# CyberX 2018: Network Architecture and Firewall

**Mohammad Salloum**
*Department of Electrical and Computer Engineering*
*Queen's University*
*Kingston, Canada*
*m.salloum@queensu.ca*

***Abstract*—CyberX is a replication of the previously known CDX (Cyber Defence Exercise) which was organized by the NSA (National Security Agency). CyberX is entirely organized and run by RMC (Royal Military College of Canada) now and it achieved the same interesting and informative experience as the original CDX. In CyberX, two teams compete in a cyber warfare, under the supervision and organization of a third team. The defensive team (blue cells), although engaged in several offensive operations, is composed of undergraduate and graduate students from RMC while the offensive team (red cells) was a team of experienced penetration testers. Each blue team member was responsible for a certain part of the blue team's network. My role was the network architecture and firewall. It was a key and critical role in the exercise and, being my first CyberX experience, I had the opportunity to learn a ton of new knowledge and gain a lot of valuable experience. In this paper, I will elaborate more on my role and go into implementation details and explain the problems that I faced and how I managed around them.**

***Index Terms*—CyberX, RMC, OpenBSD, Network Architecture, Firewall, Security**

## I. Introduction

When NSA decided to discontinue CDX, RMC professors worked together to organize the new CyberX to allow new students to benefit from the same experience that the NSA used to provide. CyberX provided a productive learning environment and a life-time experience. CyberX is composed of 3 main teams: Blue cells, Red cells and the White cells. The blue cells consist of two separate teams, the graduate team and the undergraduate team. Although these teams work separately, their objectives are the same. The blue team was responsible for building and maintaining a network that hosts multiple public services listed in the directives[3]. These services include a mail server, web server, FTP server and others. The network also includes workstations that conducted cyber operations in a simulation of a war zone. The network also included an IDS (Intrusion Detection System), IPS (Intrusion Prevention System), a Domain Controller, an NTP server and a local and a public DNS (Domain Name Server). The red team's job was to provide infected workstations to be injected in the blue team's network and to try to compromise the blue team's servers through cyber attacks from the external network. As for the white team, they were the blue team's headquarters and the overall CyberX organizers. The white team provided special operation instructions to the blue team through e-mail and IRC (Internet Relay Chat) services. Operations included attacking certain points in the warfare simulator and flying the virtual drones around and injecting new workstations into the blue team's network. The red team was allowed to infect these inject workstations as they want and the blue team was supposed to clean these stations to block any malware that allows the red team to compromise the internal network through.

My role in the exercise included the network architecture and the firewall. I provided the IPs and the routing between all the internal servers and also provided a gateway to the outside network. This was done using two different firewalls on which I also provided different packet filtering rules to protect the network from unwanted external access to our servers. These firewalls were deployed using OpenBSD [1] systems. OpenBSD was chosen as it has powerful routing and packet filtering capabilities. I had to setup the routing and the filtering for all of the network's hosts and, to provide extra security, each host was assigned a distinct VLAN with a /30 subnet that allows only 2 hosts: the actual host and its gateway. A lot of the network design factors were inferred from the work done in the previous years. This saved me a lot of time and helped me learn from what the previous network architects in CDX have achieved and allowed me to improve upon their work. I will elaborate on the implementation details in later sections.

### A. Scoring

Headquarters have designed a scoring system that will be used to rate our work at the end of the exercise. This was done by a monitoring software called RubberNeck. RubberNeck was installed on all public access services and it reported status to headquarters. Points were deducted for the loss of availability of the public services and for any breach in confidentiality or integrity of the internal machine data as well.

## II. Implementation

Implementation can be divided into two main categories: Network Architecture and Firewall. Both categories were implemented on two OpenBSD systems. Each OpenBSD system contained routing configuration and packet filtering at the same time. One system was used as an external firewall, the means of communicating with the outside world. The other system was used as an internal firewall where the main packet filtering and internal routing happened.

