Article can be accessed online at http://www.publishingindia.com

Identifying and Remediating Rogue Services Within a Cloud Based Virtual Machine

Dennis C. Guster*, Mark B. Schmidt, Karthik Paidi*****

Abstract

While the benefits of Cloud computing are well known, often the security risks involved are new and substantial. The hosts of choice in the cloud, the virtual machine (VM), are created in large numbers. This means that it becomes very difficult to keep track of each service running within the cloud. Fortunately, commands exist within the LINUX operating system that can be used to evaluate the purpose of transport layer ports related to the services running on a given host (VM). The example utilized in this paper is a complex remote procedure call (RPC) service, which generates multiple dynamically defined ports that will be evaluated using LINUX commands. Besides the expected legitimate ports there were also suspected rogue ports. These ports were created as a function of the RPC software, but were not traceable to a process id or the originating executable. The fact that these ports forked from a kernel level process made it difficult to trace their origins. Fortunately, because these ports were generated dynamically and their purpose was not known to the system administrator the firewall block definition was not updated and traffic to that port remained blocked. Simply stated by default the firewall was in place to automatically block unknown traffic whether legitimate or not. In this case the default definition served well. To remediate this problem more care needs to be used when defining/evaluating policy. Additionally, it is suggested that the port evaluation procedure be recorded and automated through the use of LINUX scripts.

Keywords: Cloud Computing, Virtual Machines, Remote Procedural Call (RPC), Port Evaluation

1. Introduction

While cloud computing offers many advantages it also poses a number of security risks (Anthes, G. (Nov. 2010)). Dealing with security issues in a cloud is not for the faint hearted because it requires extending the data trust strategy from the enterprise to a cloud architecture and more than just simple encryption is needed (Jules, A., Oprea, A. (Feb 2013)). The resulting complexity makes it difficult to track security issues within cloud computing environments given the large number of virtual machines involved (Sun, D., Chang, G., Sun, L., Wang, X. (2011)). The situation is further compounded by the fact that each virtual machine supports multiple services running within the cloud. In fact, some IT professional feel that the ramifications of cloud computing and its security challenges are still unknown (Hyman, P. (June 2013)). While the hardware needed to support 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. If the goal is to invoke green computing reducing hundreds of hosts to a platform of five physical hosts is a major advancement. However, the system administrator must still view the cloud from its logical model which is still very complex and can involve numerous levels of abstraction through the use of virtual zones, virtual hosts, replication and virtual networks. Therefore, the casual observer, given its small physical foot print, may view a cloud as not that complex and may even seem to be “disruptively simplified” (Basak, D., Toshniwal, R., Maskalik, S. (et. al). (Dec 2010)).

* Professor of Information Systems, St. Cloud State University, United States. Email: dcguster@stcloudstate.edu
** Professor of Information Systems, St. Cloud State University, United States. E-mail: mark@stcloudstate.edu
*** Information Assurance Graduate Student, St. Cloud State University, United States. E-mail: kpaidi@stcloudstate.edu

While some security issues are unique to a cloud others are present on any host supporting services. However due to its complexity, cloud computing makes the potential scope of security issues occurring on the services level much more dangerous. In the traditional hosting model, only the target host might be affected, however in a cloud structure the damage might extend to the entire cloud. For example, this is especially true if single sign-on is supported via the cloud's active directory authentication structure (LDAP). While authentication is a shared security issue in both cloud and traditional computing it is important to understand that a security issue in traditional computing which is bad could become even worse in a cloud environment (Roberts, J., Al-Hamdani, W. (2011)). Sound policy and documentation are still mainstays in helping a system administration/security team in combating security threats, but the newness and complexity of a cloud means that current policy may not be adequate. In fact, many IT professional have stated that a whole new paradigm is needed to deal with the added complexities of cloud computing (Zargari, S., Smith, A. (2013)).

