International Journal of Applied Engineering Research
ISSN 0973-4562 Volume 10, Number 10 (2015) pp. 25761-25771

http://www.ripublication.com

Fencing of Unfriendly Nodes and Optimized Network Topology With Remote Procedure Call

Ansilla J.D[1], Dr. N.Vasudevan[2] and Dr. S.Ravi[3]
[1]*Research Scholar, CSE Department, Dr. M.G.R. Educational and Research Institute, Chennai*
Email:ansilla@drmgrdu.ac.in
[2]*Dean [Academics], Hindustan University, Chennai*
Email:vasu_adth@yahoo.com
[3]*Professor & Head, ECE Department, Dr.M.G.R.Educational and Research Institute, Chennai*
Email:ravi_mls@yahoo.com

Abstract

In the network security domain, although there exist several assured secure algorithms and techniques; there exists the presence of several issues, various attacks and the enormous indecorous behavior of the attackers. Moreover, an efficient network topology is sure enough to possess a proficient network. When intruders try to gain control over the network, they are to be pointed out so that a piracy free network would be assured. This work focuses to provide an optimized network topology, fence the unfriendly nodes from the entire network and to identify the intruders.

Keywords: Ghost Vulnerability, Packet Capture, STONITH, Remote Procedure Call

Introduction

There is vast presence of attacks in the network and security domain. It is myth that Linux is more secure. But however, there exists enough vulnerability in the Linux installed servers too. These are been proven every now and then due to the rise of the security threat issues that alerts the entire network security domain. Few among the security issues that frightened the entire network security are such as the ghost vulnerability and the CommView for Wifi Packet Capture. The main objective is to optimize network topology for the cluster of nodes, identify intruder based on the characteristics of the performance of a node. The intruder is identified and STONITH / STOMITH is used to isolate the intruder. STONITH is used either to isolate the failed node while allocating the network topology or to isolate any intruder if found.

The network cluster is topologies as the dining philosophers' problem is handled. The characteristics of the node such as the response time, processing time, battery status (if battery operated) etc are analyzed to determine if they are active in the network or if the particular node act as an intruder.

Ghost Vulnerability

The Ghost vulnerability is a critically rated security threat issue reported by the Qualys researchers on Jan 26, 2015 which threatened the entire Linux based group of networked computers. The issue was termed as "Ghost" since it is triggered with the get host by name () functions in the GNU C Library (glibc) of the Linux OS. 'glibc' is the main library of C language functionality and is present on most linux distributions of the Linux OS. This GNU glibc CVE_2015_0235 Remote Heap Buffer Overflow Vulnerability (CVE_2015_0235) is a buffer overflow in the _nss_hostname_ digits_dots() function used by gethostbyname( ) function calls. This bug is reachable by both local and remote means, via, function calls. This vulnerability attracts the context-based attackers since it allows the remote attackers to access and have a full control over the victim computer although no prior knowledge of system credentials is familiar, with the permission of the user. Using this vulnerability, the attacker targets a system with specially crafted e-mail to a mail server and could do DNS type of things like a getting a remote shell prompt. This Remote Code Execution [RCE] exploit is more or less similar to the super-secure blackphone bug, in which a text message causes the software of a phone to trip over itself.

In basic, gethostbyname( ) functions is a part of Berkeley Sockets which could convert the hostname to an IPV_4 address. The gethostbyname() and gethostbyname2() find the computer friendly network number (eg.:192.16.2.180) of a host from its human friendly name (eg.: google.com), i.e., these functions perform a DNS [Domain Name System] look up which facilitates the coder to skip of the complexities or the undertakings of the DNS protocol. The function miscalculates the size needed to store the data and as a result, no enough memory is allocated and the buffer overflow occurs. The gethostbyname() function is used by any software initiating network connections, log processing, mail / spam filtering, many servers and the gethostbyname() function is unused in the software using getaddrinfo() function.

