DEVELOPING A MOBILE APP THAT USES THE ROUTING CACHE TABLE TO DETECT BRUTE FORCE SECURE SHELL ATTACKS NOT DETECTED BY THE INTRUDER DETECTION SYSTEM

Raqeeb Abdul
SAINT CLOUD STATE UNIVERSITY
rabdul@stcloudstate.edu

Laura Lebentritt
UNIVERSITY OF MARYLAND, UNIVERSITY COLLEGE
lauraleigh4097@att.net

DENNIS GUSTER
SAINT CLOUD STATE UNIVERSITY
dcguster@stcloudstate.edu

Abstract

Cyber threat indicators that can be instantly shared in real-time may often be the only mitigating factor between preventing and succumbing to a cyber attack. Detecting threats in cloud computing environment can be even more of a challenge given the dynamic and complex nature of hosts as well as the services running. Information security professionals have long relied on automated tools such as intrusion detection/prevention systems, SIEM (security information and event management), and vulnerability scanners to report system, application and architectural weaknesses. Although these mechanisms are widely accepted and considered effective at helping organizations stay more secure, each can also have unique limitations that can hinder this regard. Therefore, in addition to utilizing these resources, a more proactive approach must be incorporated to bring to light possible attack vectors and hidden places where hackers may infiltrate.

This conference proceeding shares an insightful example of such lessor known attack vectors by closely examining a host routing table cache, which unveiled a great deal of information that went unrecognized by an intrusion detection system. Furthermore, several of the authors researched and developed a robust mobile app tool that has a multitude of functions which can provide the information security community with a low-cost countermeasure that can be used in a variety of infrastructures (e.g. cloud, host-based etc.). The designed mobile app also illustrates how system administrators and other IT leaders can be alerted of brute force attacks and other rogue processes by quickly identifying and blocking the attacking IP addresses. Furthermore, it

is an Android based application that also uses logs created by Fail2Ban intrusion prevention framework for Linux. Additionally, the paper will also familiarize readers with indirect detection techniques, ways to tune and protect the routing cache, the impact of low and slow hacking techniques, as well as the need for mobile app management in a cloud.

1. Introduction

Cloud computing supported by the UNIX operating system can be quite complex. This is borne out by the fact that information stored on the system is quite frequently stored in multiple places. In some cases this process is automatically accomplished by the operating system when a related event occurs. Such is the case when an attempt is made to log into the secure shell service on a virtual machine (VM) in a cloud. Typically, the first hop of the route being used is recorded in the dynamic routing cache table. In most configurations of UNIX or variants such as LINUX (which is used in the study) this table addition occurs automatically on the server side and there is little an attacking client process can do to stop it [1]. Detailed information regarding the configuration, operation and tuning of router cache is available from [2] and provides the flexibility to tune the cache to be more effective beyond the default settings in identifying attacking routes. Of course the cache itself could be attacked and possibly disabled by a denial of service attack so mechanisms need to be in place to protect it [3]. Therefore, its internal settings need to be well thought out. For example, the garbage collector settings need to be optimized so the number of cached routes cannot grow too large [13].

To illustrate this problem a simplistic example will be delineated below using the authors' cloud based VM test-bed. First, an abbreviated routing cache table with numeric addresses appears below. In the source column the value is compared to the whitelist of Ipv4 addresses. In the first row the value observed is in the whitelist category as having a valid IP within the cloud. However the value in the second row is not in the whitelist. A lookup of this IP address via the whois command indicates that it is leased through Digital Ocean. This low cost internet service has been used in the past by hackers to devise and test new techniques [12] so it follows that further investigation is warranted. Please note however that the record includes a contact to report abuse so the ISP is taking some responsibility in the event there services are misused.

```
buster@bros:/rhome/classes$ route -Cn | more

Kernel IP routing cache

Source          Destination     Gateway         Flags Metric Ref    Use Iface
199.17.59.234   199.17.59.195   199.17.59.195         0      1     4776 eth0
188.226.139.158 199.17.59.234   199.17.59.234   l     0      0        3 lo

buster@bros:~$ cat whitelist_rt.local
IP for cloud            199.17.59.0     # Public Class C for all cloud zones
IP for Parent org       199.17.0.0      # Public Class Cs for parent org
IP for lo               127.0.0.0       # Loop back on VM
```

```
IP for internal         10.0.0.0        # Private Class A for internal nets
Ip for internal         192.0.0.0       # Private Class C for internal nets

buster@bros:~$ whois 188.226.139.158 | more
% Information related to '188.226.128.0 - 188.226.191.255'
% Abuse contact for '188.226.128.0 - 188.226.191.255' is 'abuse@digitalocean.com'

inetnum:        188.226.128.0 - 188.226.191.255
netname:        DIGITALOCEAN-AMS-4
descr:          Digital Ocean, Inc.
country:        NL
```

Figure 1: Abbreviated routing cache table

The most prevalent type of attack on this VM is a brute force secure shell attack. Which for the sake of simplicity we can assume is detected by and logged by the fail2ban sub-process within the syslog facility. The event logic to log is simply three failed login attempts and once this occurs the offending IP address is locked for ten minutes. In the example below an address starting with 222 met the intrusion status twice.

```
log for Fail2ban v0.8.6
2016-01-03 09:05:05,677 fail2ban.actions: WARNING [ssh] Ban 222.186.21.73
2016-01-03 09:15:06,569 fail2ban.actions: WARNING [ssh] Unban 222.186.21.73
2016-01-03 09:16:59,760 fail2ban.actions: WARNING [ssh] Ban 222.186.21.73
2016-01-03 09:27:00,679 fail2ban.actions: WARNING [ssh] Unban 222.186.21.73
```

