USING COMMON LINUX COMMANDS TO TRACE THE ORIGINS OF POTENTIALLY ROGUE PROCESSES WITHIN A LINUX HOST (VIRTUAL MACHINE)

DENNIS GUSTER
SAINT CLOUD STATE UNIVERSITY
dcguster@stcloudstate.edu

Martin Smith
SAINT CLOUD STATE UNIVERSITY
smma0901@stcloudstate.edu

Laura Lebentritt
UNIVERSITY of Maryland University College
lauraleigh4097@att.net

ABSTRACT

The literature indicates that cloud computing offers many advantages, but also involves new and substantial security risks. Because many virtual machines (VM) are often configured in a cloud based architectures it is difficult to keep track of each service running within the cloud. A very useful tool in evaluating the purpose of those numerous ports is the command set provided by the LINUX operating system. This paper used a complex remote procedure call (RPC) problem that generated a number of dynamically defined ports which appeared as undefined as a back drop to illustrate how LINUX commands could be used to trace the origin of what appeared to be rogue ports. It was found that the suspected rogue ports were in fact defined by the RPC software and legitimate. However, because their purpose was unknown to the system administrator that installed the RPC the firewall definition did not get updated and traffic to the port was unintentionally blocked. It was suggested that greater care be used when defining/evaluating policy and the auditing process be recorded and automated through using LINUX scripts.

INTRODUCTION

Cloud computing offers many advantages, but also involves security risks [1]. Extending the data trust strategy from the enterprise to a cloud architecture requires more than encryption [8]. Given the complexities of a cloud it is often difficult to track the security issue in cloud computing environments given the large number of virtual machines involved [14]. Specifically, in today's world of cloud based architectures it is difficult to keep track of each service running within the cloud. In fact some IT professionals feel that the ramifications of cloud computing and its security challenges are still unknown [17]. Although, the hardware for a small cloud may only involve five physical computing units, virtualization often increases the number of logical computers (virtual machines) to a value in the hundreds. From a green computing perspective reducing hundreds of hosts to a platform of 5 physical hosts is a major advancement. However, for the system administrator the logical model is still very complex and can involve numerous levels of abstraction through the use of virtual zones, virtual hosts, replication and virtual networks. To the casual observer, given its small physical footprint, a cloud may not seem that complex and may even be "disruptively simplified" [8].

Some security issues are unique to a cloud while others are present potentially on any host supporting services. However, cloud computing makes the potential scope of security issues occurring on the services level much more dangerous. Where before only the target host might be affected, now the damage might extend to the entire cloud. This is especially true if single sign-on is supported via the cloud's active directory authentication structure (LDAP). Therefore, there are shared security issues between cloud and traditional computing and it is important to understand that a security issue in traditional computing which is bad could become even worse in a cloud environment [18]. While there should be policy and documentation in place to help a system administration/security team in combating security threats, again the newness and complexity of a cloud means that current policy may not be adequate for a migration to a cloud architecture. In fact, many IT professional have stated that a whole new paradigm is needed to deal with the added complexities of cloud computing [17].

Certainly, it has been established that security issues in a cloud are complex and merit further investigation. This paper hopes to narrow the scope of such an investigation by focusing on the basic concept of auditing the service level port assignments on a virtual machine within a cloud and their potential influence on security within the whole cloud. Generally speaking one would expect that there is policy in place to verify the rationale for each service and its corresponding port. However, the sheer magnitude of services running in the cloud and their complexity can make this a difficult undertaking. The process of identifying the purpose of any given port is further complicated when complex remote procedure calls are used which require multiple ports and some of those ports are dynamically defined upon boot up of the host. Sound policy would involve being able to link a port with a process identification number (or series of related PIDs), the UNIX ID of the service owner, the command that started the process and the files the service has open. In theory this information should all be available via a LINUX command such as netstat –apeen. However, there are instances particularly when remote procedure calls are used where this information is hidden. This paper will perform an audit of a virtual

machine within the author's cloud and trace the origins and purpose of all TCP version 4 services, including the remote procedure calls and annotate the command set required to trace the origins of each service. In some cases such as secure shell it is a simple process. However, in the case of dynamically generated remote procedure call ports it can be a fairly complex process.

