KTH Royal Institute of Technology

Building Networked Systems Security EP2520

System design and implementation details for ACME

Authors
Enze Wang *(enzew@kth.se)*
Keyao Huang *(keyao@kth.se)*
Ziyang Song *(songz@kth.se)*
Zhiyuan Lin *(zhiyuanl@kth.se)*

March 2023

Contents

1 Project requirements

1.1 Basics

Employees should be able to connect to the cooperate network (two LANs). Employees can exchange files (confidentiality, integrity, and authenticity). For Mobile devices/untrusted devices, two-factor authentication is needed. Employees with laptops should be able to work remotely via VPN (except for sensitive operations). Routers in both Stockholm and London should provide wired/wireless connections (laptops and mobile devices). There will be working hours restrictions on-site for the file exchange server.

1.2 Securities

Authentication for users that request services. Secure Connection between employees and servers, employees and employees, logging network traffic and requests should be satisfied. Information exchange should be encrypted, authenticated and integrity ensured. For Wireless Access, Authorization and Authentication should also be guaranteed other aspects includes alerting for attacks, preventing clogging DoS, guaranteeing domains cannot be exploited or employees misleading, and handling equipment stolen and credentials compromised.

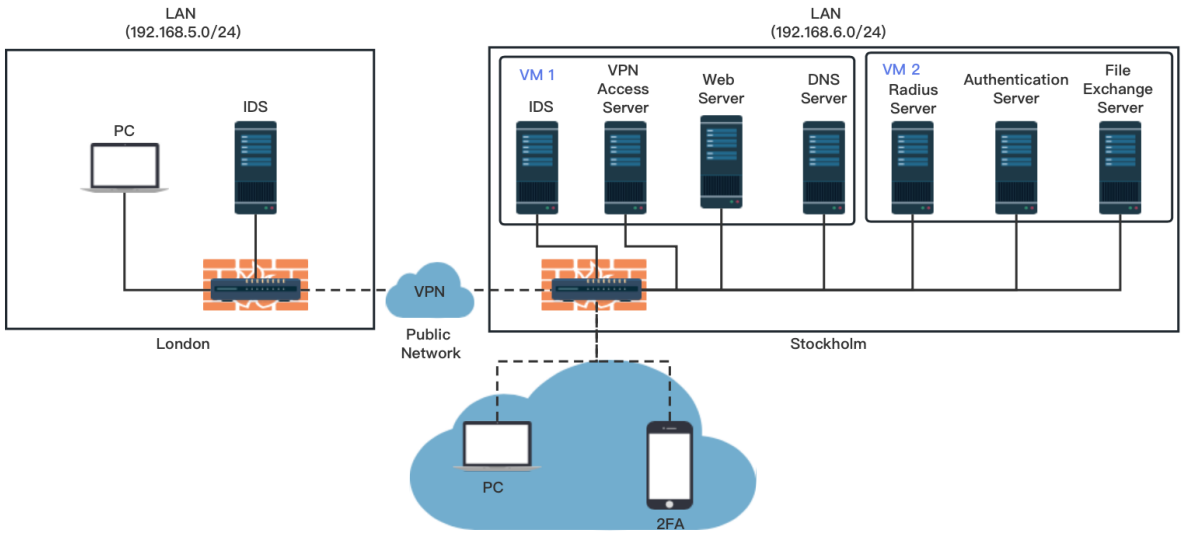

Figure 1: Topology of Network

2 Networked System Design

2.1 Explanation of Network Topology Diagram

As shown in the figure, the whole network architecture is divided into the London part and the Stockholm part, both of which have their own LANs. The two LANs communicate via VPN, and employees who are not in the company also need to connect to Stockholm's company via VPN. In both LANs we have deployed IDSs to monitor incoming and outgoing traffic. In the Stockholm company we set up six servers: VPN Access Server, DNS Server, Radius Server, Web Server, Authentication Server and File Exchange Server. It is worth mentioning that for the web server, we designed two web services, normal web service and critical web service, which are provided through different ports.

