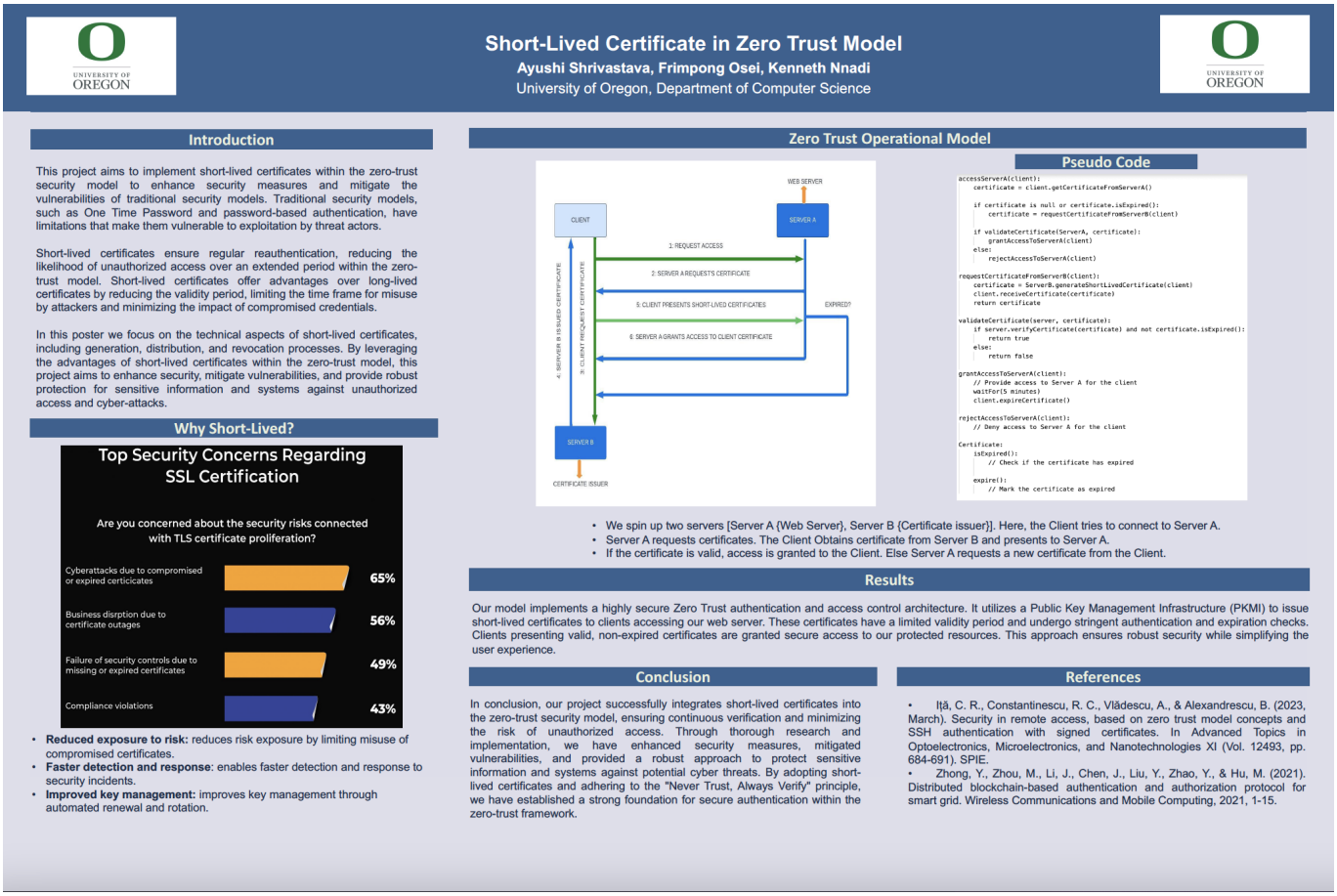
Documentation

Short-lived Certificate Authentication in Zero Trust Model

This documentation provides a step-by-step guide on short-lived authentication using the Zero Trust Model.

This is the image of our poster.



Server B Public Key Infrastructure (PKI)

Creating Directories Needed for our Certificate Authority

To begin, create the necessary directories for the root Certificate Authority (root CA), intermediate/subordinate CA (sub CA), and ServerA.

