# X.509 certificate error testing

David McLuskie
Xavier Bellekens

# X.509 Certificate Error Testing

D.McLuskie
Abertay University
Dundee, UK
d.mcluskie@abertay.ac.uk

X.Belleken
Abertay University
Dundee, UK
x.belleken@abertay.ac.uk

## ABSTRACT

X.509 Certificates are used by a wide range of technologies to verify identities, while the SSL protocol is used to provide a secure encrypted tunnel through which data can be sent over a public network. Combined both of these technologies provides the basis of the public key infrastructure (PKI). While the concept of PKI is a good idea, the different implementation of the technologies in different operating system and clients often lead to weaknesses. This paper proposes a methodology to automate the testing of SSL clients by generating both bogus and malformed certificates in order to evaluate the client's response and identify potential threats to network infrastructures.

## CCS CONCEPTS

• **Computer systems organization → Security and Privacy**; Systems Security; Network Security;

## KEYWORDS

X.509, SSL, PKI, Certificates, Certificate Authority

## 1 INTRODUCTION

Certificates play an important part in validating the authenticity of the server that a client is connecting to, while the Secure Socket Layer (SSL) protocol is used to provide an encrypted tunnel through which traffic can be securely sent. Together, certificates and the SSL protocol provide the cornerstone upon which PKI is built [1].

To verify a server, a certificate authority (CA) like Godaddy or VeriSign can be used to sign a certificate, thus leaving a signature on the signed certificate by the CA. When a client wants to verify that a server's certificate is valid the signature of the CA is examined. If the signature is determined to be valid, then the server's certificate is deemed to be valid as well.

In a recent Apple SSL security bug it was discovered that an invalid X.509 certificate present on a server allowed Apple clients to access the server as if the certificate was valid [2]. However, when using an alternative client an error screen was returned to the client indicating the invalidity of the certificate, which is the expected result.

This Apple bug highlights one of the key problems of the SSL protocol in that the developers are allowed to implement the protocol in different ways and this could lead to the protocol behaving differently between implementations. While the SSL protocol provides a secure layer, it is critical for the protocol to be implemented uniformly as described by the standard.

The contribution of this work are as follows:

- Reviewing the current state of PKI.
- Presenting a methodology which can be used for evaluating weaknesses in the PKI

The remainder of this paper is organised as follows, Section 2 presents the background on PKI and X.509 certificates while also describing the SSL handshaking process used to authenticate clients. Section 3 describes the methodology that will be used to test the SSL clients for any potential problems, while in Section 4 the test results from the execution of the methodology are presented and discussed. The paper ends with the conclusions and future work in Section 5.

## 2 BACKGROUND

The PKI is a method of providing authentication and secure communication over a public network like the Internet. The main components of a PKI are as follows [3]:

*ARES 2018,* August 27–30, 2018, Hamburg, Germany
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-6448-5/18/08...$15.00
https://doi.org/10.1145/3230833.3232820

1. Digital Certificates
2. Certificate Authority(CA)

## 2.1 Digital Certificates

A server will typically use an X.509 digital certificate to validate its identity to a client. The validation of the server's identify is underpinned by getting a CA to validate its identity.

The SSL Protocol is used in conjunction with X.509 certificates to authenticate the server that the client is connecting to, along with providing encryption mechanisms to protect the data flow between the client and the server [4].

When an SSL client connects to a server the handshake process shown in Figure 1 [5] is executed. When this process has completed the client will have authenticated the server as well as negotiated an encrypted tunnel.

As shown in Figure 1, the process begins with the client connecting to the server by sending a **ClientHello** message. This message contains a list of supported ciphers supported by the client (e.g. MD5, SHA1, etc.) along with a random number generated by the client.

The second step in the SSL protocol is for the server to respond to the client with its own message called the **ServerHello**. This message contains a list of ciphers supported by the server along with a random number generated by the server. In addition the Certificate message contains the server's public key and hostname which have been digitally signed by a CA. Note however, that it is the responsibility of the client to authenticate the certificate.