Figure 2: Failed login attempts log

However, the 188 address described above does not appear in this log at all, meaning that based on this basic logic it is not defined as a security breach event. A quick analysis of the authentication log reveals that it tried to connect via the secure shell daemon, but did not get back a value from the service that could be capitalized upon and so it never returned an authentication response. This scenario was repeated three times within a couple of hour's space between the events. This slow and low volume attack scenario is consistent with sophisticated hacking techniques designed to minimize the attack footprint [6]. Further, there were 32 instances of attacks from networks beginning with 188.226 so it may be wise to filter out all of that network traffic on a firewall level.

```
buster@bros:/var/log$ sudo cat fail2ban.log | grep 188
buster@bros:/var/log$

buster@bros:/var/log$ sudo cat auth.log | grep 188.226.139

Jan  5 10:49:21 bros sshd[5189]: Did not receive identification string from
188.226.139.158

Jan  5 12:50:13 bros sshd[6835]: Did not receive identification string from
188.226.139.158

Jan  5 15:35:49 bros sshd[8780]: Did not receive identification string from
188.226.139.158

buster@bros:/var/log$ sudo cat auth.log | grep 188.226. |wc –l      32
```

Figure 3: Slow and low volume attack scenario

So based on the scenario described above it is clear that evaluating the router cache can provide an additional tool to identify hacking attacks that may not be picked up by an intrusion detection system that might not be tuned to be overly sensitive in attempts to minimize false positives. As one would expect developing a methodology that compliments existing intrusion detection strategies and provides quick alerts when a non-whitelisted site is detected could be a valuable addition to a cloud based security strategy. Therefore, this paper will build on the basic attack methodology depicted above and implement a mobile application to remotely manage scripts that will evaluate the routing cache in relation to a whitelist in real time, send out alerts, log the offending events and provide basic performance information about the routing cache. This performance information such as table size, hit efficiency and initial round trip time will be used in part to evaluate whether the cache itself has been potentially compromised. While this paper presents an interesting security problem it also provides a series of pertinent hands-on scenarios that could be used in an educational environment. For example, the concept of a dynamic table look-up is certainly pertinent herein, but permeates throughout computing and computing security and once a basic understanding is attained that know ledge could be easily transferred to another scenario such as a dynamic ARP table.

2. Review of Literature

Indirect Detection Techniques

One of the problems in devising an effective security strategy on a dynamic system such as a cloud is make sure that the detection system is quick and adaptable. In other words, attacks against a dynamic system are best detected by another dynamic system. This concept is supported by Jichkar and Chandak [10] in their implementation of a security detection system that due to the dynamic topology of the networks any static configuration would not be sufficient. The work of [4] builds on this concept by placing the trust evaluation from their security schema on a series of dynamic systems they refer to as watchdog nodes. These nodes monitor and collect other sensors' behavior information and are tuned to dynamically spot problems within the trust interrelationships.

The work of [16] used a finite-state machine (FSM) approach to decompose programs into multiple elementary activities and used an indirect analysis of those activities to ascertain the vulnerability of the originating program. This is consistent with the prior references in this section in that a finite state machine can be used to model complex logic in dynamic systems. In this case, the FSM analysis pinpoints common characteristics among a broad range of security vulnerabilities: predictable memory layout, unprotected control data, and pointer taintedness. As one would expect the solution lies in a more dynamic resource allocation approach which would randomize vulnerable areas such as memory layout.

The work of [14] while not directly security related is very pertinent to the core purpose of this paper. This work proposed and implemented a new distributed architecture to efficiently use network-wide cache storage space based on a distributed caching algorithm. In the current paper the indirect detection system is based only on the router cache of a single host. In systems, which

used the proposed method of [14] the scope could easily be expanded form a single host to an entire cloud. The white list of know networks would just need to expand beyond those trusted by the host to those trusted by the cloud.

Tuning the routing cache

For quite some time hackers have seen the value in attacking the routing structure within internets [11]. One simple ploy would be to launch some type of denial of service attack. Besides sound isolation of the routing tables a good strategy to combat this scenario is to keep the cache well-tuned. This is in part measurable by looking at the percent of requests that are actually handled by the routing cache and do not require that packets be sent out to resolve the route, which is much slower. An abbreviated example from the authors' cloud below reveals that the cache is tuned fairly efficiently because ~95% of the incoming requests are being cached (100 - (2053864 / 43539507) *100)). The outgoing requests reach a similar efficiency with a hit percentage of ~97%.

```
buster@bros:~$  lnstat -s1 -i1 -c-1 -f rt_cache

|rt_cache|rt_cache|     |rt_cache|rt_cache|
|  in_hit|in_slow_|     | out_hit|out_slow|
|43539507| 2053864|     |11315044|  325675|
```

Figure 4: Tuned routing cache

The work of [15] provides an excellent example of what can happen if hacker attacks disrupt the expected routing performance. Specifically, they found that when routing tunnels are allocated with limited resources this allows a malicious router operators to attack such tunnels. If the tuning metrics used are insensitive to relative load the system does not adequately respond to changing conditions. This situation results in unreliable performance which may drive many users away.