2.2 Employees Connect to the Corporate Network

Configuring a VPN connection between the two routers allows the London company or external users who are not in the company to be able to access the server located in Stockholm through the router connected to London as well. Employees of the Stockholm-based company can access the servers within the LAN by connecting directly to the Stockholm router.

2.3 Exchange Files between Employees

Install and configure a file-exchange server in the network to allow employees to securely transfer files for only employees that transfer files to each other can access the files. We choose Nextcloud as our server. Implement built-in SSL/TLS encryption function for confidentiality, integrity, and authenticity. Enable user 2FA authentication to access the server and password-based authentication.

2.4 Authentication

We use FreeRADIUS[1]for wireless authentication which is a open source authentication tool based on RADIUS protocol. It will handle requests from the router which performs as NAS for clients who attempt to access the network. Only users with certain user/password combinations can access the wireless service. We choose WPA/WPA2-EAP encryption between router and server. We also activate the built-in 2FA feature of OpenVPN-as to perform secondary authentication of external users (using Google Authenticator).

2.5 Securely Work Remotely (Confidentiality)

Establish Virtual Private Network (VPN) to allow employees to securely work remotely from their laptops and allow communicate between two routers (London and Stockholm). Implement encryption and user authentication for the VPN connection. We intend to firstly flash OpenWRT[4] on both routers in order to establish VPN between them. We will use openVPN[2] and the server and client are included in OpenWRT firmware. For employees who are not in the office, they need to install the OpenVPN client on their cell phones or computers to connect to the OpenVPN server located in Stockholm.

2.6 Wireless Connections

The wireless would be implemented in IEEE 802.11. Configure the Access point to support secure connections, by enabling WPA/WPA2-EAP encryption for wireless connections.

2.7 Firewall and IDS: Internal Security

In order to be sure that only internal network users can access the servers in Stockholm, we need to place a firewall and set the firewall rules on routers to block all packets from outside the public network except VPN tunnels. We plan to use iptables to achieve the firewall or utilize the built-in firewall function of OpenWRT. we also need an Intrusion detection system for the detection of potential network attacks, located at the Stockholm site and London site, independent of but connected to the router. After discussion, we plan to use SNORT[6] to achieve IDS, which is one such open-source IDS. SNORT can also deal with Dos attacks as we can detect unwanted behaviors on our internet.

2.8 Other Security

SNORT can also deal with Dos attacks as we can detect unwanted behaviors on our internet. We can prevent employees from being misled on the internet and other DNS attacks(DNS hijack) by using DNSSEC. Any equipment stolen and credentials compromised should be reported and get the certificates revoked as soon as possible.

3 Networked System Implementation

3.1 Routers

Both routers are running OpenWRT and giving them static LAN addresses. 192.168.6.0/24 (Stockholm) and 192.168.5.0/24 (London). The two stay connected via OpenWRT's built-in OpenVPN. Once connected, the London router is assigned an address of 192.168.8.x and allows users connected to the London router to access the servers of the Stockholm LAN. The Stockholm router is used as a VPN server and only accepts connections from the London router. For external users, we also deployed an additional VPN access server, which is mentioned below. Our router also provides wireless functionality (using WPA-EAP encryption) and authenticates users with FreeRadius (discussed below). Once authenticated, the user can use Wi-Fi to access the Internet.

3.2 OpenVPN

We use OpenVPN for VPN setup. First, we download and use the OpenVPN server built into OpenWRT on the Stockholm router to handle and maintain the connections from the London router. Meaning, the Stockholm router is the VPN server and the London router is the client. While this VPN connection is stable, it is impractical to handle multiple client connections due to the limited memory of the router, etc. Therefore, we deployed an OpenVPN access server (openvpn-as) on the VM to handle connections from external clients. Users need to log in to openvpn-as and scan the provided QR code to bind to the Google Authenticator on the user's phone when they first download the .ovpn file. Each time the VPN is connected, the external user has to enter the account password and perform 2FA via Google Authenticator. Since our VM and router have different IPs, we need the router to do port forwarding to forward the VPN connection requests from external users to the OpenVPN access server. Finally, to avoid direct connection conflicts between the two servers, we will use different ports on the router to open the service. the OpenVPN access server will be on port 1194, while the router's OpenVPN server will be on port 1195.

