## Contents

Literature Review .......... 2
Asymmetric Encryption .......... 3
Symmetric Encryption .......... 3
Which Is Stronger? .......... 4
How Ssl Uses Both Asymmetric And Symmetric Encryption? .......... 4
Paper Objectives And Scope .......... 5
Grading Ssl .......... 5
Certificate Inspection .......... 6
Protocol Support .......... 6
Cipher Suite, Key Exchange, And Bits Strength .......... 7
Known Ssl Bug Detection .......... 8
Scoring Table .......... 9
Ssl Grading Script .......... 9
Machine Learning .......... 20
Data Analytics .......... 21
Bank .......... 22
Government .......... 26
Business (Other) .......... 31
Conclusion .......... 32
Items To Discuss .......... 33
Reference .......... 33

**Security Tools Lab 1**

# Project 3 – SSL Grader

Ong Tiong Heng

1005668 MSSD

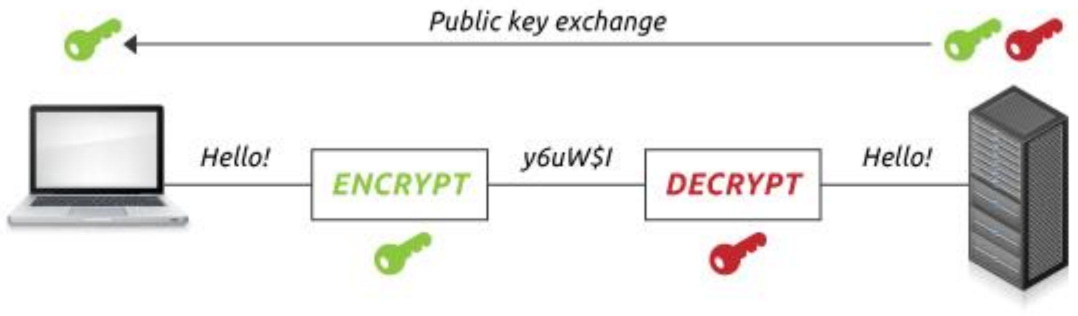
Singapore University of Technology and Design (SUTD)

## Literature Review

SSL (Secure Socket Layer) is a protocol developed by Netscape that enables a web browser and a web server to communicate securely; it allows the web browser to authenticate the web server. The SSL protocol requires the web server to have a digital certificate installed on it for an SSL connection to be made. SSL works by using a public key to encrypt data that's transferred over the SSL connection. Both Netscape Navigator and Internet Explorer support SSL, and many websites use the protocol to obtain confidential user information, such as credit card numbers. By convention, URLs that require an SSL connection start with https: instead of http. Secure Sockets Layer, or SSL, technology is one method for protecting Internet users from malicious groups or software. It is currently used in Web browsers, instant messaging programs, email clients and other software. You most likely use, or have used, several applications that depend on SSL to secure communications between you and another computer that you are connecting to, such as an email server. The Netscape Company created the SSL protocol in 1994. This technology allowed secure transmissions between computer applications on a remote server and the client's computer; however, it was never released to the public domain. SSL "provides privacy and @ Netscape continued to develop SSL technology for several years. The first version of SSL was never released because of problems regarding protection of credit card transactions on the Web. In 1994, Netscape created SSLv2, which made it possible to keep credit card numbers confidential and also authenticate the Web server with the use of encryption and digital certificates. In 1995, Netscape strengthened the cryptographic algorithms and resolved many of the security problems in SSLv2 with the release of SSLv3. SSLv3 now support more security algorithms than SSLv2 [1][4]. In 1996, the TLS protocol evolved from the Secure Sockets Layer (SSL) protocol which was developed by Netscape in the mid-1990s. In 1999, the Internet Engineering Task Force (IETF) standardized a new protocol called TLS, which is an updated version of SSL. In fact, TLS is so similar to SSL that TLS 1.0 uses the SSL protocol version number 3.1. This may seem confusing at first, but makes sense since TLS is just a minor update to SSL 3.0. Subsequent versions of TLS have followed this pattern. Since TLS is an evolution of the SSL protocol, people still use the terms TLS and SSL somewhat interchangeably [5].

## Asymmetric Encryption

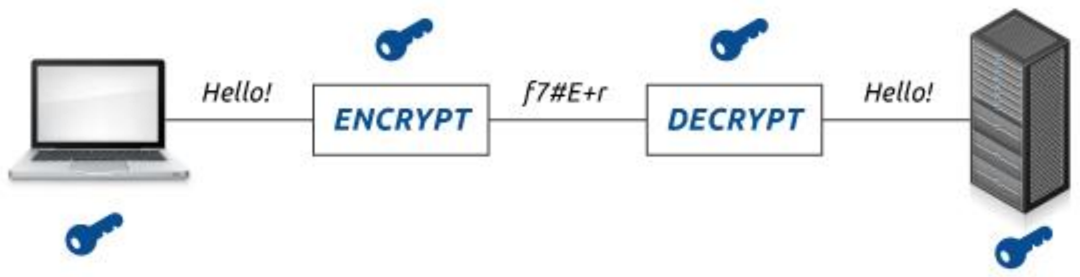
Asymmetric encryption (or public-key cryptography) uses a separate key for encryption and decryption. Anyone can use the encryption key (public key) to encrypt a message. However, decryption keys (private keys) are secret. This way only the intended receiver can decrypt the message. The most common asymmetric encryption algorithm is RSA; however, we will discuss algorithms later in this article.



Asymmetric keys are typically 1024 or 2048 bits. However, keys smaller than 2048 bits are no longer considered safe to use. 2048-bit keys have enough unique encryption codes that we won't write out the number here (it's 617 digits). Though larger keys can be created, the increased computational burden is so significant that keys larger than 2048 bits are rarely used. To put it into perspective, it would take an average computer more than 14 billion years to crack a 2048-bit certificate [6].

## Symmetric Encryption

Symmetric encryption (or pre-shared key encryption) uses a single key to both encrypt and decrypt data. Both the sender and the receiver need the same key to communicate.



Symmetric key sizes are typically 128 or 256 bits—the larger the key size, the harder the key is to crack. For example, a 128-bit key has 340,282,366,920,938,463,463,374,607,431,768,211,456 encryption code possibilities.

As you can imagine, a ‘brute force’ attack (in which an attacker tries every possible key until they find the right one) would take quite a bit of time to break a 128-bit key.

Whether a 128-bit or 256-bit key is used depends on the encryption capabilities of both the server and the client software. SSL Certificates do not dictate what key size is used [6].

## Which Is Stronger?

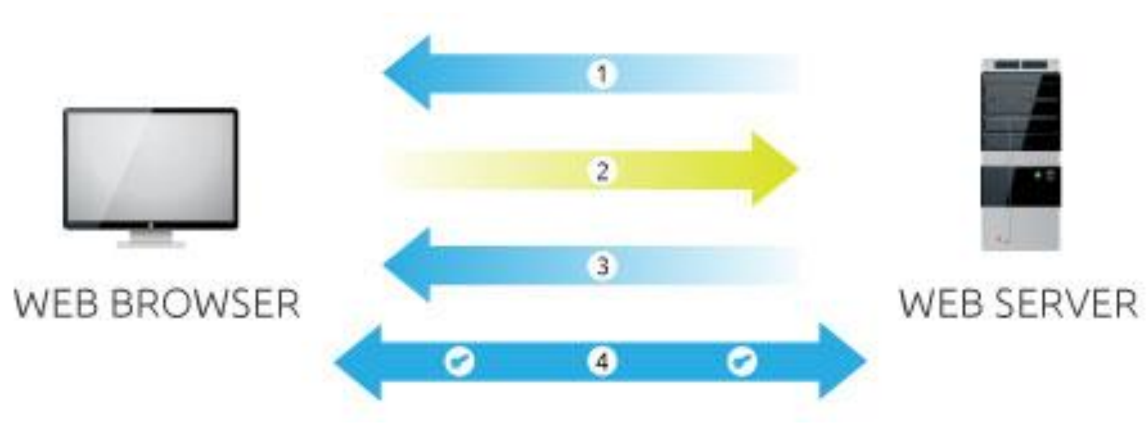
Since asymmetric keys are bigger than symmetric keys, data that is encrypted asymmetrically is tougher to crack than data that is symmetrically encrypted. However, this does not mean that asymmetric keys are better. Rather than being compared by their size, these keys should compare by the following properties: computational burden and ease of distribution.

Symmetric keys are smaller than asymmetric, so they require less computational burden. However, symmetric keys also have a major disadvantage—especially if you use them for securing data transfers. Because the same key is used for symmetric encryption and decryption, both you and the recipient need the key. If you can walk over and tell your recipient the key, this isn’t a huge deal. However, if you have to send the key to a user halfway around the world (a more likely scenario) you need to worry about data security.

Asymmetric encryption doesn’t have this problem. As long as you keep your private key secret, no one can decrypt your messages. You can distribute the corresponding public key without worrying who gets it. Anyone who has the public key can encrypt data, but only the person with the private key can decrypt it [6].

## How Ssl Uses Both Asymmetric And Symmetric Encryption?

Public Key Infrastructure (PKI) is the set of hardware, software, people, policies, and procedures that are needed to create, manage, distribute, use, store, and revoke digital certificates. PKI is also what binds keys with user identities by means of a Certificate Authority (CA). PKI uses a hybrid cryptosystem and benefits from using both types of encryption. For example, in SSL communications, the server’s SSL Certificate contains an asymmetric public and private key pair. The session key that the server and the browser create during the SSL Handshake is symmetric. This is explained further in the diagram below.



1. **Server** sends a copy of its asymmetric public key.
2. **Browser** creates a symmetric session key and encrypts it with the server's asymmetric public key. Then sends it to the server.
3. **Server** decrypts the encrypted session key using its asymmetric private key to get the symmetric session key.
4. **Server** and **Browser** now encrypt and decrypt all transmitted data with the symmetric session key. This allows for a secure channel because only the browser and the server know the symmetric session key, and the session key is only used for that session. If the browser was to connect to the same server the next day, a new session key would be created.

## Paper Objectives And Scope

The Secure Sockets Layer (SSL) protocol is a standard for encrypted network communication. Even with its ease of use and its widespread usage, it has many vulnerabilities not only through sophisticated attacks of technologies used but also through something as simple as security misconfigurations. This paper will discuss an implementation of an SSL configuration checker [8] to assess the security of any given website without the need for the user to be an SSL expert.

## Grading Ssl

This paper aims to establish a straightforward assessment methodology, allowing anyone to assess SSL server configuration confidently without any SSL knowledge [7]. In this section, there are 3 scoring sections for scores accumulation up to 100% and 1 non-scoring section that gives discount to accumulated scores upon detection of known SSL bug(s).

These automated steps consist of:

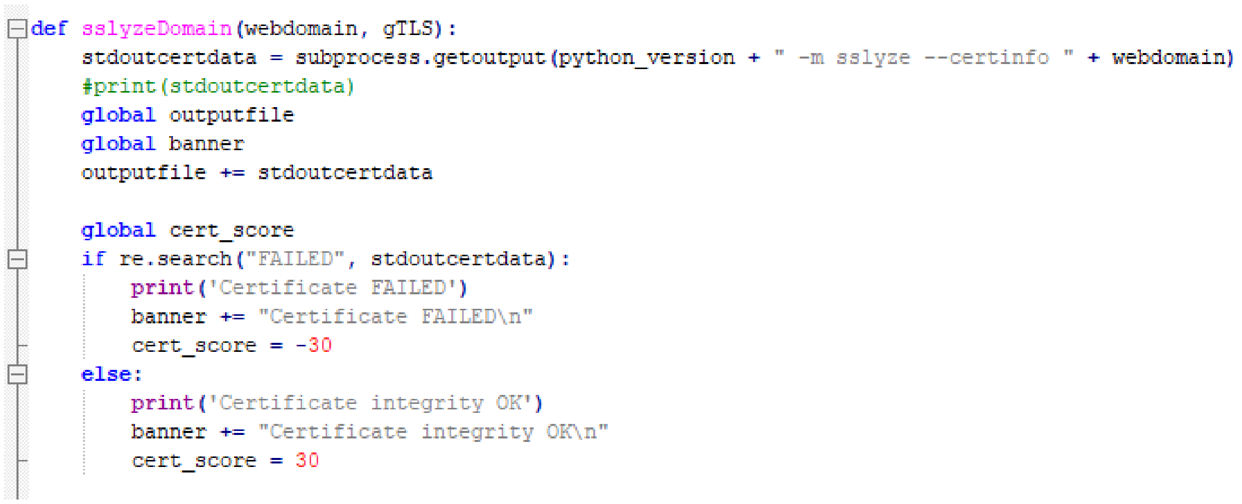
1. Certification Inspection
2. Protocol Support
3. Cipher Suite, Key Exchange, and Bits Strength Evaluation
4. Known bug scanning

## Certificate Inspection

This section the script will check the following and return a score amounting to 30% of 100%. Pass will get 30%, negative 30% for a failed.

