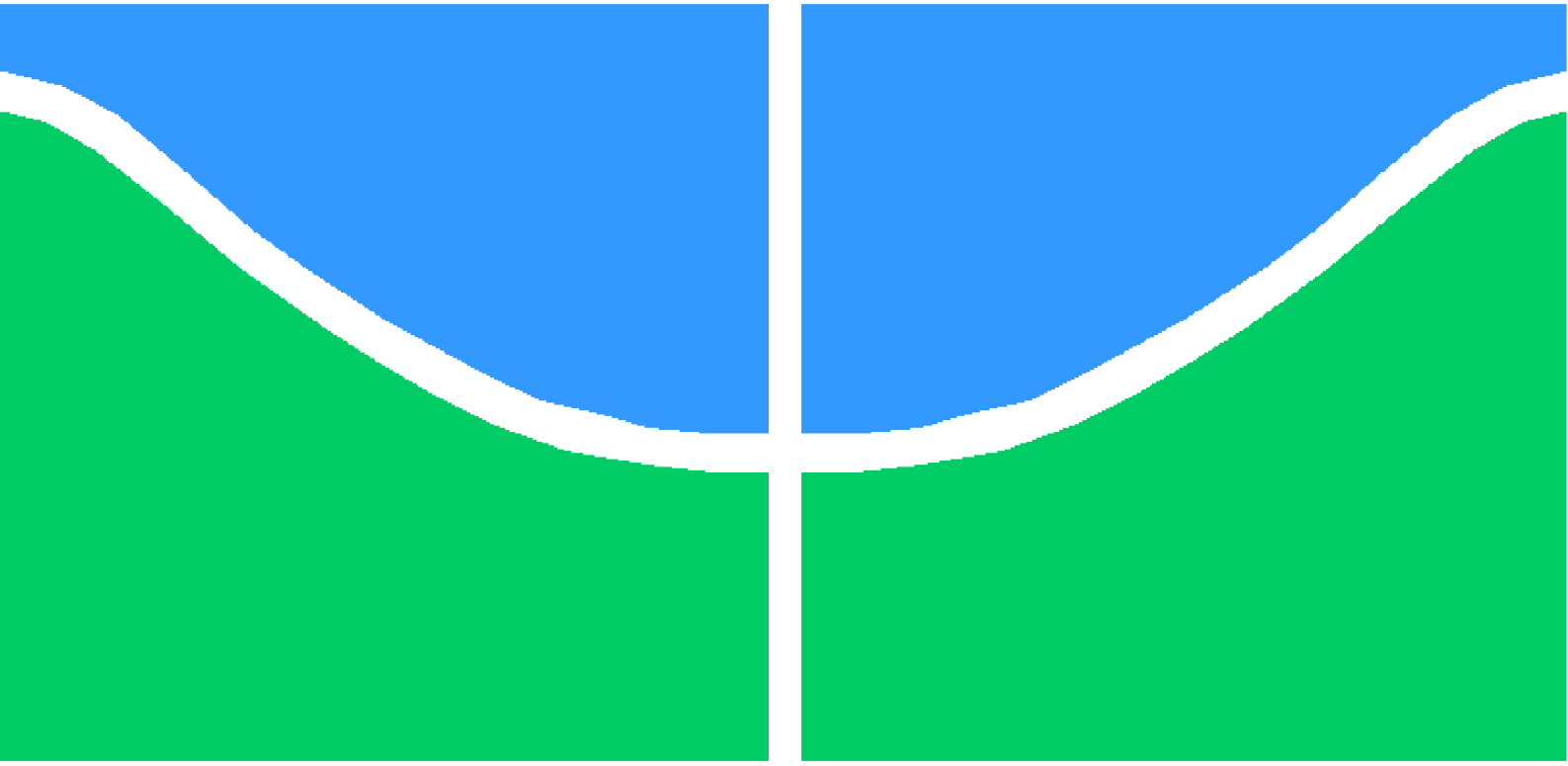


Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Engenharia de *Software*

Certificados Digitais: Exceções de Segurança e Vulnerabilidades (PRELIMINAR)

Autor: Yuri Moraes Mota

Orientador: Prof. Dr. Edson Alves da Costa Júnior

Brasília, DF

2015

Yuri Moraes Mota

Certificados Digitais: Exceções de Segurança e Vulnerabilidades (PRELIMINAR)

Monografia submetida ao curso de graduação em Engenharia de *Software* da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de *Software*.

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Prof. Dr. Edson Alves da Costa Júnior

Brasília, DF

2015

Yuri Moraes Mota
Certificados Digitais: Exceções de Segurança e Vulnerabilidades (PRELIMINAR)/ Yuri Moraes Mota. – Brasília, DF, 2015-
60 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Edson Alves da Costa Júnior

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2015.

1. Certificados Digitais. 2. Exceções de Segurança. I. Prof. Dr. Edson Alves da Costa Júnior. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Certificados Digitais: Exceções de Segurança e Vulnerabilidades (PRELIMINAR)

CDU 02:141:005.6

Yuri Moraes Mota

Certificados Digitais: Exceções de Segurança e Vulnerabilidades (PRELIMINAR)

Monografia submetida ao curso de graduação em Engenharia de *Software* da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de *Software*.

Trabalho aprovado. Brasília, DF, 27 de junho de 2014:

Prof. Dr. Edson Alves da Costa Júnior
Orientador

Prof. Dr. Fernando William Cruz
Convidado 1

Prof. Dr. Tiago Alves Fonseca
Convidado 2

Brasília, DF
2015

Este trabalho é dedicado a Deus.

Agradecimentos

Agradeço a Deus, por sustentar minha fé e me encher de esperanças quando não parecia haver solução; à minha mãe e irmãs, que me serviram de inspiração de perseverança nos momentos em que nada parecia valer à pena; Agradeço aos meus amigos, que estiveram comigo nos momentos felizes, e me apoiaram nos momentos tristes. Agradeço aos professores, que não desistiram de mim. Agradeço à minha Pequena, que me fez notar que as grandes bençãos não se prendem a grandes coisas. E finalmente, agradeço ao meu grande amigo, Maddox, eternamente vigilante.