In simple words, it is explained similar to the following scenario. On handling a 4byte network number written out as decimal digits, something not longer than 255.255.255.255 could be obtained, which takes upto 15 characters and is safe when 16 bytes of memory is allocated. But all of a sudden, when a malicious user, violating the rules uses a number like 1024.10224.102224.1022224, the program might not hold enough to reject it. Probably, the code crashes right away since the 25 bytes of the attacker overflow the available 16bytes of memory. This attack could point to Denial of Service and Distributed Denial of Service type of attacks. These attacks could be induced by the Remote Procedure Calls for the overflow of the buffer. Hence, a remote procedure call overflows the buffer memory and causes the Denial of Service / Distributed Denial of Service Attack.

Comm View for WiFi Packet Capture

CommView for WiFi is designed to capture and analyze network packets on wireless 802.11 a/b/g/n networks. It gathers information from the wireless adapter and decodes the analyzed data. With this, the list of network connections and the vital IP statistics can be seen and the individual packets could be examined. Packets can be decrypted utilizing user defined WEP or WPA keys and are decoded down to the lowest layer, with full analysis of the most widespread protocols. Full access to the raw data is available. Captured packets can be saved to log files for future analysis. It is such a flexible system that filters make it possible to drop unnecessary packets or capture the essential packets. Also, Aircrack-ng tool is used for retrieving data from captured file using "commview for wifi software. The packet capture is shown in Figure1.