3.3 Web Server

The Web Server is divided into two parts, a general web service, accessible to all employees, and a critical web service, accessible only to employees located in London and Stockholm, and not to external users using VPN. The normal web service is provided through apache and is opened on port 80 of the VM, while the important web service is deployed on the same VM (using docker) and is opened on port 8080. To achieve the requirements, we use iptables to drop all packets from the external user's VPN IP address (192.168.9.x) and accept only packets from the IP addresses of Stockholm and London's devices.

3.4 OpenSSL

For wired authentication, OpenSSL is used as a standard library for client and server to generate its private key, receive its certificate from CA and sign. For root CA server, it holds a static public IP (designed external) and grants certificates for leaf hierarchy, which is emulated as with self-signed certificate in demo case and can be configured with a real one in life. For intermediate CA server, it generates certificates for clients and servers and also maintains an internal CRL. It is designed in classical client-server mode, which listens to corresponding port waiting for CSR, and respond requested certificates back. But for simplification, it is set generating locally and distributing manually. For both client and server, mutual peer verification using certificates can be done, as it is deployed by pre-configuration (e.g. client's installments / server's loading directory settings). And TLS can be set with either default method in server's application or calling OpenSSL verify commands. Meanwhile, CRL is maintained by both root CA and intermediate CA and can be fetched by server under certain circumstances. If revocation is needed, a request with certificate to be revoked would be sent to CA and CA revokes it and updates the database with OpenSSL ca command.

3.5 Freeradius

For wireless authentication, we deploy freeradius server(192.168.6.133) in a virtual machine running Ubuntu. We add a few users in configuration file as employees who can use the information to verify their identities. We set the IP address of the router(192.168.6.1) as the IP address of the server's client. Thus, whenever an employee try to connect

to the wireless network router will send request to freeradius server. After the server verifying the user/password binding information, request-accept or request-reject message will send back.

3.6 Nextcloud

For our file-exchange server, we run Nextcloud server on Docker Desktop. We choose nextcloud: latest image for our container. After some basic configurations, from trusted domain we can access Nextcloud server at port 9006. Nextcloud has various useful applications that can be downloaded and enabled to fulfill functions such as secure file exchanging, upload/download files and etc. For our demo, we just need to enable basic file exchanging and file access control. With file access control, we set the working as 08:00 to 18:00 when the files are available. "Employees" Joe and Frank are created for testing uses and their user/password are created accordingly. There is also a admin user, we also enable the 2FA(Google Authenticator) for admin user.

3.7 SNORT

For internal security of our corporate network, we decided to use SNORT to deploy our Intrusion Detection System. At first, we planed to deploy IDS on the router to monitor the traffic from external network. However, due to the reason that the router do not have enough memory space, we chose to use a virtual machine which is deployed in the Stockholm network to set up SNORT, then we tried to install iptables-mod-tee on the router to mirror the traffic and ran some iptables commands to copy the router traffic to the IP address of the SNORT virtual machine. However, we did not realize it for unknown reasons. Finally, we decided to deploy IDS on the same virtual machines with web server and file transfer server, which can help us monitor the traffic that is coming to these servers. Furthermore, by downloading some free ruleset files from SNORT website and configuring them to SNORT, SNORT can analyze situations such as malicious port scannings, DDos attacks, SQL injections, DNS lookups and so on.

3.8 BIND9

In order to set up our own DNS server, we used to use BIND9 software to achieve the goal. The DNS server is deployed in the Stockholm branch network, which can help our company users to analyze the domain names and help access to the critical servers.

