Condor – Network Profiling Tool

Problem statement

Currently, Condor has no information about networking requirements of a job. Each job can execute multiple processes and each process in turn can send or receive date over multiple sockets.

We need some mechanism to fetch per process and per connection network stats of a Condor job, so that the same information can be used in future for better placement and scheduling decisions in Condor.

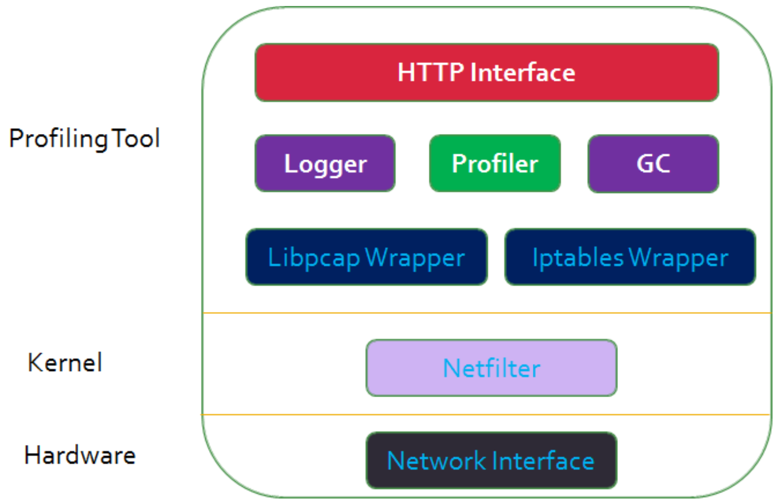
Solution

Some of the properties of the working solution should be:

a. It should utilize features which are already part of Linux kernel.
b. It should work for virtual environments too.
c. It should be generic and not tied to Condor.

We develop a profiling tool to get the stats of network activity of a Condor job. It leverages Netfilter/IPTABLES to track the TCP connections which are created by the processes corresponding to a Condor job. Using IPTABLES, we can create rule for each new network connection in Netfilter and when the connection terminates, we can fetch the stats of that specific connection.

Architecture



Approach

The profiling tool performs the following functions:

1. Exposes an HTTP interface to Condor for specifying the job, pid to be profiled.
2. Maintains a mapping between Jobs and corresponding list of processes to be profiled.
3. Creates a network filter using LIBPCAP to capture packets over specified network interface. The name of the interface is specified in the config.ini file.
4. The filter is set to TCP SYN | FIN | RST. This helps in tracking new & terminated connections. Hence whenever a TCP SYN, FIN or RST packet is received by or sent from the machine, the same packet is forwarded to our user space tool.

5. Based on the type of packet received, we can take appropriate action. The advantage of using Libpcap filter is that not all packets are sent from kernel space to user space and hence provides a performance boost.
6. For a TCP SYN received, finds the associated process (PID) using Netstat.
7. If the process is in the list of processes to be profiled, it Installs IPTABLES rules for that socket.
8. When a connection terminates i.e. a TCP FIN | RST is received, it fetches stats from IPTABLES and writes them to a log file.
9. When a process terminates, Condor informs the tool about it over the HTTP interface and the tool in turn fetches stats for all the remaining sockets for that process from IPTABLES and logs them.
10. When the number of IPTABLES rules installed on the machine crosses a specified threshold, a Garbage collector is triggered to delete the left-over IPTABLES rules from terminated processes.

Components

1. **External Interface:** The profiler tool provides an http based interface. The following calls can be made by Condor:

Interface	Description	Parameters
StartProfile	Start profiling a job	jobId, pid, owner, birthdate
EndProfile	Stop profiling a job	jobId, pid, owner, birthdate
EndProcessProfile	Stop profiling process associated to a job	jobId
Shutdown	Shutdown	None

   Since the PID can be reused, *owner* and *birthdate* help to uniquely identify a process.

2. **Libpcap Wrapper:** The profiler tool uses Libpcap (pcapy package in python) to filter out packets in the kernel. Based on the filter, only TCP SYN, FIN and RST packets are sent to the user space.

3. **IPTABLES Wrapper:** This component performs the tasks related to IPTABLES like Insertion, deletion of IPTABLES rules and collection of stats from IPTABLES.

4. **Logger**: This component logs the stats of each incoming & outgoing connection into a specified log file. The format of a log entry is:

   <JobId, PID, SrcIP, SrcPort, DestIP, DestPort, ConnectionDirection, DataTransferred>

   The log has been implemented as a rolling log file. The size of log file and number of log files to be created is tuneable. The name of the log file is the same as the name of the interface being tracked by the profiling tool.

5. **Garbage Collector:** This component deals with clean-up of left-over IPTABLES rules which had not been removed at the time of socket or connection closing.

    a. It is launched when number of installed IPTABLES rules reaches a specified threshold and the profiler tool cannot install any new rule.
    b. The threshold has to be specified in config.ini file.
    c. Current implementation walks over the list of processes being tracked by profiler tool and if a process no longer exists, all the IPTABLES rules installed for that process are deleted and hence the profiler tool can install new rules.