While [2] states that when Linux is used as a router, the inefficiency of the route cache can hinder the performances of your host he also states that information on how to tune that cache is scarce and it is difficult to find up-to-date information on how the route cache works. Of course this makes tuning it problematic. As often is the case with mundane computing topics O'Reilly Publishing comes to the rescue with "Understanding Linux Network Internals" [1]. Specifically chapter 33 deals with these issues. However, [2] provides a decent overview in regard to the basic of tuning router cache to help ensure secure operations. Specifically, decisions need to be made in regard to setting the hash table size, target average length of the queue, delay between garbage collection runs and when to remove an outdated entry. Of special note from a security perspective is the importance of tuning the garbage collection process which helps makes sure that unused resources are recovered in real time which lessens the impact of a denial of service attack.

Protecting the routing cache

Authors Zhang, Mao, and Wang discuss the need to protect against routing misbehaviors by using route normalization to protect local network traffic from erroneous and malicious routing traffic [19]. Backbone network elements such as intrusion detection systems, firewalls, and routers, all depended on the integrity and cleanliness of mechanisms like the routing cache. If it goes awry then the whole network can be compromised, therefore, it's paramount to protect routing caches due to its inherent trustworthiness. Zhang et. al. also explain how using a RouteNormalizer that mitigates violations with routing loops, missing mandatory attributes, nexthop violations, and export policy violations can mitigate and detect routing anomalies too [19].

Impact of Low and Slow Hacking Techniques

Even the most robust intrusion detection systems can be deceived by the low and slow hacking techniques that stay just enough under the radar yet often do the most damage. This is one of the main reasons why tuning and protecting the routing cache is vital to seeking out hidden places malicious hackers can lurk. An example of such surreptitious tactics is found in advanced persistent threats (APT's). APT's take "a low-and-slow approach" using social engineering "to gain access to a network and steal information quietly" [18]. Symantec describes APT's as, "An advanced persistent threat uses multiple phases to break into a network, avoid detection, and harvest valuable information over the long term. This information below details the attack phases, methods, and motivations that differentiate APTs from other "targeted attacks" [17]. Symantec further explains how APT's are carried out using the following methodology:

1.) Incursion
Attackers break into network by using social engineering to deliver targeted malware to vulnerable systems and people.

2.) Discovery
Once in, the attackers stay very much under the radar to avoid detection. They then map the organizations defenses from the inside and create a battle plan and deploy multiple parallel kill chains to ensure success.

3.) Capture
Attackers then access unprotected systems and capture information over an extended period. They may also install malware to secretly acquire data or disrupt operations.

4.) Exfiltration
Captured information is sent back to attack team's home base for analysis and further exploitation fraud and/or worse.

Larry Clinton, President of Internet Security Alliance, describes APT's as highly skilled, day job hackers who are going to get into systems and they are very persistent. Additionally, Clinton describes how they "stay for dinner and breakfast and who don't ever go away". They're going to go quiet and companies will think they're gone but they are not. They're not interested necessarily in taking down a system but rather using and gathering information about company systems and will "call home" with new information to advance their threat. They are very skilled at hiding under the radar of anti-virus software. They are usually going after a company's Internet Protocol (TCP/IP), interested in identity thefts, wire fraud and harvesting other types of classified information [5].

Need for Mobile App Management in a Cloud

Given that attacks against routing resources can be disastrous and must be dealt with in a timely manner [4], the interface used to alert the system administrator to such attacks must be agile. Therefore, the goal of any remote system administration management software should be to facilitate ease of use, speed and security. Because equipment rooms are seldom staffed anymore, the logical solution is to provide management capabilities from a mobile device. In the paper herein the examples are based on Linux platforms and it therefore is logical to evaluate the available mobile Apps that are compatible with Linux systems. There are several mobile APPS available, but typically depend on ssh or some type of remote virtual terminal program to gain access to the operating system. This situation would require that the appropriate command be entered, which may not be all that fast given the keyboard characteristics of some mobile devices [7]. However, as previously stated a mobile device will be critical in an obtaining an effective solution because it is readily available and easily accessible.

The primary goals in adapting a mobile device to support the remote identification (via the router cache) and mitigation of brute force attacks are typically related to identifying and quickly blocking the attacking IP addresses, the ease of use, and of course added security. In the case of not meeting these goals, then the solution would provide limited value. A similar security APP was devised by [8], except it was designed to identify and mediate rogue processes.
In that case the challenge was to protect the system while a true decision about the offending rogue process was being made by the system admin. Of course this decision will take time, certainly the mobile APP approach would minimize some of the end-to-end delay associated with logging in remotely via a virtual terminal program or a browser there will be some degree of asynchronous human think time. In [8], a strategy was pursued to just suspend the rogue process, and then if required, killed by the sys admin via the mobile APP. A similar approach will be applied herein where the block is temporary and can be made permanent if desired by the system administrator.