4 Supplementary Notes

Based on the feedback from the demonstration, we think it is necessary to make some additional clarifications. First, our user information is stored in different servers in a decentralized manner. When employees are hired, we create individual accounts and passwords in different servers. For example, in VPN Access Server, we go to the admin page -> User Mangement -> User Permissions to create a new VPN account. When an employee leaves or loses their account, we can delete or revoke the employee account and login information through the admin account. Secondly, we did not configure the VPN server in London Router because all the servers are located in Stockholm. Employees need access to Stockholm's LAN in order to access the servers. And the communication between employees such as file transfer should also be done in Nextcloud in Stockholm LAN.

For management of certificates, each client / server who is granted by CA would be attributed a serial number to identify (visible in serial blank in certificate). A list of serial number and CRL are maintained by CA, instead of copies of certificates. We are also queried about the physical location of database of all users' information. Considering a distributed design, a general database is not provided in this case. However if necessary, it could be set with CA server to handle fields such as `USERNAME`, `DUE`, `SERIAL`, `PASSPHASE`, `ACCESS`, e.t.c.

Appendix

Appendix A

README - Network Configuration

More detailed implementation details are available at Github (Especially the OpenSSL code). Check `ReadMe.md` file for general set up procedure and specific details and configuration files in corresponding folders.

Appendix B

Technical Details

OpenWRT

We found the corresponding OpenWRT version of the router according to the official OpenWRT documentation, here, and flashed the router according to the steps in the documentation. After installing OpenWRT, you can access the Web UI of the router through the default **192.168.1.1**. Next, we go to **Network** -> **Interfaces** -> **LAN** to set it to a static address. For London router -> 192.168.5.1/24 and for Stockholm router -> 192.168.6.1/24. To set up wireless encryption you can refer to the Freeradius section.

OpenVPN

VPN Access Server

First we need a virtual machine with Ubuntu and an OpenVPN account for downloading OpenVPN Access Server (Openvpn-as). Run the commands provided by the openvpn website in our virtual machine as root.

```
apt update && apt -y install ca-certificates wget net-tools gnupg
wget https://as-repository.openvpn.net/as-repo-public.asc -qO /etc/apt/trusted.gpg.d/as-
                                                    repository.asc
echo "deb [arch=amd64 signed-by=/etc/apt/trusted.gpg.d/as-repository.asc] http://as-
                                                    repository.openvpn.net/as/debian jammy main"
                                                    >/etc/apt/sources.list.d/openvpn-as-repo.
                                                    list
apt update && apt -y install openvpn-as
```

Once the installation is complete, you will be given your admin account and password and you should now be able to access the web interface of https://localhost:943/admin OpenVPN Access Server to create a VPN client. Go to **Network Setting**, enter the public IP address of the VM in Hostname or IP Address, then go to **VPN Setting**, enter 192.168.9.0 in Network Address, so that our client will be assigned the IP when connecting. Finally, go to **User Permissions**, where we can add the vpn client's information, including username and password, and set **Require MFA** to Enabled.

Then, we should set up port forwarding on the router to forward packets with destination ports 1194 and 943 to ports 1194 and 943 of the VM with OpenVPN Access Server installed. This is done by entering the router's luci interface (192.168.6.1), finding **port forwards** in **firewall**, and adding a new rule to forward packets arriving at the router with ports 1194 and 943 to the VM in the LAN (192.168.6.113).

Now the client will be able to access the OpenVPN server and download the login credentials (.ovpn file) via https://111.222.333.444:943 (Replace 111.222.333.444 with the router public IP). The user can use these credentials on PC or mobile to connect to the openVPN application by entering the password and 2FA. At this point, VPN users will be able to access other servers in the Stockholm LAN, such as the file transfer server.

Router VPN

We set up a dedicated VPN between the two routers.

Stockholm Router (Server): We used the official OpenWRT tutorial (here) to configure the VPN server on Stockholm's router. there are several parts to it (Preparation, Key management and Firewall). It is worth noting that we have to change its default port 1194 to port 1195 to avoid conflicts with OpenVPN Access Server. Then we can install the openvpn option for the web (here), after which we can view the VPN configuration on the web UI.