1. Trust Check for certificate to be signed by a well-known CA
2. Expiry Check to ensure certificate is not invalid
3. Invalid Check to ensure certificate is not revoked
4. Secure Check to ensure certificate is not using an insecure signature like MD5

Parsing the output into a function, FAILED keyword is used to detects SSL Certificate issue.



```python
def sslyzeDomain(webdomain, gTLS):
    stdoutcertdata = subprocess.getoutput(python_version + " -m sslyze --certinfo " + webdomain)
    #print(stdoutcertdata)
    global outputfile
    global banner
    outputfile += stdoutcertdata

    global cert_score
    if re.search("FAILED", stdoutcertdata):
        print('Certificate FAILED')
        banner += "Certificate FAILED\n"
        cert_score = -30
    else:
        print('Certificate integrity OK')
        banner += "Certificate integrity OK\n"
        cert_score = 30
```

## Protocol Support

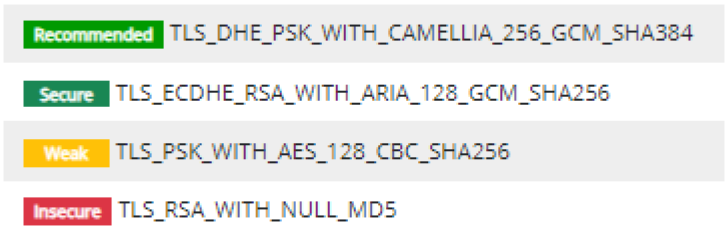
This section consists of the following scoring amounting to 30% of 100%. TLS1.3 and TLS1.2 each get 20% and 10% respectively. Older protocols such as TLS1.0 or SSLv3 will get negative 15% discount.

1. TLS 1.3 should have the highest score while TLS 1.2 the lowest
2. TLS 1.0, TLS 1.1 or SSL3 all get negative 15% score.

## Cipher Suite, Key Exchange, And Bits Strength

This section will match the pulled element tags with https://ciphersuite.info for corresponding TLS cipher suite status, key exchange type and associated bits strength amounting to 40% of 100%.

1. Secure status will get 40% (0% discount)
2. Recommended status will get 30% (10% discount)
3. Weak status will get 20% (20% discount)
4. Insecure status will get 110% discount (110% discount)



Recommended TLS_DHE_PSK_WITH_CAMELLIA_256_GCM_SHA384

Secure TLS_ECDHE_RSA_WITH_ARIA_128_GCM_SHA256

Weak TLS_PSK_WITH_AES_128_CBC_SHA256

Insecure TLS_RSA_WITH_NULL_MD5

What does insecure, weak, secure and recommended mean?

Insecure
These ciphers are very old and shouldn't be used under any circumstances. Their protection can be broken with minimal effort nowadays.

Weak
These ciphers are old and should be disabled if you are setting up a new server for example. Make sure to only enable them if you have a special use case where support for older operating systems, browsers or applications is required.

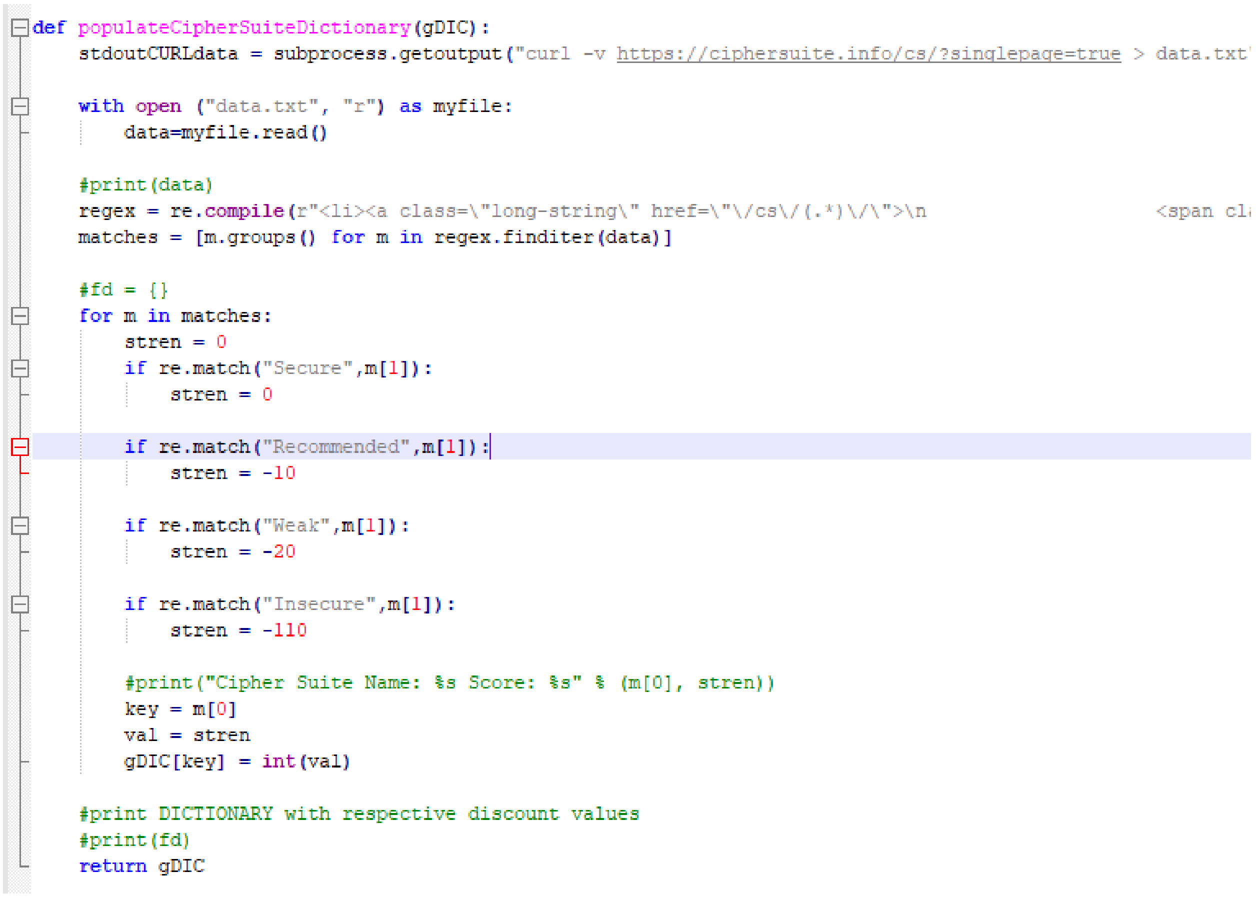
Secure
Secure ciphers are considered state-of-the-art and if you want to secure your web server you should certainly choose from this set. Only very old operating systems, browsers or applications are unable to handle them.

Recommended
All 'recommended' ciphers are 'secure' ciphers by definition. Recommended means that these ciphers also support PFS (Perfect Forward Secrecy) and should be your first choice if you want the highest level of security. However, you might run into some compatibility issues with older clients that do not support PFS ciphers.

Where does the data come from?
The list of ciphers comes from the IANA, the OpenSSL and GnuTLS library. We update them regularly to ensure our service has the most complete list and newest informations at any time.

Who determines the rating?
Most of the ratings are taken from official notes of the IETF or whitepapers by security researchers.

How element tags are used for grading?
Each tag from ciphersuite.info is mapped to a score through a function seen below.



```
def populateCipherSuiteDictionary(gDIC):
    stdoutCURLdata = subprocess.getoutput("curl -v https://ciphersuite.info/cs/?singlepage=true > data.txt

    with open ("data.txt", "r") as myfile:
        data=myfile.read()

    #print(data)
    regex = re.compile(r"<li><a class=\"long-string\" href=\"\/cs\/(.*)\/\">\n                <span cl
    matches = [m.groups() for m in regex.finditer(data)]

    #fd = {}
    for m in matches:
        stren = 0
        if re.match("Secure",m[1]):
            stren = 0

        if re.match("Recommended",m[1]):
            stren = -10

        if re.match("Weak",m[1]):
            stren = -20

        if re.match("Insecure",m[1]):
            stren = -110

        #print("Cipher Suite Name: %s Score: %s" % (m[0], stren))
        key = m[0]
        val = stren
        gDIC[key] = int(val)

    #print DICTIONARY with respective discount values
    #print(fd)
    return gDIC
```

## Known Ssl Bug Detection

This section will check again the following amounting to immediate failure with a negative 199% of 100%.

1. Robot, Test a server for the ROBOT vulnerability.
2. Openssl_ccs, Test a server for the OpenSSL CCS Injection vulnerability (CVE-2014-0224).
3. Heartbleed, Test a server for the OpenSSL Heartbleed vulnerability.
4. Fallback, Test a server for the TLS_FALLBACK_SCSV mechanism to prevent downgrade attacks.
5. Reneg, Test a server for for insecure TLS renegotiation and client-initiated renegotiation.

## Scoring Table

| | |
|---|---|
| Excellent = 100% | |
| A | >= 80% |
| B | >= 70% |
| C | >= 50% |
| F | < 50% |

Total score made up of part 1(30%) + part 2(30%) + part 3(60%) - part 4 (discount percentage -199% if found)

## Ssl Grading Script

SSLyze is a fast and powerful SSL/TLS scanning library. It allows you to analyze the SSL/TLS configuration of a server by connecting to it, in order to detect various issues (bad certificate, weak cipher suites, Heartbleed, ROBOT, TLS 1.3 support, etc.).SSLyze can either be used as a command line tool or as a Python library.

SSLyze provides core functionalities to SSLazy SSL Grader where it process the outputs from SSLyze and match them against ciphersuite.info for grading.

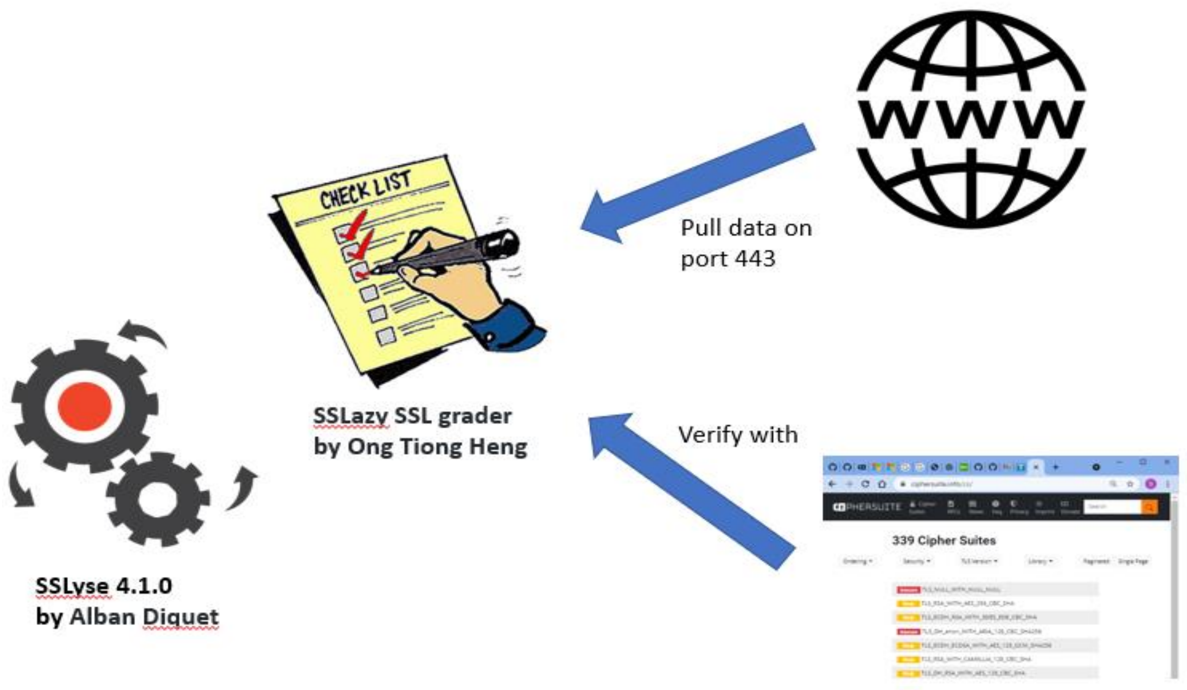

CHECK LIST
Pull data on port 443
SSLazy SSL grader by Ong Tiong Heng
Verify with
339 Cipher Suites
SSLyse 4.1.0 by Alban Diquet

Figure 1 – An Overview of the SSLazy Grading process

The python SSLazy Grader. Below shows declaratives, libraries and variables

```
import sys

import subprocess

import re

import os


python_version="/usr/bin/python3.9"

cert_score = 0

TLS_support_score = 0

discount_score = 0

ciphersuite_keyex_strength_score = 0

total_score = 0

gDIC = {}

aDIC = {}

gTLS = []

webdomain = ""

outputfile = ""

banner = ""
```

