Automating Linux Malware Analysis Using Limon Sandbox

Monnappa K A
monnappa22@gmail.com

A number of devices are running Linux due to its flexibility and open source nature. This has made Linux platform the target for malware attacks, so it becomes important to analyze the Linux malware. Today, there is a need to analyze Linux malwares in an automated way to understand its capabilities.

Limon is a sandbox developed as a research project written in python, which automatically collects, analyzes, and reports on the run time indicators of Linux malware. It allows one to inspect the malware before execution, during execution, and after execution (post-mortem analysis) by performing static, dynamic and memory analysis using open source tools. Limon analyzes the malware in a controlled environment, monitors its activities and its child processes to determine the nature and purpose of the malware. It determines the malware's process activity, interaction with the file system, network, it also performs memory analysis and stores the analyzed artifacts for later analysis. Since Limon relies on open source tools, it's easy for any security analyst to setup a personal sandbox to perform Linux malware analysis. The paper will touch on details of Linux malware analysis and features of Limon sandbox.

Why Malware Analysis?

Malware is a piece of software which causes harm to a computer system without the owner's consent. Viruses, Trojans, worms, backdoors, rootkits and spyware can all be considered as malwares.

With new malware attacks making news every day and compromising company’s network and critical infrastructures around the world, malware analysis is critical for anyone who responds to such incidents.

Malware analysis is the process of understanding the behaviour and characteristics of malware, how to detect and eliminate it.

There are many reasons why we would want to analyze a malware, below to name just a few:

- Determine the nature and purpose of the malware i.e. whether the malware is an information stealing malware, http bot, spam bot, rootkit, keylogger, RAT etc.
- Interaction with the Operating System i.e. to understand the file system, process and network activities.
- Detect identifiable patterns (network and host based indicators) to cure and prevent future infections

Types of Malware Analysis

In order to understand the characteristics of the malware three types of analysis can be performed they are:

- Static Analysis
- Dynamic Analysis
- Memory Analysis

In most cases static and dynamic analysis will yield sufficient results however Memory analysis helps in determining hidden artifacts, rootkit and stealth malware capabilities.

Static Analysis

Static Analysis involves analyzing the malware without actually executing it. Following are the steps:

- **Determining the File Type**: Determining the file type can also help you understand the type of environment the malware is targeted towards, for example if the file type is ELF (Executable and Linkable format) format which is a standard binary file format for Unix and Unix-like systems, then it can be concluded that the malware is targeted towards a Unix or Unix flavoured systems.

- **Determining the Cryptographic Hash**: Cryptographic Hash values like MD5 and SHA1 can serve as a unique identifier for the file throughout the course of analysis. Malware, after executing can copy itself to a different location or drop another piece of malware, cryptographic hash can help you determine whether the newly copied/dropped sample is same as the original sample or a different one. With this information we can determine if malware analysis needs to be performed on a single sample or multiple samples. Cryptographic hash can also be submitted to online antivirus scanners like VirusTotal to determine if it has been previously detected by any of the AV vendors. Cryptographic hash can also be used to search for the specific malware sample on the internet.

- **Strings search**: Strings are plain text ASCII and UNICODE characters embedded within a file. Strings search give clues about the functionality and commands associated with a malicious file. Although strings do not provide complete picture of the function and capability of a file, they can yield information like file names, URL, domain names, ip address, attack commands etc.

- **File obfuscation (packers, cryptors) detection:** Malware authors often use softwares like packers and cryptors to obfuscate the contents of the file in order to evade detection from anti-virus softwares and intrustion detection systems. This technique slows down the malware analysts from reverse engineering the code.

- **Determine Fuzzy Hash:** Comparing the malware samples collected or maintained in a private or public repository is an important part of file identification process. The easiest way to check for file similarity is through a process called “Fuzzy Hashing”. Fuzzy hash comparison can tell the percentage similarity between the files. Fuzzy hash comparison is a method by which identical files can be identified. This can help in determine the variants of the same malware.

- **Submission to online Antivirus scanning services**: This will help you determine if the malicious code signatures exist for the suspect file. The signature name for the specific file provides an excellent way to gain additional information about the file and capabilities. By visiting the respective antivirus vendor web sites or searching for the signature in search engines can yield additional details about the suspect file. Such information may help in further investigation and reduce the analysis time of the malware specimen.
  VirusTotal (http://www.virustotal.com) is a popular web based malware scanning services.