London Router (Client): We can configure the router client according to the tutorial on the OpenWRT website (here) and copy the contents of the client.ovpn generated in the VPN server (Stockholm router) to the London Router configuration (/etc/ openvpn/client.conf). Then we go to the Web UI of the London server (192.168.5.1), go to VPN -> OpenVPN and just open the VPN service. This should allow you to successfully access the server in Stockholm LAN via VPN.

Web Servers

We set up two Web services, one of which is a normal Web service and the second is a Critical Web service (which does not allow access to external VPN users). And, we installed Web Servers and OpenVPN Access Server on the same VM (192.168.6.113).

Normal Web Service:

```
sudo apt-get install apache2
sudo systemctl start apache2
```

The web service will be opened on the default port **80** and the web interface will be accessible when the user accesses 192.168.6.113:80.

Critical Web Service: We install **docker** in the VM and pull the httpd image and create the container so that the container web service maps to port **8080** of the VM.

```
sudo apt-get install docker.io
docker pull httpd
docker run -dit --name Criptical-Server -p 8080:80 -v "$PWD":/usr/local/apache2/htdocs/ httpd
```

To block access from external VPN clients, we use **iptable** on the VM to drop packets from VPN clients.

```
iptables -I INPUT -s 192.168.9.0/24 -d 192.168.6.113 -p tcp --dport 8080 -j DROP
```

Nextcloud

Since we use Docker Desktop for our Nextcloud server container. We need to download some essential software. First, we need to download Docker Desktop for windows. Make sure your computer have Hyper-V installed. Then, your Docker Desktop should work just fine. Now, we need to download Nextcloud image. In your windows terminal, type:

```
docker pull nextcloud
```

It should use default tag: "latest" and download the image. Now, you should see there is a nextcloud image in your Docker Desktop. The next step is making it run. Again, in your windows terminal type:

```
sudo docker run -d -p 9006:80 -v " " nextcloud
```

The content in quotes should be the directory where the data is stored on your computer. Also, we just set port 80 mapping to 9006(your choice). In Docker Desktop image, run your nextcloud image. In the pop-up setting window, configure the container name as nextcloud, ports as 9006 and set the host path and container path properly. Finally,

type: localhost:9006 , you should see the Nextcloud page. After that, you should configure your profile and start to enable applications to fulfill your purposes.

If you wish someone else from other network can access the server, you should also configure the config.php. In the config.php, find ”trust domain” and add whatever address that you wish to be able to access the server.

Freeradius

For Freeradius set-up, I recommend that try freeradius's getting started page. Before the configuration, you should make sure the router supports WAP/WAP2-EAP encryption. For example, we use OpenWRT to download wpad software for router. Also, it is very important since we use virtual machine that choose bridge connect so that radius server's IP address will be at the same network as computer.
We deploy our server in a virtual machine with a Linux Ubuntu system. The very first thing is install the server. There are a lot of ways to do it. We think the easiest way is using apt-get command. Make sure update first

```
apt-get update
apt-get install freeradius
```

Now, the server should be installed successfully. Use the following command to start debug mode

```
freeradius -X
```

You should see ”Ready to process requests”. If not, you can also see what is wrong in debug mode. If it works fine, type the command below or $Ctrl + C$ to end the service.

```
service freeradius stop
```

Now, we should edit client.conf to set the router as NAS. Under directory /etc/freeradius/3.0.

```
vim client.conf
```

Just add your router in the configuration file.

```
client router {
    ipaddr = 192.168.6.1
    secret = freeradius
}
```

Then, add some employees representing those try to access the wireless network. Still under directory /etc/freeradius/3.0.

```
vim users
```

```
frank Cleartext-Password := "password"
joey  Cleartext-Password := "password"
...
```

Don't set weak passwords as I do since it is just for the demo. It should be able to preform authentication over wireless connection now. In OpenWRT, under wireless network setting, set the radius server's IP address as 192.168.6.148 and port as default 1812 as well as password ”freeradius”. Now, users can connect to the wireless network with radius server authentication.