### A. Network Architecture

The network design and architecture played a crucial part in the exercise as it was the fundamental component that tied everything together. A solid network architecture is essential for a healthy exercise overall. Any network problems that may occur can highly impact the work of other blue teammates and therefore I had to make sure I take into account all of my teammates' work and provide them with all their networking requirements. To do this, I referred to the directives [3] provided by the CyberX headquarters and communicated with my teammates to make sure that all their expectations were met. When designing the network, I started off from the network that was implemented in CDX 2017. It was a strong network that needed little modification and so was a solid

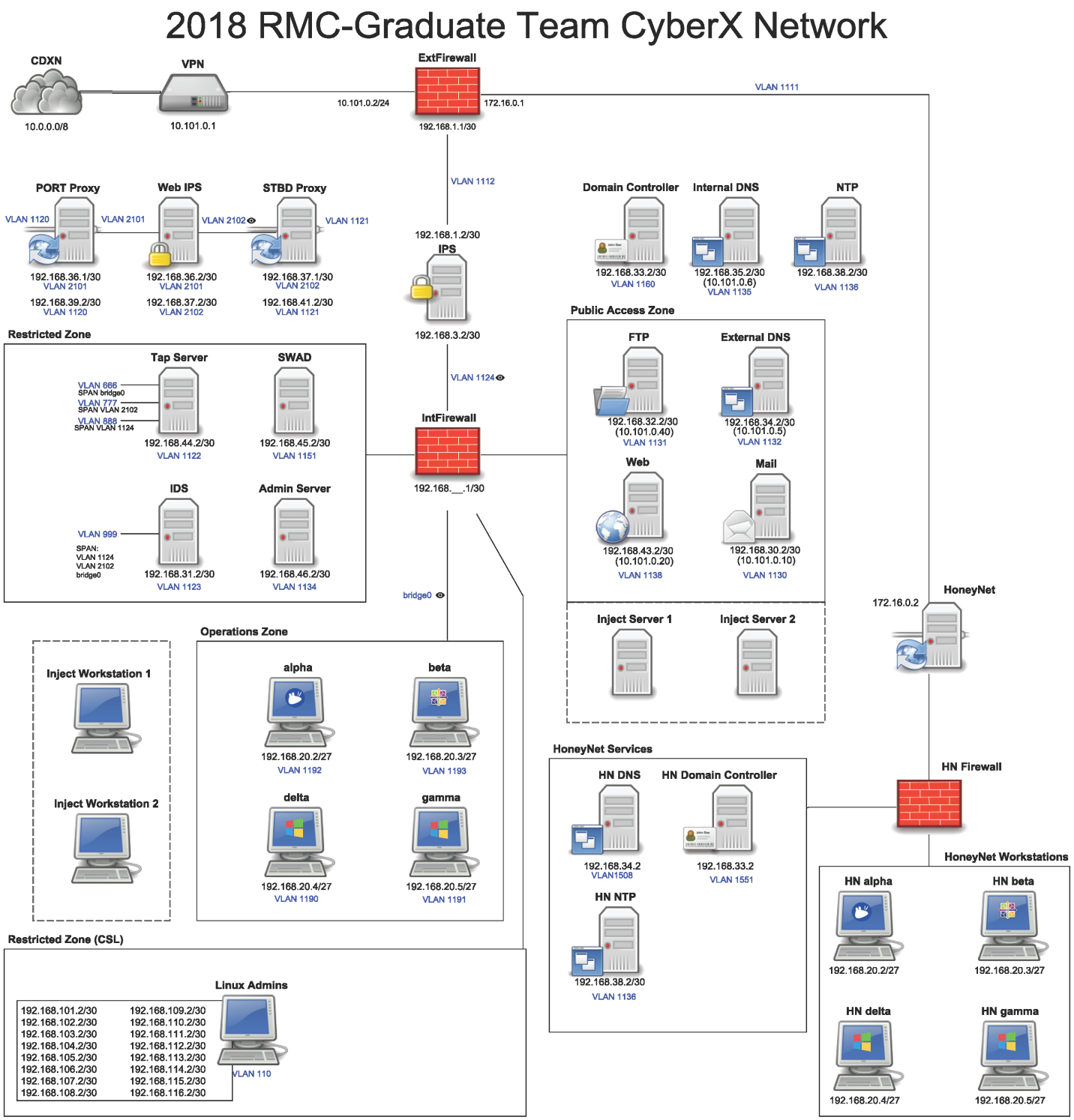


Fig. 1: CyberX 2018 Network Diagram

foundation for the new network. A lot of workstations had to be removed from the previous network to accommodate with the new CyberX directives. However, the foundation of the network was kept the same. The new network diagram can be seen in figure 1.