It is clear that security issues in a cloud are complex and merit further investigation. This paper plans to concentrate and 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. The analysis will then be expanded to the influence a port level vulnerability might cause within the whole cloud infrastructure. The coverage will be expanded to go beyond user_id and process_id in evaluating an attack or vulnerability. It is important to have policy in place to verify the rationale for each service and its corresponding port. This is not a trivial assignment given the sheer magnitude to services running in the cloud. Further,the process of identifying the purpose of any given port is further complicated when complex remote procedure calls are used which require multiple ports. This process is further complicated because some of those ports are dynamically defined upon boot up of the host. Under ideal conditions, sound policy would require 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 that service has open. One would expect that this information should all be available via a LINUX command such as netstat –apeen. Unfortunately, there are cases, particularly when remote procedure calls are used where this information is unavailable which then requires in depth analysis to ensure what appears to be a rogue port isn't dangerous. Specifically, 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. For some services 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.

2. Review of Literature

Given the available literature there is agreement that cloud computing offers many advantages, but due to its complexity involves many risks that need to be assessed (Anthes, G. (Nov. 2010)). Further, there is a sense that security challenges are the biggest obstacle to the adoption of cloud services (Bohli, J., Gruschka, N., Jensen, M., (et.al) (July-Aug 2013)). This in part is due to numerous misconceptions about cloud security such as a lack of isolation of data (Viega, J. (July-Aug. 2012)). However, when one gets past these misconceptions a properly configured a cloud can offer an improved level of trust beyond that of classical hosting solutions (Jules, A., Oprea, A. (Feb 2013)).

Security strategies in a cloud computing are still maturing and therefore, the full ramifications of the unique security concerns of this architecture have yet to be determined (Hyman, P. (June 2013)). Therefore, it becomes critical to be able to separate traditional security concerns from those that are really unique to a cloud architecture (Mell, P. (July-Aug 2012)). To streamline this process guidelines have been developed which allow users to select specific security levels and as a result become more aware of the security ramifications of cloud computing (Roberts, J., Al-Hamdani, W. (2011)). A method of dealing with the unknown risks inherent in cloud computing is to use a trusted third party. Of course, this party would need to use 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 (Zissis, D., Lekkas, D. (March 2012)). Further, this robust authentication schema can be supplemented on the data level with one of the latest public key encryption algorithms (Seo, S., Nabeel, M., Ding, X. (et.al). (Aug 2013)). Unfortunately, sound encryption is not the entire answer because the unique aspects

of cloud computing encompass various delivery and deployment models present security issues that transcend encapsulation (Takabi, H., Joshi, J., Ahn, G. (Nov-Dec 2010)). A prime example of this is side channel attacks (Guster, D. C., Lee, O. F., & Rogers, D. C. (2011)). 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 (in which the data is not encapsulated) to obtain authentication information which will then in turn to log into that target. Therefore, attacks such as these indicate that traditional perimeter security approaches will not be adequate and security must be enhanced within the cloud, its zones and on the virtual machine level (Kaufman, L. (July-Aug. 2010)).

Some of the difficulty in 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 some relatively simple involving a single static port whereas other involve multiple port of which some are assigned dynamically. Specifically, an IaaS (infrastructure as a service) client should be able to specify his/her security needs and hence flexibly defined metrics are needed to clarify this process (Caron, E., Le, A., Lefray, A., Toinard, C). A good example of this would be to run on hardware that contains no VMs controlled by other people, hence lessening the probability of a side channel attack. Due to the complexity of a cloud perhaps one of the services running in that cloud needs to directly deal with security issues. This concept has been termed "policing as a service" in addition to providing data safe guards this service attempts to empower the user by providing interactive monitoring capabilities (Zargari, S., Smith, A. (2013)). This fits with current theory about cloud design that focuses centralization of vital services (Lesk, M. (May-June 2012)). It is important to align security strategy with the type of attacks that are most prevalent. It is important to note that 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 (Green, M. (Jan-Feb 2013)).

The use of RPCs are a good example of the abstraction that takes place in a cloud which leads to complexities in analyzing how secure a given service might be. On the surface the concept of a *remote procedure call* is simple. Its purpose is to transfer process level control and data within a program running on a single computer. Of course in the context of a cloud the mechanism is extended to provide for transfer of control and data across a communication network which may involve multiple VMs. Specifically, a remote procedure is run, the calling environment is put on hold, the required parameters are sent across the network to the VM where the procedure is to execute and then the desired process or processes are executed there remotely. When the process concludes the results are sent back to the initiating VM, where the process put on hold restarts making the whole thing appear as running on a single-machine (Birrell, A., Nelson, B. (1984)).