"Aquele que habita no abrigo do Altíssimo
e descansa à sombra do Todo-Poderoso
Pode dizer ao Senhor:
Tu és o meu refúgio e minha fortaleza,
o meu Deus, em quem confio."
(Bíblia Sagrada, Salmos 91, 1-2)

Resumo

O objetivo desse trabalho é avaliar as exceções de segurança em *web browsers* decorrentes de problemas em seus certificados digitais, através da verificação destes certificados. Os erros que dão origem às principais exceções serão identificados e classificados. Posteriormente serão apontados os possíveis ataques que podem explorar tais erros, evidenciando os principais riscos a que estão expostos os sites que utilizam certificados que geram tais exceções.

Palavras-chaves: Certificação Digital. *Internet*. Verificação de Certificados. Criptografia.

Abstract

The objective of this work is to evaluate the security exceptions in *web browsers* due to problems in their digital certificates by verifying certificates. The errors that return the main exceptions are identified and classified. Posteriorly will be appointed the possible attacks that can exploit such errors, highlighting the main risks the sites are exposed to while using certificates that generate such exceptions.

Key-words: Digital Certification. Internet Security. Certificate Verification. Cryptography.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

AC	Autoridade Certificadora
ASN.1	*Abstract Syntax Notation One*
Cipher	Algoritmo de Cifração
OpenSSL	*Open Source Toolkit para SSL e TLS*
PKCS	*Public-Key Cryptography Standards* (Padrões de Criptografia de Chave Privada)
RSA	Algoritmo de Criptografia de Dados inventado por R. Rivest, A. Shamir, e L. Adleman
SSL	*Secure Sockets Layer*
TLS	*Transport Layer Security*

Sumário

1 Introdução

Certificados digitais são uma forma comum, atualmente, de comprovar a identidade de pessoas, físicas ou jurídicas, em meios digitais. Essa ferramenta de identificação precisa ser robusta, do contrário colocaria em dúvida a confiabilidade do sistema de identificação digital, entretanto, essas falhas existem e dificilmente são tratadas com a devida importância. Para garantir a segurança dos usuários de certificação digital, é necessário expor e identificar tais falhas.

Pela da exposição das falhas mais comuns entre os certificados digitais avaliados, nesse estudo, realiza-se uma análise acerca de quais fatores levam a essas falhas, se elas podem ser sanadas, e quais as principais implicações ocasionadas pela ocorrência das mesmas.

1.1 Objetivo Geral

Este trabalho se foca na identificação, por amostragem em sites que utilizam certificação digital, de quais são as principais exceções de segurança disparadas por certificados digitais utilizados em ambiente real.

1.2 Objetivos Específicos

Por estes mesmos meios, se busca (i)conhecer as estruturas internas dos certificados digitais utilizados em ambiente real; (ii)elucidar formas de contornar exceções de segurança advindas de incoerâncias no corpo de certificados digitais.

1.3 Metodologia

Dentro do padrão X.509, os tipos de erros serão divididos em grupos para que a análise esclareça quais as formas pelas quais esses erros podem se apresentar, e assim seja possível entender a origem e razão de tais erros. Todos os grupos de erros serão originários, e baseados, dos erros encontrados pela verificação realizada com a ferramenta `OpenSSL`, uma ferramenta aberta utilizada para realizar esse tipo de verificação.

1.4 Corpo do Trabalho

Na primeira parte deste trabalho, é apresentada uma introdução à teoria envolvida na criptografia, tendo como foco as razões pelas quais ela se faz necessária, os tipos de

ameaças digitais conhecidas, e as medidas tomadas para contornar esses problemas.

Uma vez que esses detalhes estão esclarecidos, será discutida a forma como a criptografia age para possibilitar a existência, e uso, dos certificados digitais e os algoritmos envolvidos em sua aplicação.

Essa informação servirá para descrever as características que permeiam os certificados digitais, e que permitem que eles sejam considerados como seguros em um ambiente real de aplicação, onde eles devem carregar informações suficientes para atestar a identidade de um indivíduo, ou grupo, em meio digital.

2 Fundamentação Teórica

Neste capítulo a principal meta é inteirar o leitor sobre a importância do uso e da qualidade de certificados digitais, de forma correta. Inicialmente serão apresentados detalhes sobre o que é comunicação segura, e a criptografia. Em seguida, serão abordados e apresentados os detalhes da criptografia, tanto simétrica quanto assimétrica. Por fim, serão discutidos certificados digitais e sua importância, assim como seu uso.

2.1 Criptografia e Protocolos

No contexto da computação, as redes de computadores têm se tornado cada vez maiores, esperasse um trânsito elevado de informações, e o uso dessas redes de computadores para a troca de informações é esperado.

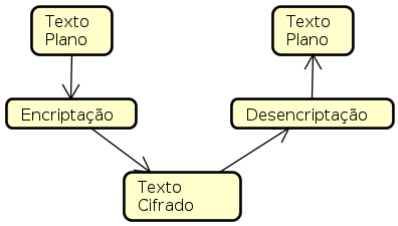
Nesse quadro, dois indivíduos pretendem trocar informações e desejam trocar suas mensagens de forma segura, eles desejam confidencialidade. Para esse fim, eles podem escolher utilizar a criptografia. A criptografia em si, possui muitos fins:

- **Confidencialidade** - Ocultar a informação contida em uma mensagem, de forma que apenas o emissor e o receptor conheçam seu conteúdo.
- **Autenticação** - Dar a possibilidade do recebedor identificar a origem de uma mensagem.
- **Integridade** - Permitir que o recebedor tenha certeza que aquela mensagem não foi modificada por um terceiro.
- **Não-Repudiação** - Um remetente não deve ser capaz de negar falsamente o envio de uma mensagem.

Os indivíduos de nosso exemplo serão, figurativamente, chamados de A e B, e suas mensagens serão chamadas de mA e mB, respectivamente. Chamando a mensagem original, mA, de **texto plano**, ela pode ser alterada através de *encriptação* para gerar um **texto cifrado**, enA, e retornará ao seu estado original passando por uma *desencriptação* (SCHNEIER, 1996, p. 15).

Um algoritmo de criptografia pode ser chamado de cipher code. Existem cipher codes cuja segurança depende de seu segredo, e existem outros cuja segurança depende do segredo de valores denominados chaves.