To implement this network, the OpenBSD was used as a router. In this section we will focus on the routing part of the OpenBSD system instead of the packet filtering part which is more of the implementation of the firewall. For the internal network, I used the IP network 192.168.0.0/16. This provided a gigantic range of IPs for me to use for the network hosts. Every host was also designated a disctinct VLAN with a subnet of /30 which tightened the security. Having a /30 subnet means that there are only 2 hosts available, the actual host and its gateway which here is the internal firewall (our OpenBSD system). This way, any communication to or from a host has to go through the internal firewall and no one can listen on that VLAN other than the firewall as there are no more available IPs to tap on.

```
inet 192.168.110.1 255.255.255.252 \\
NONE vlan 110 vlandev vmx0
inet alias 192.168.101.1 255.255.255.252
inet alias 192.168.102.1 255.255.255.252
inet alias 192.168.103.1 255.255.255.252
inet alias 192.168.104.1 255.255.255.252
inet alias 192.168.105.1 255.255.255.252
inet alias 192.168.106.1 255.255.255.252
inet alias 192.168.107.1 255.255.255.252
inet alias 192.168.108.1 255.255.255.252
inet alias 192.168.109.1 255.255.255.252
inet alias 192.168.111.1 255.255.255.252
inet alias 192.168.112.1 255.255.255.252
inet alias 192.168.113.1 255.255.255.252
inet alias 192.168.114.1 255.255.255.252
inet alias 192.168.115.1 255.255.255.252
inet alias 192.168.116.1 255.255.255.252
inet alias 192.168.18.1 255.255.255.0
inet6 alias fd00:a:6500:a110::1
inet6 alias fd00:a:6500:a101::1
inet6 alias fd00:a:6500:a102::1
inet6 alias fd00:a:6500:a103::1
inet6 alias fd00:a:6500:a104::1
inet6 alias fd00:a:6500:a105::1
inet6 alias fd00:a:6500:a106::1
inet6 alias fd00:a:6500:a107::1
inet6 alias fd00:a:6500:a108::1
inet6 alias fd00:a:6500:a109::1
inet6 alias fd00:a:6500:a111::1
inet6 alias fd00:a:6500:a112::1
inet6 alias fd00:a:6500:a113::1
inet6 alias fd00:a:6500:a114::1
inet6 alias fd00:a:6500:a115::1
inet6 alias fd00:a:6500:a116::1
```

Fig. 2: hostname.vlan110 file

On the OpenBSD system, all routing was handled and the network communication was maintained. To do this while providing a distinct VLAN for each host on the network, a virtual interface needed to be created on the internal OpenBSD system. To do this, I had to create a new interface file for each VLAN. These interfaces were named after their assigned VLAN to minimize confusion. These virtual interfaces were the gateways of the hosts of each VLAN. To create a virtual interface on OpenBSD, you simply need to create a hostname.interface_name file and place it in the /etc/ folder and restart the network. There were some special cases in which multiple hosts shared the same VLAN and a good example of this is the VLAN of the admin workstations. The admin workstations were the computers used by the blue team members to configure and maintain their servers on the main network. These can be seen in figure 1 at the bottom in the Restricted Zone. There were 16 different devices on VLAN 110, however, each had a subnet of /30 and had a virtual interface on the internal firewall as their gateway. To implement this on the firewall, I created one virtual interface for VLAN 110 and provided it with an alias IP for each gateway required by the 16 admin workstations. The file hostname.VLAN110 can be seen in figure 2. It is evident that IPv4 and IPv6 are provided in the interface file and that is because the internal network supported both IP protocols.

Another aspect handled by the internal firewall was the bridging of different network interfaces. In order for the intrusion detection team to monitor the network's traffic, they needed to listen on interfaces bridged between them and the target hosts. Bridging was implemented by creating extra virtual interfaces that included multiple VLAN interfaces. A sample bridge interface file can be seen in figure 3.

```
add vlan2102
addspan vlan777
addspan vlan999
up
```

Fig. 3: hostname.bridge2 file

Bridge2 included VLANs 2102, 777 and 999. Therefore, the intrusion detection team was able to use VLAN 2102 to monitor traffic across VLANs 777 and 999. The same method was applied to all the VLANs required by the intrusion detection team to tap on.