Not only does the use of RPCs lead to multiple VM being involved in the computing transaction but often multiple ports with their own process stack as well. For example, NFS (network file system) is a RPC that contains multiple daemons (a system level process) each requiring its own port assignment. The install information is scattered among a number of startup scripts which means that a hacker would have ample opportunity to create a rogue process and link the resulting port to the system through the port-map and then masquerade it as a legitimate service (Toxen, B. (2007)).

This complexity can often mask what user truly owns a given process associated with a rogue port. This could be especially true when the returning process is not returned to a user level but, rather handled on return by the operating system kernel (Rushby, J. (1981)). So therefore, it is critical that RPCs implemented securely which among other thing would involve using the same isolation logic that would be applied in basic cloud design (Vahidi, A. (2013)).

3. Auditing the Ports

Given the multi-protocol structure of today's hosts there are commonly UDP and TCP version 6 ports present in Linux based virtual machines, but to narrow the topic it was decided to focus the discussion on TCP version 4 ports. The first step in auditing such ports is relatively simple and involves using the netstat command to provide

basic configuration information.

```
guster@os:~$ netstat -a | more

Proto  Recv-Q  Send-Q  Local  Address
Foreign Address     State

tcp      0     0  *:36615              *:*
LISTEN

tcp      0     0  *:sunrpc             *:*
LISTEN

tcp      0     0  *:webmin             *:*
LISTEN

tcp      0     0  *:ssh                *:*
LISTEN

tcp      0     0  *:38042              *:*
LISTEN
```

The audit typically begins by evaluating the purpose of known ports (the ones with English names) and determining if their presence is legitimate. Named ports can be looked up in services file (as defined by IANA) 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 and the port assignments have meaning world-wide. For services created locally with meaning on only a given host or within the cloud it is also possible to create a services.local file. Again, the format is similar in that 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 (sunrpc, webmin, ssh). Specifically, 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. This service is fairly complex because it uses a RPC (remote procedure call) structure named NFS (network file system). The WEBMIN was installed so that system administrators could remotely manage the system and of course the SSH (secure shell) port was required so that clients would have secure virtual terminal access to the host. Unfortunately, the purpose of the other two ports (36615, 38042) remains undefined.

```
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
```