3. Methodology

Because quickness of analysis is critical, the Android Mobile application strategy facilitates the availability of information pertaining to basic router cache performance information, incursions

recorded in logs, and the efficiency of the routing cache configuration. To evaluate the implementation in an effective manner, the design will be presented in the following modules:

1) Efficiency of the router cache
2) Evaluating Routing cache from a mobile application
3) Authentication logs
4) Fail2Ban logs
    a. Push notification for Fail2Ban Logs
    b. Inserting records to database
5) Authentication and Encryption for accessing mobile application

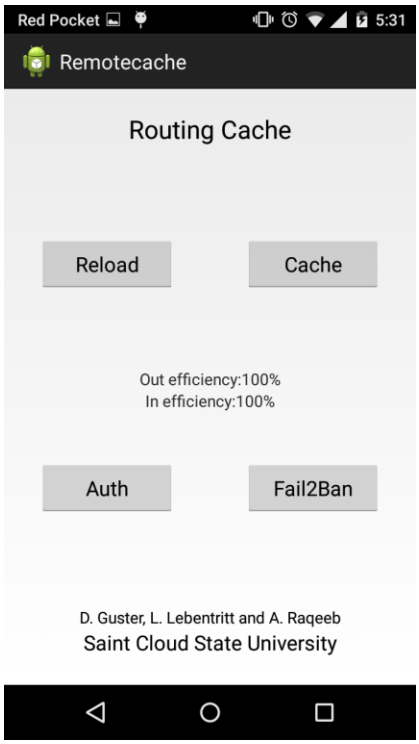


Figure 5: Mobile APP – Routing Cache

All the modules above are discussed from two different perspectives: the client side (Mobile application) and Server side (Linux Host). The server side is implemented using shell scripts and PHP scripts.

3.1 Efficiency of the router cache:

The router cache efficiency can be determined for outgoing and incoming connections. So the efficiency of the router cache is calculated on the server side. The PHP script on the server side fetches the cache from the following location "/proc/net/stat/rt_cache". The output is not structured. So, to parse this data we have identified common delimiters like comma, dot, pipe

etc. This PHP script also exposes a REST API so as to consume it on the mobile application. REST API returns the data in JSON format.

```
URL: http://<server>/performance.php
Method: GET
URL Params: <none>
Header: Authorization: Basic +
Base64(encrypt(username):encrypt(password))
Response: {"in_efficency":"100","out_efficency":"100"}
```

Figure 6: PHP Script

Below the strategy for calculating the in and out router cache efficiency is implemented in the REST API.

Incoming efficiency:
in_efficiency = 100 – (rt_cache_in_slow_tot/rt_cache_in_hit)*100
Outgoing efficiency:
Out_efficiency = 100 – (rt_cache_out_slow_tot/rt_cache_out_hit)*100

3.2 Evaluating Routing cache for mobile application:

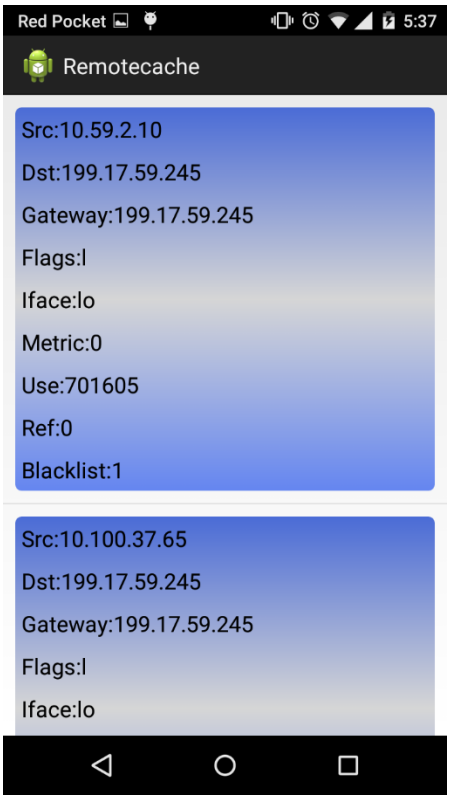


Figure 7: Mobile APP – Evaluating the Routing Cache

The routing cache can be viewed by commands “route -Cn” or “netstat -Cnre”. The whitelist_rt.local file has all the networks that are trusted on the cloud level which can basically be considered as Whitelisted IP addresses. Those networks not listed may or may not become a blacklisted address, but this APP would alert the system administrator to evaluate their presence (typically via a script).The initial evaluation of routing cache is done by comparing source ip address from output of “route -Cn” with the IPs that are listed in whitelist_rt.local. Since, the data is not structured we use the same strategy as above to parse the data. The PHP script exposes a REST API which returns the data in JSON format, which allows us to work with objects.

```
URL: http://<server>/cache.php
Method: GET
URL Params: <none>
Header: Authorization: Basic +
Base64(encrypt(username):encrypt(password))
Response:
[{"src":"199.17.59.245","dst":"199.17.59.195","gateway":"199.17.59.195","
flags":"-
","metric":"0","ref":"1","use":"1058","iface":"eth0","blacklisted":0},{"s
rc":"110.77.138.151","dst":"199.17.59.245","gateway":"199.17.59.245","fla
gs":"l","metric":"0","ref":"0","use":"2","iface":"lo","blacklisted":1}]
```

The script below provides the logic and implementation strategy for evaluating the routing cache:

```
function parse_ip($ip)
{
     $ret = 1;
     $data = shell_exec('cat whitelist_rt.local');
     $data1 = explode("\n",$data);

     foreach ($data1 as $value){
     if(strlen($value)>1)

     $ips[]
     =preg_replace("/\.{2,}/",".",str_replace(".0",".",preg_replace("/[^0-
     9.]/","",$value)));
     }
     foreach ($ips as $addr) {
         if (strpos($ip, $addr) === 0) {
            $ret =0;
         }
     }
     return $ret;
}
```

Figure 8: PHP script for evaluating the routing cache

The above parse_ip function takes only a single argument such as the IP address and compares it with the IPs stored in ‘whitelist_rt.local’. The comparison is done by checking the net portion of

whitelisted IP address against the source IP address in the router cache. In PHP the strops (string operations) returns the first occurrence of the substring. If the IP address is not a substring then it is listed as blacklisted for this phase. It returns a '1' if the IP is blacklisted and '0' if not.