REVIEW OF LITERATURE

While the literature makes it clear that cloud computing offers many advantages, but also may involve many risks that need to be assessed [1]. In fact, many would agree that security challenges are the biggest obstacle to the adoption of cloud services [14]. This in part is due to numerous misconceptions about cloud security such as a lack of isolation of data [16]. When in fact if when properly configured a cloud can offer an improved level of trust beyond classical hosting solutions [8].

Cloud computing is still relatively new and the full ramifications of the unique security concerns of this architecture are still unknown [17]. It therefore, becomes crucial to be able to separate traditional security concerns from those that are in fact unique to a cloud architecture [11]. Guidelines have been developed which allow users to select specific security levels and become more aware of the security ramifications of cloud computing [18]. One means of dealing with the uncharted risks inherent in cloud computing is to employ a trusted third party. This party would use public key encryption in concert with SSO (single sign on) and LDAP (lightweight directory access protocol) to ensure the authentication, integrity and confidentiality of the system was assured [18]. Further, this robust authentication schema can be supplemented on the data level with one of the latest public key encryption algorithms [15]. However, the unique aspects of cloud computing which encompass various delivery and deployment models present security issues that sound authentication alone will not solve [15]. A good example of this is side channel attacks [18]. In a common instance of this methodology a hacker obtains a virtual machine (VM) on the same physical host as the VM of the target within a commercial cloud and then mines the common physical memory to obtain authentication information which will then in turn be used to log into that target. Attacks such as these indicate that traditional perimeter security approaches will not be adequate and security must be enhanced on the virtual machine level [16].

Part of the problem with devising a security strategy for cloud computing stems from the lack of security metrics associated with the architecture. For example, cloud computing can support a wide variety of applications. Specifically, an IaaS (infrastructure as a service) client should be able to specify its security needs and hence flexibly defined metrics are needed to clarify this process [17]. Because of the complexity of a cloud perhaps one of the services running in that cloud needs to directly deal with security issues. This idea has been term "policing as a service" in addition to providing data safeguards this service attempts to empower the user by providing interactive monitoring capabilities [17]. This fits with current thought about cloud design which emphasizes centralization of vital services [10]. Even before the fundamental shift to cloud computing the primary attack strategy moved away from attacks on the network and the

transport layer to the application layer so therefore security within the cloud needs to succinctly address all layers. In other words, a user will be running his/her application on somebody else's hardware using somebody else's software so therefore that somebody else better have a sound cloud computing security strategy [5].

AUDITING THE PORTS

Although there are commonly UDP and TCP version 6 ports present in Linux based virtual machines, to narrow the topic it was decided to concentrate the discussion on TCP version 4 ports. To audit such ports the first step is relatively simple and involves using the netstat command to display the basic configuration information.

```
guster@os:~$ netstat -a | more
tcp        0      0 *:40811                 *:*
LISTEN
tcp        0      0 *:54478                 *:*
LISTEN
tcp        0      0 *:sunrpc                *:*
LISTEN
tcp        0      0 *:webmin                *:*
LISTEN
tcp        0      0 *:ssh                   *:*
LISTEN
```