6. **Profiler**: This is the main component of the tool. It acts as an interface between all the other components explained above and manages them.

    a. It maintains many data-structures to store mappings like JobId to PID mapping, PID to Socket mapping etc.
    b. It interfaces with IPTABLES wrapper and Inserts/deletes IPTABLES rules whenever a new socket is created or closed.
    c. It also gathers stats when a socket is closed or a process terminates and interfaces with the Logger component to log the stat for that socket into the log file.
    d. It is also responsible for triggering the Garbage Collector whenever required.

Working:

1. When the Tool is launched it cleans-up all the left-over IPTABLES rules from last execution. This is to handle the cases where the tool would have crashed abruptly because of some reason.

2. The profiler tool then creates separate threads for Profiler, Libpcap wrapper and HTTP (Web) interface. They communicate with each other via shared data-structures.

3. Profiler stops packet processing in the user space when Garbage Collection is triggered. This is to make sure that the data-structures are not in an inconsistent state while other thread is modifying them.

Tuneable parameters:

The profiler tool uses an input file (config.ini) in which the user can specify the values of tuneable parameters like:

a. Interface to be monitored

```
INTERFACE = virbr0
```

b. Max size of each log file

```
MAX_BYTES = 1048576
```

c. Number of separate backup files that will be created before roll-up

```
BACKUP_COUNT = 100
```

d. Maximum number of rules that can be installed

```
RULES_THRESHOLD = 1000
```

Execution:

1. The profiler tool has to be launched as ‘root’.
2. A port number over which the profiler tool will accept HTTP request has to be specified as well. [python main.py <port>]

Setup Instructions:

Following python packages have to be installed on the machines where Condor jobs would be running:

1. Install web.py:

sudo apt-get install python-setuptools
sudo easy_install web.py
[http://webpy.org/docs/0.3/api]

2. Install pcapy:

sudo apt-get install python-pcapy
http://www.binarytides.com/code-a-packet-sniffer-in-python-with-pcapy-extension/
http://stackoverflow.com/questions/4948043/pcap-python-library
http://stackoverflow.com/questions/8148608/network-traffic-monitor-with-pcapy-in-python
http://pylibpcap.sourceforge.net/

3. Install psutil:

http://psutil.googlecode.com/hg/INSTALL
Download the latest release
from https://pypi.python.org/pypi?:action=display&name=psutil#downloads [use sudo]
https://pypi.python.org/pypi?:action=display&name=psutil

4. Install python-dpkt

sudo apt-get install python-dpkt

Future Enhancements & performance improvements

1. Garbage collector can be modified to reclaim IPTABLES rules from running processes too. This can be done by checking against the list of sockets/connections for that processes and verifying if a socket has been closed.

2. There might be a race-condition which can lead to profiler tool unable to track network activity of some connections of a process. This can happen if Condor has already started a process and there is a delay in Condor communicating creation of the process to Profiler tool. If during this time, the process opens up any new connection, the profiler tool will not track it.

3. Currently, we are using Netstat to find PID of the process to which the received packet/socket belongs. We can think of using some other more efficient approach for doing the same.

4. During sniffing we don’t need to sniff 65536 bytes, we can reduce it to just TCP header size.

5. Combine rules together using ipsets [http://ipset.netfilter.org]

6. Rather than appending rule, insert it at the beginning, so that older/obsolete rules do not impact performance. Paper: http://people.netfilter.org/kadlec/nftest.pdf

Source Code Location

https://github.com/tihor2004/NetworkProfiler

References

1. **Libpcap:**
    a. http://www.tcpdump.org/pcap.html
    b. http://yuba.stanford.edu/~casado/pcap/section4.html
    c. Example: http://stackoverflow.com/questions/8148608/network-traffic-monitor-with-pcapy-in-python
    d. TcpDump rules: http://www.danielmiessler.com/study/tcpdump/
    e. Scapy Guide: http://theitgeekchronicles.files.wordpress.com/2012/05/scapyguide1.pdf

2. **IPTABLES**:
    a. https://openvz.org/Traffic_accounting_with_iptables
    b. https://github.com/ldx/python-iptables
    c. http://www.cyberciti.biz/faq/linux-configuring-ip-traffic-accounting/
    d. https://wiki.archlinux.org/index.php/iptables
    e. https://www.frozentux.net/iptables-tutorial/iptables-tutorial.html#HOWARULEISBUILT
    f. https://www.frozentux.net/iptables-tutorial/iptables-tutorial.html#TRAVERSINGOFTABLES
    g. IPTables: https://library.linode.com/security/firewalls/iptables
    h. IPTables tcp syn: http://www.netfilter.org/documentation/HOWTO/packet-filtering-HOWTO-7.html
    i. http://www.designsequence.net/How-Tos/iptables_bandwidth_monitoring.html
    j. Accounting: http://mohskitchen.wordpress.com/tag/iptables/