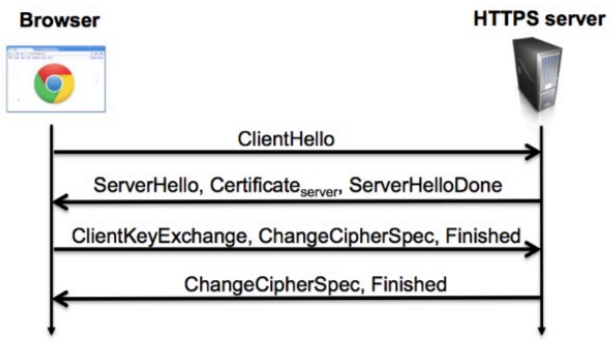

Browser
HTTPS server
ClientHello
ServerHello, Certificate$_{server}$, ServerHelloDone
ClientKeyExchange, ChangeCipherSpec, Finished
ChangeCipherSpec, Finished

**Figure 1: Basic SSL Handshake Exchange**

The client's next step is to create the pre-master key which will be used to form the session key for encrypting the traffic between the client and the server. The pre-master key will be encrypted using the server's public key, which was exchanged in the **ServerHello** message. The **ClientKeyExchange** message is then used to transmit the encrypted pre-master key to the server.

When the **ClientKeyExchange** message is received by the server the pre-master key will be decrypted using the server's private key. Both the Client and Server will now be able to derive the same session key for encrypting the traffic by using the random numbers sent by the **ClientHello** and **ServerHello** messages and the pre-master key.

The final step in the SSL process is for the client and server to negotiate the ciphers that are being used. This is done using a final message called the **ChangeCipherSpec**.

Once this process is complete a secure communication path between the server and the client will be established.

## 2.2 Certificate Authorities

When PKI is used one area of concern that needs to be addressed is how servers are authenticated. To help accomplish this a body called a CA can be used.

The CA is responsible for taking an application from a server and verifying that they are the legitimate owner of the domain that they are trying to authenticate. For example, if a server submits an application for bank.com the CA will be responsible for verifying that the application is from the server who owns the website and not a hacker trying to set up a phishing website to attract unsuspecting customers [4]

### 2.2.1 Problems with CA

The idea of getting a third party to verify the ownership of a website is a sound idea, however it has been proven that the checks that a CA performs to verify the identity of a website can be circumvented.

There have been instances in the past where a CA has not performed the necessary checks and issued certificates to customers when they should not have [4]. This behaviour led to uncertainty and lack of trust in the PKI infrastructure [6]

In order to gain a good level of trust the CA must offer a robust verification process that focuses on verifying the credentials of the customer [7].

## 2.3 X.509 Certificates

Within the PKI there is a key requirement to verify that the server, a client is trying to establish a secure communication with, is legitimate. The mechanism through which this is achieved is based upon X.509 certificates. The general structure of an X.509v3 certificate is shown in Figure 2 [8].

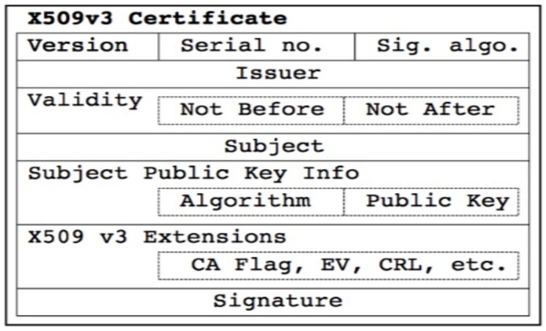


**Figure 2: X.509 Certificate Structure**

### 2.3.1 ASN Formatting

The general format of an X.509 certificate is defined by using an Abstract Syntax Notation One (ASN1) [3], which is an artificial language, used for describing data and data structures.

While the X.509 certificate is used to specify the format of a certificate, it does not dictate how the encoding is being achieved. This means that one implementation of the ASN1 encoding rules can differ from another [4], leading to potential vulnerabilities in a version of the ASN1 rules.

### 2.3.2 X.509 Certificate Field Names

This section examines the structure of an X.509v3 certificate by discussing some of the common fields used to create a certificate as shown in Figure 2.

The Common Name (CN) is the most important field as it is used to indicate the name of the website that is being connected to [8]. If the certificate is for the domain name corresponding to *www.google.com,* then when the domain name is entered as a URL in a web browser it is expected that both the URL field in the browser and in the certificate match. If the names are different then a warning message will be displayed.