3.3 Authentication logs:

The authentication logs are useful for evaluating user login patterns and determining when sudo commands are invoked. This "auth log" file can be accessed at /var/log/auth.log. In addition this log is useful for identifying malicious activities. Besides the convenience displaying information on a mobile application is better approach because the auth.log data is not structured. Of course it is easy to impose structure within the mobile APP display. The strategy for parsing the data is similar to the examples above which use delimiters like space, periods, pipes and so forth. The server also uses a REST API which returns a JSON array.

```
URL: http://<server>/auth_log.php
Method: GET
URL Params: <none>
Header: Authorization: Basic +
Base64(encrypt(username):encrypt(password))
Response:  [{"timestamp":"Feb  1 11:01:02","log":"php1 CRON[5453]
pam_unix(cron session)  session closed for user smmsp"},{"timestamp":"Feb
1 11:09:01","log":"php1 CRON[5473]  pam_unix(cron session)  session
opened for user root by (uid=0)"}]
```

The response is structured into basically timestamp and log.

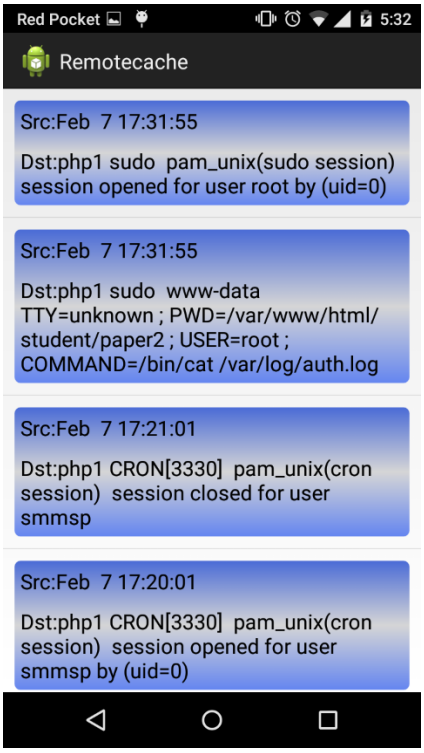


Figure 9: Imposing structure for authentication logs on mobile APP

3.4 Fail2Ban Logs:

The Fail2Ban logs are created by Fail2Ban intrusion prevention framework. These logs can be found on /var/logs/fail2ban.log. Also, these logs are unstructured. So, to make this log meaningful and secure, we have created a script which copies of records to a MySQL database. To do this in Fail2Ban framework, we need to create an action that will trigger a script which writes data to MySQL whenever a ban or unbanned has occurred. This configuration file is located at /etc/fail2ban/action.d. For the project herein, the configuration file below was created.

```
student@php1:/etc/fail2ban/action.d$ cat qshield.conf
[Definition]
actionstart =
actionstop =
actioncheck =
actionban = iptables -I INPUT -j DROP -s <ip>
            sudo php /var/www/html/student/paper2/ban.php <ip>
actionunban = iptables -D INPUT -j DROP -s <ip>
            sudo php /var/www/html/student/paper2/unban.php <ip>
```

Figure 10: Configuration file

The above configuration will call ban.php and unban.php whenever ban and unban of IPs occur respectively. Both the scripts evaluate one argument: ip-address.

Lastly, transfer the information in “qshield.conf” into “jail.local” which is the configuration file for the Fail2Ban framework. This configuration file has jails for various protocols like http,ftp,ssh etc. Since, this paper is concerned about ssh, we have created a jail only to ban ssh. Below is a sample the configuration file.

```
[ssh]

enabled  = true
port     = ssh
filter   = sshd
logpath  = /var/log/auth.log
maxretry = 3
action = qshield
```

These logs are integrated into the mobile application by using a REST API. In this case, the log files didn’t need to be parsed because they are already stored on the database. So, a simple “get” query could be used to retrieve the data.

```
URL: http://<server>/fail2ban.php
Method: GET
URL Params: <none>
Header: Authorization: Basic +
Base64(encrypt(username):encrypt(password))
```

```
Response:
[{"id":"16","status":"ban","ip":"199.17.59.234","timestamp":"2016-02-06
14:05:53","OrgName":"Minnesota State Colleges and
Universities","Address":"30 7th Street East, Suite 350","City":"St.
Paul","StateProv":"MN","PostalCode":"55101","Country":"US"},{"id":"17","s
tatus":"ban","ip":"199.17.59.234","timestamp":"2016-02-06
14:15:53","OrgName":"Minnesota State Colleges and
Universities","Address":"30 7th Street East, Suite 350","City":"St.
Paul","StateProv":"MN","PostalCode":"55101","Country":"US"}]
```