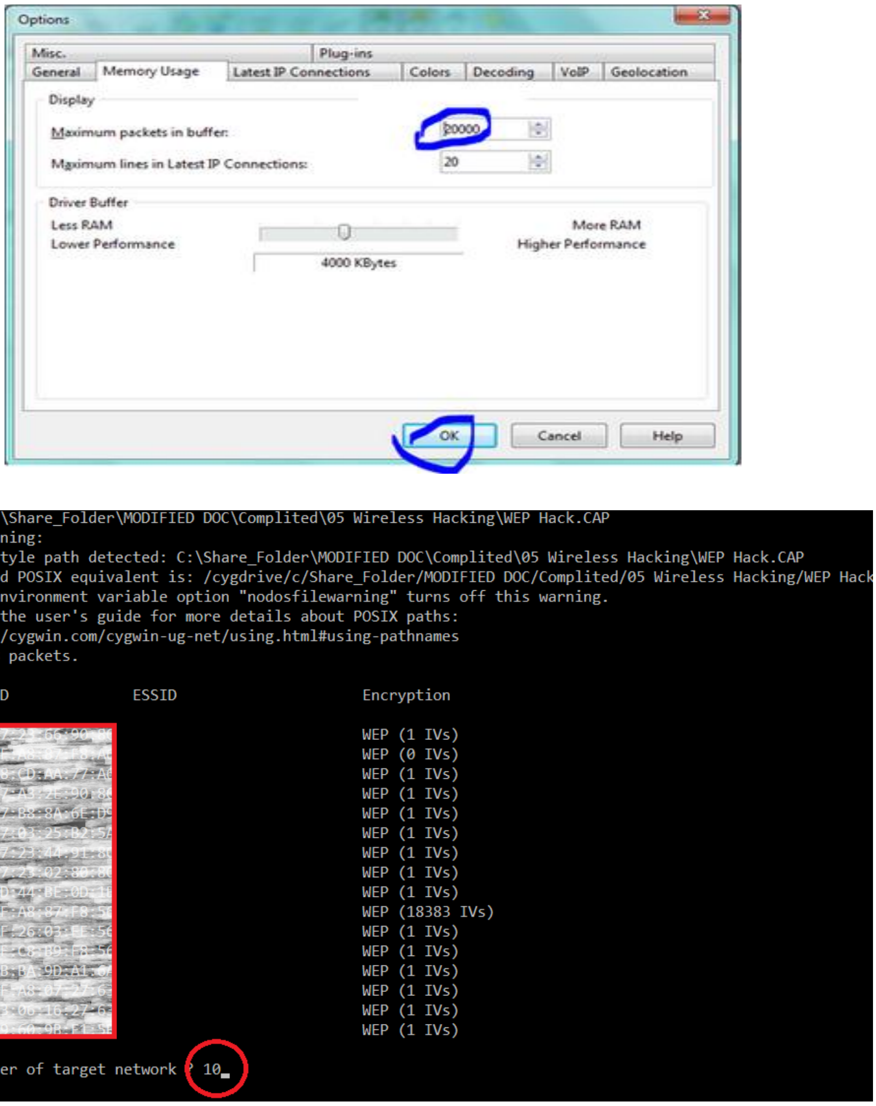


Figure 1: Screenshot demonstrating packet capture

STONITH

STONITH [Shoot The Other Node In The Head] or STOMITH [Shoot The Other Machine In The Head] is a Linux service for maintaining the integrity of nodes in a high-availability [HA] cluster. When the systems function for a longer period of time, there may be chances that either the system malfunctions or the system turns into an intruder. Hence, the each system is been under analysis. Stonith powers down a node that is not working correctly. It is implemented by network solutions that allow a cluster to talk to a physical server without involving the operating system [OS]. But the crisis involved in this is that for a network has to be installed in each server. Hence, a disk-based alternative, the split brain detection [SBD], which requires no specific network, could be implemented. In the SBD Stonith, the nodes in the Linux cluster keep each other updated by using a heartbeat mechanism. If something goes wrong with anode in the cluster, the injured node will terminate itself.

An SBD Stonith approach is simple to set up and is a reliable way to ensure data integrity in a Linux HA clusters. It is easy to implement and reliable. The only requirement is that the environment needs to have shared storage device or Storage Area Network [SAN]. Stonith fences failed nodes by resetting or powering down the failed / compromised node. In Stonith, the high availability system declares a node as head. Then it performs it in reality.

Stonith devices may be classified into five categories: UDS [Uninterruptible Power Supply], PDU [Power Distribution Unit], Blade power control devices, Lights out devices, Testing devices.

The Stonith subsystem consists of two components: Stonithd, Stonith plugins. Stonithd is a daemon which could be accessed by the local processes or over the network. It accepts the commands corresponding to the fencing operations such as reset, power-off, and power-on. If may also check the status of the fencing device. Stonithd runs on every node in the High Availability cluster. There is a Stonith plugin available for every Stonith device, ie, a device to force the nodes/resources, if necessarily must install a Stonith plugin. The Stonith plugin acts as an interface to the fencing device.

A STONITH has the following properties:

1. It must never report false positives for reset.
2. It must support the reset command [with optional ON & OFF].
3. When a RESET or OFF command is given, it must not return the control to its caller until the node is no longer running waiting until it comes up again for RESET is optional.
4. All the commands should work in all circumstances.
    a) RESET when node is ON or OFF should succeed & bring the node up.
    b) OFF when node is OFF should succeed.
    c) ON when node is ON should succeed.

Remote Procedure Call [RPC]

As any programmer knows, procedure calls are a vital software development technique. They provide the leverage necessary for the implementation of all but the most trivial of programs. Remote procedure calls (RPC) extend the capabilities of conventional procedure calls across a network and are essential in the development of distributed systems. They can be used both for data exchange in distributed file and database systems and for harnessing the power of multiple processors. Linux distributions provide an RPC version derived from the RPC facility developed by the Open Network Computing (ONC) group at Sun Microsystems.

The Remote Procedure Call Mechanism

For an RPC to execute successfully, several steps as depicted in figure 2 must take place. The list of operations happening between a client and a server, amongst their corresponding filters and data correspondingly are clearly visualized. The Client encodes data through the XDR Filter. The Client passes the XDR encoded data to the remote host across the network. Server decodes data through XDR filter. Then, the Server encodes the function call result through the XDR Filter. The Server passes the XDR encoded data across the network back to the client. The Client decodes the RPC result through XDR Filter and continues the processing.