The CA also uses this field to verify the identity of the owner for the website by taking the URL specified in the CN and performing a DNS lookup. The information specified in the DNS lookup is compared against the information supplied on the application to verify that there are no irregularities. If all fields match then a certificate is issued.

The normal procedure is to just verify the CN, but in cases where a company wants to give the users connecting to the website a higher degree of confidentiality, an additional step called the *Extended validation check* can be provided.

This extended validation process is used to try and stop hackers creating phishing websites. For example, a hacker can successfully register the domain bankofscotland.net. When an application is made to a CA for an X.509 certificate, the certificate will be granted, as the hacker is the legitimate owner of the domain name.

The extended validation on the other hand takes the process a step further by performing a more in-depth check of the domain by verifying that the domain actually belongs to the company, as well as verifying that the company is legitimate and that they are not trying to impersonate another organisation. A check will also be made against the content that the domain is serving to make sure that it is legitimate.

This process is typically undertaken by large corporations since they are able to afford the additional costs in going through the more rigorous verification process. In a web browser an extended validation certificate is displayed as a green title bar.

The signature algorithm is used to indicate the type of hash algorithms supported by the server hosting the certificate. The hash algorithms range from MD5 to SHA. Due to the proven MD5 collisions [4] and weaknesses with SHA1 it means that at a minimum SHA256 should be used for hashing [9].

The validity period is used to specify the period through which the X.509 certificate is valid for. Two fields make up the validity period and they are **notbefore** and **notafter**. The **notbefore** field is used to set a start date while the **notafter** field is used to set an end date for the certificate validity.

In most cases the time period specified for the validity period will be set to a low number, anywhere between 1 – 5 years. With the increasing computational power and the rise of quantum computing, weaknesses in the algorithms used may be discovered, hence a short time period is better for security reasons. [8].

The public key field will be populated when a certificate is generated [10]. The certificate contains general information about the holder and the public key will be embedded in the certificate. In addition to the public key being created a corresponding private key will also be created.

The public and private keys are components used in the PKI to help create a secure communication path between two entities.

When a CA first starts out it will create a root certificate that can be used to sign other certificates. This certificate will be used to create what is called an intermediate certificates (IC) and these will be used by an intermediate authoritie (IA) to sign certificates.

The main reason why the root certificate is not used to sign the certificates is that if it is stolen or compromised then all certificates signed by the CA would be at risk. By getting the IA to sign the certificates using the IC, it means that the root CA is protected and only used to create ICs.

Once an IA has obtained the IC, it can in turn create its own IC. Each time an IC is created the chain that is included in the certificate will be updated. This chain shows the path that has been taken to create the certificate. This means that all ICs used to create the current certificate will be included, terminating with the root CA.

By following this process, if one IC is violated then only that certificate is made invalid and therefore only the certificates signed with it will required to be reprocessed [5].

Each time a CA issues a certificate a serial number identifying the certificate is associated with it. This serial number should be unique and not repeated in any other certificate issued by the CA [11].

The country field is a 2 character field used to identify the country of origin the certificate is located in.

The final field that will be examined is the Organisation Unit (OU) field. This field is used to indicate the organisation that the certificate belongs to.

## 2.4 Creating X.509 Certificates

There are two options for creating certificates, the first option is called self-signed while the second option uses what is called a CA.

Self-signing allows system administrators to create their own root certificate and use it to sign their own certificates. While this is a cost saving option, it has the disadvantage that the identity of their server is not proven by a recognised authority [1].

The self-signing process should only be used when the client is connecting to internal servers and there is absolute trust in the server they are establishing a connection with. It is also important to note that the root certificate generated is not included in the root store of the web browser. This means that when the SSL client browses to the website an error message will be shown indicating that the connection cannot be trusted.

The user will typically ignore the warning message and proceed to the website. This behaviour is fine if the user is going to a trusted website that they know, but there is evidence that they also do this when going to untrusted websites which can end up being a security risk [12].

An alternative to using self-signed certificates is to use a CA. The purpose of the CA is to provide a mechanism whereby the identity of the owner of the website can be verified by a third party and thus increasing trust.

#### 2.4.1 OpenSSL

For creating X.509 certificates there are a range of Application Program Interfaces (APIs) available that can be used. The most popular API is called OpenSSL [13].