Figura 1 – Processo de Encriptação e Decriptação



Porém, a comunicação em si pode não estar completamente segura apenas com o uso da criptografia, caso um algoritmo ou chave de encriptação secretos sejam expostos, esses casos caracterizam quebras de segurança (STALLINGS, 2011, p. 399):

- **Quebra Total** - Um indivíduo externo descobriu a *chave* do algoritmo.
- **Dedução Global** - Um indivíduo externo encontra um algoritmo equivalente, sem conhecer a *chave*.
- **Dedução Seletiva** - Um indivíduo externo consegue acesso ao *texto plano* de um *texto cifrado*.
- **Dedução Existencial** - Um indivíduo externo tem acesso a uma parcela de informação da *chave* ou do *texto plano*.

Para reduzir as chances de possíveis vazamentos de mensagens, é possível seguir protocolos de comunicação, esses *protocolos* se fazem úteis tanto para padronizar a forma como as informações são trocadas quanto para ajudar na interpretação do que está sendo falado pelos envolvidos.

A criptografia em si pode ser reforçada por outros métodos que tornariam a troca de mensagens ainda mais segura, é possível então que os indivíduos insiram na sua forma de comunicação um **protocolo**. Um *protocolo* é um processo realizado pelos participantes de uma atividade (no caso a comunicação) para realizar essa atividade, sendo que esse processo apresenta passos bem definidos e que devem ser seguidos (SCHNEIER, 1996).

Um **protocolo criptográfico** é aquele que envolve protocolo que envolve criptografia (SCHNEIER, 1996, p. 31), como por exemplo o protocolo de comunicação *HTTPS*. Os protocolos ajudam a gerar confiança entre os envolvidos, eles podem exigir uma troca de chaves secretas ou uma linguagem específica. Isso pode ser feito para convencer os envolvidos de que são quem verdadeiramente afirmam. Deve-se ter em mente, também, que o protocolo deve se limitar ao que se propõe em cada caso, para aumentar a segurança dos indivíduos.

Como exemplo é possível imaginar dois estranhos se vendo pela primeira vez, imagine que o protocolo social que se segue é cumprimentar as pessoas com um aperto

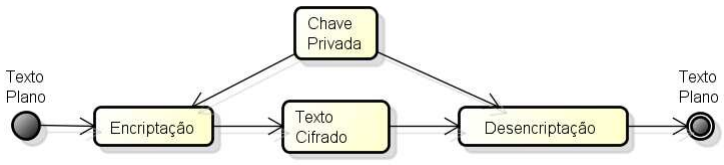
de mão. Caso os dois estranhos se encontrem na rua, e se cumprimentem com um aperto de mãos, eles dirão um ao outro seus respectivos nomes, e iniciarão uma conversa. Caso o aperto de mãos não ocorra, isso seria uma quebra de protocolo, e a comunicação entre eles terminaria ali.

2.2 Algoritmos e Chaves

Existem duas formas mais populares de se tratar com chaves de segurança, mantendo-as em segredo (*chaves secretas ou privadas*), na **criptografia de chave simétrica**, ou tornando-as acessíveis (*chaves públicas*) em par com chaves privadas, na **criptografia de chave assimétrica**. Cada um dos casos apresenta suas próprias vantagens, e desvantagens, sendo possível trabalhar com ambos objetivando uma troca segura de informações.

Nos algoritmos de chave privada, a chave é compartilhada pelos envolvidos na troca de informações. Utilizando de um mesmo algoritmo conhecido, eles irão enviar suas mensagens, e recebê-las, utilizando uma mesma chave secreta para o processo de encriptar e *desencriptar* as informações trocadas. Na atualidade, é comum que essas chaves privadas sejam monitoradas por **KDCs** (*Key Distribution Centers*), **Centros de Distribuição de Chaves**, afim de arbitrar a distribuição e consenso em relação à chave utilizada em uma troca.

Figura 2 – Processo de Encriptação e Desencriptação com Chave Privada



Já os algoritmos de chave pública seguem uma ideia diferente, existe uma chave conhecida publicamente e uma chave secreta, idealmente conhecida apenas pelo usuário. Através da encriptação utilizando a chave pública de um indivíduo, os algoritmos de chave assimétrica irão garantir que apenas a chave privada do indivíduo seja capaz de desencriptar a mensagem. Da mesma forma, utilizar a própria chave privada para encriptar um conteúdo irá garantir que apenas a chave pública seja utilizada para desencriptar corretamente aquela mensagem, atestando a origem daquela mensagem.

Nesse contexto, a encriptação com a própria chave privada serviria como meio de assinar um conteúdo, de forma a garantir a autenticidade e a não-repudiação. Em se tratando de uma comunicação verdadeiramente séria, é possível empregar mais de uma encriptação, com o objetivo de garantir que um conteúdo seja lido apenas pelo destinatário, e que seja provada a identidade do remetente.

Figura 3 – Processo de Encriptação em Chave Pública

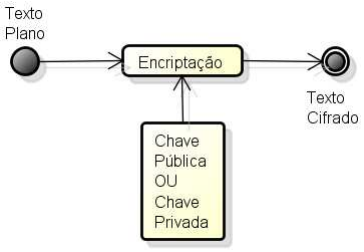
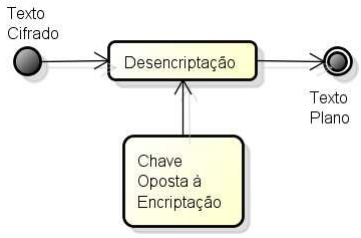


Figura 4 – Processo de Desencriptação em Chave Pública



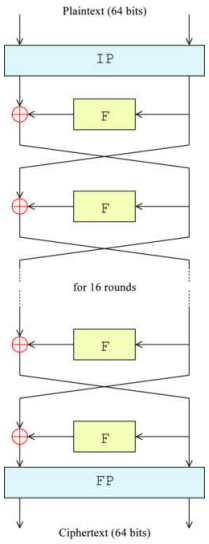
Entretanto, para garantir que essas trocas sejam feitas, e que haja consenso em como encriptar e desencriptar essas informações, é necessário que os algoritmos utilizados no processo sejam conhecidos por ambas as partes. Diferente de algoritmos que se tornam seguros através do segredo do algoritmo em si, algoritmos que utilizam chaves, geralmente, tem sua segurança embasada nas chaves em si, sendo os algoritmos públicos. Historicamente, existem diversos algoritmos de criptografia conhecidos e utilizados, vale citar dois desses algoritmos: o DES e o RSA.