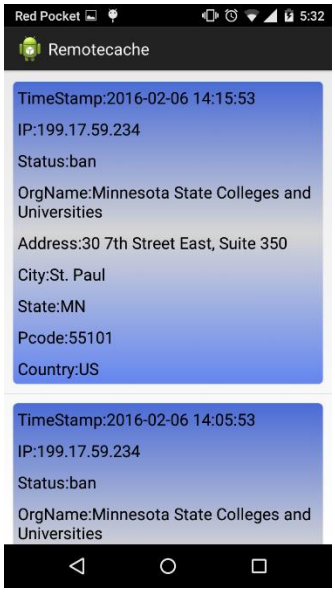
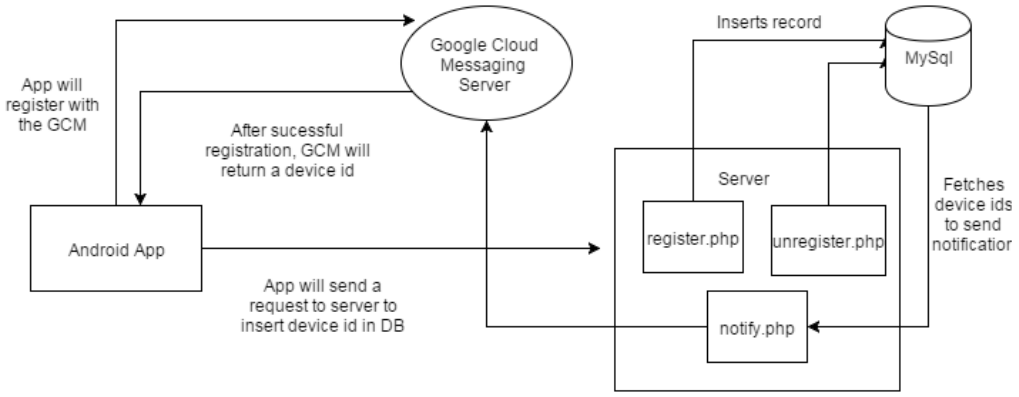



Figure 11: Configuration file integrated to mobile APP

3.5 Push notification for Fail2Ban Logs

The push notification is a message which is delivered by the server to the mobile application automatically without any need for a request from the mobile application. In the implementation herein, the push notification is used to notify the user about the bans and unbans of the IP address. This is also useful in identifying false positives:



The mobile device will register with the Google Cloud Messaging server (GCM) upon successful login to the application. If the registration is successful, the GCM will return a device id which is a unique id to identify the device and application within the cloud. To send a notification to a mobile application this device id is imperative. To facilitate future development and in the spirit of event logging this device id is being saved in a MySQL database. This process is accomplished when the registration API is called. To send notifications, a script called notify.php was created which in turn has a function called “notify” containing the arguments message and the IP-address that are captured from fail2ban.The ban.php and unban.php files which are discussed above call this “notify” function along with the arguments message and IP-address. Below is the format of a request to the GCM server.

```
URL:https://android.googleapis.com/gcm/send
Method: POST
URL Params: <none>
Header: Authorization: key=YOUR_API_KEY, Content-Type: application/json
Parameters: registration_ids = <array of device ids>, data = <array
of key value pair>
Response:
{"multicast_id":6123852703457412158,"success":3,"failure":0,"canonical_id
s":2,"results":[{"message_id":"0:1454882086684937%39ad5476f9fd7ecd"},{"re
gistration_id":"APA91bGD3SNy6XfBr0o8x9d6Eh_TPSKRVjDfO8IHmmNqIfkXZjxaQKZqr
r-
DQywAo7rJypbNcXhnEznAjsluVBuw78Ow6QgxPYVesIFg79ZHt55_0FNmXBuXhq3bvo26TArT
bKho2ras","message_id":"0:1454882086684941%39ad5476f9fd7ecd"}]}
```

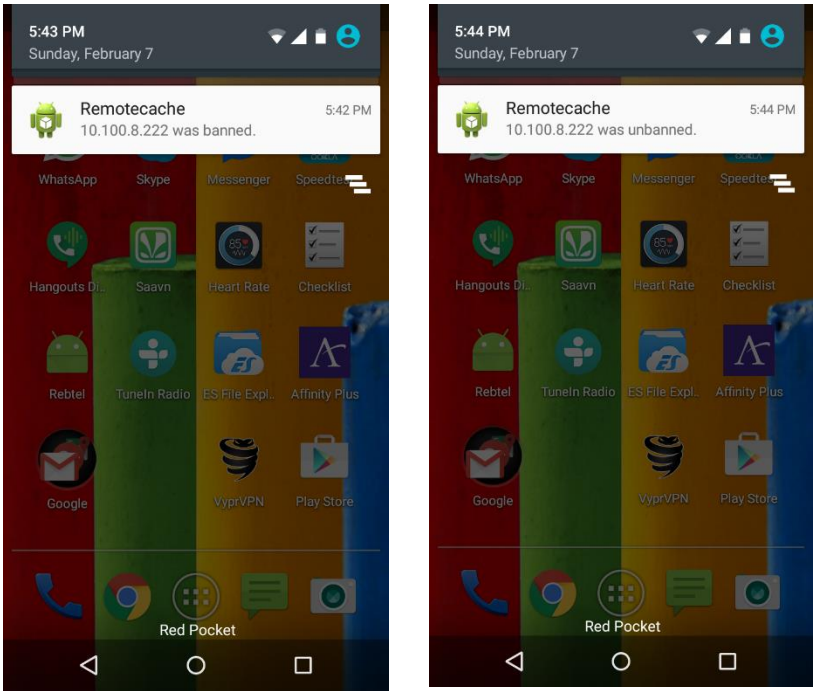


Figure 12: Push notifications for Fail2Ban Logs

3.6 Inserting records to database

The idea behind this strategy is to secure the logs. It is similar to taking backup of fail2ban logs. This strategy will be more efficient if the database is on some other remote server. But in our design for convenience we have used same host as the fail2ban is installed. This database can be used to create the REST API that can be consumed on the mobile application. To make this data more meaningful we have included the location, host organization, etc. of the IP-address. To achieve this we have used “dig” command. Below is the function that takes one argument as IP-address and returns the array of details

```
function getLoc($ip){
        $data = shell_exec("whois ".$ip." | grep
'OrgNam\|City\|Address\|StateProv\|PostalCode\|Country'");
        $data1 =  explode("\n",$data);
        foreach($data1 as $value){
        $data2 = explode(":",$value);
        if(sizeof($data2)>=2)
        $output[$data2[0]] = trim($data2[1]);
}

return $output;
}
```