- **Inspecting File Dependencies:** Executable loads multiple shared libraries and call api functions to perform certain actions like resolving domain names, establishing an http connection etc. Determining the type of shared library and list of api calls imported by an executable can give an idea on the functionality of the malware.

- **Examining ELF File Structure:** ELF stands for "Executable and Linkable Format" this is a standard binary file format for Linux systems. Examining the ELF file structure can yield wealth of the information including Sections, Symbols and other file metadata information.

- **Disassembling the File:** Examining the suspect program in a disassembler allows the investigator to explore the instructions that will be executed by the malware. Disassembly can help in tracing the paths that are not usually determined during dynamic analysis.

Dynamic Analysis

Dynamic Analysis involves executing the malware sample in a controlled environment and monitoring as it runs. Sometimes static analysis will not reveal much information due to obfuscation, packing in such cases dynamic analysis is the best way to identify malware functionality. Following are some of the steps involved in dynamic analysis:

- **Monitoring Process Activity:** This involves executing the malicious program and examining the properties of the resulting process and other processes running on the infected system. This technique can reveal information about the process like process name, process id, child processes created, system path of the executable program, modules loaded by the suspect program.

- **Monitoring File System Activity:** This involves examining the real time file system activity while the malware is running; this technique reveals information about the opened files, newly created files and deleted files as a result of executing the malware sample.

- **Monitoring Network Activity:** In addition to monitoring the activity on the infected host system, monitoring the network traffic to and from the system

during the course of running the malware sample is also important. This helps to identify the network capabilities of the specimen and will also allow us to determine the network based indicator which can then be used to create signatures on security devices like Intrusion Detection System.

- **System Call Tracing:** System calls made by malware can provide insight into the nature and purpose of the executed program such as file, network and memory access. Monitoring the system calls can help determine the interaction of the malware with the operating system.

Memory Analysis

Memory Analysis also referred to as Memory Forensics is the analysis of the memory image taken from the running computer. Analyzing the memory after executing the malware sample provides post-mortem perspective and helps in extracting forensics artifacts from a computer's memory like:

- running processes
- network connections
- loaded modules
- code injections
- Hooking and Rootkit capabilities.
- API Hooking

Limon Linux Sandbox

Limon is a sandbox for automating Linux malware analysis. It was developed as a research project for learning Linux malware analysis. It is written in python and uses custom python scripts and various open source tools to perform static, dynamic/behavioural and memory analysis.

Working of Limon

Limon performs below steps for analyzing the linux malware samples.

- Takes sample as input
- Performs static analysis
- Starts the VM
- Transfers the malware to VM
- Runs the monitoring tools (to monitor process, file system, network activity etc)
- Executes the malware for the specified time
- Stops the monitoring tools
- Suspends the VM
- Acquires the memory image
- Performs memory analysis using Volatility framework
- Stores the results (Final reports, destkop screenshot, pcaps and malicious artifacts for later analysis)

Tools Used by Limon

Limon relies on below tools to perform static, dynamic and memory analysis

- Custom python scripts
- YARA-python
  - https://github.com/plusvic/yara
- VirusTotal Public api
  - https://www.virustotal.com/en/documentation/public-api/
- ssdeep
  - http://ssdeep.sourceforge.net/
- strings utility
  - http://linux.die.net/man/1/strings
- ldd
  - http://linux.die.net/man/1/ldd
- readelf
  - https://sourceware.org/binutils/docs/binutils/readelf.html
- Inetsim
  - http://www.inetsim.org/downloads.html
- Tcpdump
  - http://www.tcpdump.org/
- strace
  - http://linux.die.net/man/1/strace
- Sysdig
  - http://www.sysdig.org/
- Volatility memory forensics framework
  - http://www.volatilityfoundation.org/#!releases/component_71401

Supported File Types

Limon can analyze below file types (both with and without parameters) :

- ELF Executable(both x86 and x86_64)
- Perl Script
- Python script
- Shell script

- Bash script
- PHP script
- Loadable kernel module(LKM)