Terminologies:

- **private**: Used to store the private key
- **certs**: Used to store the certificates
- **newcerts**: Used to store newly issued certificates for clients
- **crl**: Stands for Certificate Revocation List
- **csr**: Stands for Certificate Signing Request

```
mkdir -p ca/{root-ca,sub-ca,serverA}/{private,certs,newcerts,crl,csr}
```

Change the permissions of the private directories that hold the private keys of the root CA, sub CA, and ServerA to 700.

```
chmod -v 700 ca/{root-ca,sub-ca,serverA}/private
```

Create an index file in the root CA and sub CA directories. The index file serves as a database to store all the certificates generated in the PKI.

```
touch ca/{root-ca,sub-ca}/index
```

Generate a random hexadecimal number for the root and sub CA and save it in the serial directory. Each certificate generated is assigned a unique serial number.

```
openssl rand -hex 16 > ca/root-ca/serial
```

```
openssl rand -hex 16 > ca/sub-ca/serial
```

Private Key Generation

Change your current directory to the private directory to generate the private keys.

```
cd ca
```

Creating Root CA Private Key

Generate the private key for the root CA. The **-aes256** flag adds a passphrase to the private key for additional security, making it difficult for unauthorized individuals to use the key without knowing the correct passphrase. The **4096** key size provides stronger encryption.

```
openssl genrsa -aes256 -out root-ca/private/ca.key 4096
```

Creating Sub CA Private Key

Generate the private key for the intermediate/subordinate CA (sub CA). This key will be used for signing the root CA.

```
openssl genrsa -aes256 -out sub-ca/private/sub-ca.key 4096
```

Creating ServerA Private Key

Generate the private key for ServerA. No passphrase is added to the server's private key, and the key size is reduced to prevent overloading the server.

```
openssl genrsa -out serverA/private/serverA.key 2048
```

Creating Configuration Files

Create the configuration files for the root CA (root-ca.conf) and sub CA (sub-ca.conf).

Public Key for our CA (Root CA)

Change to the root CA directory.

```
cd root-ca/
```

Generate the public key for the root CA using the root-ca.conf configuration file. The key is self-signed, valid for 7500 days, and uses the SHA256 hash algorithm. The resulting certificate is saved as "ca.crt" in the certs directory.

```
openssl req -config root-ca.conf -key private/ca.key -new -x509 -days 7500
-sha256 -extensions v3_ca -out certs/ca.crt
```

To view the details of the generated certificate, use the following command:

```
openssl x509 -noout -in certs/ca.crt -text
```

Creating a Certificate Signing Request for Sub CA

Change to the sub CA directory.

```
cd ../sub-ca/
```

Generate a certificate signing request (CSR) for the

sub CA using the sub-ca.conf configuration file.

```
openssl req -config sub-ca.conf -key private/sub-ca.key -sha256 -out
csr/sub-ca.csr
```