Figure 13: Inserting records to database

3.7 Encryption

Because this projected attempted to use object oriented programming whenever possible a well-respected library [9] was used for encryption on both the mobile APP and the server. This library makes use of AES 128. When the mobile application sends a request to the server the data is encrypted before being transmitted. For example, one of the parameters to be passed would be the process ID which could be considered to be sensitive data. So under this scenario, every process id is encrypted and then sent over the REST API, of course then the, REST API is capable of decrypting the data.

The Initial Vector and Key both are initially shared between mobile app and the server. Below is sample implementation of the Encryption on the mobile app.

```
String credentials = MCrypt.bytesToHex(mcrypt.encrypt("raqeeb")) + ":"+
MCrypt.bytesToHex(mcrypt.encrypt("superman2"));
```

The decryption on the server side is accomplished with the logic below:

```
$mcrypt = new MCrypt();
$username =$mcrypt->decrypt($user);
$password = $mcrypt->decrypt($pass);
```

3.8 Authentication for accessing mobile application:

In this application authentication is done by the HTTP Basic Auth routine in conjunction with along with the AES-128 encryption class. The main file for handling the authentication and login API on the server is “auth.php”. Every request from the mobile APP is then authenticated using that HTTP Basic Auth routine. Below is the example of adding the Auth header to the mobile application. Below is the sample java code for a request on client side:

```
credentials = MCrypt.bytesToHex(mcrypt.encrypt("raqeeb")) + ":"         +
MCrypt.bytesToHex(mcrypt.encrypt("superman2"));

String credBase64 = Base64.encodeToString(
credentials.getBytes(), Base64.DEFAULT).replace("\n","");
HttpClient httpclient = new DefaultHttpClient();
HttpGet httpGet = new HttpGet(params[0]);
httpGet.setHeader("Authorization", "Basic " + credBase64);
HttpResponse response = httpclient.execute(httpGet);
```

The above credentials are stored in the persistent memory of mobile device once the user has successfully logged in. This facilitates the efficiency of evaluating future requests to the server. Below is the sample PHP code used to handle the authentication requests.

```
function pc_validate($user,$pass) {
        $mcrypt = new MCrypt();
        $username =$mcrypt->decrypt($user);
        $password = $mcrypt->decrypt($pass);
        $pass_md5 = md5($password);

        $query = "select * from users where username = '$username' and
password
= '$pass_md5'";

        $result=mysql_query($query);
        $count=mysql_num_rows($result);

        if($count==1)
        {
                return true;
        }
        else
        {
         return false;
        }
}
```

Figure 14: PHP code used for authentication requests

To achieve better security, the username and password have also been encrypted using AES-128 class before being transmitted to the server. In turn, the server will handle this request by decrypting and following the rest of the authentication process. A much needed future addition to this project would be implementing an administration control panel for controlling user accounts

within the mobile application. Keeping this in mind the users are being stored in a MYSQL table called ‘users’. To ensure added security the passwords are stored as a MD5 hash.

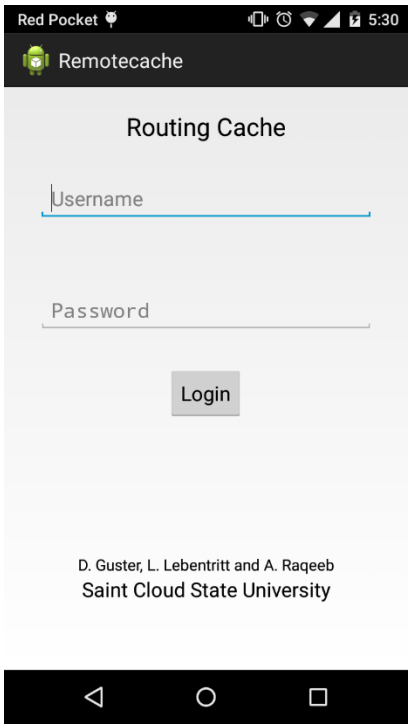


Figure 15: Authentication for Mobile APP

4. Discussion/Conclusion

The results generally indicate that an indirect measurement strategy such as using the router cache can detect attacks that may not trip an intrusion detection system. This is due in part to trying to minimize false positives within intrusion detection system and the fact that hacker have learned to utilize less intense attacks to minimize the probability of getting caught and to minimize their foot print from a forensics point of view. The data evaluated revealed that scenario. There were typically about ten attacks per hour that didn’t trip the intrusion detection system, but were caught by the router cache. However, there were also several events per hour where the IP address was not on the whitelist and hence was mistakenly identified as an attack. Future research by the authors needs to address how the whitelist can be expanded beyond the list of current trusted domains. In many cases outside address are related to a service provider such as Charter and Charter related client addresses in the authors’ service area could be added to the white list. Also, use of reoccurring client addresses if used ethically could be added as well.