To illustrate the complexity of a remote procedure call (RPC) a list of modules loaded on the host level is provided below. Because multiple modules related to NFS are loaded multiple ports on the service level are required to support all of the required sub-services. To get a better understanding of this process let's take a look at the structure of the modules associated with the NFS daemon (a system level process at the top of the NFS hierarchy). A quick analysis reveals that there is a specific module that deals with data concurrency issues (lockd), one that deals with authentication issues (auth_rpcgss), one that controls queries to the access control list (nfs_acl) and of course the main NFS module is present to deal with transmission of the actual data. Because RPC port assignment is often dynamic 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 does the dynamic assignment of port addresses take place? Just the portmap (sunrpc) and the prime NFS service itself are defined in the services file, the rest of the ports are dynamically assigned upon booting up the host. This situation can make access outside of the cloud (or even the host) difficult if some means of updating the firewall after every reboot is not considered. Of course if the data is to be shared within a specific zone of the cloud then the port map can be suppressed outside of that zone providing an additional layer of security. Because RPCs use the client/server model it is important to look at the port maps on both sides of that model. 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 which are needed on both the client and server. However, note that the server side is more complex and therefore also running the NLOCKMANAGER which manages daemons related to data locking and the (mount daemon) which answers requests from clients related to file system mounts. Analysis of the information from the port map below allows reconciliation of one of the two undefined ports from the original netstat output, 36615 which provides the status function on the client side and now can 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  36615 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  38192 status
   100003    4   tcp   2049 nfs
   100021    4   tcp  41275 nlockmgr
   100005    3   tcp  52771 mountd
```

An analysis of the sockets (net.node.port) used for the client/server connection can be used to illustrate how the data is transferred between the client and server. Additionally, if more detail is desired then using packet dumps would be warranted. By running the display file systems command (df) we see that the server side of the NFS file system is nfs.bcrc.local (10.18.59.35) and is mounted to the client side as /myhome. To obtain the full socket level connection information the netstat command is used and as one would expect the server side port is 2049 which matches its assigned value in the services file. However, the client port is 957 it is protected (root required to modify) and not in the services file. The need for 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 results in less 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  10.18.59.34:957         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: 10.18.59.34.82046249. 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, 82046249, 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 be reading the dump. By looking in the second row, third group of four hex numbers we see a value of 03bd which when converted to decimal yields a value of 957. 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
10:02:20.06204500:50:56:8c:05:a8 > 00:50:56:8c:0a:0e, ethertype IPv4 (0x0800),
length 306: (tos 0x0, ttl 64, id 27675, offset 0, flags [DF], proto TCP (6), length
292)
  10.18.59.34.82046249 >10.18.59.35.2049: 236 getattr fh 0,0/22
    0x0000: 4500 0124 6c1b 4000 4006 c7e80a12 3b22 E..$l.@.@.....;.
    0x0010: 0a12 3b2303bd 0801 8deb 08e5 cc05 c66c ..;.............l
    0x0020: 8018 08ed 06e7 0000 0101 080a 185e 6990 .............^i.
    0x0030: 185c fb53 8000 00ec 04e3 ed29 0000 0000 .\.S.......)....
    0x0040: 0000 0002 0001 86a3 0000 0004 0000 0001 ................
    0x0050: 0000 0001 0000 0030 011f 18ff 0000 000f .......0........
    0x0060: 6572 6f73 2e62 6372 6c2e 6c6f 6361 6c00 eros.bcrl.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 4305 0c00 ............C...
    0x00b0: 92e9 7494 0000 0026 0000 0001 3f2c b054 ..t....&....?,.T
    0x00c0: c800 0000 cc05 0000 0000 0000 0000 0000 ................
    0x00d0: 0000 0002 0000 003c 5468 6973 2069 7320 .......<This.is.
    0x00e0: 6120 7361 6d70 6c65 2070 6179 6c6f 6164 a.sample.payload
    0x00f0: 2c20 756e 666f 7274 7561 7465 6c79 2069 ,.unfortuately.i
    0x0100: 7420 6973 206e 6f74 2065 6e63 7279 7074 t.is.not.encrypt
    0x0110: 6564 210a 0000 0009 0000 0002 0000 0018 ed!.............
    0x0120: 0030 0000                .0..
```

Unfortunately, there still is one port that needs to be resolved, port 38042! Typically a sound starting strategy would be to try to obtain the user, process ID and command that started the process/port. A sophisticated configuration of the netstat command is used below and run as the root (via sudo). While the results are useful for the NFS status port (36615) and the port map (111) they provided limited information about the rogue port (38042). The desired information was obtained for port 36615 which is owned by user 106 (statd = status daemon). This process is running with process ID 838 and was started by the command rpc.statd. As previously stated, the information is very limited for port 38042 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. Therefore, another strategy is needed to track down the origins of this port. There is an inode address list for the port 38042 which is 504.

```
guster@os:~$ sudo netstat -apeen |more

tcp 0.0.0.0:38042  0.0.0.0:* LISTEN   0
504 -

tcp  0.0.0.0:36615    0.0.0.0:*  LISTEN
106   574 838/rpc.statd

tcp 0.0.0.0:111   0.0.0.0:* LISTEN   0
649    973/rpcbind
```

Using this strategy can be relatively effective in some cases by simply using the lsof command (list open files). In the example below we look up the inode associated with the statd and are able to glean, the process_id, user, portname and originating program.

```
guster@os:~$ sudo lsof | grep " 574"

rpcbind    838        root   8u    IPv4
574    0t0     TCP *:sunrpc (LISTEN)
```

However, when using the inode address associated with rogue port 38042 we get no results.

```
guster@os:~$ sudo lsof | grep " 504"

guster@os:~$
```

The next step is to evaluate the files in the file system used by the operating system which is simply "/". The basic tree and find commands provided the output below.

```
guster@os:/$ tree -a -L 10 --inodes |
grep " 504]"

|   |   |   |   |   |-- [  504] os_mac.txt


guster@os:/$ sudo find . -inum 504

./usr/share/vim/vim73/doc/os_mac.txt

./sys/devices/system/cpu/cpu2/node0
```