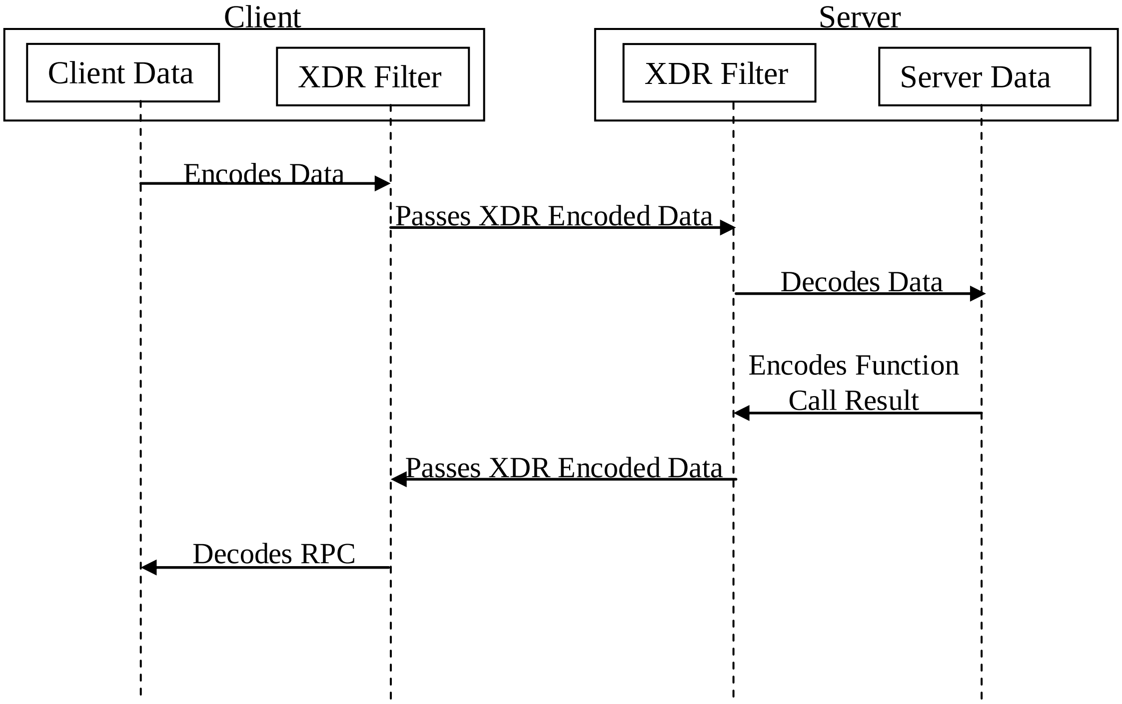


Figure 2: Remote Procedure Call Mechanism

External Data Representation

An RPC can be executed between two hosts that run completely different processor hardware. Data types, such as int and float, can have different physical representations on different machines. The solution to this problem involves the adoption of a standard for data interchange. One such standard is the Open Network Computing external data representation (XDR). XDR is essentially a collection of C functions and macros that enable conversion from machine specific data representations to the corresponding standard representations and vice versa. It contains primitives for simple data types and provides the capability to define and transport more complex ones. Most of the XDR functions require the passing of a pointer to a structure of "XDR" type.

Possible Value Of XDR	**Function**
XDR_ENCODE Operation	instructs the XDR routine to convert the passed data to XDR format
XDR_DECODE Operation	indicates the conversion of XDR represented data back to its local representation
XDR_FREE	provides a means to de-allocate memory that was dynamically allocated for use by a variable that is no longer needed