Setting up passing of argument on Main function

```
def main():

    # print command line arguments

    for arg in sys.argv[1:]:

        print(arg)


if __name__ == "__main__":

    main()
```

Fetching CipherSuite.info data feeds and mapped them to grading scores.

```
def populateCipherSuiteDictionary(gDIC):

    stdoutCURLdata = subprocess.getoutput("curl -v https://ciphersuite.info/cs/?singlepage=true > data.txt")



    with open ("data.txt", "r") as myfile:

        data=myfile.read()



    #print(data)

    regex = re.compile(r"<li><a class=\"long-string\" href=\"\/cs\/(.*)\/\">\n                        <span class=\"badge bg-fixed-width bg-.*\">(.*)</span>", re.MULTILINE)

    matches = [m.groups() for m in regex.finditer(data)]



    #fd = {}

    for m in matches:

        stren = 0

        if re.match("Secure",m[1]):

            stren = 0



        if re.match("Recommended",m[1]):

            stren = -10



        if re.match("Weak",m[1]):

            stren = -20



        if re.match("Insecure",m[1]):

            stren = -110



        #print("Cipher Suite Name: %s Score: %s" % (m[0], stren))

        key = m[0]

        val = stren

        gDIC[key] = int(val)
    #print DICTIONARY with respective discount values
    #print(fd)
    return gDIC
```

## Matching TLS lines on SSLyze Outputs.

```
def findTLS(stdout):
    matched = re.findall(r"\s+(TLS\S+)\s+\d+", stdout, re.MULTILINE)
    #print(matched)
    return(matched)
```

## Analyze SSLyze Outputs for success, failure and known bug.

```
def sslyzeDomain(webdomain, gTLS):
    stdoutcertdata = subprocess.getoutput(python_version + " -m sslyze --certinfo " + webdomain)
    #print(stdoutcertdata)
    global outputfile
    global banner
    outputfile += stdoutcertdata

    global cert_score
    if re.search("FAILED", stdoutcertdata):
        print('Certificate FAILED')
        banner += "Certificate FAILED\n"
        cert_score = -30
    else:
        print('Certificate integrity OK')
        banner += "Certificate integrity OK\n"
        cert_score = 30

    stdoutTLS13data = subprocess.getoutput(python_version + " -m sslyze --tlsv1_3 " + webdomain)
    #print(stdoutTLS13data)
    outputfile += stdoutTLS13data
    gTLS += (findTLS(stdoutTLS13data))
    global TLS_support_score
    if re.search("The server accepted the", stdoutTLS13data):
        print('TLS13 OK')
        banner += "TLS13 OK\n"
        TLS_support_score += 20
    else:
        print('TLS13 Not Found')
        banner += "TLS13 Not Found\n"

    stdoutTLS12data = subprocess.getoutput(python_version + " -m sslyze --tlsv1_2 " + webdomain)
    #print(stdoutTLS12data)
    outputfile += stdoutTLS12data
    gTLS += (findTLS(stdoutTLS12data))

    if re.search("The server accepted the", stdoutTLS12data):
        print('TLS12 OK')
        banner += "TLS12 OK\n"
        TLS_support_score += 10
    else:
        print('TLS12 Not Found')
        banner += "TLS12 Not Found\n"

    stdoutTLSOtherdata = subprocess.getoutput(python_version + " -m sslyze --sslv3 --sslv2 --tlsv1 --tlsv1_1 " +
webdomain)
    #print(stdoutTLSOtherdata)
    outputfile += stdoutTLSOtherdata
    gTLS += (findTLS(stdoutTLSOtherdata))

    if re.search("The server accepted the", stdoutTLSOtherdata):
        print('Older SSL/TLS Found')
        banner += "Older SSL/TLS Found\n"
        TLS_support_score += -15
    else:

        print('Older SSL/TLS Not Found')

        banner += "Older SSL/TLS Not Found\n"



    stdoutDISdata = subprocess.getoutput(python_version + " -m sslyze --robot --openssl_ccs --heartbleed --fallback --
reneg  " + webdomain)

    #print(stdoutDISdata)

    outputfile += stdoutDISdata



    global discount_score

    if re.search("VULNERABLE", stdoutDISdata):

        print('Vulnerable Issue Found')

        banner += "Vulnerable Issue Found\n"

        discount_score += -199

    else:

        print('Vulnerable Issue Not Found')

        banner += "Vulnerable Issue Not Found\n"
```

## Building a dynamic Dictionary for matching purposes

```
def append_value(dict_obj, key, value):

    # Check if key exist in dict or not

    if key in dict_obj:

        # Key exist in dict.

        # Check if type of value of key is list or not

        if not isinstance(dict_obj[key], list):

            # If type is not list then make it list

            dict_obj[key] = [dict_obj[key]]

        # Append the value in list

        dict_obj[key].append(value)

    else:

        # As key is not in dict,

        # so, add key-value pair

        dict_obj[key] = value
```

## Grading function

```
def determine_grade(scores):
    if scores == 100:
        return 'Excellent'
    elif scores >= 80 and scores <= 99:
        return 'A'
    elif scores >= 70 and scores <= 79:
        return 'B'
    elif scores >= 50 and scores <= 69:
        return 'C'
    else:
        return 'F'
```

## Firing up SSL checker, matching the outputs with dynamic dictionary and save to CSV.

```
print("<<< Start SSL Grading <<<")
banner = "<<< Start SSL Grading <<<\n" + banner
#print(populateCipherSuiteDictionary(gDIC))
populateCipherSuiteDictionary(gDIC)
sslyzeDomain(str(sys.argv[1]), gTLS)
#print(gTLS)

for k, v in gDIC.items():
    if k in gTLS:
        #print(k)
        append_value(aDIC, k, v)

# PRINT out the temp DICTIONARY used to store the pull TLS scoring
#print(aDIC)

# Using min() + list comprehension + values()
# Finding min value keys in dictionary
```

```
#ciphersuite_keyex_strength_score = min(aDIC.values())

#print(ciphersuite_keyex_strength_score)


try:

    ciphersuite_keyex_strength_score = min(aDIC.values())

except:

    ciphersuite_keyex_strength_score = -99


ciphersuite_keyex_strength_score = 40 + ciphersuite_keyex_strength_score


print("<<< End SSL Grading <<<")

banner += "<<< End SSL Grading <<<\n"

total_score = cert_score + TLS_support_score + discount_score + ciphersuite_keyex_strength_score

banner1 = ">>>Start Computing score>>>" \

            + "\nCertificate score is:" +  str(cert_score) \

            + "\nTLS support score is:" + str(TLS_support_score) \

            + "\nTLS discount score is:" + str(discount_score) \

            + "\nCipher Suite score is:" + str(ciphersuite_keyex_strength_score) \

            + "\n>>>Total SSL grade for " + str(sys.argv[1]) + " is " + str(total_score) + "/100. Grade is " + determine_grade(int(total_score)) + ". >>>\n"


print(banner1)


try:

    os.makedirs('results')

except OSError as e:

    pass


new_path = "results/" + str(sys.argv[1]) + ".txt"

new_handler = open(new_path,'w')

new_handler.write(banner + banner1 + outputfile)

new_handler.close()
```

```
print(banner + banner1 + outputfile)

                                   )

new_path = "SSLayzeSummary.txt"

new_handler = open(new_path,'a')

new_handler.write(str(sys.argv[1]) + "," \

                             + determine_grade(int(total_score)) + "," \

                             + str(total_score) + "," \

                             + str(cert_score) + "," \

                + str(TLS_support_score) + "," \

                + str(discount_score) + "\n" \

                )


new_handler.close()
```

Output Brief:

Brief Sample 1 – Graded C

```
kali@kali:~/Downloads/sslyze$ python3 sslgrader.py www.google.com
www.google.com
<<< Start SSL Grading <<<
Certificate integrity OK
TLS13 OK
TLS12 OK
Older SSL/TLS Found
Vulnerable Issue Not Found
<<< End SSL Grading <<<
>>>Start Computing score>>>
Certificate score is: 30
TLS support score is: 15
TLS discount score is: 0
Cipher Suite score is: 20
>>>Total SSL grade for  www.google.com  is  65 /100. Grade is  C . >>>
```

Brief Sample 2 – Graded A

```
kali@kali:~/Downloads/sslyze$ python3 sslgrader.py www.ocbc.com
www.ocbc.com
<<< Start SSL Grading <<<
Certificate integrity OK
TLS13 OK
TLS12 OK
Older SSL/TLS Not Found
Vulnerable Issue Not Found
```

```
<<< End SSL Grading <<<
>>>Start Computing score>>>
Certificate score is: 30
TLS support score is: 30
TLS discount score is: 0
Cipher Suite score is: 30
>>>Total SSL grade for  www.ocbc.com  is  90 /100. Grade is  A . >>>
```

Brief Sample 3 – Graded F

```
kali@kali:~/Downloads/sslyze$ python3 sslgrader.py www.bankofchina.com
www.bankofchina.com
<<< Start SSL Grading <<<
Certificate integrity OK
TLS13 Not Found
TLS12 OK
Older SSL/TLS Found
Vulnerable Issue Found
<<< End SSL Grading <<<
>>>Start Computing score>>>
Certificate score is:30
TLS support score is:-5
TLS discount score is:-199
Cipher Suite score is:20
>>>Total SSL grade for www.bankofchina.com is -154/100. Grade is F. >>>
```

Output Detail:

Detail Sample 1 – www.bankofchina.com

```
kali@kali:~/Downloads/sslyze$ cat results/www.bankofchina.com.txt
<<< Start SSL Grading <<<
Certificate integrity OK
TLS13 Not Found
TLS12 OK
Older SSL/TLS Found
Vulnerable Issue Found
<<< End SSL Grading <<<

>>>Start Computing score>>>
Certificate score is:30
TLS support score is:-5
TLS discount score is:-199
Cipher Suite score is:20
>>>Total SSL grade for www.bankofchina.com is -154/100. Grade is F. >>>

CHECKING HOST(S) AVAILABILITY

www.bankofchina.com:443                    => 123.124.191.45

SCAN RESULTS FOR WWW.BANKOFCHINA.COM:443 - 123.124.191.45

* Certificates Information:
   Hostname sent for SNI:            www.bankofchina.com
   Number of certificates detected:   1

  Certificate #0 ( _RSAPublicKey )
   SHA1 Fingerprint:                 24e635c8dbb7783ba9232285eaa58ec4895ee6c1
   Common Name:                      www.bankofchina.com
   Issuer:                           Secure Site Pro Extended Validation CA G2
   Serial Number:                    20984255454740395246010106896176242556
   Not Before:                       2019-11-26
   Not After:                        2021-11-26
   Public Key Algorithm:             _RSAPublicKey
   Signature Algorithm:              sha256
   Key Size:                         2048
   Exponent:                         65537
   DNS Subject Alternative Names:    ['www.bankofchina.com']

  Certificate #0 - Trust
   Hostname Validation:              OK - Certificate matches server hostname
   Android CA Store (9.0.0_r9):      OK - Certificate is trusted
   Apple CA Store (iOS 14, iPadOS 14, macOS 11, watchOS 7, and tvOS 14):OK - Certificate is trusted
   Java CA Store (jdk-13.0.2):       OK - Certificate is trusted
   Mozilla CA Store (2021-01-24):    OK - Certificate is trusted, Extended Validation
   Windows CA Store (2021-02-08):    OK - Certificate is trusted
   Symantec 2018 Deprecation:        OK - Not a Symantec-issued certificate
   Received Chain:                   www.bankofchina.com --> Secure Site Pro Extended Validation CA G2
   Verified Chain:                   www.bankofchina.com --> Secure Site Pro Extended Validation CA G2 -->
DigiCert High Assurance EV Root CA
   Received Chain Contains Anchor:   OK - Anchor certificate not sent
   Received Chain Order:             OK - Order is valid
   Verified Chain contains SHA1:     OK - No SHA1-signed certificate in the verified certificate chain

  Certificate #0 - Extensions
   OCSP Must-Staple:                 NOT SUPPORTED - Extension not found
   Certificate Transparency:         OK - 3 SCTs included

  Certificate #0 - OCSP Stapling
                                     NOT SUPPORTED - Server did not send back an OCSP response

  * TLS 1.3 Cipher Suites:
   Attempted to connect using 5 cipher suites; the server rejected all cipher suites.

  * TLS 1.2 Cipher Suites:
   Attempted to connect using 156 cipher suites.

  The server accepted the following 4 cipher suites:
    TLS_RSA_WITH_AES_256_CBC_SHA                      256
    TLS_RSA_WITH_AES_128_CBC_SHA                      128
    TLS_RSA_WITH_3DES_EDE_CBC_SHA                     168
    TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256             128       ECDH: prime256v1 (256 bits)

   The group of cipher suites supported by the server has the following properties:
   Forward Secrecy                   OK - Supported
   Legacy RC4 Algorithm              OK - Not Supported

   * TLS 1.1 Cipher Suites:
   Attempted to connect using 80 cipher suites.
```

```
The server accepted the following 3 cipher suites:
  TLS_RSA_WITH_AES_256_CBC_SHA                    256
  TLS_RSA_WITH_AES_128_CBC_SHA                    128
  TLS_RSA_WITH_3DES_EDE_CBC_SHA                     168

The group of cipher suites supported by the server has the following properties:
 Forward Secrecy                   INSECURE - Not Supported
 Legacy RC4 Algorithm               OK - Not Supported

* TLS 1.0 Cipher Suites:
Attempted to connect using 80 cipher suites.

The server accepted the following 3 cipher suites:
  TLS_RSA_WITH_AES_256_CBC_SHA                    256
  TLS_RSA_WITH_AES_128_CBC_SHA                    128
  TLS_RSA_WITH_3DES_EDE_CBC_SHA                     168

The group of cipher suites supported by the server has the following properties:
  Forward Secrecy                   INSECURE - Not Supported
  Legacy RC4 Algorithm               OK - Not Supported

* SSL 3.0 Cipher Suites:
Attempted to connect using 80 cipher suites; the server rejected all cipher suites.

* SSL 2.0 Cipher Suites:
 Attempted to connect using 7 cipher suites; the server rejected all cipher suites.

* ROBOT Attack:
                              VULNERABLE - Strong oracle, a real attack is possible.

* OpenSSL Heartbleed:
                              OK - Not vulnerable to Heartbleed

* Downgrade Attacks:
 TLS_FALLBACK_SCSV:                OK - Supported

* Session Renegotiation:
 Client Renegotiation DoS Attack:   OK - Not vulnerable
 Secure Renegotiation:              OK - Supported

* OpenSSL CCS Injection:
                              OK - Not vulnerable to OpenSSL CCS injection
```