Unfortunately, neither file provides a clue as to the origins of port 38042. One file is a text file that is related to a visual editor and may be related to recording error messages and the other file is related to supporting multi-core symmetric multi-processing. In an effort, to ascertain if port 38042 was truly functional, it was decided to try to connect to it using a simple client such as telnet. The connection was successful from the local host (127.0.0.1) and data was passed, but the connection was dropped because a protocol mismatch occurred (telnet was not the appropriate client). Note there is no firewall on the localhost interface to filter the ports.

```
guster@os:~$ telnet 127.0.0.1 38042

Trying 127.0.0.1...

Connected to 127.0.0.1.

Escape character is '^]'.

letmein

Connection closed by foreign host.
```

While this connection was in place it was decide to see of lsof might now provide useful information. Unfortunately, neither searching by port (38042 or by the inode of the connection (8323843) yielded anything new.

```
guster@os:/$ sudo lsof | grep 38042

telnet   31181     dguster   3u    IPv4
8323843    0t0     TCP localhost:60059-
>localhost:38042 (ESTABLISHED)

guster@os:/$ sudo lsof | grep 8323843

telnet   31181     dguster   3u    IPv4
8323843    0t0     TCP localhost:60059-
>localhost:38042 (ESTABLISHED)
```

Therefore, 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 and the associated log files that support the ufw (uncomplicated firewall) function. The dmesg command allows direct access to the kernel ring buffer log information, when specifically searching for information on port 38042 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 38042 |
more

kern :warn : [Wed Feb 5 12:55:22 2015]
[UFW BLOCK] IN=eth0 OUT= MAC=00:50:5
6:8c:05:a9:00:50:56:8c: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=725 DPT=38042 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. This pattern would indicate a drop or in firewall terms a block. However, as was shown earlier the port is accessible from inside the VM, but the below shows that the no traffic can reach the port from outside its hosting VM. In fact the dump shows that the traffic was blocked at the firewall level of the nfs client and the SYN_SENT indicates that it was sent out by the NFS server. Once again in looking at the SYN_SENT entry there is no information provided about the process id or the command that generated the connection request. So once again we are limited with the inode entry. Because the SYN_SENT is transient there is limited time to try to trace it, but once again that route provided no useful information.

```
guster@os:~$ sudo tcpdump port 38042 -n –c2                    tcpdump: verbose
output suppressed, use -v or -vv for full protocol decode

listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes

11:55:02.921273  IP  10.18.59.35.44586  >  10.18.59.34.38042:  Flags  [S],  seq
2709921128, win 14600, options [mss 1460,sackOK,TS val 927308390 ecr 0,nop,wscale
4], length 0

11:55:03.918150  IP  10.18.59.35.44586  >  10.18.59.34.38042:  Flags  [S],  seq
2709921128, win 14600, options [mss 1460,sackOK,TS val 927308640 ecr 0,nop,wscale
4], length 0


tcp    0   1 10.18.59.35:725    10.18.59.34:38042   SYN_SENT                    0
4248126   -
```

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 38042 from the host nfs (10.18.59.35). So on the surface it appears that port 38042 could be a legitimate port that is used by the NFS service, but communication is being blocked by the firewall. This further supported by the fact that the source port is a protected port which might mean that it is designed to support multiplexed traffic. This is similar to what was seen with the connection from 10.18.59.34:957 to 10.18.59.35:2049. 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 os kernel: [14738071.383243] [UFW BLOCK] IN=eth0 OUT= MAC=0e
:50:56:8c:05:a9:00:50:56:8c: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=725 DPT=38042 WINDOW=14600
RES=0x00 SYN URGP=0


guster@nfs:~$ telnet 10.18.59.34 38042

Trying 10.18.59.34...

telnet: Unable to connect to remote host: Connection timed out
```

That investigation needs to center on the kernel itself and its interaction with the NFS daemons (system level processes). That paper would need to evaluate the process ids of the nfs daemons and examine their interaction with the kernel. From the output below one can see that there are multiple daemons and by looking at the detailed information about any one of those daemons leads to a link to an executable. However, this kind of analysis is typically outside a generic port auditing and is very time consuming as well as expensive. The important outcome of this audit was that although the port 38042 may or may not have been a byproduct of the NFS remote procedure call installation of NFS, because of a host level firewall the port was not assessable from outside of the host. Also, because the client that is trying to connect to that port based on the firewall logs is the trusted NFS host one would assume it is an unwanted byproduct and not a truly rogue port.