OpenSSL

OpenSSL is used with default extensions to distinguish the type and purpose of files. The following shell script codes display procedure outline of certificates distribution and emulated authentication e.t.c. (Full codes and notes in repositories.) Note certificate is prerequisite even if user account and password are hold. Certificate is not bind with user's identity nor machine, but only with private key. Therefore, private key should be stored locally and ought not be transmitted. Immediately revoked if leaked and re-requested if lost. The realization is mainly for demo and not much efficient, but the structure can be inherited in further developments. (This link is a pratical tutorial.)

CA

`ROOT_SELF_SIGNED_CERT_GEN.sh` is to generate certificate a self-signed root CA and intermediate CA for test, which can be replaced if a real root CA's certificate is povided. `CA_GEN.sh` is designed for CA to manually locally grant a single new user's key and certificate (For simple demo, shown draft below). It can be reformed easily to bash process a list of users (Not implemented). `CA_GEN_CERT.sh` is using client-server mode to listen to a fixed server port and receive CSR using `nc` command, and to sign certificates (For better demo, deprecated).

```
#!/bin/bash
# Make sure ca.key cert-bundle.pem (...)
# Get name and dir for local key, cert, chain (...)
# Generate CSR
if [ -f $dir_var/$file_name.cnf ]; then
  openssl req -new -key $dir_var/$file_name.key -out $dir_var/$file_name.csr -config
else
  read -p "Input common name (CN), default if omitted:" cn
  if [ ! $cn ]; then cn="${USER}.demo.com"; fi
  openssl req -new -key $dir_var/$file_name.key -out $dir_var/$file_name.csr -subj "/C=SE/ST=
    Stockholm/L=Stockholm/O=ACME/OU=Headquarters/CN=$cn/emailAddress=info@demo.com"
fi
# Sign Certificate for local
openssl x509 -req -in $dir_var/$file_name.csr -days 60 -CA $dir_ca/ca.pem -CAkey $dir_ca/ca.key -
    CAcreateserial -out $dir_var/$file_name.pem
chmod g+rw $dir_var/$file_name.pem
# See Certificate
openssl x509 -in $dir_var/$file_name.pem -noout -text
# Generate new chain
cat $dir_ca/cert-bundle.pem $dir_var/$file_name.pem > $dir_var/cert-bundle_new.pem
chmod g+rw $dir_var/cert-bundle_new.pem
# Set $cert_bundle_path (...) and show verification
openssl verify -CAfile $dir_ca/root.pem $cert_bundle_path
# Generate p12 for client
openssl pkcs12 -export -in $dir_var/$file_name.pem -inkey $dir_var/$file_name.key -CAfile $dir_ca
    /cert-bundle.pem -name "ACME ${file_name} certificate" -out $dir_var/$file_name.p12
# Complete
```

A CRL is generated and a bad certificate is revoked as shown in `CA_CRL.sh`.

```
openssl ca -gencrl -keyfile ca.key -cert ca.pem -config ca.cnf -out ca.crl
openssl ca -config ca.cnf -revoke client_bad.pem
```

Client

Client has to install its key and certificate in a p12 format file to get connect to servers. In original proposal, if a new client with no certificate arrives, one can be requested by `CLIENT_GET_CERT.sh` with the co-orperation of `CA_GEN_CERT.sh` (For better demo, deprecated). If a client wants to access a server, it must finish mutual authentication which can be tested with a TLS handshake given by `CLIENT_TEST_TLS.sh` (Shown below).

```
read -p "Input server's IP" SERVER_IP
openssl s_client -connect $SERVER_IP:4433 -servername server -brief -cert client.pem -key client.
    key -verify_return_error -state -CAfile ca.pem
```