General Features of Limon

- Can run in sandbox mode (does not allow to connect to c2)
- Can run in internet mode (connects to c2)
- Simulates all services (like dns, http and other protocols) when run in sandbox mode
- Option to run malware for specified time (default is 60 seconds)
- Captures desktop screenshot
- Reports on the malware behaviour

Static Analysis Features

Below are the static analysis capabilities of Limon:

- Determines File Type
- Determines File Size
- Determines md5 hash
- Determines fuzzy hash(ssdeep hash)
- Comparison of fuzzy hash with previously submitted samples to determine similar variants
- Display ELF header Structure
- Dumps ASCII and UNICODE strings
- Determines packers using YARA rules
- Determines malware capability using YARA rules (ability to run custom YARA rules will be added soon)
- Perfoms md5 search on VirusTotal(does not submit samples)
- Displays dependencies of the malware (shared objects)
- Displays program header structures
- Displays section header information
- Displays symbol table (both static and dynamic symbols)

Dynamic Analysis Features

Limon gives different options for performing dynamic analysis to track activity of the malware(during execution), below are the different options:

- Filtered call trace for tracing system calls related to file, process, network activity
- Unfiltered call trace - traces all system calls (more noisy)
- Filtered system event montioring to track file, process, network activity (less noisy)
- Unfiltered system even monitoring to track file, process, network, memory allocations/unallocations, signals etc (more noisy)
- Shows DNS summary
- Shows TCP conversations
- Stores packet captures
- Stores event trace dump

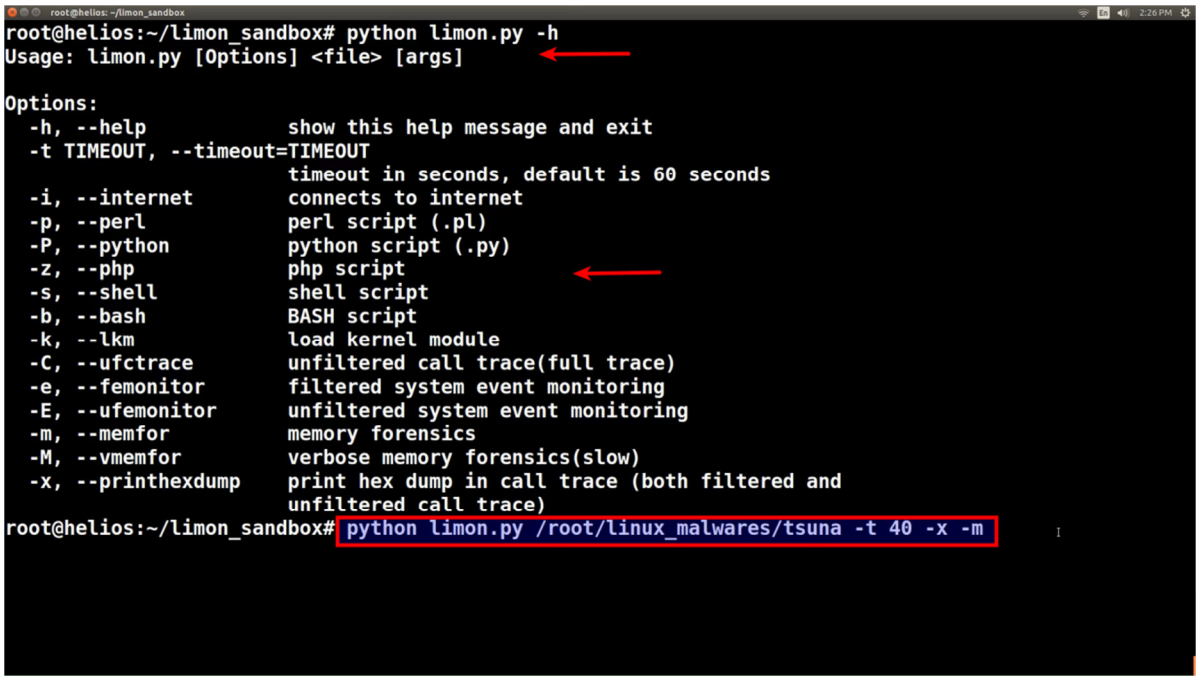
Memory Analysis features

Limon performs post-mortem analysis by performing memory analysis using Volatility framework. This feature should help in detecting stealthy rootkits and malwares performing Anti-Forensic tricks. Below are the memory analysis features:

- option to perform verbose memory forensics (slow)
- Process Listing (using different methods)
- Process tree listing
- Process listing with process arguments
- Displays thread associated with each process
- Dispays Network connections (TCP and UDP)
- Displays Interface Information
- Displays processes running with RAW sockets
- Displays shared libaries associated with the processes (using different methods)
- Displays kernel modules
- Dislays kernel modules hidden from module list but present in SYSFS
- Displays Kernel modules hidden from both module list and SYSFS
- Displays files opened within kernel
- Displays processes sharing credential structures
- Checks for keyboard notifier hooks
- Checks for TTY hooks
- Checks for system call table modification
- Displays BASH history
- Checks for modified file operation structures
- Checks hooked network operation function structures
- Checks netfilter hooks
- Check inline kernel hooks
- Checks for code or binary injection
- Check for PLT/GOT hooks (only in verbose mode)
- Checks for userland api hooks (only in verbose mode)

Analysis of Linux Malware Tsunami using Limon

To demonstrate the working of Limon, Linux malware sample “Tsunami” was run in Limon for 40 seconds as shown in the screenshot below. This section contains the analysis details of the Linux malware “Tsunami”. The screenshots also shows different options in Limon.

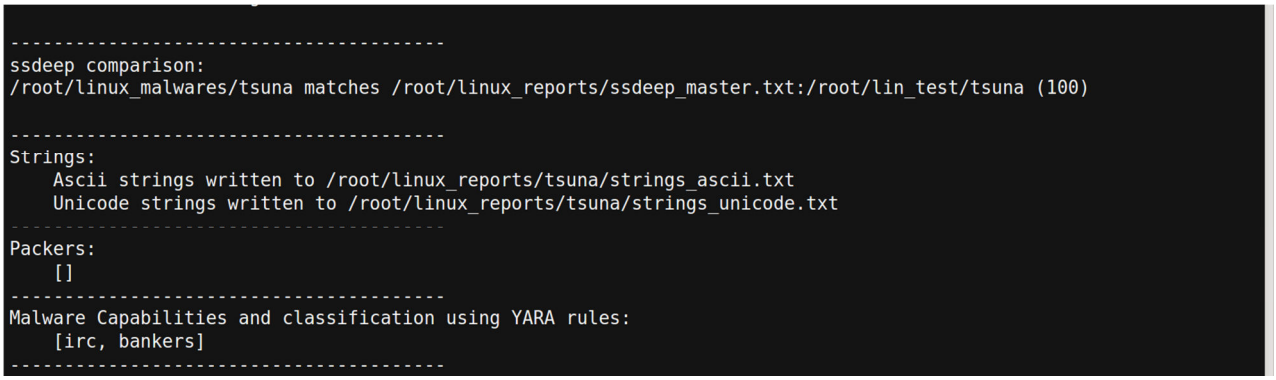


Below screenshot shows some of the static analysis results after analyzing the malware in Limon. The malware is 32 bit ELF executable, its dynamically linked and the symbols are not stripped.