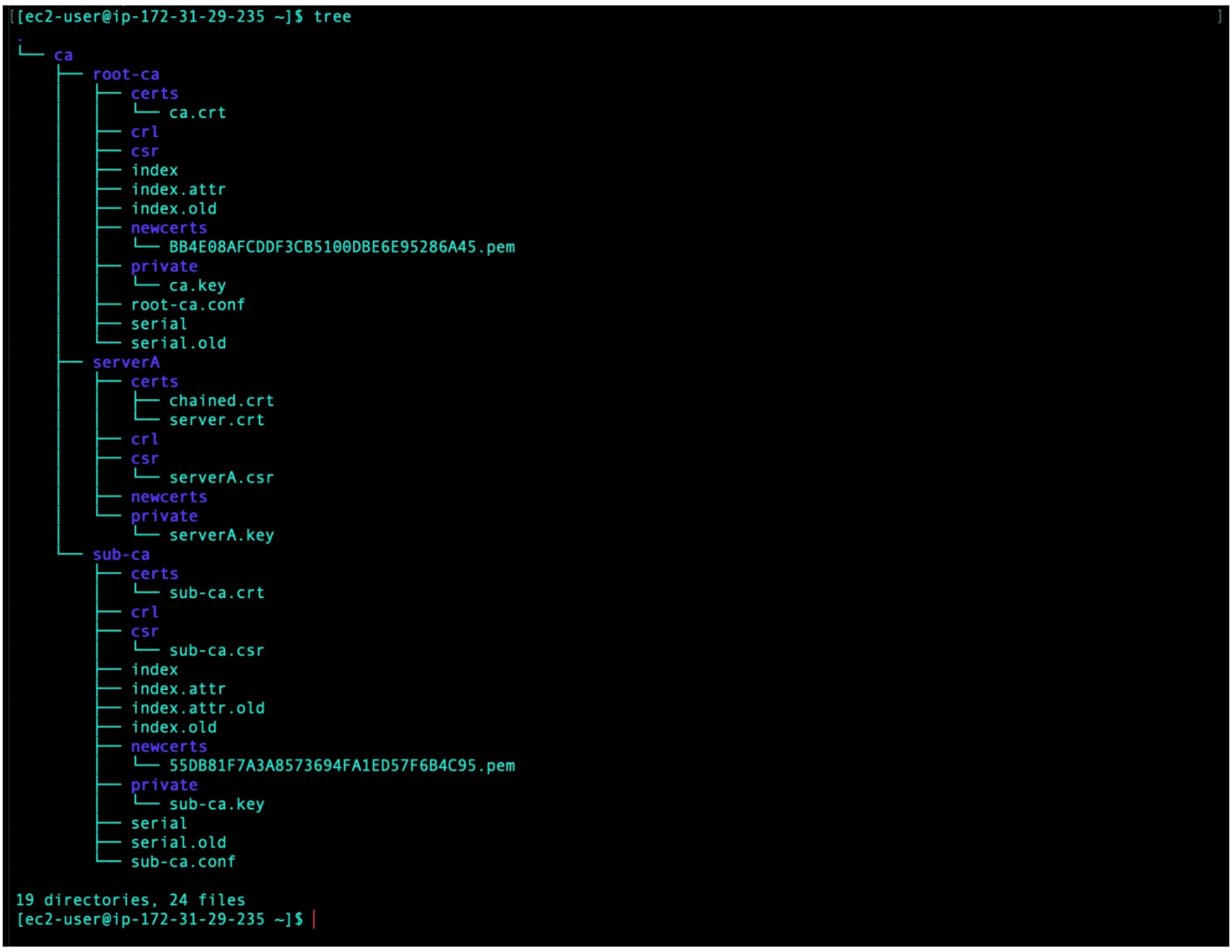
To sign the sub-CA certificate, return to the root-ca directory.

```
openssl ca -config root-ca.conf -extensions v3_intermediate_ca -days 3650
-notext -in ../sub-ca/csr/sub-ca.csr -out ../sub-ca/certs/sub-ca.crt
```

To view the details of the sub-CA certificate, use the following command:

```
openssl x509 -noout -in ../sub-ca/certs/sub-ca.crt -text
```

Show the structure of the certificates created using the **tree** command.



Generating ServerA Certificate

Note:

- Configuration files are not required for creating the server certificate.
- Keep the certificate signing request (CSR) file local to the server for reuse when requesting a new certificate.

Generating Certificate Signing Request (CSR)

Change to the serverA directory.

```
cd ../serverA
```

Generate a certificate signing request (CSR) for ServerA.

```
openssl req -key private/serverA.key -new -sha256 -out csr/serverA.csr
```

You will be prompted to provide the required information. Ensure that the common name matches the fully qualified domain name (FQDN) used by clients to access the server. The server certificate request is now generated. The next step is to sign the server CSR using the signing authority, which is the intermediate CA (sub CA).

```
cd ../sub-ca
openssl ca -config sub-ca.conf -extensions server_certs -days 0 -seconds
300 -notext -in ../serverA/csr/serverA.csr -out
../serverA/certs/serverA.crt
```

To view the database of all certificates, use the following command:

```
cat index
```

To view the newly created server certificate, use the following command:

```
ls newcerts
```

To test the certificate on ServerA, which hosts the Nginx web server, concatenate the server certificate (serverA.crt) and the sub-ca.crt into a file named chained.crt. This file will be used in the ssl_certificate section.

```
cat serverA.crt ../../sub-ca/certs/sub-ca.crt > chained.crt
```

Testing the Certificate

The hostname on the certificate must match the configured certificate.

```
echo "127.0.0.2 www.example.com" >> /etc/hosts
ping www.example.com
```

Different Testing Methods

Testing with OpenSSL

Open two terminal windows. In the **first window**, start the server.

```
openssl s_server -accept 443 -www -key private/server.key -cert
certs/server.crt -CAfile ../sub-ca/certs/sub-ca.crt
```

In the **second window**, run the following command to test the server using cURL:

```
curl https://example.com
```

Since the operating system does not trust the root CA, the connection will fail.