Using the router cache for this purpose places an additional burden on that cache in the sense it will get read more often. So it will be important to make sure the cache is well tuned. On the

authors' system the virtual machine tested was getting an excellent hit ratio of about 95% in each direction. This is a good indicator of the health of the configuration and hopefully can be maintained when the router cache is also used to identify intrusions.

Last, for this technique to be viable a quick and convenient interface is required. It makes sense to develop this interface as a mobile APP. Equipment rooms are seldom manned 24/7 anymore, so assuming that a systems administrator carried a smart phone at all times, intrusion could be identified and dealt with remotely. Certainly, this remote management would require that sensitive information travel the airways. Therefore, the design presented herein attempted to use encryption mechanisms to ensure secure transmission of data. Further, it was clear that this security tool could be just one of many that could be managed via a mobile APP. In fact this is the second tool developed by the authors, see [8]. To accommodate such a mission creating the MYSQL data base as a depository of security related data will allow the sharing of such data across multiple security tools deployed as mobile APPs.

Hackers are consistently use a dynamic strategy, in other words they are constantly devising new attack techniques and are aware of the importance of minimizing their attack fingerprint. Indirect methods such as this can be effective because adding to the routing cache table can be viewed as a subsequent event to a primary event such as attacking the secure shell daemon. This is an unconventional strategy and a hacker would really need to think outside the box to thwart its effectiveness. Ultimately, this makes it more difficult for a hacker to: go undetected, cover their tracks and ultimately succeed.

References

[1] Benvenuti, C. (2006). Understanding LINUX Network Internals. O'Reilly Media, Inc.

[2] Bernat, V. (2011). Tuning Linux IPv4 route cache. Retrieved from: http://vincent.bernat.im/en/blog/2011-ipv4-route-cache-linux.html.

[3] Bernat, V. (2011). Tuning Linux IPv4 route cache. Retrieved from: https://github.com/vincentbernat/www.luffy.cx/blob/master/content/en/blog/2011-ipv4-route-cache-linux.html

[4] Cho, Y.)., Qu, G.)., & Wu, Y.). (2012). Insider threats against trust mechanism with watchdog and defending approaches in wireless sensor networks. Proceedings - IEEE CS Security and Privacy Workshops, SPW 2012), 134-141. doi:10.1109/SPW.2012.32

[5] Clinton, L. (2013, March). The evolution of the cyber threat and public policy. BrightTalk. Retrieved February 19th, 2015 from: https://www.brighttalk.com/community/it-security/webcast/8883/67297

[6] Dev. L. (2014). The Five Most Common Cyberattack Myths-Revealed. Retrieved from: http://www.cybereason.com/five_most_common_cyberattacks_myths_revealed/.

[7] Geier, Eric. (2015). 10 Android Apps for Linux Server Admins. Retrieved from http://www.linuxplanet.com/linuxplanet/reviews/7301/2.

[8] Guster, D., Abdul, R., and Erich Rice. (December, 2015). Mitigating Virtual Machine Denial of Service Attacks from Mobile APPS. Journal of Network and Information Security Volume 3. Issue 2. pp. 21-31.

[9] Github – serpro. (2014). Android-PHP-Encrypt-Decrypt. Retrieved from: https://github.com/serpro/Android-PHP-Encrypt-Decrypt/.

[10] Jichkar, M. A., & Chandak, D. B. (2014). An implementation on detection of trusted service provider in mobile ad-hoc networks. doi:10.14445/22315381/IJETT-V11P213.

[11] Meyer, D. and A. Partan. (June, 2003). S-BGP/soBGP Panel: What Do We Really Need and How Do We Architect a Compromise to Get It? NANOG 0306.

[12] Munsell, A. (2014). Setup Hack and HHVM on Digital Ocean. Retrieved from: https://www.andrewmunsell.com/blog/setup-hack-and-hhvm-on-digital-ocean/.

[13] Nguyen, B. (2004). LINUX File System Hierarchy. Retrieved from: http://www.tldp.org/LDP/Linux-Filesystem-Hierarchy/html/proc.html.

[14] Saha, S., Lukyanenko, A., & Ylä-Jääski, A. (2015). Efficient cache availability management in Information-Centric Networks. Computer Networks, 8432-45. doi:10.1016/j.comnet.2015.04.005.

[15] Snader, R., Borisov, N. (February 2008), A tune-up for Tor: Improving security and performance in the Tor network. In Proceedings of the Network and Distributed Security Symposium - NDSS'08, Internet Society.

[16] Shuo, C., Jun, X., Kalbarczyk, Z., & Iyer, R. K. (2006). Security Vulnerabilities: From Analysis to Detection and Masking Techniques. Proceedings Of The IEEE, 94(2), 407-418. doi:10.1109/JPROC.2005.862473.

[17] Symantec. Advanced Persistent Threat (APT): The Uninvited Guest. (n.d.). Retrieved February 19, 2015, from: http://www.symantec.com/theme.jsp?themeid=apt-infographic-1

[18] Teller, T. (2012). The Biggest Cybersecurity Threats of 2013. Forbes. Retrieved from http://www.forbes.com/sites/ciocentral/2012/12/05/the-biggest-cybersecurity-threats-of-2013-2/

[19] Zhang, Y., Mao, Z. & Wang, J.). (2007). A firewall for routers: Protecting against routing misbehavior. Proceedings Of The International Conference On Dependable Systems And Networks, (Proceedings - 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2007), 20-29. doi:10.1109/DSN.2007.7