The first step in evaluating such information is to evaluate the known ports (the ones with English names) and ascertain they are supposed to be running and are consistent with site policy. Named ports typically will appear in the services file and that file can be used to determine their numeric value. For example the WEBMIN port is displayed below and has been assigned port 10000. This port listing is defined and maintained by IANA. It is also possible to create a services.local file where site specific ports can be documented and such an example is also displayed below. In this case the audit revealed that all three of the named ports were legitimate. The SUNRPC port was needed to support the centralized file system within the cloud that allowed for centralized storage/management of data. Further, this file system could be exported to potentially all hosts in the cloud and therefore a user would have his/her data follow him/her no matter which host he/she logged into. Specifically, this file system is a common RPC (remote procedure call) named NFS (network file system). The WEBMIN port was needed so that system administrators could remotely manage the system and the SSH port was need so that client would have secure virtual terminal access to the host. However, the other two ports at this point remain unaccounted for.

```
guster@os:~$ cat /etc/services | grep webmin
webmin          10000/tcp

guster@os:~$ cat services.local
udptest         44444/udp    #Guster's special test port for udp
traffic
```

It is not unusual for remote procedure calls to be complex and hence use multiple ports. To get a better understanding of this process let's take a look at the structure of the modules associated with the NFS daemon. One can see that there is a specific module that deals with data concurrency issues (lockd), one that deals with authentication issues (auth_rpcgss) and one that controls queries to the access control list (nfs_acl). Also note that the port that ties all of the subparts together termed the "port map" (sunrpc) is also listed.

```
guster@os:~$ lsmod | grep nfs
nfsd                  281980  2
nfs                   436929  1
lockd                  90326  2 nfsd,nfs
fscache                61529  1 nfs
auth_rpcgss            53380  2 nfsd,nfs
nfs_acl                12883  2 nfsd,nfs
sunrpc                255224  12 nfsd,nfs,lockd,auth_rpcgss,nfs_acl
```

How then do these modules get linked to port addresses? Other than the port map (sunrpc) and the prime NFS service itself which are defined in the service file, the rest of the ports are dynamically assigned upon booting up the host. Which can make access outside of the cloud (or even the host) tricky if some means of updating the firewall after every reboot is not considered. Because RPCs use the client/server model it is important to look at the port maps for both sides. In the example below 10.18.59.34 is the NFS client and 10.18.59.35 is hosting the NFS service. On both sides, of course SUNRPC is running to provide the port mapping. The status service provides crash-and-recovery functions for the locking services and is needed on both the client and server. However, note that the server is also running the NLOCKMANAGER which manages daemons related to data locking and the (mount daemon) which answers requests from clients for file system mounts. By reading the information from the port map below we can reconcile one of the two unaccounted for ports from the original netstat output, 54478 which is providing the status function and therefore can now be deemed legitimate.

```
guster@os:~$ rpcinfo -p 10.18.59.34
   program vers proto   port  service
    100000    4   tcp    111  portmapper
    100000    3   tcp    111  portmapper
    100024    1   tcp  54478  status

guster@os:~$ rpcinfo -p 10.18.59.35
   program vers proto   port  service
    100000    4   tcp    111  portmapper
    100000    3   tcp    111  portmapper
    100024    1   tcp  41364  status
    100003    4   tcp   2049  nfs
    100021    4   tcp  40141  nlockmgr
    100005    3   tcp  54093  mountd
```

How then is data transferred between the client and server and how secure is it? By displaying the file systems (df) we can see that the NFS file system is nfs.bcrc.local (10.18.59.35) and is mounted on the client side as /myhome. The full connection information can be obtained by using the netstat command and as expected the server side port is 2049 as would be listed in the services file. However, the client port is 671 it is protected and not in the services file. Its protected status is due to the fact that it is a "multiplexed" client port, meaning multiple users on the client side will use the port rather than forming their own connection when using the NFS service. This reduces the amount connection management overhead.

```
guster@os:~$ df
Filesystem      1K-blocks    Used Available Use% Mounted on
/dev/sdbe        15480816 3826724  10867712  27% /
udev              2015964       4   2015960   1% /dev
tmpfs              809912     284    809628   1% /run
nfs.bcrc.local:  41283968 1180416  38006400   4% /myhome

guster@os:~$ netstat -tn | grep 2049
tcp        0      0 10.18.59.34:671       10.18.59.35:2049
ESTABLISHED
```