OpenSSL is an open source implementation of the SSL protocol that allows the use of various cryptographic algorithms such as AES, DES, and RSA to create X.509 certificates [14]. The advantage of OpenSSL is that it is available on many different platforms including Windows and Linux.

However one of OpenSSL greatest strength is one of its weaknesses in that with it being an open source project there is the danger that the code has not been fully verified via the use of code reviews [15]. This could lead to exploits as was seen with the heart bleed bug [16].

The rest of the paper will focus on describing a methodology which can be used for testing for weaknesses in how the SSL protocol has been implemented in clients.

## 3. PROCEDURE

This section provides a high-level overview of the proposed methodology for testing SSL. This methodology will be used for finding weaknesses in an SSL implementation as well as being able to be used to identify differences in how SSL functions between different implementations.

The block diagram in Figure 3 shows how the methodology works.

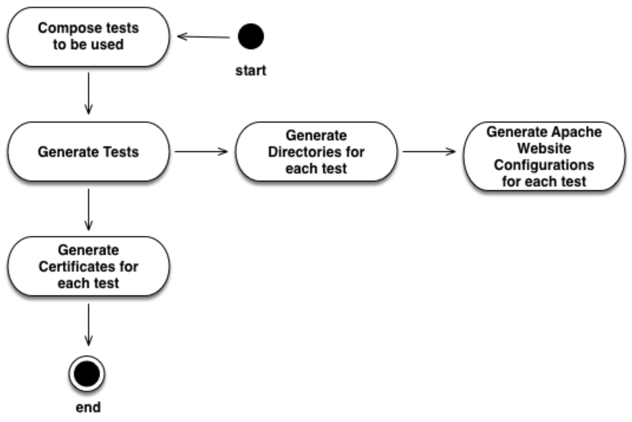
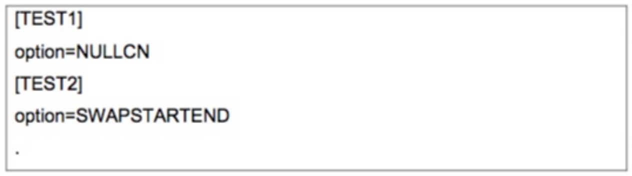

Compose tests to be used
start
Generate Tests
Generate Directories for each test
Generate Apache Website Configurations for each test
Generate Certificates for each test
end

**Figure 3: Methodology Block Diagram**

### 3.1 Compose Tests

The compose tests procedure requires the creation of a configuration file containing the tests that are going to be generated by the methodology. The format of the configuration file is shown in Figure 4.

The tests will be created programmatically via the use of the Python programming language and the OpenSSL API. The code will be designed to be modular so that a programmer will be able to create new tests as required.



```
[TEST1]
option=NULLCN
[TEST2]
option=SWAPSTARTEND
.
```

**Figure 4: Test Configuration File**

The format of the file starts with a test name, in this case the first test is named **TEST1.** The test name is used to create a directory containing the generated HTML page alongside the generated X.509 certificates.

The option parameter in the configuration file is used to specify the test that is going to be executed. Figure 4 shows that **TEST1** executes the **NULLCN** option, which produces an X.509 certificate with a null character in the common name field.

### 3.2 Generate Tests Function

The purpose of the Generate Test function is to take the generated configuration file that has been created and perform the following steps :

1. Generate Directories for each test
2. Generate an Apache configuration file for each test
3. Generate an HTML page for each test

Each of these steps will be examined in the next sections

### 3.2.1 Generate Directories

The aim of the generate directories function is to create a unique directory based upon the test name. This function will first examine the current directory structure and if it detects that there are files left over from a previous test then it will delete them. This ensures that the test will be starting from a blank slate.

### 3.2.2 Generate Apache Website Configuration

Once the directory structure has been created the next step is to create the Apache configuration file. This configuration file contains the directory locations of the root of each website generated along with where the generated X.509 certificate and key are located.

The final step executed in the Generate Apache Website function is to generate a unique HTML page for each test based upon the option parameter in the configuration file. The generate HTML file process will be used to uniquely identify each test that will be executed by the client.

Once this process has finished the methodology will then proceed to the next step in Figure 4, which is the Generate Certificate procedure.