Notice in figure 1 the connection between the internal and the external firewall. There is an inline IPS (intrusion prevention server) which is placed this way to monitor all traffic incoming and outgoing from the internal network. To implement this structure, the default gateway on the internal firewall was set to one of the interfaces of the IPS and on the external firewall I had to route all traffic destined to the 192.168.0.0/16 network to the other IPS interface.

All network communication within the internal network was handled by the internal firewall. As for the external firewall, it acted as the gateway to the outside network, the 10.0.0.0/8 range. Its main job was to NAT from the 192.168.0.0/16 network to that 10.0.0.0/8 network. This was handled using

special PF (Packet Filter) [2] rules. The NAT rule can be seen in figure 4. Specific routing associated with the external connection is handled on the external firewall too.

```
pass out on vmx1 inet from 192.168.0.0/16
to any nat-to 10.101.0.2
```

Fig. 4: NAT using PF rules

*1) What did not Work:* There were no major issues with the network during the exercise. All required network communications were maintained throughout the exercise. One thing to mention that never worked was IPv6 connection to the external network. Note that IPv6 was working perfectly within the internal network. This was a low priority case and so it was not given enough time to be resolved. I was able to narrow down the problem to the routing of IPv6 to the external network from the internal firewall. What that means is that the packets destined to an IPv6 address on the external network was getting stuck in the internal firewall when it had to be forwarded through the external interface, the VLAN1124 interface on figure 1.

### *B. Firewall*

In this part, things get more interesting. The OpenBSD systems provide a DSL (Domain Specific Language) for filtering packets called PF. PF is very intuitive and powerful, I was able to effectively and flexibly filter traffic going through the network with no restrictions. Using PF, I was able to monitor traffic by PF rule, this helped me a lot when I was troubleshooting my rule set. PF also is able to read hosts from system files which allowed me to efficiently organize my PF code.

To start writing my PF rules, I referred first to the previous year's PF file. From that file, I extracted all the main filtering criteria and zone structures. The hosts were distributed into zones on which the PF rules were applied. These zones were distinguished within the PF file using hosts list variables. These zones can be seen in figure 1. We can see that we have three main zones: a Restricted Zone (RZ), a Public Access Zone (PAZ) and Operations Zone (OZ). There are still some servers that were not categorized into a main zone, these devices were handled using PF rules specific to them. We will not go into details of the Honey Net zone as it was outside the scope of my work.

PAZ was allowed to be accessed from the outside network. In fact, points were deducted for any availability failure of any of the services in the PAZ section. All access for the PAZ was specified by port number; only ports that host public services were open and the rest were all blocked. The OZ was allowed to communicate with the external network as well, however, direct access to the OZ from the outside network was all blocked. The admin workstations were allowed to access the PAZ through SSH, and that is to allow the blue team members to manage their servers remotely. In addition, inject servers and workstations were also given access in accordance to their requirements which were specified by Headquarters. I also added a black list file in which all communication was blocked to any host specified in this file. This was extremely effective when we were being attacked by the red team and we needed rapid IP blocking. We will see these PF rules in details later on in the paper.

What makes PF powerful is that not only it blocks traffic, it has redirecting and forwarding capabilities. This was essential in setting up the web proxy IPS (Intrusion Prevention System). All incoming and outgoing HTTP and HTTPS traffic was being forwarded through the proxy. This allowed the Web IPS team to monitor all web traffic, whether encrypted or not. However, we faced several problem associated with the web proxy setup and I had to bypass the web proxy for most of the exercise.

*1) Default Filter:* This is one of the most important rules in the PF file, the default case. This case is triggered if a packet did not match any of the rules. By default, this packet is blocked and dropped right away. The only traffic allowed through the network is what has been specified in the PF rules. This rule can be seen in figure 5.

```
# default

anchor "DEFAULT-BLOCK"{
  block all
}

anchor "DEFAULT-PASS" {
  #pass log all
  pass in log all
}
```

Fig. 5: Default PF rules

We can see that also by default, I allowed all incoming packets. This is to avoid confusion and maintain consistency, all traffic restrictions was applied on the outgoing traffic and not the incoming, except for some special cases which we will see later on.

These rules are wrapped around anchors. These anchors allowed me to filter traffic on my firewall that is triggering these rules. This was particularly helpful in PF rule debugging.