To make multiplexing work another addressing layer is needed. To illustrate this, the packet sniffer TCPDUMP is used to trap a NFS packets. The packets trapped are recorded in a file called dump2049 and one packet from that file is displayed below. Note the source address: 10.18.59.34.1421250189. What appears to be a large undefined client port is really a transaction ID (XID). Because the maximum port address space is only 64KB the number displayed, 1421250189, is too large. Its job is to uniquely identify a data transaction and make sure it gets ultimately delivered to the correct user and session. Interestingly, it is still possible to determine the client port by reading the dump. By looking in the third row, second group of four hex numbers we see a value of 029f which when converted to decimal yields a value of 671. Looking back at the connection information from the netstat command we see that this value matches the client port. As implemented here this version of NFS is not encrypted, although there are certainly versions of NFS that are. Because this host is used for educational purposes (such as illustrating security vulnerabilities) encryption is not deemed necessary. However, access is limited to inside the cloud and a private network address used so that the service is not available directly to the internet.

```
guster@os:~$ sudo tcpdump port 2049 -e -n -vvv -X -c111 > dump2049

guster@os:~$ cat dump2049 | more
10:02:20.062045 00:50:56:8c:05:a8 > 00:50:56:8c:0a:0e, ethertype IPv4 (0x0800),
length 338: (tos 0x0, ttl 64, id 2082, offset 0, flags [DF], proto TCP (6),
leng
th 324)10.18.59.34.1421250189 > 10.18.59.35.2049: 268 getattr fh 0,0/22
        0x0000:  4500 0144 0822 4000 4006 2bc2 0a12 3b22  E..D."@.@.+...;.
        0x0010:  0a12 3b23 029f 0801 3201 aa9c e815 37bb  ..;.....2.....7.
        0x0020:  8018 577f 0707 0000 0101 080a c836 82d2  ..W..........6..
        0x0030:  8694 2242 8000 010c 54b6 8e8d 0000 0000  .."B....T.......
```

```
0x0040:  0000 0002 0001 86a3 0000 0004 0000 0001  ................
0x0050:  0000 0001 0000 0030 01d3 297b 0000 000f  .......0..){....
0x0060:  0000 6f73 2e62 6372 632e 6c6f 6361 6c00  ..os.bcrc.local.
0x0070:  3be0 0471 3be0 0201 0000 0003 3be0 0201  ;..q;.......;...
0x0080:  3be0 0466 3be0 0940 0000 0000 0000 0000  ;..f;..@........
0x0090:  0000 0000 0000 0000 0000 0003 0000 0016  ................
0x00a0:  0000 0010 0100 0101 0000 0000 6c17 0c00  ............l...
0x00b0:  6841 f8df 0000 0026 0000 0001 4f83 6c52  hA.....&....O.lR
0x00c0:  c20e 0000 5d94 0000 0000 0000 0000 0000  ....]...........
0x00d0:  0000 0002 0000 0059 7468 6973 2069 7320  .......Ythis.is.
0x00e0:  6120 7365 6372 6574 2066 696c 6520 646f  a.secret.file.do
0x00f0:  6e27 7420 7465 6c6c 206f 7572 2063 6f6d  n't.tell.our.com
0x0100:  7065 7469 746f 7273 2061 7420 4d53 552d  petitors.at.MSU-
0x0110:  4d61 6e6b 6174 6f20 7468 6520 6b65 7977  Mankato.the.keyw
0x0120:  6f72 6420 6973 2064 6f75 6768 6e75 7421  ord.is.doughnut!
0x0130:  0a00 0000 0000 0009 0000 0002 0000 0018  ................
0x0140:  0030 0000                                .0..
```