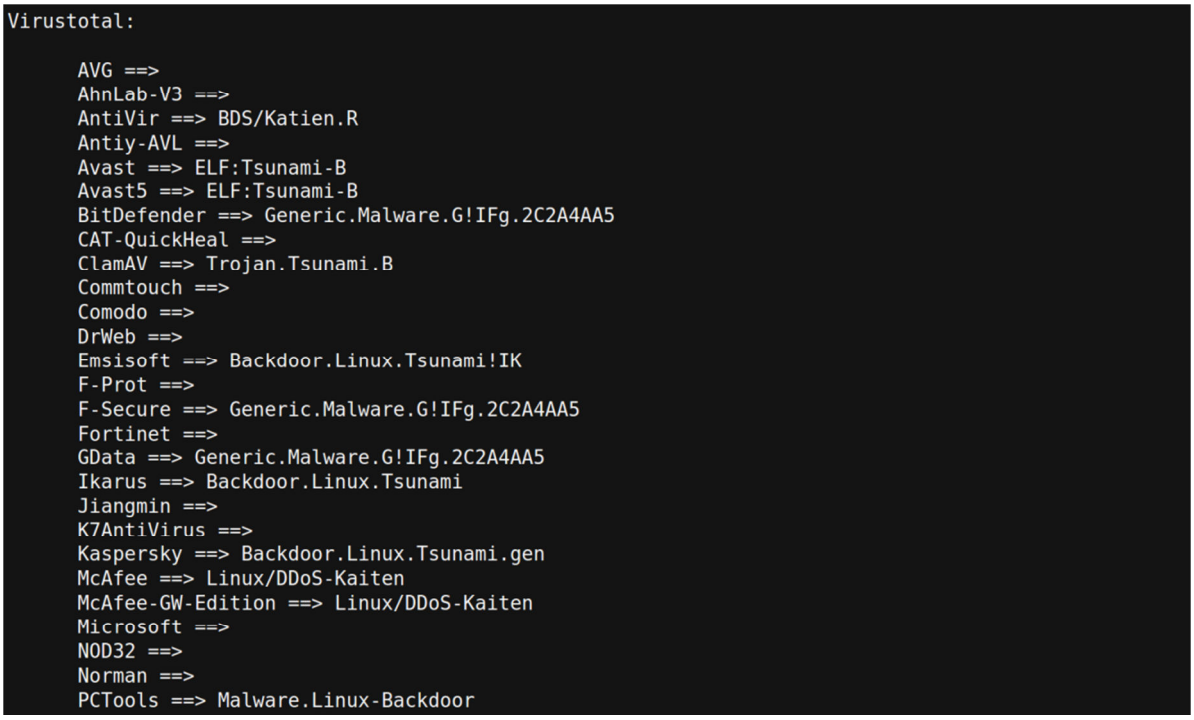
```
==========[STATIC ANALYSIS RESULTS]==========

Filetype: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked (uses shared libs), for
GNU/Linux 2.6.8, not stripped
File Size: 28.63 KB (29318 bytes)
md5sum: 1610768b1524e24d840ae25964d02c8e
ssdeep: 384:fJp2sVqQvqRFP514VWPE898bTyJGb0GnfknfXI0yIUQhLxJs+C3POCtZ8ax0h/49:BpRkQiVHAbTyJGb01fXI+9w9f5+R4wC
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Intel 80386
  Version:                           0x1
  Entry point address:               0x8048e10
  Start of program headers:          52 (bytes into file)
  Start of section headers:          23172 (bytes into file)
  Flags:                             0x0
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         7
  Size of section headers:           40 (bytes)
  Number of section headers:         36
  Section header string table index: 33
```

When a malware is submitted to Limon, Limon determines the ssdeep hash (fuzzy hash) and compares the fuzzy hash with the master list of fuzzy hashes of previously submitted samples. In this case the malware fuzzy hash has 100% match with the previously submitted sample, indicating that it is the same malware sample and the

malware sample was also run against YARA rules to determine malware capabilities. As shown in the below screenshot it looks like malware has IRC capabilities.



When the malware is submitted to Limon, it determines the md5 hash of the sample and uses the md5sum to search the VirusTotal using its public api. In this case the sample is detected by AV vendors as Tsunami.