## Machine Learning

A SSLayzeSummary.txt file is created in parent folder to arrange the data into comma separated values (CSV) for pre-Machine Learning analysis such as Tableau or Weka. Manually purging of SSLayzeSummary.txt is needed as the script will only append data to it.

A Sample CSV

```
kali@kali:~/Downloads/sslyze$ cat SSLayzeSummary.txt

www.ocbc.com,A,90,30,30,0,30

www.bankofchina.com,F,-154,30,-5,-199,20

test-dv-rsa.ssl.com,F,5,-30,15,0,20

expired-ecc-ev.ssl.com,F,5,-30,15,0,20
```

CSV Schema

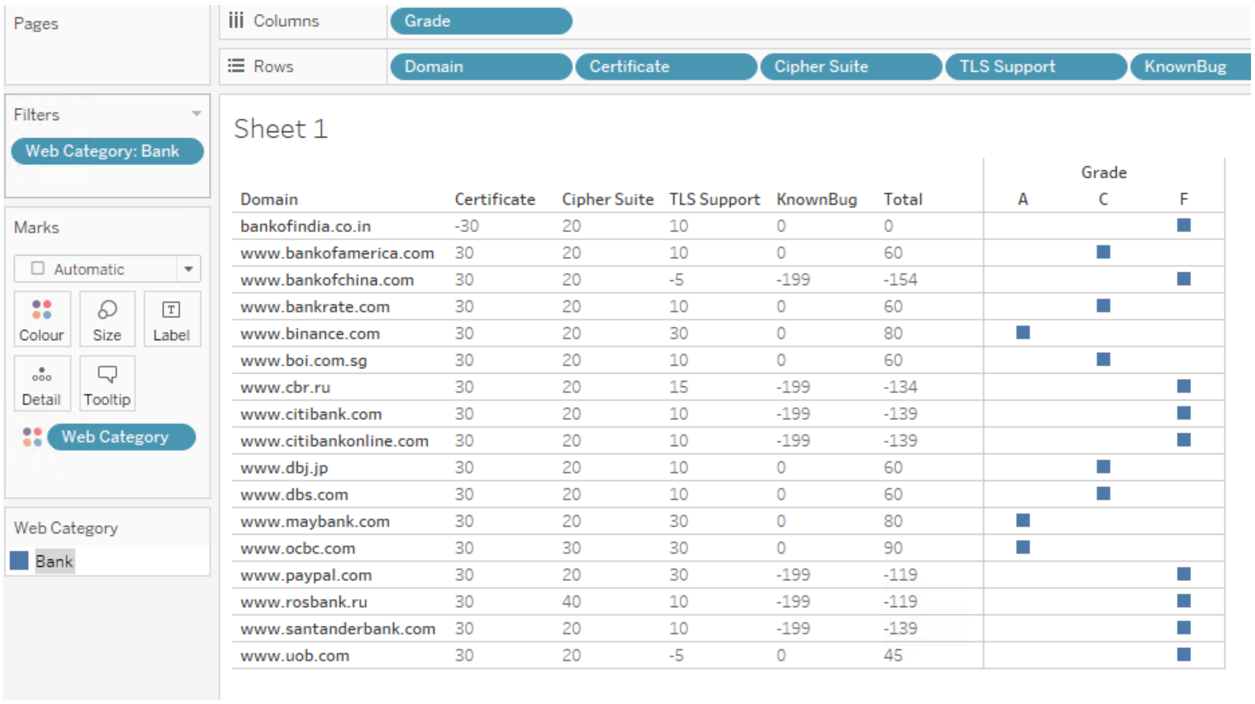
1. Domain
2. Grade
3. Total Score
4. Certificate Score
5. TLS Support Score
6. Discount Score
7. Cipher Suite i.e. Key Exchange, Bits strenght score

## Data Analytics

Data visualization can produce significant levels of insight, provided that said data is extracted in a clear and non-convoluted manner. Such insights can be used to delve deeper into data sets and can extract actionable information for businesses and clients alike. In this paper, a research on the SSL certificates, the questions arise with the respective to different sectors.

## Bank

Finance institutions and Bank are important and crucial infrastructure for each country alike. Ensure gaining insights on the current SSL standing can help to prevent any future cyber-attack or fraud.



Columns: Grade

Rows: Domain, Certificate, Cipher Suite, TLS Support, KnownBug

Filters: Web Category: Bank

Marks: Web Category

Web Category: 🟦 Bank

Sheet 1

| Domain | Certificate | Cipher Suite | TLS Support | KnownBug | Total | Grade A | Grade C | Grade F |
|---|---|---|---|---|---|---|---|---|
| bankofindia.co.in | -30 | 20 | 10 | 0 | 0 | | | 🟦 |
| www.bankofamerica.com | 30 | 20 | 10 | 0 | 60 | | 🟦 | |
| www.bankofchina.com | 30 | 20 | -5 | -199 | -154 | | | 🟦 |
| www.bankrate.com | 30 | 20 | 10 | 0 | 60 | | 🟦 | |
| www.binance.com | 30 | 20 | 30 | 0 | 80 | 🟦 | | |
| www.boi.com.sg | 30 | 20 | 10 | 0 | 60 | | 🟦 | |
| www.cbr.ru | 30 | 20 | 15 | -199 | -134 | | | 🟦 |
| www.citibank.com | 30 | 20 | 10 | -199 | -139 | | | 🟦 |
| www.citibankonline.com | 30 | 20 | 10 | -199 | -139 | | | 🟦 |
| www.dbj.jp | 30 | 20 | 10 | 0 | 60 | | 🟦 | |
| www.dbs.com | 30 | 20 | 10 | 0 | 60 | | 🟦 | |
| www.maybank.com | 30 | 20 | 30 | 0 | 80 | 🟦 | | |
| www.ocbc.com | 30 | 30 | 30 | 0 | 90 | 🟦 | | |
| www.paypal.com | 30 | 20 | 30 | -199 | -119 | | | 🟦 |
| www.rosbank.ru | 30 | 40 | 10 | -199 | -119 | | | 🟦 |
| www.santanderbank.com | 30 | 20 | 10 | -199 | -139 | | | 🟦 |
| www.uob.com | 30 | 20 | -5 | 0 | 45 | | | 🟦 |

- Which bank(s) or financial institution does/do not a valid SSL certificate? And why?

  Bank of India (-30 on Certificate). Reason being CA not trusted.

  See detail on Figure 1

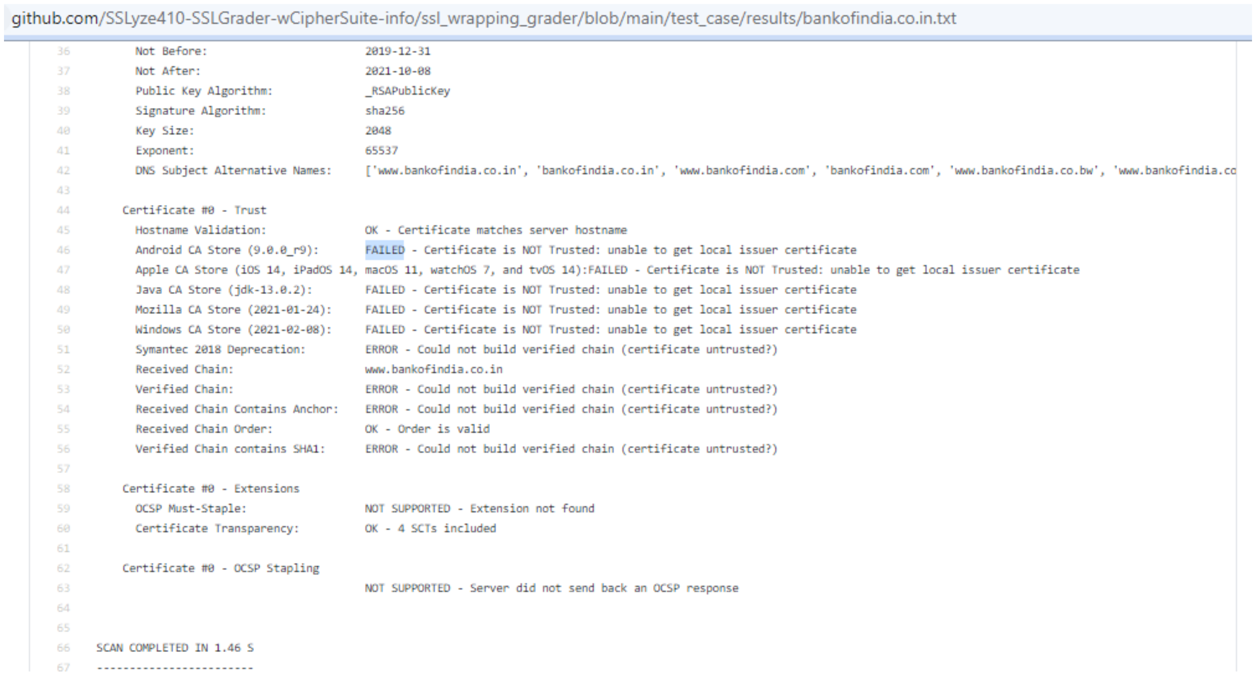
- Which local bank(s) failed the SSL inspection check? And why?

  UOB (-5 on TLS support) Reason being decommissioned TLS1.1 is used.

  See detail on Figure 2

- Which bank(s) has/have known Bug? And Why?

