(r'^socket:\[(\d+)\]$', os.readlink(join(pid_fds, fd)))
        if not match:
            continue
        try:
            match = conns[match.group(1)]
        except KeyError:
            continue
        yield tuple(tuple(int(v, 16) for v in ep.split(':')) for ep in match)

def ipset_update(cmd, name, ip, port):
    if name is None:
        return
    assert cmd in ['add', 'del'], cmd
    log.debug('Updating ipset {!r} ({!r} {!r})'.format(name, cmd, ip, port))
    if Popen(['ipset', cmd, name, '{},{}'.format(ip, port)]).wait():
        raise RuntimeError('Failed running ipset command, see error output above.')

class TCPBreaker(Automaton):

```

```
ss_filter_port_local = ss_filter_port_remote = None
ss_remote, ss_intercept = None, None
```

```
def __init__(self, port_local, port_remote, timeout=False, **atmt_kwz):
    self.ss_filter_port_local, self.ss_filter_port_remote = port_local, port_remote
    self.ss_timeout = timeout
    atmt_kwz.setdefault('store', False)
    atmt_kwz.setdefault('filter',
        ('tcp and ((src port {sport} and dst port {dport})'
         ' or (dst port {sport} and src port {dport})))').format(
            sport=port_local, dport=port_remote)
    )
    super(TCPBreaker, self).__init__(**atmt_kwz)
```

```
def pkt_filter(self, pkt):
    'Filter packets by ports in case of capture filter leaks or dissector bugs.'
    if {pkt[TCP].sport, pkt[TCP].dport} == {self.ss_filter_port_local, self.ss_filter_port_remote}:
        return pkt[IP].dst if pkt[TCP].sport == self.ss_filter_port_local else pkt[IP].src
    raise KeyError
```

```
@ATMT.state(initial=1)
```

```
def st_seek(self):
    log.debug('Waiting for noise on the session')
```

```
@ATMT.receive_condition(st_seek)
```

```
def check_packet(self, pkt):
    log.debug('Session check, packet: {}'.format(pkt.summary()))
    try:
        remote = self.pkt_filter(pkt)
    except KeyError:
        return
    raise self.st_collect(pkt).action_parameters(remote)
```

```
@ATMT.action(check_packet)
```

```
def break_link(self, remote):
    if not self.ss_remote:
        log.debug('Found session (remote: {})'.format(remote))
        self.ss_remote = remote
```

```
@ATMT.timeout(st_seek, 1) # yep, hard-coded 1s timeout
```

```
def interception_done(self):
    if self.ss_remote and self.ss_intercept:
        log.debug('Captured seq, proceeding to termination')
        raise self.st_rst_send()
```

```
@ATMT.state()
```

```
def st_collect(self, pkt):
    log.debug('Collected: {}'.format(pkt))
    self.ss_intercept = pkt # only last one matters
    raise (self.st_rst_send if not self.ss_timeout else self.st_seek)()
```

```
@ATMT.state()
```

```

def st_rst_send(self):
    pkt_ip, pkt_tcp = op.itemgetter(IP, TCP)(self.ss_intercept)
    ordered = lambda k1, v1, k2, v2, pkt_dir=(pkt_ip.dst == self.ss_remote):\
        dict(it.izip((k1, k2), (v1, v2) if pkt_dir else (v2, v1)))
    rst = IP(**ordered('src', pkt_ip.src, 'dst', pkt_ip.dst)) /\
        TCP(**dict(it.chain.from_iterable(p.viewitems() for p in (
            ordered('sport', pkt_tcp.sport, 'dport', pkt_tcp.dport),
            ordered('seq', pkt_tcp.seq, 'ack', pkt_tcp.ack),
            dict(flags=b'R', window=pkt_tcp.window))))))
    rst[TCP].ack = 0
    log.debug('Sending RST: {!r}'.format(rst))
    send(rst)

# Finish
os.kill(os.getpid(), signal.SIGINT)
self.stop()

```

```

def main(argv=None):
    import argparse
    parser = argparse.ArgumentParser(
        description='TCP connection breaking tool.'
        ' Finds connection with a specified parameters and adds local'
        ' ip:port to the specified ipset, so firewall can cut it on first activity'
        ' (with something like "-j REJECT --reject-with tcp-reset").'
        ' Also uses captured traffic to get connection sequence number and send'
        ' RST packet to remote endpoint, removing ip:port from ipset upon success.'
    )

    parser.add_argument('ipset_name', nargs='?',
        help='Name of the ipset to temporarily insert local ip:port to.'
        ' Should be used in some iptables filter rule, rejecting outgoing'
        ' packets matching it with tcp-reset packet (see README for examples).'
        ' If not specified, no ipset updates will be performed and raw-socket'
        ' capture interface will be used (as packet will presumably make it through
netfilter).')

    parser.add_argument('--pid', type=int,
        help='Process ID to terminate connection of.')

    parser.add_argument('--port', type=int,
        help='TCP port of local (unless --remote-port is specified) connection to close.')
    parser.add_argument('-d', '--remote-port', action='store_true',
        help='Treat port argument as remote, not local one.')

    parser.add_argument('-c', '--capture', metavar='driver',
        help='Either "raw" or "nflog". Default is "nflog" if ipset name is'
        ' specified as an argument (since packet should not pass through'
        ' netfilter in this case), otherwise "raw".')
    parser.add_argument('--debug', action='store_true', help='Verbose operation mode.')
    optz = parser.parse_args(sys.argv[1:] if argv is None else argv)

    import logging

```

```

logging.basicConfig(level=logging.DEBUG if optz.debug else logging.INFO)
global log
log = logging.getLogger()

if not optz.capture:
    optz.capture = 'nflog' if optz.ipset_name else 'raw'
    if optz.capture == 'nflog':
        log.debug('Using nflog packet capture interface')
    elif optz.capture not in ['nflog', 'raw']:
        parser.error('Only "nflog" or "raw" values are'
            ' supported with --capture option, not {!r}'.format(optz.capture))

# It should be traffic to/from same machine - no promisc-mode necessary
conf.promisc = conf.sniff_promisc = False
# Disables "Sent 1 packets." line
conf.verb = False

if optz.capture == 'nflog':
    from scapy_nflog import install_nflog_listener
    install_nflog_listener()

if optz.pid:
    local, remote = get_endpoints(optz.pid, optz.port if not optz.remote_port else None,
        None if not optz.remote_port else optz.port)
else:
    local, remote = get_endpoints_by_pid(optz.pid)

log.debug('local = {}; remote = {}'.format(local, remote))
if optz.ipset_name:
    ipset_update('add', optz.ipset_name, *local)

TCPBreaker(local[1], remote[1]).run()

if optz.ipset_name:
    ipset_update('del', optz.ipset_name, *local)

return 0

if __name__ == '__main__':
    sys.exit(main())

```

Soccerdude6000 - October 24, 2024, 12:17 pm

Can you dm me the download for this?

Soccerdude6000 - October 25, 2024, 5:05 pm

(Oct 25, 2024 03:29 AM)m0hx Wrote: [\[CITE\]](#) the python script is included in the thread, just copy paste into new file.py
Thanks

Pimping - October 25, 2024, 5:11 pm

Looks interesting
thanks for the contribution