There still is one port unaccounted for, port 40811! A good strategy might be to try to obtain the user, process ID and command that started the process/port. A sophisticated version of the netstat command is used below and run as the root (via sudo). The results are useful for the NFS status port (54478) and the port map (111). For example, port 54478 is owned by user 106 which is the statd (status daemon), running with process ID 1005 and was started by the command rpc.statd. However, the information is very limited for port 40811 with only the user ID being provided which is 0 (root). The problem is that the root starts lots of software and using the ID 0 for the root is a no brainer for a hacker. So obviously another strategy is needed to track down the origins of this port. There is an inode address list for the port 40811 which is 8379, however using that route proved relatively ineffective, time consuming and complex. So it was decided to go to the heart of the operating system and try to determine if there was useful information in the kernel ring buffer.

```
guster@os:~$ sudo netstat -apeen |more
tcp  0.0.0.0:40811   0.0.0.0:*  LISTEN      0        8379      -
tcp  0.0.0.0:54478   0.0.0.0:*  LISTEN      106      681       1005/rpc.statd
tcp  0.0.0.0:111     0.0.0.0:*  LISTEN      0        649        973/rpcbind
```

The dmesg command allows direct access to the kernel ring buffer log information. When specifically searching for information on port 40811 the information below appeared. It was clear that a connection attempt was being made from 10.18.59.34. This was reassuring because that host was a trusted host within the same cloud and the NFS service provider for the cloud.

```
guster@os:~$ dmesg -kTx | grep 40811 | more
kern  :warn  : [Wed Feb  5 12:55:22 2014] [UFW BLOCK] IN=eth0 OUT=
MAC=00:50:56:8c:05:a8:00:50:56:8c:0a:0e:08:00 SRC=10.18.59.
35 DST=10.18.59.34 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=18996 DF
PROTO=TCP SPT=744 DPT=40811 WINDOW=14600 RES=0x00 SYN URGP=0
```

To get a better idea of the characteristics of that connection attempt a packet sniffer was employed. The session consisted of only SYN packets (request to connect) coming from 10.18.59.35 which were evidently not received because there was no response.

```
guster@os:~$ sudo tcpdump port 40811 -e -n -vvv -X -c111 > dump40811

guster@os:~$ cat dump40811
09:36:58.060989 00:50:56:8c:0a:0e > 00:50:56:8c:05:a8, ethertype IPv4 (0x0800),
length 74: (tos 0x0, ttl 64, id 23833, offset 0, flags [DF], proto TCP (6),
length 60)10.18.59.35.725 > 10.18.59.34.40811: Flags [S], cksum 0xf904
(correct), seq 2056391044, win 14600, options [mss 1460,sackOK,TS val
2257478464 ecr 0,nop,wscale 4], length 0
        0x0000:  4500 003c 5d19 4000 4006 d7d2 0a12 3b23  E..<].@.@.....;.
        0x0010:  0a12 3b22 02d5 9f6b 7a92 0984 0000 0000  ..;....kz.......
        0x0020:  a002 3908 f904 0000 0204 05b4 0402 080a  ..9.............
        0x0030:  868e 6340 0000 0000 0103 0304            ..c@........
09:37:14.092788 00:50:56:8c:0a:0e > 00:50:56:8c:05:a8, ethertype IPv4 (0x0800),
length 74: (tos 0x0, ttl 64, id 23834, offset 0, flags [DF], proto TCP (6),
length 60)10.18.59.35.725 > 10.18.59.34.40811: Flags [S], cksum 0xe95c
(correct), seq 2056391044, win 14600, options [mss 1460,sackOK,TS val
2257482472 ecr 0,nop,wscale 4], length 0
        0x0000:  4500 003c 5d1a 4000 4006 d7d1 0a12 3b23  E..<].@.@.....;.
        0x0010:  0a12 3b22 02d5 9f6b 7a92 0984 0000 0000  ..;....kz.......
        0x0020:  a002 3908 e95c 0000 0204 05b4 0402 080a  ..9..\..........
        0x0030:  868e 72e8 0000 0000 0103 0304            ..r.........
```