RPC Data Flow

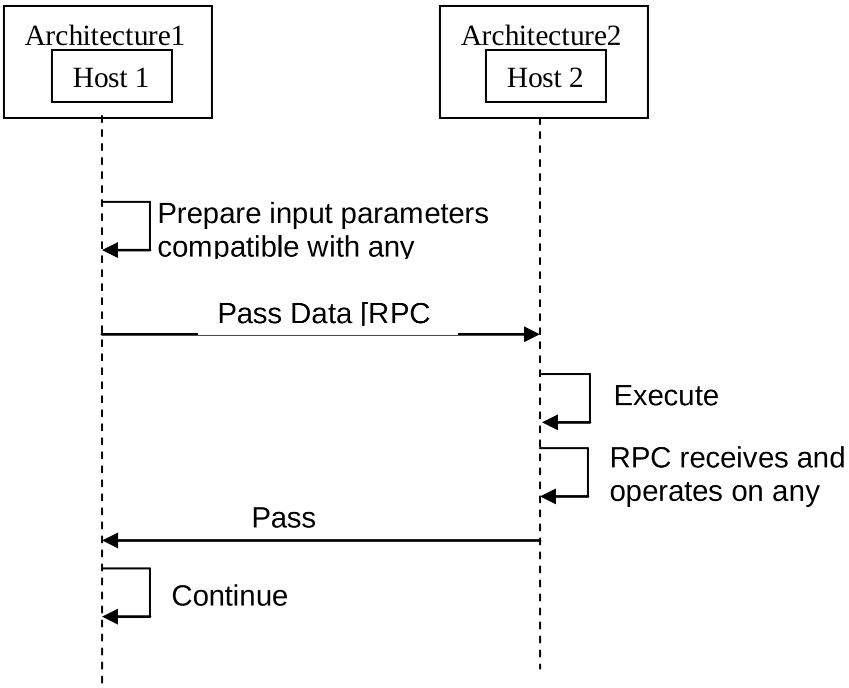


Figure 3: RPC Dataflow

The flow of data from caller to called and back again is illustrated in Figure 3. The calling program executes as a client process and the RPC runs on a remote server. All data movement between the client and the network and between the server and the network pass through XDR filter routines. In principle, any type of network transport can be used, but our discussion of implementation specifics centers on ONC RPC which typically uses either Transmission Control Protocol routed by Internet Protocol (the familiar TCP/IP) or User Datagram Protocol also routed by Internet Protocol (the possibly not so familiar UDP/IP). Similarly, any type of data representation could be used, but our discussion focuses on XDR since it is the method used by ONC RPC.

Implementation of RPC

RPCGEN: The Protocol Compiler

Source File fileremote.x

If the discussion of the mechanisms supporting RPC sounds complex, that's because it is. Fortunately, the development of RPC applications can be greatly simplified through the use of rpcgen, the protocol compiler. rpcgen has its own input language which is used to declare programs, their procedures and the data types for the procedures' parameters and return values. The source code is stored in a file called fileremote.x and rpcgen is invoked with the command “rpcgenfileremote.x”

Header File fileremote.h

The header file fileremote.h contains all of the function prototypes and data declarations needed for the development of the application. It also generated three other source files:

a. **fileremote_clnt.c** : the stub program for client (caller) process
b. **fileremote_svc.c** : the main program for server (called) process
c. **fileremote_xdr.c** : the XDR routines used by both the client and the server

These sources are to be used “as is” and must not be edited.

Server End

To complete the application at the server end, code is necessary to provide the actual “smarts” required to correctly process the input data. This is created manually. This code takes the XDR decoded array from the client and separates and averages the values. It returns the result which is then XDR encoded for transmission back to the client.

Client End

To complete the application at the client end, the input data must be packed into XDR format, so that it can be sent to the server. The client program is also generated manually.

Remote Procedures

The procedures stored on the Server will be remotely called from the Client running on the board. The following are the Remote procedures that the client calls on the server:

- openremote_1_svc
- storeremote_1_svc
- closeremote_1_svc

Source Code For Client

The details of the client code includes

1. Get server name and remote file name from the command line
2. Setup client connection (per *rpcgen* documentation)
3. Open the remote file using openremote_1 ()
4. Generate *struct* for the server to store and store it using storeremote_1()
5. Close the remote file using closeremote_1 () and exit

Source Code For Client

The server just hosts three methods that will be called from the client

- **openremote_1_svc** : opens the file for read write
- **storeremote_1_svc** : Stores the data sent from the client to the file
- **closeremote_1_svc** : closes the open file