To trust the root CA, copy the `ca.crt` file to the appropriate directory. The exact directory location depends on the operating system. In this example, the directory is `/etc/pki/trust/anchors/`.

```
cp ca/root-ca/certs/ca.crt /etc/pki/trust/anchors/
```

Update the certificate database:

```
sudo update-ca-trust
```

Rerun the cURL command to test the connection:

```
curl https://example.com
```

Nginx Setup

To compress our configuration in the Nginx configuration file, we mapped Nginx to extract the `certificate issuer`, `state`, `canonical name`, and `email address` of each client presenting their certificate. We then compare this information with the certificate issuer, which is our Certificate Authority (CA), and reject any client that presents a certificate not issued by our CA.

To transfer the required keys for nginx configuration, we need to configure SSH in serverB and copy the public key of serverB to the authorized files.

```
ssh-keygen -t rsa -b 4096
```

```
ssh-copy-id @serverAipaddress
```

```
scp path/to/required_keys
serverAusername@serverAipaddress:path/to/destination-folder
```

We also securely transferred the `ca.crt`, `serverA.crt`, `serverA.key` and `chained.crt` (concatenation of the `sub-ca.crt` and `serverA.crt`) using Scp (Secure Copy Protocol).

Edit the Nginx configuration file:

```
nano /etc/nginx/nginx.conf
```

Scroll down to the `ssl` section and uncomment the following lines:

```
server_name www.example.com;
ssl_certificate /root/ca/server/certs/chained.crt;
ssl_certificate_key /root/ca/server/private/server.key;
```

The nginx configuration file will look like this:

```
# For more information on configuration, see:
#   * Official English Documentation: http://nginx.org/en/docs/
#   * Official Russian Documentation: http://nginx.org/ru/docs/

user nginx;
worker_processes auto;
error_log /var/log/nginx/error.log notice;
pid /run/nginx.pid;

# Load dynamic modules. See /usr/share/doc/nginx/README.dynamic.
include /usr/share/nginx/modules/*.conf;

events {
    worker_connections 1024;
}
```

```
http {
    log_format  main  '$remote_addr - $remote_user [$time_local]
"$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;

    sendfile            on;
    tcp_nopush          on;
    keepalive_timeout   65;
    types_hash_max_size 4096;

    include             /etc/nginx/mime.types;
    default_type        application/octet-stream;

    # Load modular configuration files from the /etc/nginx/conf.d
directory.
    # See http://nginx.org/en/docs/ngx_core_module.html#include
    # for more information.
    include /etc/nginx/conf.d/*.conf;

    server {
    listen       80;
        listen       [::]:80;
        server_name  _;
        root         /usr/share/nginx/html;

        # Load configuration files for the default server block.
        include /etc/nginx/default.d/*.conf;

        error_page 404 /404.html;
        location = /404.html {
        }

    error_page 500 502 503 504 /50x.html;
        location = /50x.html {
        }
    }
  # Define a map to extract the client certificate information
    map $ssl_client_s_dn $client_certificate_info {
        default "";
        ~.CN=(?<CN>[^,]+).*C=(?<C>[^,]+).*ST=(?<ST>[^,]+).*O=(?<O>
[^,]+).*emailAddress=(?<emailAddress>[^,]+). $C $ST $O $CN $emailAddress;
    }

# Settings for a TLS enabled server.
#
    server {
    listen       443 ssl http2;
        listen       [::]:443 ssl http2;
        server_name  www.example.com;
        root         /srv/htdocs;
        index        index.html;
```

```
        ssl_certificate /home/ec2-user/ca/serverA/certs/chained.crt;
        ssl_certificate_key /home/ec2-user/ca/serverA/private/serverA.key;
        ssl_session_cache shared:SSL:1m;
        ssl_session_timeout  10m;
        ssl_ciphers PROFILE=SYSTEM;
        ssl_prefer_server_ciphers on;
#
   # Specify the CA certificate used to verify the client certificate
        ssl_client_certificate /ca/root-ca/cert/ca.crt;
        ssl_verify_client on;

        # Add access control based on the client certificate issuer
        if ($ssl_client_verify != SUCCESS) {
            return 403;
        }

        # Compare the client certificate issuer with the CA issuer
        if ($ssl_client_issuer != "C=US, ST=Oregon, O=ZeroTrust Ltd,
CN=SubCA, emailAddress=kennethnnadi14@gmail.com") {
            return 403;
        }
#        # Load configuration files for the default server block.
        include /etc/nginx/default.d/*.conf;
#
        error_page 404 /404.html;
        location = /404.html {
        location = /404.html {
        }

        error_page 500 502 503 504 /50x.html;
        location = /50x.html {
        }
    }

}
```