To determine if port 40811 was functional, a telnet session was tried and the result logged. It was determined that the port was functional because a connection was made and data could be transferred, although not intelligently because the telnet client was unable to match the protocol that port 40811 was expecting.

```
guster@os:~$ sudo tcpdump port 40811 -e -n -vvv -X -c111 -i lo >> dump40811

guster@os:~$ telnet localhost 40811
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
letmein
Connection closed by foreign host.

guster@os:~$ sudo tcpdump port 40811 -e -n -vvv -X -c111 -i lo >> dump40811

guster@os:~$ cat dump40811 | more
10:20:43.417265 00:00:00:00:00:00 > 00:00:00:00:00:00, ethertype IPv4 (0x0800),
length 75: (tos 0x10, ttl 64, id 7566, offset 0
, flags [DF], proto TCP (6), length 61)
    127.0.0.1.44737 > 127.0.0.1.40811: Flags [P.], cksum 0xfe31 (incorrect ->
0xe856), seq 1:10, ack 1, win 513, options [nop,n
op,TS val 3359291472 ecr 3359287999], length 9
        0x0000:  4510 003d 1d8e 4000 4006 1f1b 7f00 0001  E..=..@.@.......
        0x0010:  7f00 0001 aec1 9f6b 7200 b734 9106 d419  .......kr..4....
        0x0020:  8018 0201 fe31 0000 0101 080a c83a b850  .....1.......:.P
        0x0030:  c83a aabf 6c65 746d 6569 6e0d 0a         .:..letmein..
```

```
10:20:43.419461 00:00:00:00:00:00 > 00:00:00:00:00:00, ethertype IPv4 (0x0800),
length 66: (tos 0x0, ttl 64, id 32422, offset 0
, flags [DF], proto TCP (6), length 52)
    127.0.0.1.40811 > 127.0.0.1.44737: Flags [.], cksum 0xfe28 (incorrect ->
0x9917), seq 1, ack 10, win 512, options [nop,nop,
TS val 3359291473 ecr 3359291472], length 0
        0x0000:  4500 0034 7ea6 4000 4006 be1b 7f00 0001  E..4~.@.@.......
        0x0010:  7f00 0001 9f6b aec1 9106 d419 7200 b73d  .....k......r..=
        0x0020:  8010 0200 fe28 0000 0101 080a c83a b851  .....(.......:.Q
        0x0030:  c83a b850                                .:.P
```

A quick look at the UNIX firewall log verifies that the traffic was in fact being blocked on the firewall level. This is further confirmed by trying to telnet to port 40811 from the host nfs (10.18.59.35). So on the surface it appears that port 40811 could be a legitimate port that is used by the NFS service, but communication is being blocked by the firewall. Certainly based on the investigation herein there does not appear to be any dangerous traffic getting to that port. However, further investigation is warranted.

```
guster@os:/var/log$ sudo cat ufw.log | more

Feb 23 11:47:44 eros kernel: [14738071.383243] [UFW BLOCK] IN=eth0 OUT=
MAC=0e:50:56:8c:05:a6:00:50:56:8c:8a:0f:08:00 SRC=10.18.59.35
DST=10.18.59.34 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=52280 DF PROTO=TCP
SPT=1009 DPT=40811 WINDOW=14600 RES=0x00 SYN URGP=0

guster@nfs:~$ telnet 10.18.59.34 40811
Trying 10.18.59.34...
telnet: Unable to connect to remote host: Connection timed out
```

DISCUSSION/CONCLUSIONS

Most installations adopting a cloud architecture can be expected to have sound policy in place regarding port assignment. Typically, the ports that are supporting services are defined through a services file and there is documentation that describes their need and purpose. The example used in this paper illustrated that when complex remote procedure calls are used that additional ports can be generated dynamically and may slip through or affect the efficiency of the defined security policy.