```
root@nfs:/# pidof nfsd

1254 1253 1252 1251 1250 1249 1248 1247
1246



root@nfs:/# lsof | grep 1246

nfsd4_cal  1246      root  cwd      DIR
8,1   4096     2 /

nfsd4_cal  1246      root  rtd      DIR
8,1   4096     2 /

nfsd4_cal  1246     root  txt   unknown
/proc/1246/exe
```

4. Discussion/Conclusions

The first step to facilitate sound port assignments within a cloud architecture is to have sound policy in place. In most cases, the ports that are supporting services are defined through a services file and there is documentation that describes their need and purpose. However non-typical ports are often needed. Specifically, 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 challenge the effectiveness of the defined security policy.

To illustrate this concept port 38042 running on a VM in the author's cloud served as an example of a port that appeared to have been created by malware used by hackers and provide them a "back door" to the host. However, this potentially rogue port appeared to be created by the installation of the NFS software (or perhaps a contaminated version). So, it is crucial when installing or updating software to take a snapshot of the port structure and compare the before and after status of the ports. Especially when ports are being defined dynamically knowing the purpose is critical. This means that there should be a way of linking the port to a process, startup command, user id and inode. The example used herein showed that this auditing required a multistep strategy using several LINUX commands. If this strategy was well documented the steps could be streamlined and performed quicker. Unfortunately, in the case of port 38042 the audit could not be effectively complete because both the server-side and client side were started on the kernel level which means that other than residual error message files no data could be linked to the port via its available inode.

Besides, auditing the purpose of the ports the steps taken herein performed an informal audit of the firewall process within the cloud. Of course the main filtering agent to prevent bad data coming into the cloud or within the cloud, of course, would be the firewall. Because one of the primary security strategies in the cloud involves layering, it is common 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 38042 the host level firewall prevented the client process from the host nfs from even forming a connection and this block was recorded in the UFW firewall log. These log files serve two purposes: detecting intrusion problems and evaluating if policy is being followed as well as its degree of effectiveness. The fact that this (possibly 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. Without a doubt, this is the correct policy. However, 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 and ports and the associated processes are started on the kernel level. In this case there is no trace of the process id or the program that started the port. As we have seen this seriously restricts the ability to audit the purpose and the characteristics of the port. Besides the issues associated with unknown ports, complex remote procedure calls may lead to other dangerous security issues. Under

certain predefined trust conditions the RPC's require no authentication or 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 transfer the same rights set to "host "B". However, this can be easily remediated by invoking the "nosuid" parameter which combats this inheritance problem.

It is clear that when performing a port level audit, that the basic LINUX operating system commands were quite effective. Determining which commands and options to use was not a simple process. For an inexperienced user, the process often involves trial and error and how quickly it can be performed is a function of the experience of the investigator. However, it is hoped that papers such as our current offering clarify the process. Further, often by entering the command and the key work LINUX into a search engine will provide the needed syntax. Typically, in most cloud installation one can expect to find some form of UNIX hosts as well as Windows hosts. Therefore, a cloud installation would be well advised to have a port auditing plan for both operating systems. Once devised that process should be well documented. In both cases, the process can be automated through scripts so that the code can be reused which in the long run will quicker and more cost effective. Of course, one would expect that there will still be new intrusion related cases occurring in which the problem must be attacked from scratch. So for cloud installations, it is imperative to have an analyst on staff that truly knows LINUX and how it interacts with the cloud/network infrastructure. Unfortunately, upper level management often assumes that they can rely totally on an icon based tools, but what happens when there is no icon to solve the problem available? Without a doubt this icon logic 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. This ability to apply critical thinking skills in connection with the structure of the operating system and trace down problems such as rogue ports is still a valuable commodity especially if there is no pertinent icon to click on.