Um dos primeiros algoritmos utilizados em larga escala, o DES (Data Encryption Standard) na verdade se trata de um padrão, detentor do algoritmo DEA (Data Encryption Algorithm). Em seu funcionamento, esse algoritmo adota a Função de Feistel, que realiza a encriptação através de quatro estágios. A função-F (como é conhecida a função de Feistel) é repetida diversas vezes dentro do algoritmo, de forma que ao final de diversas repetições será alcançado o texto cifrado.

Já o RSA, nomeado assim em referência aos seus criadores, é utilizado para a encriptação utilizando chaves assimétricas. Seu funcionamento difere, pois certas formalidades são exigidas para gerar as chaves a serem utilizadas nesse algoritmo. As chaves utilizadas dependem de dois números primos grandes (o que matematicamente diminui as chances de uma quebra de segurança) que irão ser utilizados em sua geração, que se dá de forma que:

1. Dois números primos grandes **p** e **q**, são escolhidos;
2. É calculado $\mathbf{n} = pq$;
3. Utiliza-se a função-φ onde $n : \varphi(n) = (p-1)(q-1)$;

Figura 5 – Processo de Encriptação do DEA (CRYPTO, 2005)



4. É escolhido um valor **e** (inteiro) maior que 1 e menor que $\varphi(n)$, de forma que e e $\varphi(n)$ sejam primos entre si.

5. Calcula-se **d** como inverso multiplicativo de e, onde mod($\varphi(n)$)

Assim gerando uma chave pública através do par *(e, n)*, e uma chave privada do conjunto *(p, q, d)* ou apenas *(d)*, o segundo caso sendo uma abordagem mais lenta, pois o desencriptador não conhece os primos originais. A encriptação e desencriptação se dão, respectivamente, pelas fórmulas: *(onde m é o texto plano, e c é o texto cifrado)*

1. $c = m^e mod n$

2. $m = c^d mod n$

Existem muitos algoritmos de cifração de chave pública, porém a maioria é impraticável por questões de desempenho. É importante notar que a complexidade envolvida nesses algoritmos desse tipo se deve a existência da própria chave pública, pois os procedimentos envolvidos para gerá-la não são simples, nem mais eficientes que algoritmos simétricos (STALLINGS, 2011). Para se trabalhar com esse tipo de sistema, é necessário que o algoritmo e a chave pública não sejam suficientes para descobrir a chave privada (RSA, 1993). Essa mesma capacidade de gerar chaves públicas em par com chaves privadas permite que algoritmos assimétricos gerem um texto cifrado a partir de um texto plano usando uma chave pública, e que a chave privada seja a única capaz de decifrar aquele texto cifrado gerado (RSA, 2012).

Originalmente o que foi proposto por Diffie-Hellman para as aplicações de algoritmos de chave pública permitia apenas a Troca de Chaves, já na abordagem RSA se tornou

possível utilizar Assinaturas Digitais (RSA, 2012, p. 12) e esquemas de Encriptação/Decriptação (RSA, 2012, p. 15); o uso do RSA se tornou difundido pela sua aplicabilidade e a impraticabilidade de ataques por conta do tamanho e administração de suas chaves.

Pela capacidade de gerar novas chaves públicas, e possuir uma chave privada, os algoritmos assimétricos servem aos seguintes propósitos (STALLINGS, 2011, p. 275):

- **Encriptação/Decriptação** - O remetente pode encriptar uma mensagem com a *chave pública* do receptor.
- **Assinatura Digital** - Um remetente pode assinar uma mensagem com sua *chave privada*. A assinatura pode ser alcançada por métodos diferentes, de acordo com os *protocolos* utilizados.
- **Troca de Chaves** - Duas entidades em conjunto geram uma **chave de sessão**. Assim como na assinatura digital, existem diversas aproximações para que isso seja alcançado.

2.3 Certificados Digitais

No contexto da comunicação segura, a ideia de saber a identidade do outro indivíduo é essencial; Não é desejável que um desconhecido se faça passar pela receptor, ou mesmo pelo emissor, um ̈homem do meio.̈ Para garantir essa comunicação de um para um, é desejável que sejam apresentadas credenciais que garantam que a mensagem foi transmitida sem desvios, e pelo emissor esperado. Acerca disso, Schneier exemplifica:

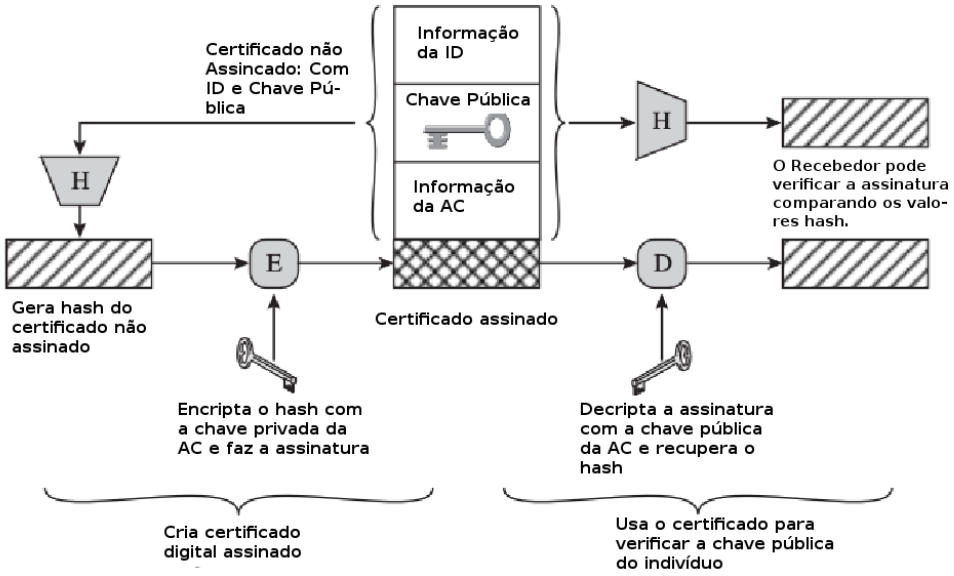
"Um certificado público é a chave pública de alguém, assinada por alguém confiável. Certificados são usados para frustar tentativas de substituir uma chave por outra [879]. O certificado de Bob, numa base de dados de chaves-públicas, possui muito mais que sua chave pública. Ela contém informações sobre Bob -seu nome, endereço, e por aí vai- e é assinado por alguém em quem Alice confia: Trent (comumente conhecido como uma autoridade certificadora, ou AC)..." (SCHNEIER, 1996, p. 163, com adaptações).