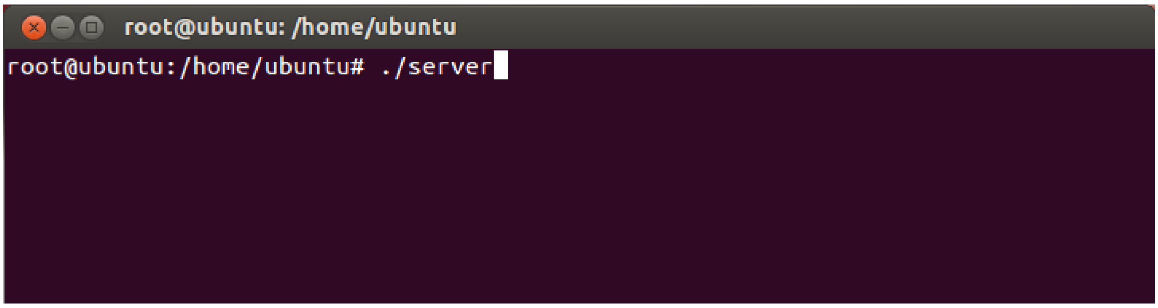
Output

STEP 1 : Server side

sudo./server

If Server does not run on the Ubuntu PC, an additional step is required.

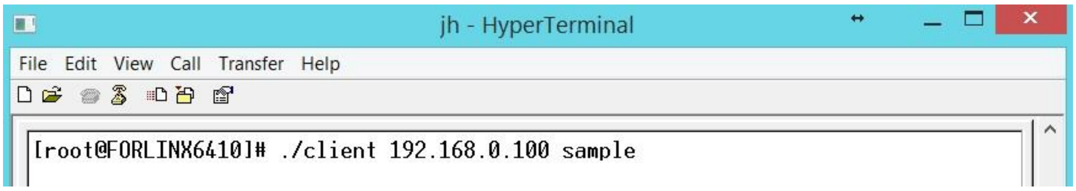
apt-get install rpcbind



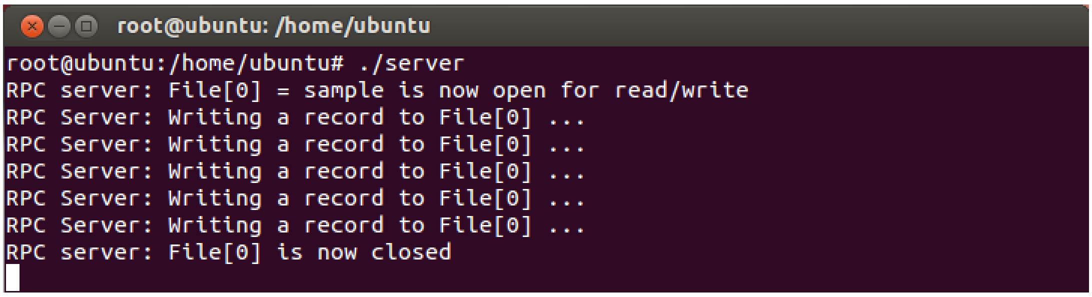
STEP 2 : Run the client on the board:

./client 192.168.0.100 sample

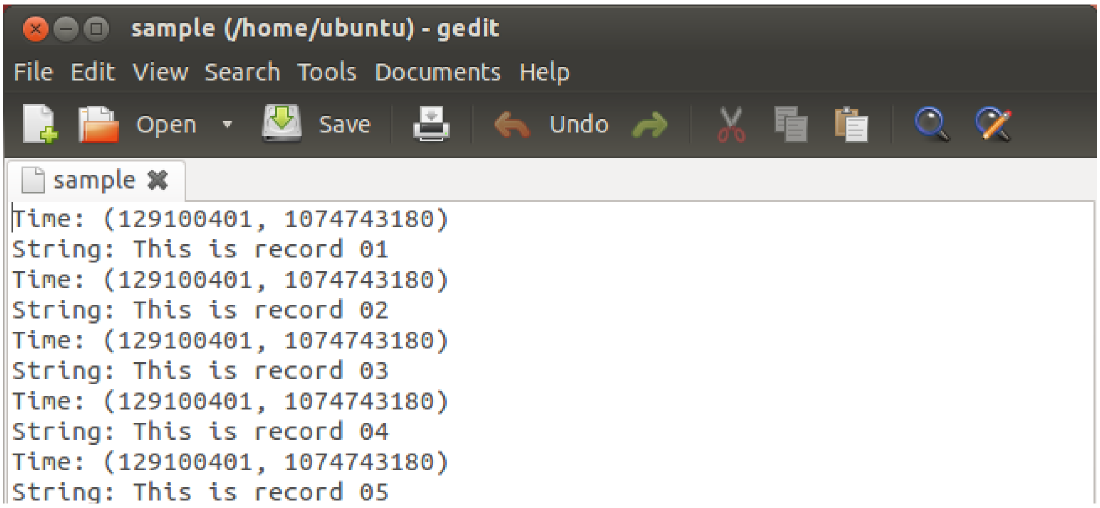
where 192.168.0.100 is the ip address of the Ubuntu PC; sample is the filename that will be created on the server.