References

1. Anthes, G. (2010). *Security in the cloud*. Communications of the ACM, November, 53(11), 16-18.
2. Basak, D., Toshniwal, R., & Maskalik, S. (et al). (2010). Virtualizing networking and security in the cloud. *ACM SIGOPS Operating Systems Review,* December, 44(4), 86-94, Retrieved from http://bit.ly/1fj5ICO
3. Birrell, A., & Nelson, B. (1984). *Implementing remote procedure calls*. ACM Transactions on Computer Systems, 2(1), 39-59.
4. Bohli, J., Gruschka, N., & Jensen, M., (et.al) (2013). *Security and privacy-enhancing Multi-cloud architectures*. IEEE Transactions on Dependable and Secure Computing, July-August, 10(4), 212-224. Retrieved from http://www.computer.org.ezproxy.umuc.edu/csdl/trans/tq/2013/04/ttq2013040212.html
5. Caron, E., Le, A., Lefray, A., & Toinard, C. (2013). *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
6. Green, M. (2013). *The Threat in the Cloud*. IEEE Security & Privacy, 11(1), 86-89. Retrieved from *http://www.computer.org.ezproxy.umuc.edu/csdl/mags/sp/2013/01/msp2013010086.html*
7. 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.
8. Hyman, P. (2013). *Augmented-reality glasses bring cloud security into sharp focus*. Communications of the ACM, June, 56(6), 18-20. Retrieved from http://bit.ly/MGQGyW
9. Jules, A., & Oprea, A. (2013). *New approaches to security and availability for cloud data*. Communications of the ACM, February, 56(2), 64-73. Retrieved from http://bit.ly/1gz6XDh
10. Kaufman, L. (2010). *Can Public-Cloud Security Meet Its Unique Challenges?* IEEE Security & Privacy, July-August, 8(4), 55-57. Retrieved from *http://doi.ieeecomputersociety.org.ezproxy.umuc.edu/10.1109/MSP.2010.120*
11. Lesk, M. (2012). *The Clouds Roll By*. IEEE Security & Privacy, May-June, 10(3), 84-87. Retrieved from http://www.computer.org.ezproxy.umuc.edu/csdl/mags/sp/2012/03/msp2012030084.html
12. Mell, P. (2012). *What's Special about Cloud Security?* IT Professional, July-August, 14(4), 6-8. Retrieved

from http://www.computer.org.ezproxy.umuc.edu/csdl/mags/it/2012/04/mit2012040006.html

13. 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, (pp. 15-19). Retrieved from http://bit.ly/1bfH3jn
14. Rushby, J. (1981). *Design and verification of secure systems*. Proceedings of SOSP, the 8th ACM Sysmposium on operating System Principles, (pp. 12-21).
15. Seo, S., Nabeel, M., & Ding, X. (et.al). (2013). *An Efficient Certificate less Encryption for Secure Data Sharing in Public Clouds*. IEEE Transactions on Knowledge and Data Engineering. IEEE computer Society Digital Library. Retrieved from http://www.computer.org.ezproxy.umuc.edu/csdl/trans/tk/preprint/06574849.pdf (accessed on August 5, 2013)
16. Sun, D., Chang, G., Sun, L., & Wang, X. (2011). *Surveying and Analyzing Security, Privacy and Trust Issues in Cloud Computing Environments*. Procedia Engineering, 15, 2852-2856.
17. Takabi, H., Joshi, J., & Ahn, G. (2010). Security and Privacy Challenges in Cloud Computing Environments. IEEE Security & Privacy, November-December, 8(6), 24-31. Retrieved from http://www.computer.org.ezproxy.umuc.edu/csdl/mags/sp/2010/06/msp2010060024.html
18. Toxen, B. (2007). The seven deadly sins of Linux security. *Security,* May-June, 5(4), 38-47.
19. Vahidi, A. (2013). *Secure RPC in embedded systems: Evaluation of some global platform implementation alternatives*. Proceedings of the Workshop on Embedded Systems Security, 4.
20. Viega, J. (2012). Cloud Security: Not a Problem. IEEE Security & Privacy, July-August, 10(4), 3. Retrieved from http://doi.ieeecomputersociety.org.ezproxy.umuc.edu/10.1109/MSP.2012.93
21. Zargari, S., & Smith, A. (2013). *Policing as a Service in the Cloud.* Proceedings from 2013 4th 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
22. Zissis, D., & Lekkas, D. (2012). Addressing cloud computing security issues. *Future Generation Computer Systems,* March, 28(3), 583-592. Retrieved from http://www.sciencedirect.com.ezproxy.umuc.edu/science/article/pii/S0167739X10002554#