Save the changes and restart the Nginx web server:

```
sudo systemctl restart nginx
```

or

```
sudo nginx -s reload
```

```
echo "
<!DOCTYPE html>
```

```
<html>
<head>
    <title>Short Lived Configuration Test</title>
</head>
<body>
    <h1>We are Testing the short-lived certificate</h1>
    <h2>Hello there, your Short Lived Configuration is working
effectively.</h2>
</body>
</html>

" > /srv/www/htdocs/index.html
```

```
curl https://www.example.com
```

Testing

During our testing phase, we utilized servers without a graphical user interface and relied on the curl client to test our models. The server's domain name we used for testing purposes is example.com, a public and insecure domain accessible to anyone. Although we could have used our own domain, time constraints prevented us from doing so.

To test our web server configured on serverA, we used the curl client. Inside the serverA configuration file, we included the necessary certificate and implemented an access control code block within the server block. This code block verifies the credentials of the presented certificate. If the curl client is used to access the insecure website, the HTML document of example.com will be displayed.

```
ubuntu@ip-172-31-16-178:~/certs$ curl https://www.example.com
<!doctype html>
<html>
<head>
    <title>Example Domain</title>

    <meta charset="utf-8" />
    <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <style type="text/css">
    body {
        background-color: #f0f0f2;
        margin: 0;
        padding: 0;
        font-family: -apple-system, system-ui, BlinkMacSystemFont, "Segoe UI", "Open Sans", "Helvetica Neue", Helvetica, Arial
, sans-serif;

    }
    div {
        width: 600px;
        margin: 5em auto;
        padding: 2em;
        background-color: #fdfdff;
        border-radius: 0.5em;
        box-shadow: 2px 3px 7px 2px rgba(0,0,0,0.02);
    }
    a:link, a:visited {
        color: #38488f;
        text-decoration: none;
    }
    @media (max-width: 700px) {
        div {
            margin: 0 auto;
            width: auto;
        }
    }
    </style>
</head>

<body>
<div>
    <h1>Example Domain</h1>
    <p>This domain is for use in illustrative examples in documents. You may use this
    domain in literature without prior coordination or asking for permission.</p>
    <p><a href="https://www.iana.org/domains/example">More information...</a></p>
</div>
</body>
</html>
ubuntu@ip-172-31-16-178:~/certs$
```

However, we configured the nginx web server to point to our customized index.html file and display its contents. When using the curl command with the URL "https://www.example.com", instead of the original or default example.com page, we obtained the index of our HTML file. This outcome confirms that our certificate is functioning correctly.

```
[ec2-user@ip-172-31-29-235 certs]$ sudo nano /srv/htdocs/index.html
[ec2-user@ip-172-31-29-235 certs]$ curl https://www.example.com
<!DOCTYPE html>
<html>
<head>
    <title>Short Lived Configuration Test</title>
</head>
<body>
    <h1>We are Testing the short-lived certificate</h1>
    <h2>Hello there, your Short Lived Configuration is working effectively.</h2>
</body>
</html>


[ec2-user@ip-172-31-29-235 certs]$
```

After the specified certificate validity period of 15 minutes, we attempted the same access using the curl client on the client's machine. At this point, the default `example.com` is displayed, indicating that the certificate has expired.

Furthermore, the server also verifies if the certificate originates from the root CA and if it matches the client's name. If these conditions are not met, the example.com page will display the default HTML file.