## 3.3 Generate Certificate

The purpose of the generate certificate function is to generate unique X.509 certificates that contain errors. Each item in the configuration file will be processed and an X.509 certificate will be generated and placed in the corresponding directory that was created during the **Generate Test** function.

## 3.4 Test Scenarios

This section discusses the test Scenarios generated for the methodology along with what the expected result of each test is. Table 1 shows the list of tests that are generated by the methodology along with the expected outcomes.

**Table 1: List of Tests**

| Test Name | Result |
|---|---|
| WrongKey | Fail ⊗ |
| SwapStartEnd | Fail ⊗ |
| MissingStart | Fail ⊗ |
| MissingEnd | Fail ⊗ |
| LongEnd | Pass √ |
| NullCN | Fail ⊗ |
| FOOCN | Fail ⊗ |
| TabCN | Fail ⊗ |
| BackspaceCN | Fail ⊗ |
| LongOU | Fail ⊗ |
| LongRandomSerial | Fail ⊗ |
| SameSerial | Fail ⊗ |

### 3.4.1 WrongKey Test

The **WrongKey** test will sign the X.509 certificate using the wrong key. This means that the certificate will be invalid. When an SSL client encounters the certificate it is expected that an error message will be generated and as result block the website from loading.

### 3.4.2 SwapStartEnd Test

The **SwapStartEnd** test generates a certificate with the start and the end date swapped. Thus a certificate which is supposed to have a start date of 25/01/18 and an end date of 25/08/24 will instead be swapped so that the start data will be 25/08/24 and the end date will be 25/01/18.

It is expected that when an SSL client encounters this certificate that a warning page is displayed stating that the certificate is invalid and the website will be blocked from loading.

### 3.4.3 MissingStart Test

The **MissingStart** test will generate a certificate that has the start date missing from the certificate. It is expected that when an SSL client encounters this certificate that a warning page will be displayed stating that the certificate is invalid and the website will be blocked from loading.

### 3.4.4 MissingEnd Test

The **MissingEnd** test will generate a certificate that has the end date missing from the certificate. It is expected that when an SSL client encounters this certificate that a warning page will be displayed stating that the certificate is invalid and the website will be blocked from loading.

### 3.4.5 LongEnd Test

The **LongEnd** test will generate a certificate with an end date that is far off in the future, which in this case will be 50 years.

Normally a certificate will have an end date that is 5-10 years in the future and it will be interesting to see if an SSL client can handle an end date that is 50 years in the future. When an SSL client encounters this certificate it is expected that no error messages will be displayed and it will allow access to the website.

### 3.4.6 NullCN Test

The **NullCN** test will insert a null character (\0) in the common name of an X.509 certificate. Ideally the encoding routine used should reject this invalid common name, as NULL characters should not be allowed.

### 3.4.7 FooCN Test

The **FooCN** test will include a different common name than what is expected. The website will be using the domain name of localhost.example.com but when the FOOCN option is specified a different common name will be used which in this case will be localhost.foo.com.

It is expected that when the browser encounters a certificate with a common name that is different from the domain name then a warning message will be displayed and the website will not be loaded.

#### 3.4.8 TabCN Test

The **TabCN** test will insert a tab escape character into the common name. Like with the **NULLCN** test it is expected that the tab escape character will be rejected by the encoding routines.

#### 3.4.9 BackspaceCN Test

The **BackspaceCN** test will insert a backspace escape character into the common name. Like with the **NULLCN** option it is expected that the tab escape character will be rejected by the encoding routines.

#### 3.4.10 LongOU Test

The**LongOU** test will use a long character string and attempt to overflow the string buffer used to store the organizational unit name.

#### 3.4.11 LongRandomSerial Test

The **LongRandomSerial** test will generate a large number that would not normally be seen on an X.509 certificate. It is expected that there will be an upper limit to the length of the serial number and that the test will fail due to an invalid number being used.

#### 3.4.12 SameSerial Test

The **SameSerial** will generate the same serial number for two X.509 certificates. It is expected that when an SSL client encounters a certificate with the same serial number from a different certificate then the website will fail to load.

## 4. RESULTS AND DISCUSSION

In this section, the results from the experiments are presented in the same order as described in the test scenario section. Each of the tests will be executed using the following software configurations as shown in Table 2.