Em suma, o que o autor diz é que a forma mais simples de atestar a identidade de uma pessoa é utilizando um **certificado de chave pública**. Pensando numa situação onde aparecem os indivíduos **A**, **B**, e **C**, um certificado apresenta várias informações sobre um *indivíduo A*, e é assinado por um *indivíduo C* externo, para que o *indivíduo B* passe a confiar no *indivíduo A*. Para que esse certificado seja confiável e válido, é necessário que o *indivíduo A* e o *indivíduo B* confiem no *indivíduo C*. No caso o indivíduo C seria então conhecido como uma **Autoridade Certificadora**.

Para que esses certificados possam seguir um padrão, foram criados os **Public-Key Cryptography Standards**, ou *PKCS*, e dentre elas, especificamente a *PKCS #10*

(RSA, 2000), cujo objetivo é padronizar os dados contidos em *certificados digitais*, assim como definir quais tipos de informações devem estar contidas em um *certificado digital* e o modelo que deve ser seguido para se requisitar um. O certificado funciona da seguinte forma 2.2:

Figura 6 – Certificado de Chave Pública em Uso (STALLINGS, 2011, p. 430)

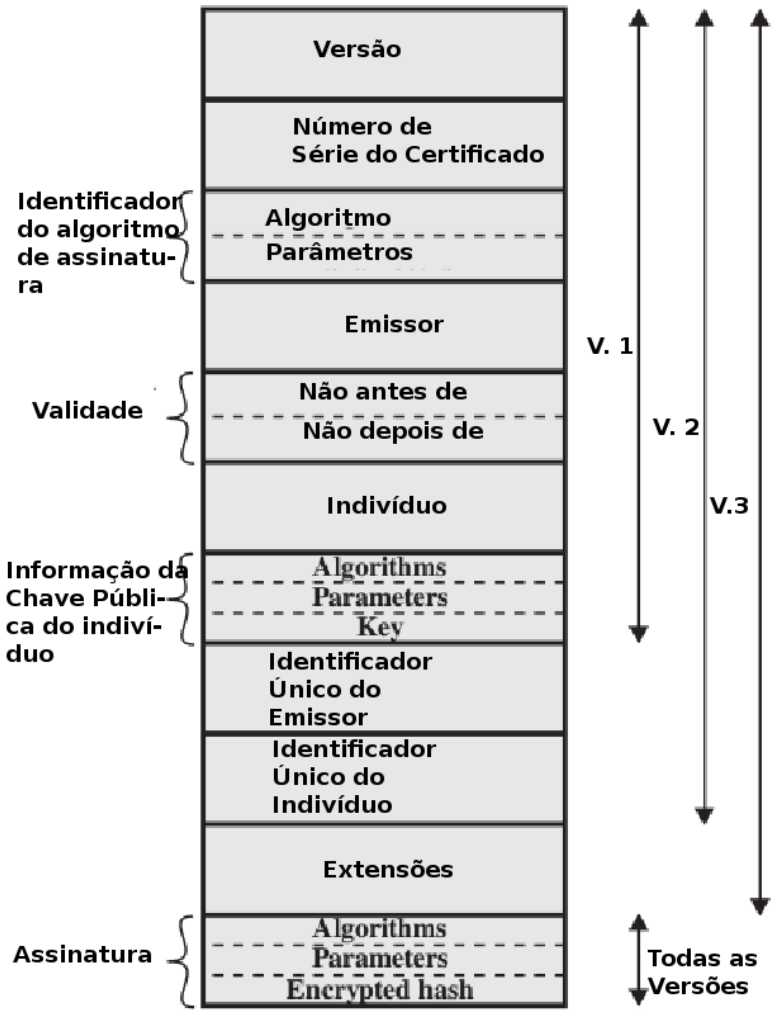


Esses certificados utilizam o padrão **X.509**. O *X.509* não dita um *cipher* específico, entretanto ele aconselha que seja utilizado o *RSA* para a implementação da *criptografia de chave pública* e das *assinaturas digitais*. Na figura abaixo se observa a estrutura de um certificado:

E esses certificados *X.509* podem ainda se mostrar de duas formas, em formato **DER** ou **PEM**. Quando se encontram em codificação *DER* (*Distinguished Encoding Rules*), esses certificados se apresentam em formato binário, legíveis apenas através dos devidos interpretadores por parte da máquina. Por outro lado, os *PEM* (*Privacy Enhanced Mail*) usam codificação **ASCII**, facilitando sua interpretação e leitura, assim como a distinção das informações contidas em seu corpo.

Como já foi dito, a ideia de um certificado é atestar a identidade dos indivíduos envolvidos numa transação de informações. Para isso, pode ser necessário que uma terceira parte se envolva, caso aqueles que se comunicam não conheçam os certificados um do outro de antemão, e esse terceiro envolvido seria uma autoridade certificadora, de confiança mútua entre os envolvidos. Essa terceira entidade, geralmente, conhecida como **Autoridade Certificadora**, é responsável por emitir e assinar os certificados digitais que circulam em determinados domínios. Por exemplo, no domínio brasileiro, temos a **AC Raiz**, como autoridade certificadora raiz, definindo quem pode ser visto como um agente certificador e emitindo certificados para tais indivíduos.

Figura 7 – Certificado X.509 (STALLINGS, 2011, p. 431)



Abaixo, podemos observar a estrutura simplificada da árvore de certificação brasileira.

Entretanto, esses certificados também podem apresentar erros, ou falhas, e comprometer a confiabilidade da troca de informações por diversos fatores, que podem acarretar em quebras de segurança, esses erros são explicitados através de verificações realizadas por ferramentas de geração e análise de certificados digitais (entre outras funções). Abaixo, a tabela de erros identificáveis através da ferrammenta **OpenSSL**.