The example used herein involving port 40811 was a good example of a port that appeared to have been created by malware used by hackers to perhaps provide them a "back door" to the host. However, this potentially rogue port was created by the installation of the NFS software. Therefore, it is critical when installing or updating software to take a snapshot of the port structure and compare the before and after status of the ports. In the case of ports being defined dynamically knowing the purpose is critical and the process or command set to determine that process must be well defined. In the example used herein it was a multistep process using several LINUX commands. If

the process was well documented the process could be streamlined and performed quicker.

The primary sentinel for filtering out bad data coming into the cloud or within the cloud, of course, would be the firewall. It is common place to have a firewall at the internet access point of the cloud and other firewalls within the cloud even down to the host level. In the case of port 40811 the host level firewall prevented the client process from the host nfs from even forming a connection and this block was recorded in the UNIX firewall log. Log files such as these are valuable in not only detecting problems, but in evaluating if policy is being followed and if it is its degree of effectiveness. The fact that this legitimate port was blocked relates to the local firewall policy which basically states to block all ports initially and then open only those truly needed as defined by policy. Certainly this is the correct policy but it results in a legitimate port being blocked occasionally. This is especially true when that port is being generated dynamically by a complex remote procedure call. In addition to issues with unknown ports appearing complex remote procedure call can also create other dangerous security issues. In some cases the RPC's require no authentication in other cases they allow inheritance of rights. For example, a user of the RPC on host "A" may have super user rights and unless specifically limited may carry the same rights set to "host "B". For example, if this is a concern in an NFS application there is a "nosuid" parameter that can be set to combat this inheritance problem.

In performing the audit to trace down the rogue port, it is clear that commands provided by the LINUX operating system were quite effective. However, determining which commands and options to use was not a simple process. In fact, the process often involves trial and error and how quickly it can be performed is a function of the experience of the investigator. A cloud installation would be well advised to document and automate the process through scripts so that the code can be reused which in the long run will be quicker and more cost effective. However, there will still be new cases occurring where the problem must be attacked from scratch so having an analyst on staff that truly knows LINUX and how it interacts with the cloud/network infrastructure will be invaluable. Too often upper level management assumes that they can rely totally on an icon based tools, but what happens when there is no icon to solve the problem available? This culture has reduced the number of such analysts needed, but in cases where new problems need to be solved there is still a place for the human problem solver with strong critical thinking skill and LINUX commands are a powerful tool which can assist him/her.

REFERENCES

[1] Anthes, G. (Nov. 2010). Security in the cloud. *Communications of the ACM , vol. 53 Issue 11, pp.16-18.*

[2] Basak, D., Toshniwal, R., Maskalik, S. (et.al). (Dec 2010). Virtualizing networking and security in the cloud. *ACM SIGOPS Operating Systems Review, vol. 44 Issue 4, pp. 86-94, Retrieved from http://bit.ly/1fj5ICO*

[3] Bohli, J., Gruschka, N., Jensen, M., (et.al) (July-Aug 2013). Security and Privacy-Enhancing Multicloud Architectures. *IEEE Transactions on Dependable and Secure Computing, vol. 10, no. 4, pp. 212-224. Retrieved from http://www.computer.org.ezproxy.umuc.edu/csdl/trans/tq/2013/04/ttq2013040212.html*

[4] Caron, E., Le, A., Lefray,A., Toinard,C. Definition of Security Metrics for the Cloud Computing and Security-Aware Virtual Machine Placement Algorithms. *International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, pp.125-131.* Retrieved from http://www.computer.org.ezproxy.umuc.edu/csdl/proceedings/cyberc/2013/5106/00/5106a125.pdf

[5] Green, M. (Jan-Feb 2013). The Threat in the Cloud. *IEEE Security & Privacy, vol. 11, no. 1, pp. 86-89, Retrieved from http://www.computer.org.ezproxy.umuc.edu/csdl/mags/sp/2013/01/msp2013010086.html*

[6] Guster, D. C., Lee, O. F., & Rogers, D. C. (2011). Pitfalls of devising a security policy in virtualized hosts. *Journal of Information Security Research, 2*(2), 75-83.