*2) ICMP Filter:* For proper network debugging and being a relatively safe protocol, ICMP was allowed through the network with no restrictions. The rules can be seen in figure 6.

```
# ICMP
anchor "PING" {
  pass out log proto icmp
  pass out proto icmp6
}
```

Fig. 6: ICMP PF rules

*3) Web Filter:* As mentioned earlier, all web traffic should be re-routed to the proxy servers. There are two proxy servers,

```
anchor "WEB" {
  # Web proxy - to proxy1
  pass in log on $ext_if proto tcp to
  {$web_server $ext_net} port {www https}
  route-to ( $proxy1_if $proxy1)

  pass in log on {$inject_if $oz_if
  $admin_if}   inet proto tcp from
  {<injects> <oz> <admins>} to {$web_server
  $ext_net } port {www https} route-to
  ($proxy1_if $proxy1)

  pass out log on $proxy1_if inet proto
  tcp to {$web_server $ext_net} port
  {www https}

  # Web proxy - from proxy2
  pass out log on $ext_if proto tcp from
  {$proxy2 $proxy2_6} to {$ext_net} port
  {www https}

  pass out log on $web_if proto tcp from
  {$proxy2 $proxy2_6} to {$web_server} port
  {www https}
}
```

Fig. 7: Web PF rules

I will refer to them as proxy1 and proxy2. Web traffic, incoming or outgoing, is forwarded to proxy1. In turn, proxy1 forwards the packets it receives to the Web IPS which then forwards the traffic to proxy2. Proxy2 forwards the traffic back to the firewall which routes this traffic to its original destination. To handle this, I wrote the rules that are shown in figure 7.

*4) FTP Filter:* A requirement from HQ was to set an FTP proxy on the internal firewall, this was easily handled by the PF rules. The FTP server was also allowed to be accessed from the external network and from the internal one. The rules for FTP packet filtering are shown in figure 8.

*5) SSH:* For SSH, PF rules were implemented a bit differently than other rules. Here, I made use of the tagging feature in PF. All SSH traffic coming from an admin workstation is tagged with a special ssh_admin tag. Then, all outbound SSH traffic that is going to an internal host or to the 10.101.0.0/24 network and that has been tagged with ssh_admin is allowed to pass. Note that the 10.101.0.0/24 is the external network that belongs to the graduate team. The SSH filter rule can be seen in figure 9.

Note how there are also SSH filtering for inject servers. These servers were introduced by HQ during the exercise and so they were handled separately. There were no SSH problems and everything worked properly.

*6) Black Listing:* Occasionally, the intrusion detection team identified public hosts doing malicious activity against our network and would report their IP addresses to me to block

```
anchor "ftp-proxy/*"
anchor "FTP" {
  # FTP proxy handler
  pass in quick log on $ext_if inet proto
  tcp to $ftp_server port ftp divert-to
  127.0.0.1 port 8021

  pass in quick log on {$oz_if $fac_lap_if}
  inet proto tcp to {$ftp_server $ext_net}
  port ftp divert-to 127.0.0.1 port 8021

  pass in quick log on {$admin_if $dc_if
  $ext_if} inet proto tcp to {$ftp_server}
  port ftp divert-to 127.0.0.1 port 8021

  pass out log on $ext_if inet proto tcp
  from (self) to $ext_net port ftp keep state

  pass out log on $ftp_if inet proto tcp
  from (self) to $ftp_server port ftp
}
```

Fig. 8: FTP PF rules

it completely from the PF rules. Therefore, I created a rule that blocks all traffic of hosts listed in a black list file in the system and I simply only added an IP that I want to block to this file and refreshed the filtering rules. The black list filter is in figure 10.

This is a straight forward rule but what made it work effectively is that it was placed at the very end of the PF file and in PF, the last matching rule is executed.

Other PF rules were of similar structure of the ones already mentioned and therefore I will not go into their details.

*7) What did not Work:* All main PF issues were handled and dealt with effectively with no problems. However, there was a low-priority issue that was never fixed. Accessing the FTP in the passive mode from the external network did not work and since it was an FTP issue it had a very low priority from my side and so I never found the time to address it. If given the time, I was told that it may be handled using PF but as I already mentioned, I never found the time to implement it.