Valor Associado	***Retorno***
0	*ok*
2	*unable to get issuer certificate*
3	*unable to get certificate CRL*
4	*unable to decrypt certificate's signature*
5	*unable to decrypt CRL's signature*

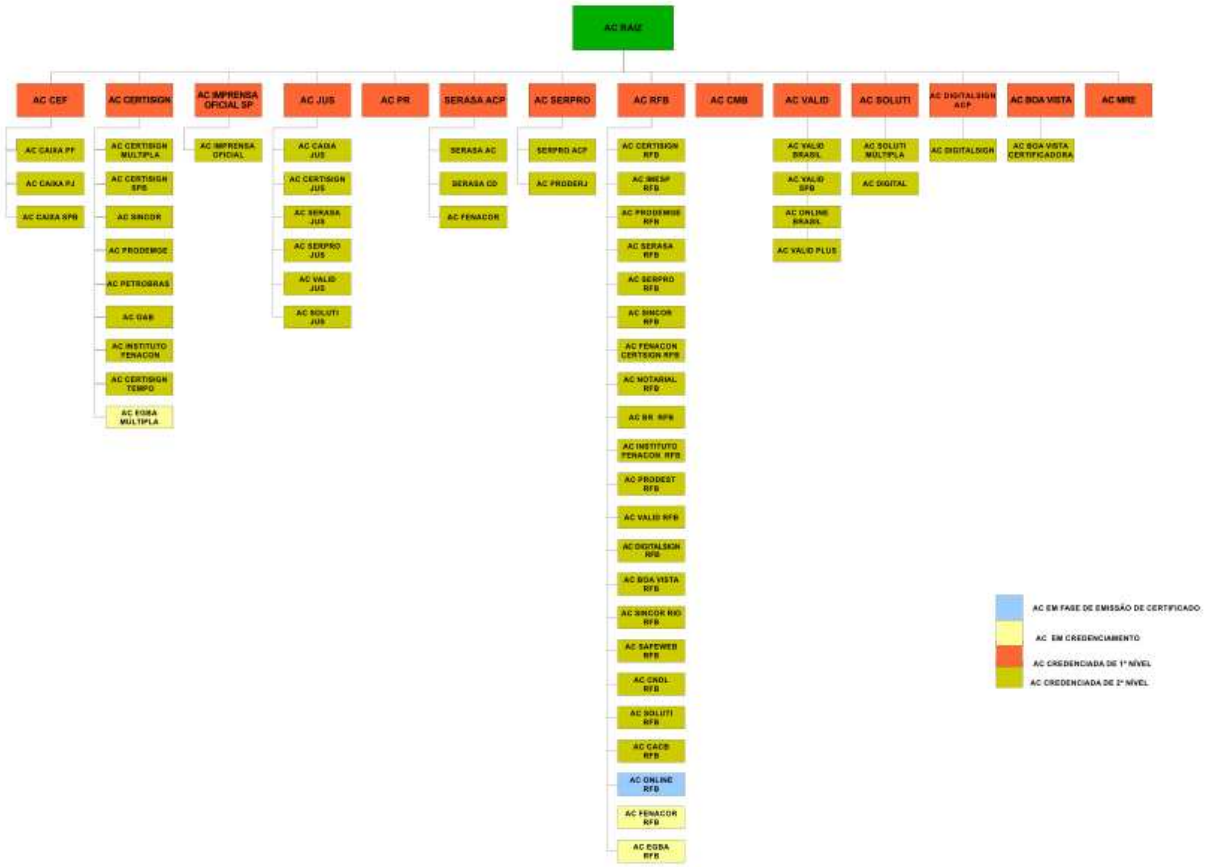
6	*unable to decode issuer public key*
7	*certificate signature failure*
8	*CRL signature failure*
9	*certificate is not yet valid*
10	*certificate has expired*
11	*CRL is not yet valid*
12	*CRL has expired*
13	*format error in certificate's notBefore field*
14	*format error in certificate's notAfter field*
15	*format error in CRL's lastUpdate field*
16	*format error in CRL's nextUpdate field*
17	*out of memory*
18	*self signed certificate*
19	*self signed certificate in certificate chain*
20	*unable to get local issuer certificate*
21	*unable to verify the first certificate*
22	*certificate chain too long*
23	*certificate revoked*
24	*invalid CA certificate*
25	*path length constraint exceeded*
26	*unsupported certificate purpose*
27	*certificate not trusted*
28	*certificate rejected*
29	*subject issuer mismatch*
30	*authority and subject key identifier mismatch*
31	*authority and issuer serial number mismatch*
32	*key usage does not include certificate signing*
50	*application verification failure*

Tabela 1 – Tabela de Retornos do Comando Verify

Cada um desses retornos (*com excessão do caso zero*) envolve uma vulnerabilidade de segurança, e se não tratados, certificados que apresentam essas falhas podem representar brechas de segurança. Notemos os Erros 02, 10, 18, e 20.

O retorno **02 - unable to get issuer certificate** caracteriza que um ou mais certificados da cadeia não puderam ser encontrados, ou acessados, e por essa razão, não é possível atestar que aquele certificado foi realmente emitido por aquela autoridade.

Figura 8 – Árvore Simplificada de Autoridades Certificadoras Brasileiras (ITI, 2015)



Já o retorno **10 - certificate has expired** determina que a data de validade do certificado já expirou, e por essa razão o certificado não deve ser considerado válido em datas posteriores.

No retorno **18 - self signed certificate** vemos um certificado auto-assinado. Essa situação deveria ser comum, caso o certificado em questão fosse emitido por uma autoridade certificadora do topo de sua cadeia de certificação; entretanto, essa situação não deve ocorrer em outros casos, pois não existe uma relação de confiança para atestar a identidade do indivíduo, a não ser que aquele certificado já fosse conhecido.

E o retorno **20 - unable to get local issuer certificate** mostra que ao recuperar os certificados, alguns dos certificados de emissores podem não coincidir com o que era esperado, ou não podem ser lidos. Por alguma razão, esses certiificados que deveriam ter sido providos/acessados não estão presentes, impedindo que a verificação siga adiante.

3 Metodologia

Neste capítulo o objetivo é apresentar as ferramentas e métodos que foram adotados com o objetivo de alcançar resultados significativos para o trabalho. Na primeira parte do capítulo serão apresentadas as ferramentas, e as razões pelas quais foram escolhidas certas tecnologias capazes de auxiliar e facilitar o alcance dos resultados. Em seguida, são apresentados os métodos e scripts gerados, de forma a automatizar e auxiliar o processo de análise dos certificados digitais obtidos.