**Table2: Test Configurations**

| Test Configuration | Operating System | SSL Client |
|---|---|---|
| 1 | Windows 10 Enterprise version 10.0.17134 | Firefox Web Browser 60.0.2 |
| 2 | MAC OSX 10.13.4 | Safari Web Browser 11.1 |

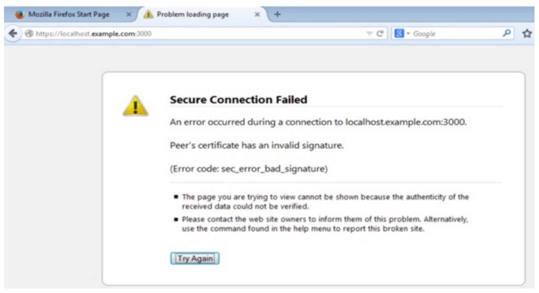
### 4.1 Test Configuration 1 Results

The results from the test configuration 1 are shown in Table 3.

**Table 3: Test configuration 1 results**

| Test Option | Expected Result | Result |
|---|---|---|
| WrongKey | Fail ⊗ | Fail ⊗ |
| SwapStartEnd | Fail ⊗ | Fail ⊗ |
| MissingStart | Fail ⊗ | Fail ⊗ |
| MissingEnd | Fail ⊗ | Fail ⊗ |
| LongEnd | Pass √ | Pass √ |
| NullCN | Fail ⊗ | Pass √ |
| FOOCN | Fail ⊗ | Fail ⊗ |
| TabCN | Fail ⊗ | Fail ⊗ |
| BackspaceCN | Fail ⊗ | Fail ⊗ |
| LongOU | Fail ⊗ | Pass √ |
| LongRandomSerial | Fail ⊗ | Pass √ |
| SameSerial | Fail ⊗ | Fail ⊗ |

The first test that was executed was the *WrongKey* test and as can be seen from the results in Table 3 the test failed. When the test web page was loaded a Secure Connection Failed error message was displayed as shown in Figure 6. This is the expected result since the wrong key was used to sign the certificate.


Mozilla Firefox Start Page
Problem loading page
https://localhost.example.com:3000
Google
Try Again

Secure Connection Failed

An error occurred during a connection to localhost.example.com:3000.

Peer's certificate has an invalid signature.

(Error code: sec_error_bad_signature)

- The page you are trying to view cannot be shown because the authenticity of the received data could not be verified.
- Please contact the web site owners to inform them of this problem. Alternatively, use the command found in the help menu to report this broken site.

**Figure 6: Wrongkey Secure Connection Error**

The *SwapStartEnd* test is where the start and end dates are swapped in the X.509 certificate. When the website was loaded a warning message was displayed stating that the website was not trusted. This is the expected result.

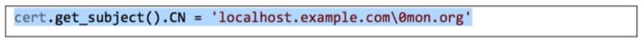
The next two tests *MissingStart* and *MissingEnd* were unable to be run since the Apache webserver would not allow the websites to load. This shows that Apache performs an internal validity check on the certificates that are being deployed on the server. If Apache deems that the certificates are invalid then Apache will not allow the website to start until the problems have been resolved.

The *LongEnd* test is were the end date is set 50 years into the future. The result of this test was a pass as the website was loaded with no errors encountered, which was as expected.

The next four tests NullCN, FOOCN, TabCN and BackspaceCN focus on testing the Common Name (CN) field.

Three of the tests ( FOOCN, TabCN and BackspaceCN) failed as expected. The reason for this is because the CN field, which is the name of the website being visited (localhost.example.com), is being modified. So this means that when the common name is compared to the website name then an error message is displayed as they are different.

The only test that passed when it should have failed is the NullCN test. In this test the common name was set to the value shown in Figure 7.



```
cert.get_subject().CN = 'localhost.example.com\0mon.org'
```

**Figure 7: Setting a NullCN**

In this case the '*\0*' is acting as null termination character for the string so that the common name is localhost.example.com instead of localhost.example.common.org. This means that the CA is being fooled into signing a certificate for a domain which the customer is not the legitimate owner, and thus is able to spoof the website.