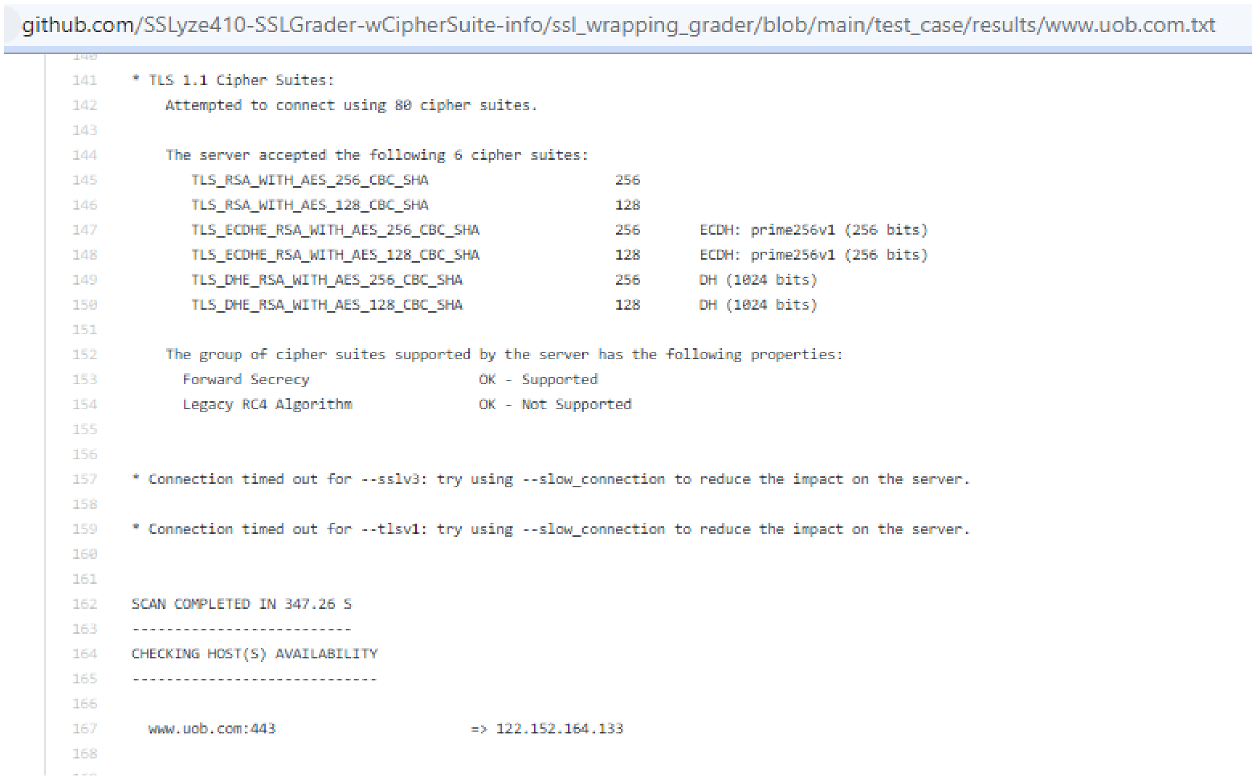
  Bank of China (-199 on KnownBug), Cbr.ru, Citibank, paypal.com, Rosbank.ru and SantanderBank. See samples' details on Figure 3, 4 and 5


github.com/SSLyze410-SSLGrader-wCipherSuite-info/ssl_wrapping_grader/blob/main/test_case/results/bankofindia.co.in.txt

```
       Not Before:                       2019-12-31
       Not After:                        2021-10-08
       Public Key Algorithm:             _RSAPublicKey
       Signature Algorithm:              sha256
       Key Size:                         2048
       Exponent:                         65537
       DNS Subject Alternative Names:    ['www.bankofindia.co.in', 'bankofindia.co.in', 'www.bankofindia.com', 'bankofindia.com', 'www.bankofindia.co.bw', 'www.bankofindia.co

     Certificate #0 - Trust
       Hostname Validation:              OK - Certificate matches server hostname
       Android CA Store (9.0.0_r9):      FAILED - Certificate is NOT Trusted: unable to get local issuer certificate
       Apple CA Store (iOS 14, iPadOS 14, macOS 11, watchOS 7, and tvOS 14):FAILED - Certificate is NOT Trusted: unable to get local issuer certificate
       Java CA Store (jdk-13.0.2):       FAILED - Certificate is NOT Trusted: unable to get local issuer certificate
       Mozilla CA Store (2021-01-24):    FAILED - Certificate is NOT Trusted: unable to get local issuer certificate
       Windows CA Store (2021-02-08):    FAILED - Certificate is NOT Trusted: unable to get local issuer certificate
       Symantec 2018 Deprecation:        ERROR - Could not build verified chain (certificate untrusted?)
       Received Chain:                   www.bankofindia.co.in
       Verified Chain:                   ERROR - Could not build verified chain (certificate untrusted?)
       Received Chain Contains Anchor:   ERROR - Could not build verified chain (certificate untrusted?)
       Received Chain Order:             OK - Order is valid
       Verified Chain contains SHA1:     ERROR - Could not build verified chain (certificate untrusted?)

     Certificate #0 - Extensions
       OCSP Must-Staple:                 NOT SUPPORTED - Extension not found
       Certificate Transparency:         OK - 4 SCTs included

     Certificate #0 - OCSP Stapling
                                         NOT SUPPORTED - Server did not send back an OCSP response


 SCAN COMPLETED IN 1.46 S
 ------------------------
```

Figure 1 – Bank of India Detail Report

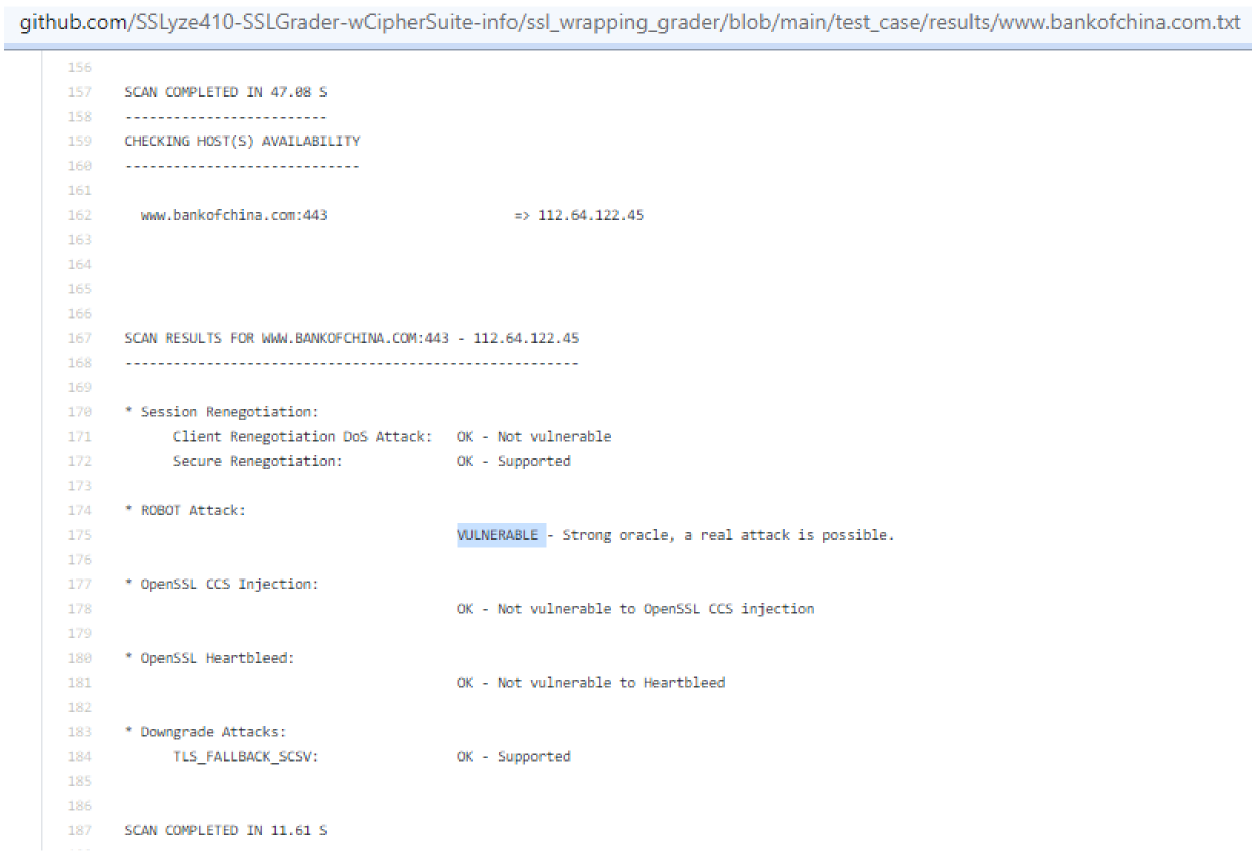
Bank of India uses bad SSL certificate.


github.com/SSLyze410-SSLGrader-wCipherSuite-info/ssl_wrapping_grader/blob/main/test_case/results/www.uob.com.txt

```
 * TLS 1.1 Cipher Suites:
     Attempted to connect using 80 cipher suites.

     The server accepted the following 6 cipher suites:
        TLS_RSA_WITH_AES_256_CBC_SHA                      256
        TLS_RSA_WITH_AES_128_CBC_SHA                      128
        TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA                256       ECDH: prime256v1 (256 bits)
        TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA                128       ECDH: prime256v1 (256 bits)
        TLS_DHE_RSA_WITH_AES_256_CBC_SHA                  256       DH (1024 bits)
        TLS_DHE_RSA_WITH_AES_128_CBC_SHA                  128       DH (1024 bits)

     The group of cipher suites supported by the server has the following properties:
       Forward Secrecy                    OK - Supported
       Legacy RC4 Algorithm               OK - Not Supported


 * Connection timed out for --sslv3: try using --slow_connection to reduce the impact on the server.

 * Connection timed out for --tlsv1: try using --slow_connection to reduce the impact on the server.


 SCAN COMPLETED IN 347.26 S
 --------------------------
 CHECKING HOST(S) AVAILABILITY
 -----------------------------

   www.uob.com:443                      => 122.152.164.133

```

Figure 2 – UOB Detail Report

This bank uses legacy TLS 1.1 protocol.

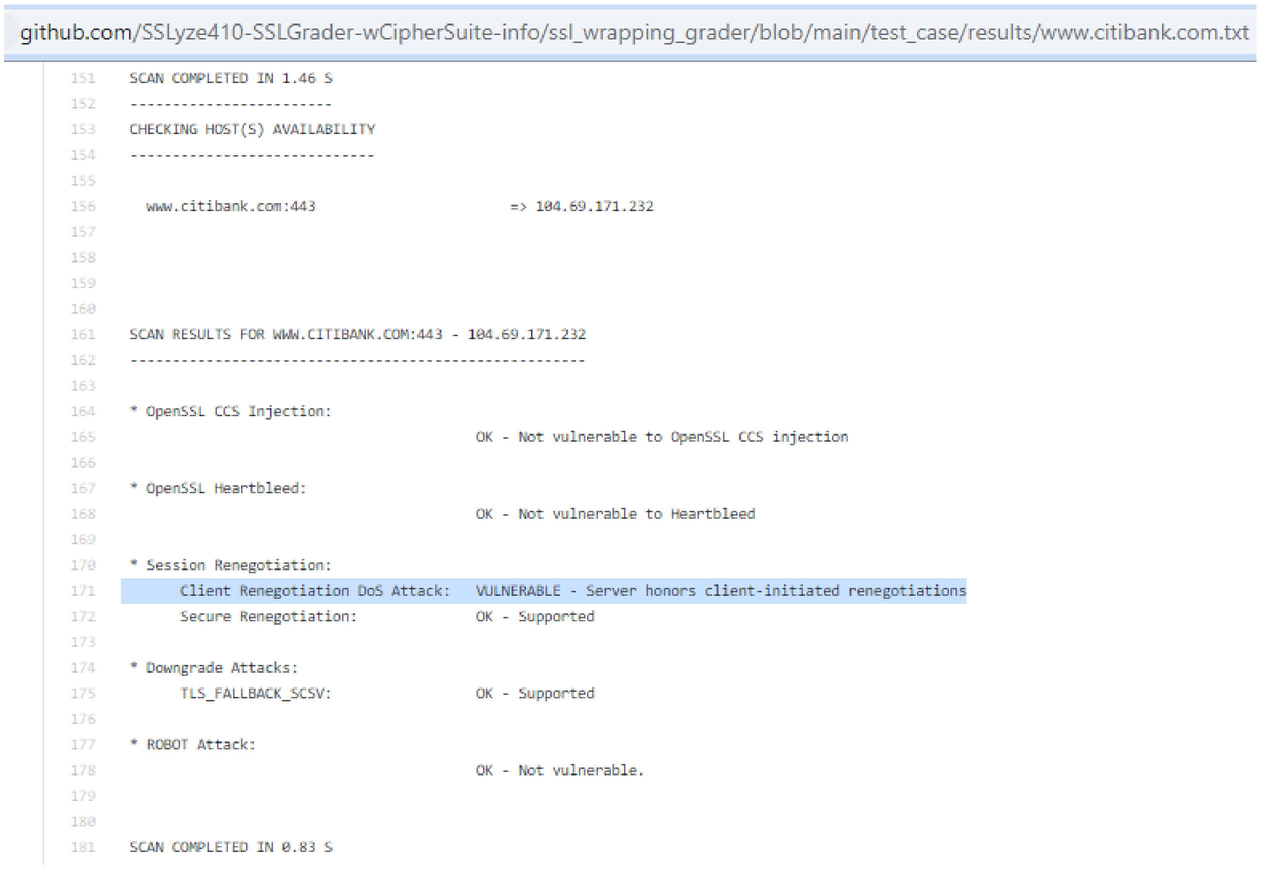


github.com/SSLyze410-SSLGrader-wCipherSuite-info/ssl_wrapping_grader/blob/main/test_case/results/www.bankofchina.com.txt

```

SCAN COMPLETED IN 47.08 S
-------------------------
CHECKING HOST(S) AVAILABILITY
-----------------------------

  www.bankofchina.com:443                       => 112.64.122.45




SCAN RESULTS FOR WWW.BANKOFCHINA.COM:443 - 112.64.122.45
--------------------------------------------------------

* Session Renegotiation:
      Client Renegotiation DoS Attack:   OK - Not vulnerable
      Secure Renegotiation:              OK - Supported

* ROBOT Attack:
                                         VULNERABLE - Strong oracle, a real attack is possible.

* OpenSSL CCS Injection:
                                         OK - Not vulnerable to OpenSSL CCS injection

* OpenSSL Heartbleed:
                                         OK - Not vulnerable to Heartbleed

* Downgrade Attacks:
      TLS_FALLBACK_SCSV:                 OK - Supported


SCAN COMPLETED IN 11.61 S
```