3.1 Ferramentas

Para o desenvolvimento deste trabalho de avaliação dos certificados digitais, foi configurado um ambiente com as seguintes especificações apresentadas nas tabelas abaixo.

Componente	***HardwareEspecificação***
Processador	*Intel Core i3-3217U 1.8GHz*
Memória	*4,00 Gb*
Placa de Rede	*Realtek RTL8139/810x Fast Ethernet Adapter*

Tabela 2 – Tabela de Componentes de Hardware

Software	***Versão***
Sistema Operacional	*Ubuntu GNOME 12.04*
Python	*3.4.3*
Java VM	*OpenJDK 1.7*
OpenSSL	*1.0.0*
OpenOffice	*4.0.0*

Tabela 3 – Tabela de Softwares Utilizados

A conexão com a internet é um pré-requisito para a execução do trabalho, para que seja possível a busca de links, e o download dos certificados digitais utilizados em cada um dos domínios identificados, que empregam o protocolo HTTPS em suas vias de acesso.

3.2 Aquisição dos Certificados

Para a coleta dos certificados são utilizadas duas etapas bem definidas:

1. A coleta da URL de sites que apresentem o protocolo HTTPS;
2. O download dos certificados através de comunicação via porta 443;

Os certificados digitais identificados foram adquiridos através de comunicação com a porta 443, padrão utilizado no protocolo HTTPS para a requisição e comunicação de segurança. O processo de handshake foi ignorado durante a aquisição dos certificados, de modo a evitar que certificados que apresentam erros sejam descartados durante a coleta. Esta escolha permitiu que todos os certificados fossem avaliados da mesma forma, posteriormente, pelo OpenSSL.

Os passos definidos foram executados através de scripts. Todos os scripts desenvolvidos e/ou utilizados neste trabalho realizam tarefas bem definidas, e devem ser executados na ordem descrita para seu devido funcionamento. O primeiro script a ser executado é apresentado no Código 3.1.

Listing 3.1 – FunnyCrawler.java

```
import java.io.IOException;
import java.util.HashSet;
import java.util.Set;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;

public class FunnyCrawler {

    private static Pattern patternDomainName;
    private Matcher matcher;
    private static final String DOMAIN_NAME_PATTERN
    = "([a-zA-Z0-9]([a-zA-Z0-9\\-]{0,61}[a-zA-Z0-9])?\\.)+[a-
       zA-Z]{2,6}";
    static {
        patternDomainName = Pattern.compile(
           DOMAIN_NAME_PATTERN);
```

```
    }

    public static void main(String[] args) {

        FunnyCrawler obj = new FunnyCrawler();
        Set<String> result;
        for(int i = 0; i < 100; i++){
            result = obj.getDataFromGoogle("+https", i);
            for(String temp : result){
                System.out.println(temp);
            }
            System.out.println(result.size());
        }
    }

    public String getDomainName(String url){

        String domainName = "";
        matcher = patternDomainName.matcher(url);
        if (matcher.find()) {
            domainName = matcher.group(0).toLowerCase().trim
               ();
        }
        return domainName;

    }

    private Set<String> getDataFromGoogle(String query, int
       number) {

        Set<String> result = new HashSet<String>();
        //String request = "https://www.google.com.br/search?
           q=login+%2Bhttps&start=" + number + "0";
        //String request = "https://www.google.com.br/search?
           q=sign+in+%2Bhttps&start=" + number + "0";
        //String request = "https://www.google.com.br/search?
           q=register+%2Bhttps&start=" + number + "0";
        //String request = "https://www.google.com.br/search?
           q=%2Bhttps&start=" + number + "0";
```

```
52          String request = "https://www.google.com.br/search?q=
               buy%2Bhttps&start=" + number + "0";

54          System.out.println("Sending request... " + request);

56          try {

58              // need http protocol, set this as a Google bot
                   agent :)
                Document doc = Jsoup
60                      .connect(request)
                        .userAgent("Mozilla/5.0 (Macintosh; U;
                           Intel Mac OS X 10.4; en-US; rv
                           :1.9.2.2) Gecko/20100316 Firefox/3.6.2
                           ")
62                      .referrer("http://www.google.com")
                        .timeout(5000).get();
64
                // get all links
66              Elements links = doc.select("a[href]");
                for (Element link : links) {
68
                    String temp = link.attr("href");
70                  if(temp.startsWith("/url?q=")){
                        //use regex to get domain name
72                      result.add(getDomainName(temp));
                    }
74
                }
76
            } catch (IOException e) {
78              e.printStackTrace();
            }
80
            return result;
82      }

84  }
```

O 3.1 foi escrito a partir de um tutorial da ferramenta JSON, da plataforma Java, disponibilizado na internet (YONG, 2015). Sua função é analisar os resultados de uma pesquisa Google e retornar os links disponibilizados pela mesma na forma de um conjunto (SET). As seguintes strings de busca foram utilizadas:

- “https://www.google.com.br/search?q=login+%2Bhttps&start=”+ number + “0”;
- “https://www.google.com.br/search?q=sign+in+%2Bhttps&start=”+ number + “0”;
- “https://www.google.com.br/search?q=register+%2Bhttps&start=”+ number + “0”;
- “https://www.google.com.br/search?q=%2Bhttps&start=”+ number + “0”;
- “https://www.google.com.br/search?q=buy%2Bhttps&start=”+ number + “0”.

* - O parâmetro “number” é utilizado para a navegação em diferentes páginas, durante a busca.

Os retornos, no caso links para resultados das buscas, foram direcionados para um arquivo de texto (UTF-8), onde os links são colocados de forma que cada link ocupa uma única linha do arquivo.
“...
www.wpbeginner.com
www.wpwhitesecurity.com
stackoverflow.com
en.support.wordpress.com
security.stackexchange.com
github.com
webcache.googleusercontent.com
...”

Utilizando a lista de links que foi organizada, o script 3.2 inicia conexões via porta 443 utilizando o OpenSSL e recuperando os certificados, quando existentes, salvando localmente uma cópia .pem do certificado que foi tocado durante o processo de comunicação.