*8) What Went Wrong:* As the network grew, the PF rules became more and more complicated. I took my time to make sure that no rules overwrite one another and to keep all unwanted traffic blocked while allowing what was needed. However, some mistakes were made but most problems that occurred had quick fixes.

One issue that I faced was the order of the rules. I had to make sure to always take the order into consideration, especially when I attempted special bypass policies for specific hosts. PF executes the last matching rule only. I had a server completely blocked just because of the rules' order. The best solution for this problem is to avoid writing generic rules and always using comments to remind myself what each rule is

```
anchor "SSH" {
  # SSH

  # tag reliable sources
  pass in log on $admin_if inet proto
  tcp to port 22 tag ssh_admin

  pass in log on $SWAD_if inet proto
  tcp to $tap_server port ssh tag ssh_admin

  # only allow tagged ssh packets
  pass out log on $RZ_if tagged ssh_admin
  pass out log on $PAZ_if tagged ssh_admin
  pass out log on {$proxy1_if $proxy2_if}
  tagged ssh_admin

  pass out log on $ext_if inet proto tcp to
  {$int_net 10.101.0.0/24} tagged ssh_admin

  pass out log on {$oz_if $alpha_if
  $beta_if $delta_if $gamma_if} tagged
  ssh_admin

  # Inject Web server (FACServ)
  pass in log on $ext_if inet proto tcp
  to $FAC_server port 22 tag ssh_admin

  pass out log on $ext_if from $FAC_server
  tagged ssh_admin

  # Inject Web Server (CSAPServer)
  pass in log on $ext_if inet proto tcp
  to $CSAP_server port 22 tag ssh_admin

  pass out log on $CSAP_if inet proto tcp
  to $CSAP_server port 22 tagged ssh_admin
}
```

Fig. 9: SSH PF rules

```
anchor "BLACK" {
  block on $ext_if to <black_list>
  block on $ext_if from <black_list>
}
```

Fig. 10: Black list PF rules

for.

There was one particular mistake that actually exposed one of our servers. The server is allowed to access the token agent server on port 443, however, I allowed the server to connect to all of the outside network on that port. The faulty rule is in figure 11.

It is unclear if the issue was due to a miscommunication within the team or if it was my mistake. This problem was identified as I was going through the code just for a check

```
# allow DC to connect to token agent on
port 443

pass out log on $ext_if inet proto tcp
from $dc_server to $ext_net port {https}
keep state
```

Fig. 11: Wrong PF rule, server exposed

and was fixed immediately. Its fix was very straight forward: specify the destination. The fixed rule can be seen in figure 12.

```
# allow DC to connect to token agent on
port 443

pass out log on $ext_if inet proto tcp
from $dc_server to $token_agent port {https}
keep state
```

Fig. 12: Corrected PF rule, server protected

There was a moment of panic when the problem was identified, we counted on the red team not identifying this vulnerability. Later on, the red team informed us that they were able to get through this vulnerability in the firewall and compromise the server. It was a lesson well learned.

## III. Acknowledgements

I want to thank Dr Leblanc and Dr Knight and all the other RMC professors who worked hard to replicate the CDX experience, it was a job very well done. I also want to thank Andrew Mckay who was a secondary on the Network Architecture and Firewall role, his help has sped up setting up the firewalls a lot and this was especially needed as I had other courses to devote some time to.

## IV. Conclusion

Overall, there were no issues with the network or the firewall. They had a stable implementation which proved to be strong throughout the exercise. It was a great responsibility to be in charge of these two key factors of CyberX and it was a very fruitful experience in which I gained a lot of valuable knowledge that is not easily found. I found that the network and the firewall roles can be broken into two separate roles in order to distribute the gigantic work load, however, I enjoyed this work load because it resulted in me learning a ton of valuable information. I included in the paper some of the problems and mistakes that happened, this is to help the next network and firewall architect improve upon this year's work and learn from what has been done. I would enjoy participating in the next CyberX, although I would want to take on a smaller role this time.

## REFERENCES

[1] Openbsd. https://www.openbsd.org/. Accessed: 2018-04-22.
[2] Openbsd pf manual. https://www.openbsd.org/faq/pf/. Accessed: 2018-04-22.
[3] Royal Military College of Canada. *CyberX 2018 Exercise Directives*. Computer Security Lab, 2018.