Figure 3 – Bank of China Detail Report

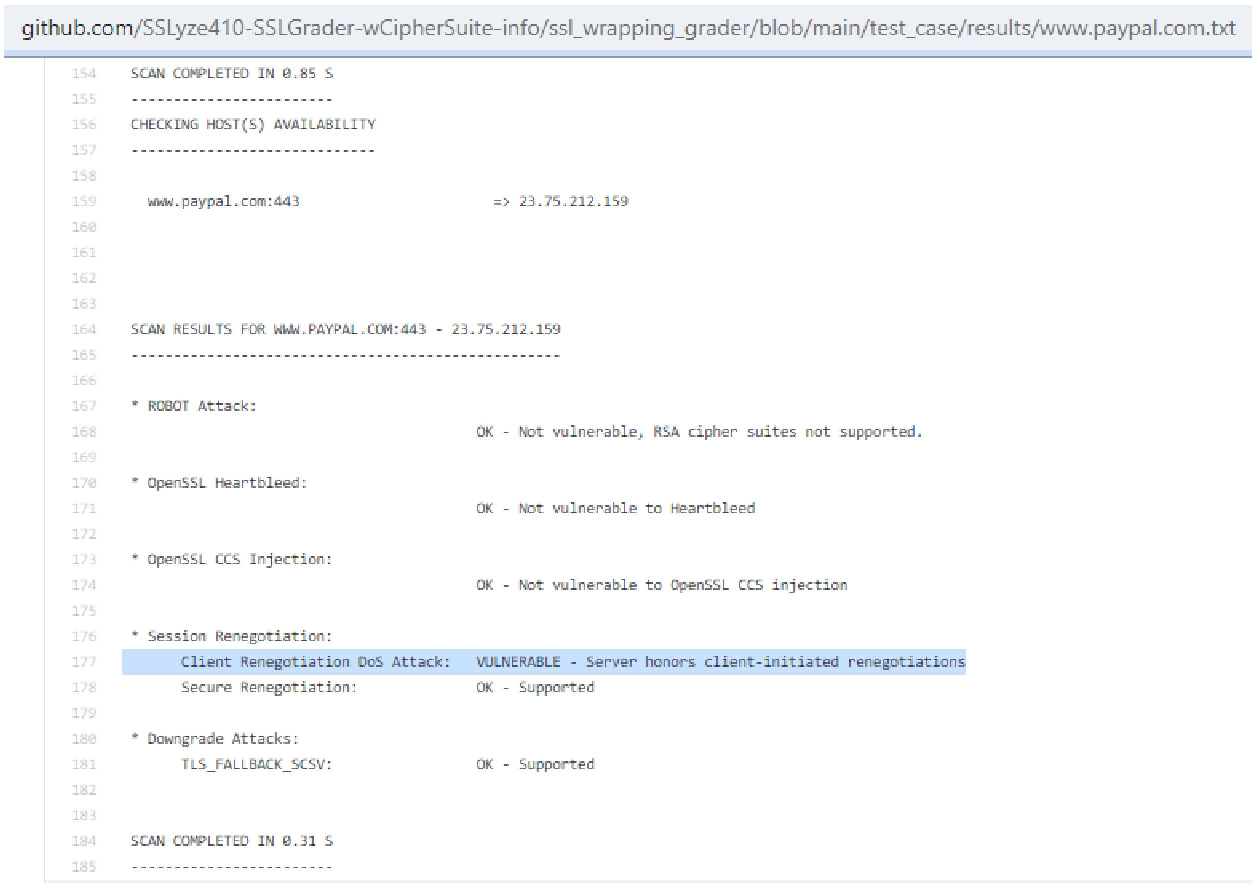
BOC is vulnerable to ROBOT attack.


github.com/SSLyze410-SSLGrader-wCipherSuite-info/ssl_wrapping_grader/blob/main/test_case/results/www.citibank.com.txt

```
SCAN COMPLETED IN 1.46 S
------------------------
CHECKING HOST(S) AVAILABILITY
-----------------------------

  www.citibank.com:443                       => 104.69.171.232




SCAN RESULTS FOR WWW.CITIBANK.COM:443 - 104.69.171.232
------------------------------------------------------

* OpenSSL CCS Injection:
                                          OK - Not vulnerable to OpenSSL CCS injection

* OpenSSL Heartbleed:
                                          OK - Not vulnerable to Heartbleed

* Session Renegotiation:
      Client Renegotiation DoS Attack:    VULNERABLE - Server honors client-initiated renegotiations
      Secure Renegotiation:               OK - Supported

* Downgrade Attacks:
      TLS_FALLBACK_SCSV:                  OK - Supported

* ROBOT Attack:
                                          OK - Not vulnerable.


SCAN COMPLETED IN 0.83 S
```

Figure 4 – Citibank Detail Report

Citibank is vulnerable to client renegotiation DOS attack.



github.com/SSLyze410-SSLGrader-wCipherSuite-info/ssl_wrapping_grader/blob/main/test_case/results/www.paypal.com.txt

```
SCAN COMPLETED IN 0.85 S
------------------------
CHECKING HOST(S) AVAILABILITY
-----------------------------

  www.paypal.com:443                        => 23.75.212.159




SCAN RESULTS FOR WWW.PAYPAL.COM:443 - 23.75.212.159
---------------------------------------------------

* ROBOT Attack:
                                          OK - Not vulnerable, RSA cipher suites not supported.

* OpenSSL Heartbleed:
                                          OK - Not vulnerable to Heartbleed

* OpenSSL CCS Injection:
                                          OK - Not vulnerable to OpenSSL CCS injection

* Session Renegotiation:
      Client Renegotiation DoS Attack:    VULNERABLE - Server honors client-initiated renegotiations
      Secure Renegotiation:               OK - Supported

* Downgrade Attacks:
      TLS_FALLBACK_SCSV:                  OK - Supported


SCAN COMPLETED IN 0.31 S
------------------------
```

Figure 5 – Paypal Detail Report

Paypal is vulnerable to client renegotiation DOS attack.

## Government

*As more businesses conduct their activities online in the light of public health restrictions, more data is generated and exchanged. This increases the risk of data being exposed.*

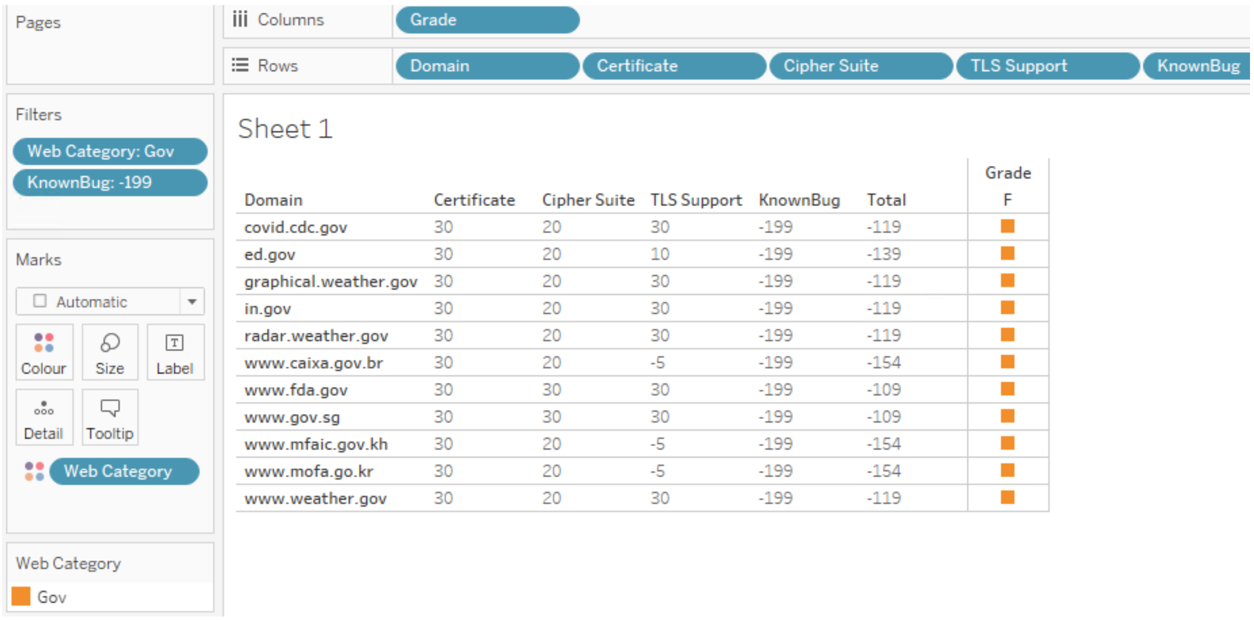
*"Work-from-home arrangements and the use of unsecured home networks may also raise the risk of data incidents," SNDGO said.*

*"These trends highlight the increased data security risks faced by the private and public sectors and... the urgency of implementing the necessary measures to safeguard personal data."*

*On its part, the public sector has committed to roll out 24 key measures by the end of 2023 as part of its $1 billion investment to better safeguard citizens' personal data.*

*These measures were recommended by the Public Sector Data Security Review Committee (PSDSRC), formed in March 2019 after a spate of cyber-security breaches, including Singapore's worst data breach involving 1.5 million SingHealth patients' data in June 2018 [10].*

Hence gaining insights into SSL data is a necessity, no longer a luxury item. Let's explore the CSV data on Tableau further.



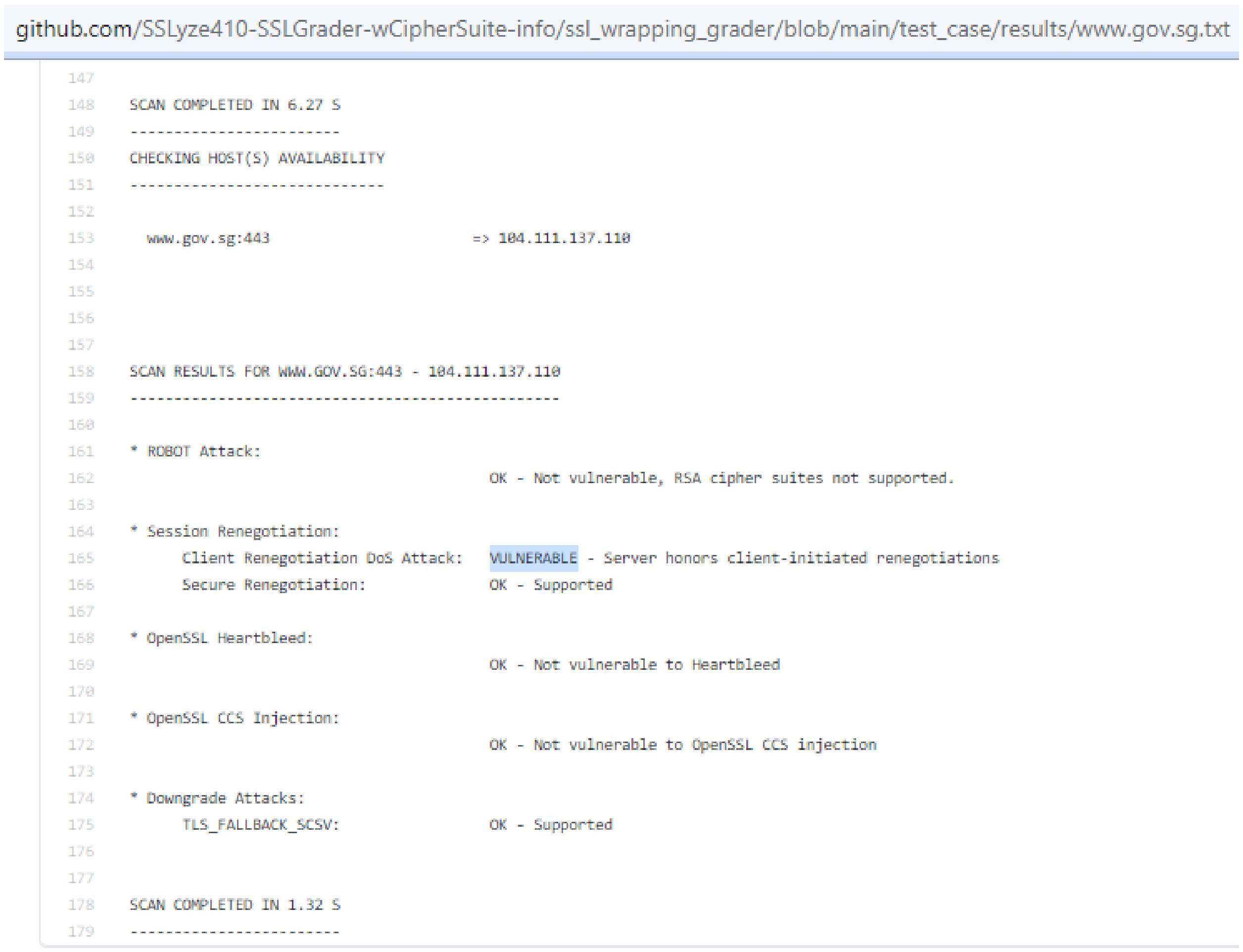
Sheet 1

| Domain | Certificate | Cipher Suite | TLS Support | KnownBug | Total | Grade F |
|---|---|---|---|---|---|---|
| covid.cdc.gov | 30 | 20 | 30 | -199 | -119 | 🟧 |
| ed.gov | 30 | 20 | 10 | -199 | -139 | 🟧 |
| graphical.weather.gov | 30 | 20 | 30 | -199 | -119 | 🟧 |
| in.gov | 30 | 20 | 30 | -199 | -119 | 🟧 |
| radar.weather.gov | 30 | 20 | 30 | -199 | -119 | 🟧 |
| www.caixa.gov.br | 30 | 20 | -5 | -199 | -154 | 🟧 |
| www.fda.gov | 30 | 30 | 30 | -199 | -109 | 🟧 |
| www.gov.sg | 30 | 30 | 30 | -199 | -109 | 🟧 |
| www.mfaic.gov.kh | 30 | 20 | -5 | -199 | -154 | 🟧 |
| www.mofa.go.kr | 30 | 20 | -5 | -199 | -154 | 🟧 |
| www.weather.gov | 30 | 20 | 30 | -199 | -119 | 🟧 |