A concerning fact is the escape characters that are being allowed in the fields of the X.509 certificates. There have been instances in the past where SQL injection commands (SQLi) have been stored in the fields, therefore if a database is being used to store information about the certificates then there is a possibility that it could be vulnerable to SQLi attacks if input sanitisation is not being performed. We would recommend that the X.509 standard is modified to stipulate that escape characters are forbidden from being used within the fields of the certificate. thus only standard ASCII and UNICODE characters would be accepted.

The next test that was executed was the LongOU test where an attempt is made to overflow the OU field of the X.509 certificate. This is being done to ascertain if this field is vulnerable to a buffer overflow attack.

The expected result for this test is to fail but as can be seen from the results this test passed. The reason for this result is because when it came to creating the certificate the OU field was not included as it exceeded the maximum permissible value for this field. This also proves that the OU field is not required for a valid certificate to be created since the website is loaded successfully when test is being executed.

The LongRandomSerial test is a test whereby an attempt to overflow the field storing the serial number is made. During the development of this test it was attempted to come up with a large enough number that would result in the serial number being rejected, but the serial number is converted into hex and the field grows to accommodate the larger maximum number used to generate the serial number. There did not appear to be an upper limit to the length of the serial number and thus the test passed. In the test the largest number tried was 321961016614730452433428166410946947104475398322062851583796333

The final test that was undertaken was the SameSerial test that uses the scenario of two certificates on the same web server having the same serial number.

When executing this test it was noted that the first website that was visited on port 3009 resulted in the website being displayed properly, with no error messages appearing. However when the second part of the test was loaded on port 3010 an error message was encountered stating that an invalid certificate was encountered due to a duplicate serial number

The same test was repeated again with new certificates, and this time the site accessed through port 3010 was first visited which result in a successful view of the website and then the site accessed on port 3009 was then visited which resulted in the error message stating that an invalid certificate was encountered due to a duplicate serial number. This process was repeated ten times yielding the same result. This appears to indicate that the first website visited will be classified as the legitimate website while the second website will be viewed as the fake.

This means that if an attacker can get the user to visit the illegitimate website first, then the attacker will have a higher chance of being successful. This result proves that there is a need for a methodology, such as the one developed for this paper, that can be used to find SSL weaknesses like this.

With having successfully executed the first set of tests the next step was to execute the same tests but using the second configuration.

## 4.2 Test Configuration 2 Results

The results from the test configuration 2 are shown in Table 4.

**Table 4: Test Configuration 2 Results**

| Test Option | Expected Result | Actual Result |
|---|---|---|
| WrongKey | Fail ⊗ | Fail ⊗ |
| SwapStartEnd | Fail ⊗ | Fail ⊗ |
| MissingStart | Fail ⊗ | Fail ⊗ |
| MissingEnd | Fail ⊗ | Fail ⊗ |
| LongEnd | Pass √ | Pass √ |
| NullCN | Fail ⊗ | Incomplete |
| FOOCN | Fail ⊗ | Fail ⊗ |
| TabCN | Fail ⊗ | Fail ⊗ |
| BackspaceCN | Fail ⊗ | Fail ⊗ |
| LongOU | Fail ⊗ | Pass √ |
| LongRandomSerial | Fail ⊗ | Fail ⊗ |
| SameSerial | Fail ⊗ | Pass √ |

The majority of the tests using test configuration 2 had the same result as the test 1 configuration, however there are three tests that had different results. The first result that is different is with the NullCN test. When an attempt was made to generate the test it was found that the code would not compile and an error message was returned which stated that NULL characters could not be encoded into the CN, this is exhibiting different behavior from when the same code was run in the test 1 configuration.

The next result that was different is the SameSerial Test. In the test 1 configuration the SameSerial failed as expected when the second web page was visited, however in the test 2 configuration the test passed. The final result that was different was the LongRandomSerial test. The test failed but in the test 1 configuration it passed.

The reason as to why the NullCN, SameSerial and LongRandomSerial tests had different results is in the way that the SSL client is implemented, since the standard does not insist upon a standard implementation, it is left up to the programmers to implement and this can result in differences in the way the clients implement certain functionality. This again highlights the need for a methodology, such as the one described in the paper, for testing SSL as it can be used to illustrate differences in behaviour between SSL implementations.

From the test results it shows that the Firefox web browser is more stringent in checking for errors in the X.509 certificates, while the Safari web browser is more lax in checking for errors.