STEP 3 :
Once the Client program is run on the board, the output could be seen on the Ubuntu PC:



STEP 4 :
The contents of the file can be viewed on the Ubuntu PC by issuing the following command: **gedit sample**



Conclusion
The serious security threat issue pointed out by the Qualys researchers recently is been analyzed. Although, the existence of patches from various Linux distributions solved the problem, the buffer overflow caused due to the remote procedure call is been identified and is been implemented. The presence of the intruders is being identified and they are been fenced out. On the analysis of the intruders, as and when the unfriendly nodes are eliminated from the network, the network topology is also

analyzed and an efficient network topology is provided. Thus, the work nevertheless fails to provide an intruder free, optimized network topologised and implemented through the remote procedure calls methods.

References

[1] P. Andris and K. Dobrovodsky, “Developing an embedded system based on a real-time version of Linux,”*23rd International Conference on Robotics in Alpe-Adria-Danube Region*, 2014, pp. 1–7.

[2] Y. Chen and A. Zhu, “Implementation of Linux centralized user authentication and cloud storage in teaching,”*International Conference on Information Science, Electronics and Electrical Engineering*, 2014, vol. 1, pp. 626–628.

[3] Y. A. Enaya and K. Deb, “Network path optimization under dynamic conditions,” *IEEE Congress on Evolutionary Computation*, 2014, pp. 2977–2984.

[4] X. Li, P. Djukic, and H. Zhang, “Zoning for hierarchical network optimization in software defined networks,”*IEEE Network Operations and Management Symposium*, 2014, pp.1–8.

[5] J. Networks, “Juniper Secure Analytics Packet Capture Quick Reference Guide,” 2015.

[6] W. Paper, “Ethanalyzer : Cisco NX-OS Software Built-In Packet Capture Utility,” no. March, pp. 1–12, 2015.

[7] H. Sarbazi-Azad and /A/. Zomaya, *Large Scale Network-Centric Distributed Systems*. Hoboken, New Jersey: John Wiley & Sons, Inc., 2013.

[8] M. G. Sobell, *A Practical Guide to Ubuntu Linux*. Pearson Education, 2010, p. 1320.

[9] J. Stuckman and J. Purtilo, “Mining Security Vulnerabilities from Linux Distribution Metadata,”*IEEE International Symposium on Software Reliability Engineering Workshops*, 2014, pp. 323–328.

[10] H. Subramoni, K. Kandalla, et.al., “Design and Evaluation of Network Topology-/Speed- Aware Broadcast Algorithms for InfiniBand Clusters,” in *2011 IEEE International Conference on Cluster Computing*, 2011, pp. 317–325.

[11] S. Van Vugt, *Practical Guide to Xen High Availability: Configuring Enterprise Virtualization on Suse Linux Enterprise Server*. Books4Brains, 2010, p. 134.

[12] J. Wei, E. Juarez, et.al., “A Linux implementation of the energy-based fair queuing algorithm on an ARM-based embedded system,”*IEEE International Conference on Consumer Electronics*, 2014, pp. 546–547.

[13] “Ghost Vulnerability.”Online: https://www.youtube.com/watch?v =zHRRLsZtWAA.

[14] "How to Capture, Filter and Inspect Packets." [Online]. Available: http://www.howtogeek.com/104278/how-to-capture-filter-and-inspect-packets/.

[15] "Chapter 4. Capturing Live Network Data." [Online].https://www.wireshark.org/docs/ wsug_html_chunked/ChapterCapture.html. [Accessed: 26-Mar-2015].