Server

Server has to well configure corresponding directory and path to enable SSL authentication, which is built-in in different application's configuration file. Take *Apache2* as example, it has a set up of `default-ssl.conf` and an add on `ssl-param.conf` if needed. Server and client's mutual authentication can be tested with a TLS handshake given by `SERVER_TEST_TLS.sh` (Shown below), in which the HTTP field can be set as web-server's IP for client trying to fetch.

```
openssl s_server -cert server.pem -key server.key -state -Verify 5 -CAfile root.pem -HTTP
    192.168.6.113 -verify_return_error
```

By fetching CRL from CA can server verify given certificate (For simple demo).

```
cat root.crl root.pem > root_with_crl.pem
openssl verify -extended_crl -verbose -CAfile root_with_crl.pem -crl_check client.pem
```

If a built-method is failed to provide, server has to run an authenticator `SERVER_AUTH` in background to perfrom TLS, which is programmed and generated by C with a more extended working flow (Shown draft below, deprecated).

```
// ...
int main(int argc, char **argv)
{
    get_current_directory(); ctx = create_context();
    configure_context(ctx); sock = create_socket(4433);
    /* Handle connections */
    while(1) {
        struct sockaddr_in addr; unsigned int len = sizeof(addr); SSL *ssl;
        int client = accept(sock, (struct sockaddr*)&addr, &len);
        if (client < 0) { perror("Unable to accept"); exit(EXIT_FAILURE); }
        ssl = SSL_new(ctx); SSL_set_accept_state(ssl); SSL_set_fd(ssl, client);
        SSL_set_verify(ssl, SSL_VERIFY_PEER | SSL_VERIFY_FAIL_IF_NO_PEER_CERT, NULL);
        if (SSL_accept(ssl) <= 0) { ERR_print_errors_fp(stderr); /* Handshake failed. */}
        else { // Handshanke success.
            int res = SSL_get_verify_result(ssl);
            if (!(X509_V_OK == res)) { fprintf(stdout, "Verify Failed\n"); }
            else { fprintf(stdout, "Verify Successful\n");
                /* Authentication Success! Do Next. */}
        }
        SSL_shutdown(ssl); SSL_free(ssl); close(client);
    }
    close(sock); SSL_CTX_free(ctx);
}
```

SNORT

Installation and configuration

Firstly, we used Ubuntu20.10 as the virtual machine and SNORT2.9.20 to deploy IDS server, run the following commands to install dependencies on the virtual machine before installing SNORT.

```
    sudo apt-get install -y build-essential libpcap-dev libpcre3-dev libdumbnet-dev bison flex
    zlib1g-dev liblzma-dev openssl libssl-dev
    sudo apt-get install -y libnghttp2-dev
    wget https://snort.org/downloads/snort/daq-2.0.6.tar.gz
    tar -xvzf daq-2.0.6.tar.gz
    cd daq-2.0.6
    ./configure
    make
    sudo make install
```

Then Install SNORT,

```
    cd ~/snort_src
    wget https://snort.org/downloads/snort/snort-2.9.9.0.tar.gz
    tar -xvzf snort-2.9.9.0.tar.gz
    cd snort-2.9.9.0
    ./configure --enable-sourcefire
    make
    sudo make install
    sudo ldconfig
    sudo ln -s /usr/local/bin/snort /usr/sbin/snort
```

Once the installation is complete, we need to create new files and folder that SNORT needs and modify the permissions of the files and folders. Now we can download some ruleset files form SNORT website or edit our own

local.rules file, and move these rule files to /etc/snort/rules,which is the path that stores SNORT detection rules. Then we can start to edit our SNORT configuration document.

Firstly, we need to configure the snort.conf where the user can specify the subnet of which to analyze traffic. We change the subnet information in this file:

```
    ipvar HOME_NET 192.168.6.0/24
    ipvar EXTERNAL_NET !$HOME_NET
```

Secondly, we need to tell SNORT all the locations of the rule files we created and edit in snort.conf file:

```
    var RULE_PATH /etc/snort/rules
    var SO_RULE_PATH /etc/snort/so_rules
    var PREPROC_RULE_PATH /etc/snort/preproc_rules
    var WHITE_LIST_PATH /etc/snort/rules/iplists
    var BLACK_LIST_PATH /etc/snort/rules/iplists
```

Thirdly, enable rule file in snort.conf, such as:

```
    include $RULE_PATH/local.rule
```

Then we can test and run SNORT.