Symbol information shows references to network related system calls indicating the network capability of the malware.

```
Symbol table '.dynsym' contains 56 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 00000000    29 FUNC    GLOBAL DEFAULT  UND __errno_location@GLIBC_2.0 (2)
     2: 00000000    49 FUNC    GLOBAL DEFAULT  UND sprintf@GLIBC_2.0 (2)
     3: 00000000   141 FUNC    GLOBAL DEFAULT  UND popen@GLIBC_2.1 (3)
     4: 00000000    96 FUNC    GLOBAL DEFAULT  UND srand@GLIBC_2.0 (2)
     5: 00000000   108 FUNC    GLOBAL DEFAULT  UND connect@GLIBC_2.0 (2)
     6: 00000000    49 FUNC    GLOBAL DEFAULT  UND getpid@GLIBC_2.0 (2)
     7: 00000000     0 NOTYPE  WEAK   DEFAULT  UND __gmon_start__
     8: 00000000   192 FUNC    GLOBAL DEFAULT  UND vsprintf@GLIBC_2.0 (2)
     9: 00000000   555 FUNC    GLOBAL DEFAULT  UND inet_network@GLIBC_2.0 (2)
    10: 00000000   108 FUNC    GLOBAL DEFAULT  UND recv@GLIBC_2.0 (2)
    11: 00000000    34 FUNC    GLOBAL DEFAULT  UND inet_addr@GLIBC_2.0 (2)
    12: 00000000   198 FUNC    GLOBAL DEFAULT  UND strncpy@GLIBC_2.0 (2)
    13: 00000000   112 FUNC    GLOBAL DEFAULT  UND write@GLIBC_2.0 (2)
    14: 00000000   108 FUNC    GLOBAL DEFAULT  UND sendto@GLIBC_2.0 (2)
    15: 00000000    55 FUNC    GLOBAL DEFAULT  UND listen@GLIBC_2.0 (2)
    16: 00000000    50 FUNC    GLOBAL DEFAULT  UND toupper@GLIBC_2.0 (2)
    17: 00000000   369 FUNC    GLOBAL DEFAULT  UND fgets@GLIBC_2.0 (2)
    18: 00000000    88 FUNC    GLOBAL DEFAULT  UND memset@GLIBC_2.0 (2)
    19: 00000000   441 FUNC    GLOBAL DEFAULT  UND __libc_start_main@GLIBC_2.0 (2)
    20: 00000000     7 FUNC    GLOBAL DEFAULT  UND ntohl@GLIBC_2.0 (2)
    21: 00000000    14 FUNC    GLOBAL DEFAULT  UND htons@GLIBC_2.0 (2)
    22: 00000000   251 FUNC    GLOBAL DEFAULT  UND free@GLIBC_2.0 (2)
    23: 00000000   108 FUNC    GLOBAL DEFAULT  UND accept@GLIBC_2.0 (2)
    24: 00000000    58 FUNC    GLOBAL DEFAULT  UND ioctl@GLIBC_2.0 (2)
    25: 00000000    55 FUNC    GLOBAL DEFAULT  UND socket@GLIBC_2.0 (2)
    26: 00000000   539 FUNC    GLOBAL DEFAULT  UND fclose@GLIBC_2.1 (3)
```

Strings from the malware sample shows references to the C2 ip and references to http and IRC commands

```
80.243.54.131
NOTICE %s :Unable to comply.
/usr/dict/words
%s : USERID : UNIX : %s
NOTICE %s :GET <host> <save as>
NOTICE %s :Unable to create socket.
http://
NOTICE %s :Unable to resolve address.
NOTICE %s :Unable to connect to http.
GET /%s HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.75 [en] (X11; U; Linux 2.2.16-3 i686)
Host: %s:80
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
NOTICE %s :Receiving file.
NOTICE %s :Saved as %s
NOTICE %s :Spoofs: %d.%d.%d.%d
NOTICE %s :Spoofs: %d.%d.%d.%d - %d.%d.%d.%d
NOTICE %s :Kaiten wa goraku
NOTICE %s :NICK <nick>
NOTICE %s :Nick cannot be larger than 9 characters.
```

Strings extracted from the malware also shows the references to the attacks commands of the malware, from the strings it looks like the malware has DOS/DDOS capabilities.