## 5. CONLUSIONS

In this work we have presented a methodology that can be utilised to test for weaknesses in how a client has implemented the SSL protocol.

One of the main results that were achieved was to highlight how the different operating systems implement the SSL protocol as can be seen with the NullCN, SameSerial and LongRandomSerial test. These tests show that the methodology is able to be used to detect differences in how the SSL protocol has been implemented and proves the usefulness of the proposed methodology.

One area of concern that was discovered during the testing was that escape characters can be encoded in the X.509 certificates fields like Organisational Unit, Common Name and so on. This fact could be used to encode SQLi attacks against the CA database that is used to store certificate details. One recommendation that is made is that the specification should change to ban these types of characters from being allowed in the X.509 certificate and this should help to remediate against this problem.

From the evaluation of the results it was shown that it was not possible to exceed the limitations of the fields in an X.509 certificate and force a buffer overflow attack. The reason as to why the limits could not be exceeded is because the ASN1 structure was predefined by the use of OpenSSL API and it was not possible to change this.

To test for buffer overflows a different approach would have to be taken whereby the methodology would create its own ASN1 structure and not be reliant upon the structure defined by OpenSSL.

Another area that could be investigated is to examine the encoding of SQLi attacks within the X.509 certificates as numerous agencies store the certificates using databases.

## REFERENCES

[1] Liddy, C. and Sturgeon, A. The evolution of certificate model architecture. *Information management & computer security*, 7, 2 (1999), 95-100.
[2] Langley, A. *Apple's SSL/TLS bug, 2014*. City.
[3] Toma, C. Security Issues of the Digital Certificates within Public Key Infrastructures. *Informatica Economica*, 13, 1 (2009), 16.
[4] Kaminsky, D., Patterson, M. L. and Sassaman, L. *PKI layer cake: New collision attacks against the global X. 509 infrastructure*. Springer, City, 2010.
[5] Huang, L. S., Rice, A., Ellingsen, E. and Jackson, C. *Analyzing forged SSL certificates in the wild*. IEEE, City, 2014.
[6] Kim, T. H.-J., Huang, L.-S., Perrig, A., Jackson, C. and Gligor, V. *Accountable key infrastructure (AKI): A proposal for a public-key validation infrastructure*. ACM, City, 2013.
[7] Kamp, P.-H. Please Put OpenSSL Out of Its Misery. *ACM Queue*, 12, 3 (2014), 20-23.
[8] Holz, R., Braun, L., Kammenhuber, N. and Carle, G. *The SSL landscape: a thorough analysis of the x. 509 PKI using active and passive measurements*. ACM, City, 2011.
[9] Ouvrier, G., Laterman, M., Arlitt, M. and Carlsson, N. Characterizing the HTTPS trust landscape: a passive view from the edge. *IEEE Communications Magazine*, 55, 7 (2017), 36-42.
[10] Huang, J. and Nicol, D. M. An anatomy of trust in public key infrastructure. *International Journal of Critical Infrastructures*, 13, 2-3 (2017), 238-258.
[11] Basney, J., Fleury, T. and Gaynor, J. CILogon: A federated X. 509 certification authority for cyberinfrastructure logon. *Concurrency and Computation: Practice and Experience*, 26, 13 (2014), 2225-2239.
[12] Reeder, R. W., Felt, A. P., Consolvo, S., Malkin, N., Thompson, C. and Egelman, S. An Experience Sampling Study of User Reactions to Browser Warnings in the Field (2018).
[13] Bhargavan, K., Fournet, C. and Kohlweiss, M. mitls: Verifying protocol implementations against real-world attacks. *IEEE Security & Privacy*, 14, 6 (2016), 18-25.
[14] Saraiva, P. M. C. OpenSSL acceleration using Graphics Processing Units. *PhD diss., TECNICO LISBOA* (2013).
[15] Jimenez, M., Papadakis, M. and Le Traon, Y. *An empirical analysis of vulnerabilities in openssl and the linux kernel*. IEEE, City, 2016.
[16] Momani, E. M. H. and Hudaib, A. A. Z. Comparative Analysis of Open-SSL Vulnerabilities & Heartbleed Exploit Detection. *International Journal of Computer Science and Security*, 8, 4 (2014), 159-176.