Web Category
🟧 Gov

- Which SG Gov domain failed? And why?
  www.gov.sg (-199 on KnownBug)
  See detail report on Figure 6

- Which Gov domain(s) has/have legacy TLS running? And why?
  Caixa.gov.br, mfaic.gov.kh and mofa.go.kr
  See samples' details report on Figure 7 and 8



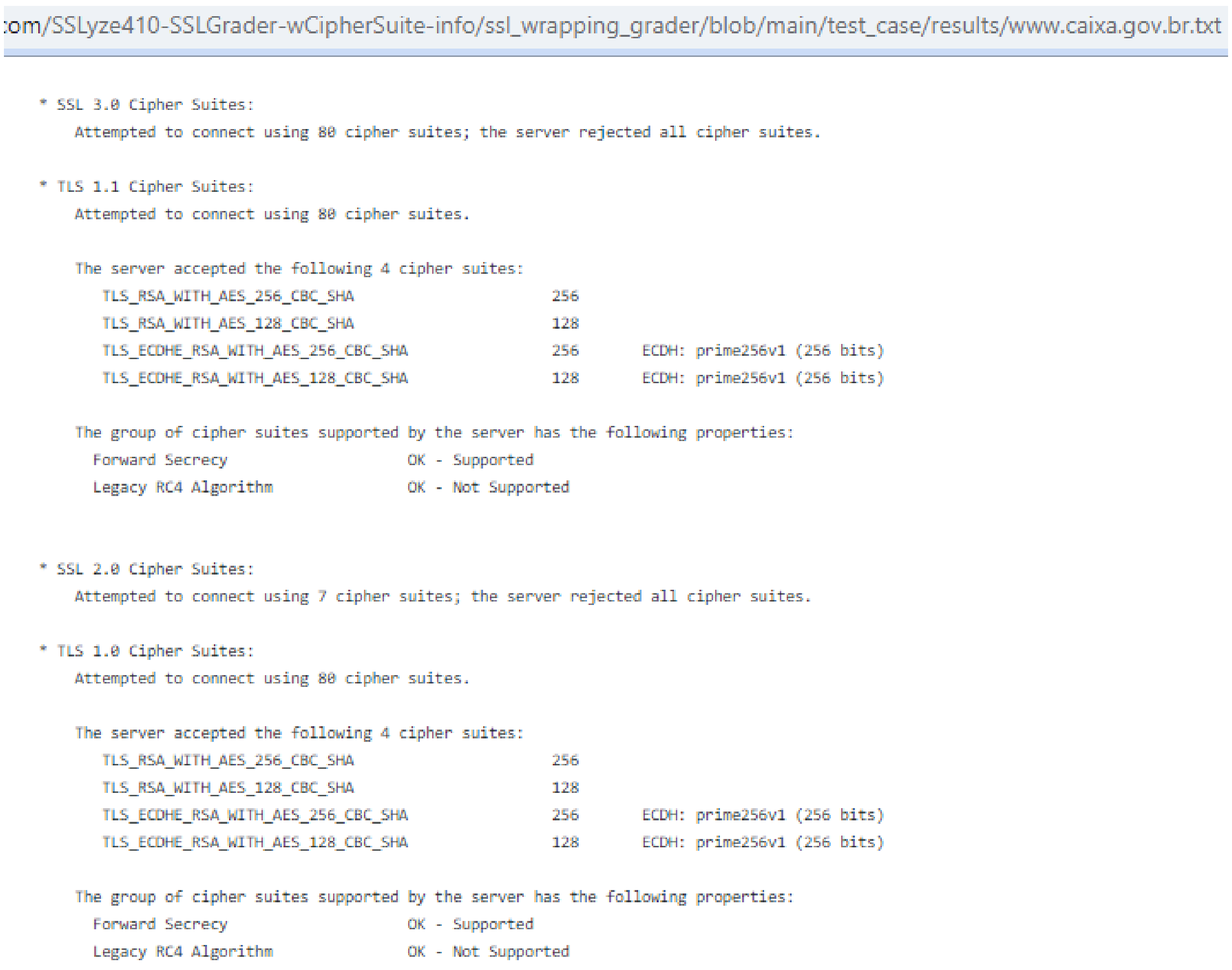
github.com/SSLyze410-SSLGrader-wCipherSuite-info/ssl_wrapping_grader/blob/main/test_case/results/www.gov.sg.txt

```

SCAN COMPLETED IN 6.27 S
------------------------
CHECKING HOST(S) AVAILABILITY
-----------------------------

  www.gov.sg:443                       => 104.111.137.110




SCAN RESULTS FOR WWW.GOV.SG:443 - 104.111.137.110
-------------------------------------------------

* ROBOT Attack:
                                         OK - Not vulnerable, RSA cipher suites not supported.

* Session Renegotiation:
      Client Renegotiation DoS Attack:   VULNERABLE - Server honors client-initiated renegotiations
      Secure Renegotiation:              OK - Supported

* OpenSSL Heartbleed:
                                         OK - Not vulnerable to Heartbleed

* OpenSSL CCS Injection:
                                         OK - Not vulnerable to OpenSSL CCS injection

* Downgrade Attacks:
      TLS_FALLBACK_SCSV:                 OK - Supported


SCAN COMPLETED IN 1.32 S
------------------------
```

Figure 6 – www.gov.sg Detail Report

Here goes our $1 billion investment with a known SSL Bug!

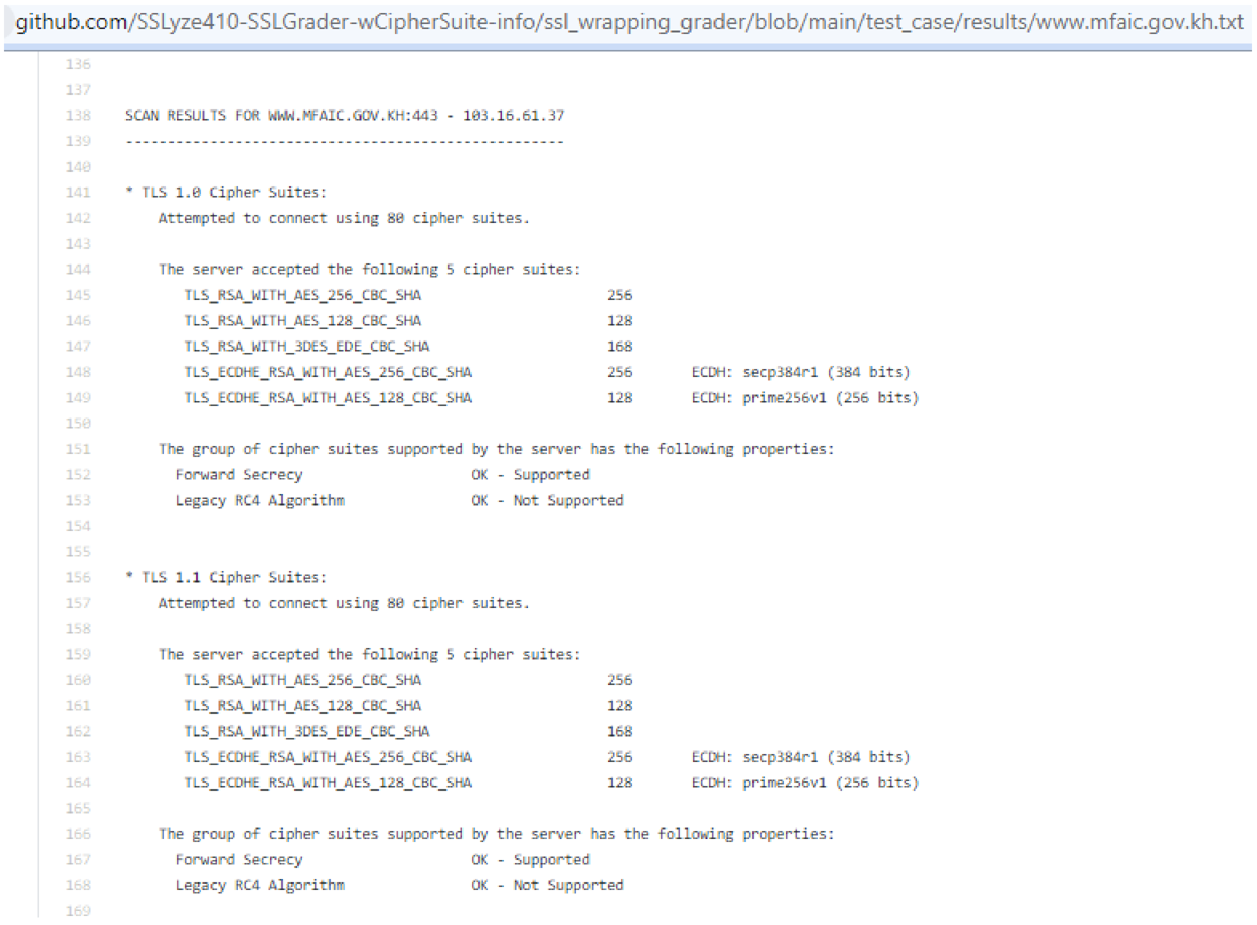
www.gov.sg is vulnerable to client renegotiation DOS attack.



```
:om/SSLyze410-SSLGrader-wCipherSuite-info/ssl_wrapping_grader/blob/main/test_case/results/www.caixa.gov.br.txt

 * SSL 3.0 Cipher Suites:
     Attempted to connect using 80 cipher suites; the server rejected all cipher suites.

 * TLS 1.1 Cipher Suites:
     Attempted to connect using 80 cipher suites.

     The server accepted the following 4 cipher suites:
        TLS_RSA_WITH_AES_256_CBC_SHA                      256
        TLS_RSA_WITH_AES_128_CBC_SHA                      128
        TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA                256       ECDH: prime256v1 (256 bits)
        TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA                128       ECDH: prime256v1 (256 bits)

     The group of cipher suites supported by the server has the following properties:
       Forward Secrecy                    OK - Supported
       Legacy RC4 Algorithm               OK - Not Supported


 * SSL 2.0 Cipher Suites:
     Attempted to connect using 7 cipher suites; the server rejected all cipher suites.

 * TLS 1.0 Cipher Suites:
     Attempted to connect using 80 cipher suites.

     The server accepted the following 4 cipher suites:
        TLS_RSA_WITH_AES_256_CBC_SHA                      256
        TLS_RSA_WITH_AES_128_CBC_SHA                      128
        TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA                256       ECDH: prime256v1 (256 bits)
        TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA                128       ECDH: prime256v1 (256 bits)

     The group of cipher suites supported by the server has the following properties:
       Forward Secrecy                    OK - Supported
       Legacy RC4 Algorithm               OK - Not Supported
```

Figure 7 – Caixa.gov.br Detail Report

This website uses legacy TLS 1.0 and TLS 1.0 protocol.



github.com/SSLyze410-SSLGrader-wCipherSuite-info/ssl_wrapping_grader/blob/main/test_case/results/www.mfaic.gov.kh.txt