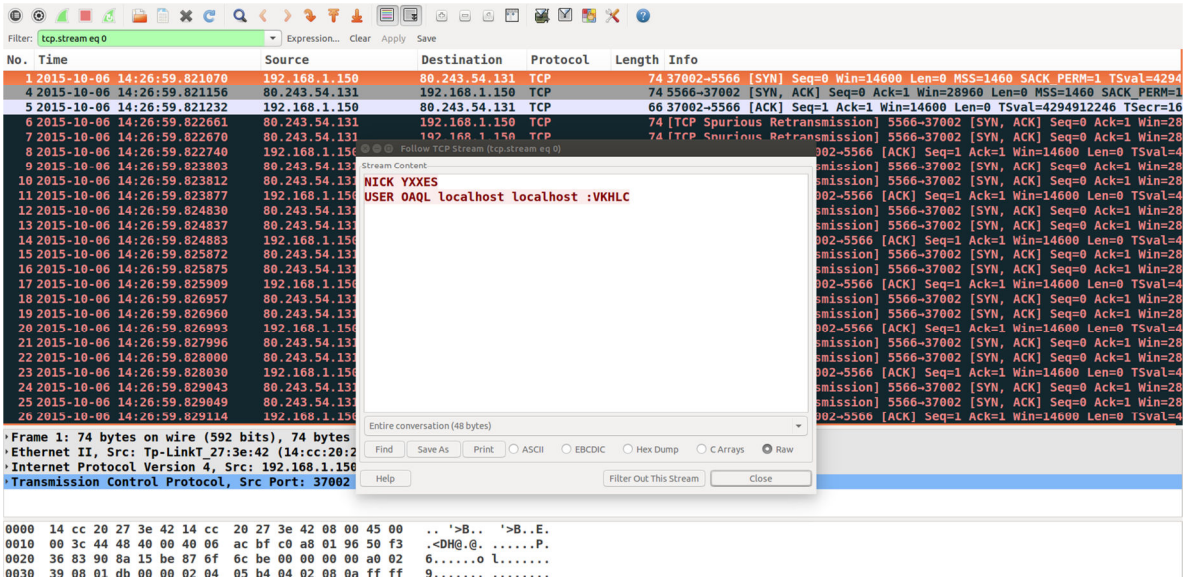
```
NOTICE %s :Tsunami heading for %s.
NOTICE %s :UNKNOWN <target> <secs>
NOTICE %s :Unknowning %s.
NOTICE %s :MOVE <server>
NOTICE %s :TSUNAMI <target> <secs>                          = Special packeter that wont be blocked by
most firewalls
NOTICE %s :PAN <target> <port> <secs>                       = An advanced syn flooder that will kill
most network drivers
NOTICE %s :UDP <target> <port> <secs>                       = A udp flooder
NOTICE %s :UNKNOWN <target> <secs>                          = Another non-spoof udp flooder
NOTICE %s :NICK <nick>                                      = Changes the nick of the client
NOTICE %s :SERVER <server>                                  = Changes servers
NOTICE %s :GETSPOOFS                                        = Gets the current spoofing
NOTICE %s :SPOOFS <subnet>                                  = Changes spoofing to a subnet
NOTICE %s :DISABLE                                          = Disables all packeting from this client
NOTICE %s :ENABLE                                           = Enables all packeting from this client
NOTICE %s :KILL                                             = Kills the client
NOTICE %s :GET <http address> <save as>                     = Downloads a file off the web and saves
it onto the hd
NOTICE %s :VERSION                                          = Requests version of client
NOTICE %s :KILLALL                                          = Kills all current packeting
NOTICE %s :HELP                                             = Displays this
NOTICE %s :IRC <command>                                    = Sends this command to the server
NOTICE %s :SH <command>                                     = Executes a command
NOTICE %s :Killing pid %d.
```

The screenshots below shows the dynamic analysis results. The malware was successfully executed by Limon, after execution the malware creates a child process (with pid 2674). The child process tries to read a file /usr/dict/words which does not exist. From the name of the file it looks like it's a dictionary file which malware uses for some kind of password cracking. Also the malware creates a network socket, establishes a connection with the C2 ip on port 5566 and writes some content on the socket.

```
=========================[DYNAMIC ANALYSIS RESULTS]=========================

CALL TRACE ACTIVITIES
=====================================
2673  execve("/root/malware analysis/tsuna", ["/root/malware analysis/tsuna"], [/* 50 vars */]) = 0
2673  open("/usr/lib/vmware-tools/libconf/lib/tls/i686/sse2/cmov/libc.so.6", O_RDONLY|O_CLOEXEC) = -1
ENOENT (No such file or directory)
2673  open("/usr/lib/vmware-tools/libconf/lib/tls/i686/sse2/libc.so.6", O_RDONLY|O_CLOEXEC) = -1
ENOENT (No such file or directory)
2673  open("/usr/lib/vmware-tools/libconf/lib/tls/i686/cmov/libc.so.6", O_RDONLY|O_CLOEXEC) = -1
```

```
2673  clone(child_stack=0, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0) =
2674
2674  open("/usr/dict/words", O_RDONLY) = -1 ENOENT (No such file or directory)
2674  open("/usr/dict/words", O_RDONLY) = -1 ENOENT (No such file or directory)
2674  open("/usr/dict/words", O_RDONLY) = -1 ENOENT (No such file or directory)
2674  socket(PF_INET, SOCK_STREAM, IPPROTO_TCP) = 3
2674  connect(3, {sa_family=AF_INET, sin_port=htons(5566), sin_addr=inet_addr("80.243.54.131")}, 16)
= -1 EINPROGRESS (Operation now in progress)
2674  connect(3, {sa_family=AF_INET, sin_port=htons(5566), sin_addr=inet_addr("80.243.54.131")}, 16)
= 0
2674  write(3, "NICK YXXES\nUSER OAQL localhost localhost :VKHLC\n", 48) = 48
 | 00000  4e 49 43 4b 20 59 58 58  45 53 0a 55 53 45 52 20  NICK YXX ES.USER  |
 | 00010  4f 41 51 4c 20 6c 6f 63  61 6c 68 6f 73 74 20 6c  OAQL loc alhost l |
 | 00020  6f 63 61 6c 68 6f 73 74  20 3a 56 4b 48 4c 43 0a  ocalhost  :VKHLC. |
```