Test and Run

SNORT can also be tested explicitly with the following command:

```
    sudo snort -T -c /etc/snort/snort.conf -i enp0s3
```

Make sure to correct the network interface that we try to listen to. After Snort successfully validated the configuration, we start to run:

```
    snort -i enp0s3 -c /etc/snort/snort.conf -A fast
```

Then we can see the alert log file in /var/log/snort/ path.

BIND9

We need a virtual machine with ubuntu20.10 to install BIND9:

```
     apt install bind9 dnsutils
```

BIND9 will create a path /etc/bind/ to create and store configuration files.

Then we can configure vim named.conf.options file firstly,

```
options {
    directory "/var/cache/bind";
    listen-on port 53 { any; }; //
    allow-query { any; }; //
    // If there is a firewall between you and nameservers you want
    // to talk to, you may need to fix the firewall to allow multiple
    // ports to talk.  See http://www.kb.cert.org/vuls/id/800113

    // If your ISP provided one or more IP addresses for stable
    // nameservers, you probably want to use them as forwarders.
    // Uncomment the following block, and insert the addresses replacing
    // the all-0's placeholder.
     forwarders {
            223.5.5.5;
            223.6.6.6;
            //114.114.114.114;
```

```
     };
    //========================================================================
    // If BIND logs error messages about the root key being expired,
    // you will need to update your keys.  See https://www.isc.org/bind-keys
    //========================================================================
    dnssec-validation auto;//aut
    auth-nxdomain no;//
    //listen-on-v6 { any; };
    //include "/etc/rndc.key";
};
```

Then configure named.conf.local file to be certained about zones,

```
// Do any local configuration here
//
// Consider adding the 1918 zones here, if they are not used in your organization
// include "/etc/bind/zones.rfc1918";
zone "demo.com" {
        type master;
        file "/etc/bind/db.demo.com";
};

zone "168.192.in-addr.arpa"  {
    type master;
    file "/etc/bind/db.168.192";
};
```

Then we should configure db.demo.com and db.168.192 to achieve Forward parsing and reverse parsing,

```
;
; BIND data file for local loopback interface
;
$TTL    604800
@       IN      SOA     demo.com. root.demo.com. (
                              2         ; Serial
                         604800         ; Refresh
                          86400         ; Retry
                        2419200         ; Expire
                         604800 )       ; Negative Cache TTL
;
@       IN      NS      localhost.
@       IN      A       192.168.6.233
web     IN      A       192.168.6.113
;@      IN      AAAA    ::1
* IN  A  192.168.6.233   ;
```

```
;
; BIND reverse data file for local loopback interface
;
$TTL    604800
@       IN      SOA     demo.com. root.demo.com. (
                              1         ; Serial
                         604800         ; Refresh
                          86400         ; Retry
                        2419200         ; Expire
                         604800 )       ; Negative Cache TTL
;
@       IN      NS      localhost.
1.0.0   IN      PTR     localhost.
233.6   IN      PTR     demo.com.   ;
113.6.168.192.in-addr.arpa.  IN   PTR   web.demo.com.
```

The configuration is initially complete, restart the bind service, then we can go to the hosts in the Stockholm, and modify their resolv.conf file to change the IP address of DNS server. After that, we achieve basic function of DNS server.

References

[1] FreeRADIUS, https://freeradius.org/documentation

[2] OpenVPN, https://openvpn.net

[3] OpenSSL, https://www.openssl.org/

[4] OpenWRT, https://openwrt.org/docs/guide-user/services/vpn/start

[5] Iptables, https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/security_guide/sect-security_guide-iptables

[6] SNORT, https://www.snort.org/

[7] BIND9, https://www.isc.org/bind/