[7] Hyman, P. (June 2013). Augmented-reality glasses bring cloud security into sharp focus. *Communications of the ACM, vol. 56 Issue 6, pp. 18-20.* Retrieved from http://bit.ly/MGQGyW

[8] Jules, A., Oprea, A. (Feb 2013). New approaches to security and availability for cloud data. *Communications of the ACM, vol. 56, issue. 2, pp. 64-73. Retrieved from http://bit.ly/1gz6XDh*

[9] Kaufman, L. (July-Aug. 2010). Can Public-Cloud Security Meet Its Unique Challenges?. *IEEE Security & Privacy*, vol. 8, no. 4, pp. 55-57. Retrieved from http://doi.ieeecomputersociety.org.ezproxy.umuc.edu/10.1109/MSP.2010.120

[10] Lesk, M. (May-June 2012). The Clouds Roll By. *IEEE Security & Privacy, vol. 10, no. 3, pp. 84-87, Retrieved from http://www.computer.org.ezproxy.umuc.edu/csdl/mags/sp/2012/03/msp2012030084.html*

[11] Mell, P. (July-Aug 2012). What's Special about Cloud Security?. *IT Professional, vol. 14, no. 4, pp. 6-8, Retrieved from http://www.computer.org.ezproxy.umuc.edu/csdl/mags/it/2012/04/mit2012040006.html*

[12] Roberts, J., Al-Hamdani, W. (2011) Proceedings from the 2011 Information Security Curriculum Development Conference. *Who can you trust in the cloud?: a review of security issues within cloud computing. Pages 15-19. Retrieved from http://bit.ly/1bfH3jn*

[13] Seo, S., Nabeel, M., Ding, X. (et.al). (Aug 2013). An Efficient Certificateless Encryption for Secure Data Sharing in Public Clouds. *IEEE Transactions on Knowledge and Data Engineering, 05 Aug. 2013. IEEE computer Society Digital Library*. Retrieved from http://www.computer.org.ezproxy.umuc.edu/csdl/trans/tk/preprint/06574849.pdf

[14] Sun, D., Chang, G., Sun, L., Wang, X. (2011). Surveying and Analyzing Security, Privacy and Trust Issues in Cloud Computing Environments. *Procedia Engineering, vol. 15, pp. 2852-2856.*

[15] Takabi, H., Joshi, J., Ahn, G. (Nov-Dec 2010). Security and Privacy Challenges in Cloud Computing Enviroments. *IEEE Security & Privacy, vol. 8, no. 6, pp. 24-31, Retrieved from http://www.computer.org.ezproxy.umuc.edu/csdl/mags/sp/2010/06/msp2010060024.html*

[16] Viega, J. (July-Aug. 2012). Cloud Security: Not a Problem. *IEEE Security & Privacy, vol. 10, no. 4, pg.3.* Retrieved from http://doi.ieeecomputersociety.org.ezproxy.umuc.edu/10.1109/MSP.2012.93

[17] Zargari, S., Smith, A. (2013). Policing as a Service in the Cloud. *Proceedings from 2013 Fourth International Conference on Emerging Intelligent Data and Web Technologies. Retrieved from http://www.computer.org.ezproxy.umuc.edu/csdl/proceedings/eidwt/2013/2141/00/5044a589.pdf*

[18] Zissis, D., Lekkas, D. (March 2012). Addressing cloud computing security issues. *Future Generation Computer Systems, vol. 28, issue 3, pp. 583-592, Retrieved from http://www.sciencedirect.com.ezproxy.umuc.edu/science/article/pii/S0167739X10002554#*