The packet capture shows the IRC communication made by the malware to the C2 ip on port 5566. The malware is an IRC bot.



Process listing from the memory analysis results shows the malicious process “tsuna” running with a pid 2674

```
2015-10-06 08:56:06 UTC+0000
0xffff88001c332de0 goa-daemon            2603             0                0      0x0000000006cb8000
2015-10-06 08:56:06 UTC+0000
0xffff88001a250000 gnome-screensav       2608             0                0      0x0000000009126000
2015-10-06 08:56:11 UTC+0000
0xffff88001a35dbc0 aptd                  2662             0                0      0x000000000c8cf000
2015-10-06 08:56:52 UTC+0000
0xffff8800020b8000 vmtoolsd              2671             0                0      0x000000001a5c7000
2015-10-06 08:56:59 UTC+0000
0xffff88001a35ade0 strace                2672             0                0      0x000000001a3d8000
2015-10-06 08:56:59 UTC+0000
0xffff88001b992de0 tsuna                 2674             0                0      0x00000000005aa000
2015-10-06 08:56:59 UTC+0000
0xffff88001a555bc0 dnsmasq               2691             65534            30     0x000000001a1a8000
2015-10-06 08:57:44 UTC+0000
0xffff88001efedbc0 dbus-daemon           2698             102              105    0x000000001cc07000
2015-10-06 08:57:44 UTC+0000
0xffff88001a5544d0 dbus-daemon-lau       2700             0                0      0x00000000178b3000
2015-10-06 08:57:44 UTC+0000
```

Network connections from the memory analysis shows that the process “tsuna” (with pid 2674) established the connection to the C2 ip on port 5566

```
NETWORK CONNECTIONS
===================================

UDP      0.0.0.0         : 5353 0.0.0.0          :    0                    avahi-daemon/677
UDP      ::              : 5353 ::               :    0                    avahi-daemon/677
UDP      0.0.0.0         :38766 0.0.0.0          :    0                    avahi-daemon/677
UDP      ::              :43148 ::               :    0                    avahi-daemon/677
TCP      127.0.0.1       :  631 0.0.0.0          :    0 LISTEN                  cupsd/752
TCP      192.168.1.150   :39549 91.189.89.144    :   80 CLOSE_WAIT       ubuntu-geoip-pr/2455
TCP      192.168.1.150   :37002 80.243.54.131    : 5566 ESTABLISHED            tsuna/2674
UDP      127.0.0.1       :   53 0.0.0.0          :    0                    dnsmasq/2691
TCP      127.0.0.1       :   53 0.0.0.0          :    0 LISTEN             dnsmasq/2691
```

Conclusion

Linux is growing in its popularity and with multiple devices running Linux it has become target for malware attacks, so it becomes important to analyze the Linux malware in an automated way to determine the network and host based indicators. This paper provided a high level introduction to malware analysis and also introduced a tool "Limon" to perform static, dynamic and memory analysis of Linux malwares. The paper also covered the analysis of a Linux malware called "tsunami" using Limon, which helped in determining the various capabilities of the malware.

References

https://en.wikipedia.org/wiki/Linux_malware

https://securelist.social-kaspersky.com/en/descriptions/iframe/Backdoor.Linux.Tsunami.gen

http://malware.wikia.com/wiki/Tsunami

http://www.intego.com/mac-security-blog/tsunami-backdoor-can-be-used-for-denial-of-service-attacks/