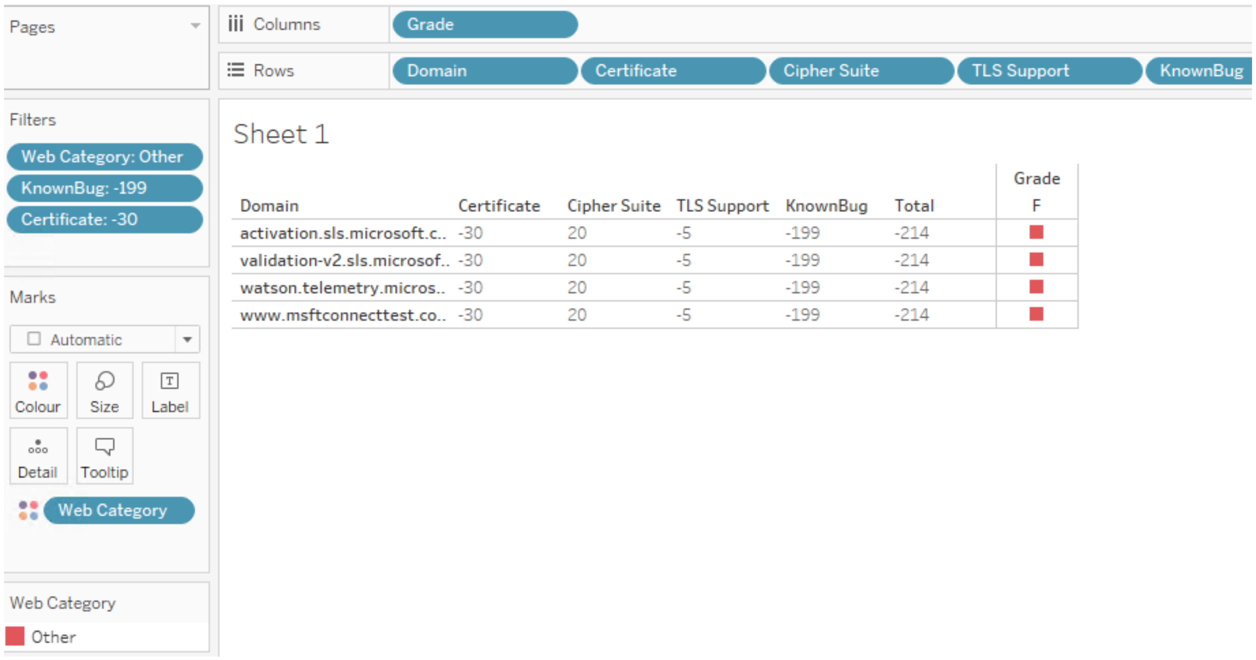
```


SCAN RESULTS FOR WWW.MFAIC.GOV.KH:443 - 103.16.61.37
----------------------------------------------------

* TLS 1.0 Cipher Suites:
    Attempted to connect using 80 cipher suites.

    The server accepted the following 5 cipher suites:
       TLS_RSA_WITH_AES_256_CBC_SHA                      256
       TLS_RSA_WITH_AES_128_CBC_SHA                      128
       TLS_RSA_WITH_3DES_EDE_CBC_SHA                     168
       TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA                256       ECDH: secp384r1 (384 bits)
       TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA                128       ECDH: prime256v1 (256 bits)

    The group of cipher suites supported by the server has the following properties:
      Forward Secrecy                   OK - Supported
      Legacy RC4 Algorithm              OK - Not Supported


* TLS 1.1 Cipher Suites:
    Attempted to connect using 80 cipher suites.

    The server accepted the following 5 cipher suites:
       TLS_RSA_WITH_AES_256_CBC_SHA                      256
       TLS_RSA_WITH_AES_128_CBC_SHA                      128
       TLS_RSA_WITH_3DES_EDE_CBC_SHA                     168
       TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA                256       ECDH: secp384r1 (384 bits)
       TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA                128       ECDH: prime256v1 (256 bits)

    The group of cipher suites supported by the server has the following properties:
      Forward Secrecy                   OK - Supported
      Legacy RC4 Algorithm              OK - Not Supported

```

Figure 8 – mfaic.gov.kh Detail Report

This website uses legacy TLS 1.1 protocol.

## **Business (Other)**

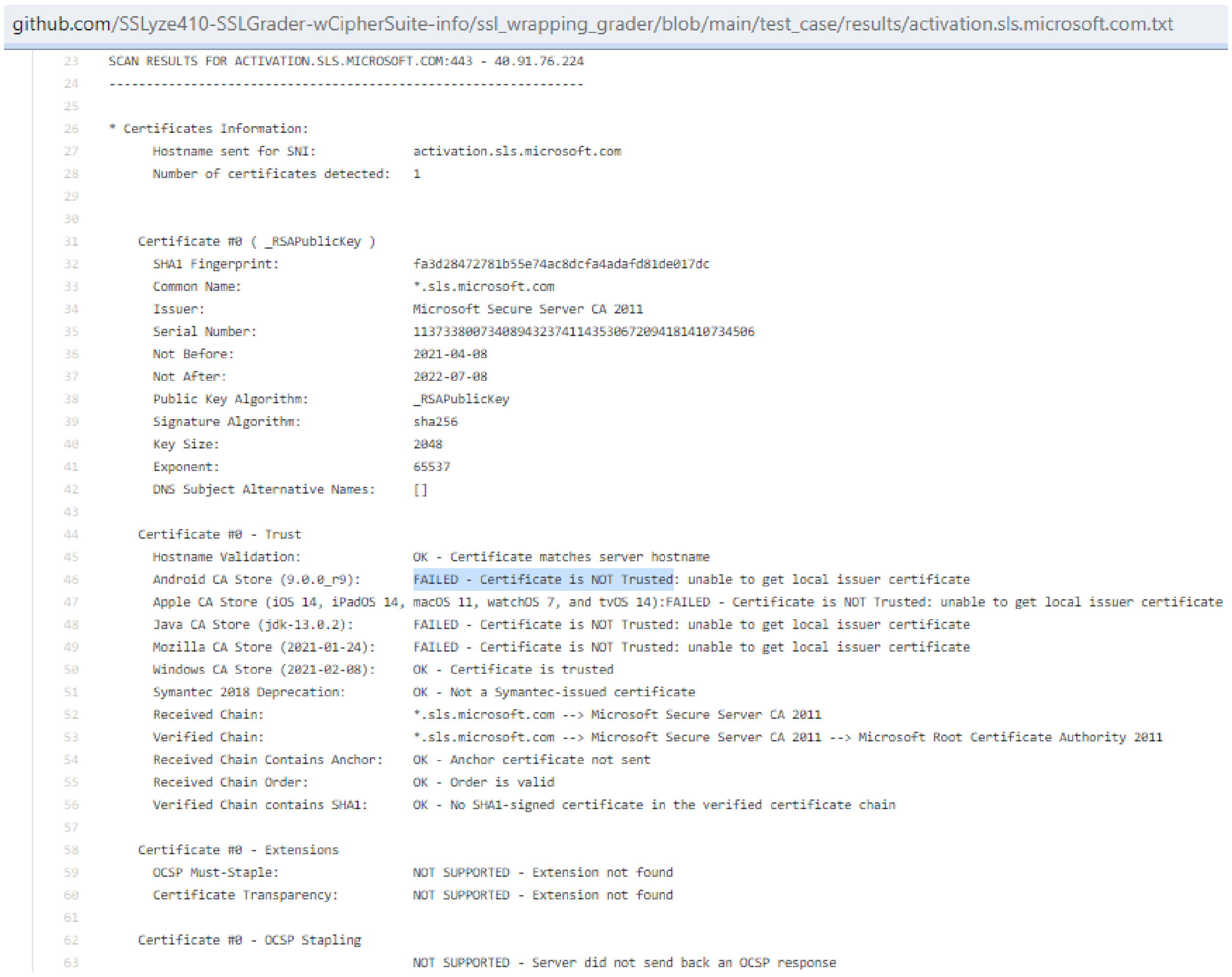
Last but not least business under other category are explored with filter KnownBug and Certificate (-30 aka Failed), we need to narrow down at the onset as there are a lot more data from this grouping.



Sheet 1

| Domain | Certificate | Cipher Suite | TLS Support | KnownBug | Total | Grade F |
|---|---|---|---|---|---|---|
| activation.sls.microsoft.c.. | -30 | 20 | -5 | -199 | -214 | 🟥 |
| validation-v2.sls.microsof.. | -30 | 20 | -5 | -199 | -214 | 🟥 |
| watson.telemetry.micros.. | -30 | 20 | -5 | -199 | -214 | 🟥 |
| www.msftconnecttest.co.. | -30 | 20 | -5 | -199 | -214 | 🟥 |

- Which domain(s) failed? And why?
  Microsoft, 4 domains.
  See sample detail report on Figure 9



github.com/SSLyze410-SSLGrader-wCipherSuite-info/ssl_wrapping_grader/blob/main/test_case/results/activation.sls.microsoft.com.txt

```
SCAN RESULTS FOR ACTIVATION.SLS.MICROSOFT.COM:443 - 40.91.76.224
-----------------------------------------------------------------

 * Certificates Information:
       Hostname sent for SNI:             activation.sls.microsoft.com
       Number of certificates detected:   1


     Certificate #0 ( _RSAPublicKey )
       SHA1 Fingerprint:                  fa3d28472781b55e74ac8dcfa4adafd81de017dc
       Common Name:                       *.sls.microsoft.com
       Issuer:                            Microsoft Secure Server CA 2011
       Serial Number:                     1137338007340894323741143530672094181410734506
       Not Before:                        2021-04-08
       Not After:                         2022-07-08
       Public Key Algorithm:              _RSAPublicKey
       Signature Algorithm:               sha256
       Key Size:                          2048
       Exponent:                          65537
       DNS Subject Alternative Names:     []

     Certificate #0 - Trust
       Hostname Validation:               OK - Certificate matches server hostname
       Android CA Store (9.0.0_r9):       FAILED - Certificate is NOT Trusted: unable to get local issuer certificate
       Apple CA Store (iOS 14, iPadOS 14, macOS 11, watchOS 7, and tvOS 14):FAILED - Certificate is NOT Trusted: unable to get local issuer certificate
       Java CA Store (jdk-13.0.2):        FAILED - Certificate is NOT Trusted: unable to get local issuer certificate
       Mozilla CA Store (2021-01-24):     FAILED - Certificate is NOT Trusted: unable to get local issuer certificate
       Windows CA Store (2021-02-08):     OK - Certificate is trusted
       Symantec 2018 Deprecation:         OK - Not a Symantec-issued certificate
       Received Chain:                    *.sls.microsoft.com --> Microsoft Secure Server CA 2011
       Verified Chain:                    *.sls.microsoft.com --> Microsoft Secure Server CA 2011 --> Microsoft Root Certificate Authority 2011
       Received Chain Contains Anchor:    OK - Anchor certificate not sent
       Received Chain Order:              OK - Order is valid
       Verified Chain contains SHA1:      OK - No SHA1-signed certificate in the verified certificate chain

     Certificate #0 - Extensions
       OCSP Must-Staple:                  NOT SUPPORTED - Extension not found
       Certificate Transparency:          NOT SUPPORTED - Extension not found

     Certificate #0 - OCSP Stapling
                                          NOT SUPPORTED - Server did not send back an OCSP response
```

Figure 9 – Microsoft Detail Report

Microsoft SSL Certificate is toxic to Non-Microsoft brand CA. Hence there are implication for cross platform application that read this. i.e., Android, Java and Mozilla cannot authenticate Microsoft certificate.

## Conclusion

As we all know nowadays internet banking, online learning, online marketing, online businesses have expanded its horizon. Lots of people regularly access various types of platforms to speed up the process of business. In this case, Internet security is concerned with protecting digital assets, networks, and computers from attack or unauthorized access of data, therefore SSL/TLS is become the most important part of the internet world.

As we observed in this paper, a lot of known brands are affected by BAD SSL. If we talk about data security, the website/domain should have at least grade. Currently there are still websites running on legacy TLS 1.0 and TLS 1.1.

## Items To Discuss

I and the other MSSD student Zaw Myothet came out with the idea to make more stricter in SSL grading. The basic idea is, below TLS1.2 consider as the legacy systems. The program is based on SSLyze 4.1.0 with API - https://ciphersuite.info

## Reference

[1] Ms.Dipti S.Charjan et al, An Overview of Secure Sockets Layer , http://www.researchpublications.org/IJCSA/NCAICN-13/245.pdf

[2] Mr Narbehai, SSL checker, https://github.com/narbehaj/ssl-checker

[3] Ciphersuite.info, https://ciphersuite.info

[4] Steve Lloyd et al, Understanding Certificate Path Construction, 2002, PKI Forum, 2002

[5] Cloudfare, https://developers.cloudflare.com/fundamentals/internet/protocols/tls

[6] Digicert, https://www.digicert.com/faq/ssl-cryptography.htm

[7] SSLLabs, https://github.com/ssllabs/research/wiki/SSL-Server-Rating-Guide

[8] SSL Grading script, POTH SUTD, https://github.com/SSLyze410-SSLGrader-wCipherSuite-info/ssl_wrapping_grader

[9] SSLyse, SSL Checker Core, https://github.com/nabla-c0d3/sslyze

[10] Straits Times News Paper, CyberCrime, https://www.straitstimes.com/tech/tech-news/public-sector-data-leaks-total-108-last-year-up-from-75-cases-in-2019