Listing 3.2 – DOWNLOAD.py

```
from subprocess import *
from glob import glob
#coding: utf-8

def download(url):
    httpsPort = str(url.replace("\n",":443"))
    resultado = str(url.replace("\n",".pem"))
```

```
8
      comandoCriarArquivo = ['touch', resultado]
10
      processoCriarArquivo = Popen(comandoCriarArquivo, stdout=
         PIPE)
12    [response, error] = processoCriarArquivo.communicate()
      processoCriarArquivo.wait()
14
      certificado = open(resultado, 'w')
16
      comando_a = ['openssl', 's_client', '-showcerts', '-
         connect', httpsPort]
18    comando_b = ['openssl', 'x509', '-outform', 'PEM']
      #openssl s_client -showcerts -connect matriculaweb.unb.br
         :443 | openssl x509 -outform PEM > mycert.pem
20
      process_a = Popen(comando_a, stdout=PIPE)
22    process_b = Popen(comando_b, stdin=process_a.stdout,
         stdout=certificado)
      [response, error] = process_b.communicate()
24    process_b.wait()

26    certificado.close()
      pass
28
  if __name__ == '__main__':
30    with open('links.lnk') as f:
          listaDeLinks = f.readlines()
32
      f.close()
34
      for link in listaDeLinks:
36        download(link)
```

Nesse momento, os certificados salvos localmente estão prontos para a análise posterior. Porém, os certificados não são baixados com todas sua cadeia de certificação, em alguns dos casos, e por conta disso o certificado digital do seu emissor, a CA, preciso ser copiado localmente, para evitar falso-positivos entre os resultados.

Para solucionar essa falta, se utiliza o seguinte script.

Listing 3.3 – LOAD_ROOT.py

```
from subprocess import *
from glob import glob

def download_root(certificado):
    comando_um = ['openssl', 'x509', '-in', certificado, '-
       text']

    process = Popen(comando_um, stdout=PIPE)
    [response, error] = process.communicate()
    process.wait()

    tokens = response.split()

    foundIssuers = False
    foundScore = False

    download_link = ""

    #procura o link do CA Issuers e joga em download_link
    for token in tokens:
        if foundIssuers and foundScore:
            download_link = str(token)
            foundIssuers = False
        if token == 'Issuers':
            foundIssuers = True
            foundScore = False
        if token == '-':
            foundScore = True

    ##tirando os primeiros quatro caracteres "URI:" e
       impurezas
    download_link = download_link[4:]
    #download_link = download_link.split(":")[-1]
    #download_link = 'http:' + download_link
    print download_link

    #nome do arquivo baixado
    nome_download = download_link.split("/")[-1]
```

```
        print nome_download
38
        comando_dois = ['wget', download_link]
40
        process = Popen(comando_dois, stdout=PIPE)
42      [response, error] = process.communicate()
        process.wait()
44
        #determinando o nome final do arquivo
46      na_root = str(certificado.replace(".crt", ""))
        na_root = na_root + '.root'
48      print na_root

50      #comando para passar de DER para PEM
        comando_tres = ['openssl', 'x509', '-in', nome_download,
           '-inform', 'der', '-out', na_root, '-outform', 'pem']
52
        process = Popen(comando_tres, stdout=PIPE)
54      [response, error] = process.communicate()
        process.wait()
56
        #comando para deletar o DER original
58      comando_quatro = ['rm', '-r', nome_download]

60      process = Popen(comando_quatro, stdout=PIPE)
        [response, error] = process.communicate()
62      process.wait()


64


66  if __name__ == '__main__':

68      files = glob("*.crt")

70      for certificado in files:
            download_root(certificado)
```

No 3.3, a função download_root() realiza todo o trabalho de acessar o conteúdo textual de um certificado, separá-lo em tokens, e a partir desses tokens extrair o endereço para

download do certificado do emissor. Uma vez com o certificado do emissor em mãos, esse certificado é tratado e passado do formato DER para o formato PEM, e esse novo arquivo PEM tem sua extensão definida, para fins de controle, como .root.

Uma vez recuperados os certificados dos emissores, foi executado o script apresentado no 3.4, com a finalidade de listar quais certificados possuem o certificado da CA armazenado localmente.

Listing 3.4 – LISTA.py

```
from subprocess import *
from glob import glob
# coding: utf-8

def listar():
    file = open('lista.dupla', 'w')

    temp_a = ""
    temp_b = ""
    output = ""

    CRT = glob("*.crt")
    ROOT = glob("*.root")

    match = False

    for temp_a in CRT:
        temp_a = str(temp_a.replace(".crt",""))
        match = False
        for temp_b in ROOT:
            temp_b = str(temp_b.replace(".root",""))
            if (temp_a == temp_b) and (match == False):
                dados = (temp_a +'.crt',temp_b + '.root')
                output = '%s,%s\n'%dados
                match = True
            if (match == False):
                dados = (temp_a +'.crt')
                output = '%s,-\n'%dados

        if match == True:
            file.write(output)
```

```
32          else:
                file.write(output)
34
        file.close()
36
if __name__ == '__main__':
38
        listar()
```

A função do script Código xyz é pegar todos os arquivos .pem presentes no mesmo diretório que ele e pesquisar se para eles existe um arquivo .root associado. O script então gera uma lista pareada de .pem's e .root's na qual, caso não haja um par, o .pem será pareado com um hífen (-), para identificação dos arquivos sem .root quando for feita a verificação dos certificados.

3.3 Validação dos Certificados

A OpenSSL foi amplamente utilizada no processo de aquisição e verificação dos certificados por permitir, através de suas funções, que a comunicação pela porta 443 fosse feita e que os processos de comunicação pela mesma fossem estabelecidos. Em um momento posterior, esta mesma biblioteca foi utilizada para permitir a verificação local dos certificados coletados.

A tabela abaixo apresenta os comandos OpenSSL utilizados para a verificação e validação dos certificados coletados.

Comando	***Descrição***
$ openssl	*O comando básico*
$ openssl verify certificado	*Dentro do OpenSSL*
$ openssl verify -CAfile root certificado	*A flag -CAfile é uti*
$ openssl x509 certificado	*Esse comando é uti*
$ openssl x509 -in certificado -text	*Abre um certificado*
$ openssl x509 -in certificado_der -inform der -out certificado_pem -outform pem	*Esse comando lê ur*

Tabela 4

Esses comandos, entretanto, não são utilizados diretamente via terminal, estando presentes no corpo dos scripts (quando se fazem necessários) responsáveis por sua execução

automatizada. A automatização do processo de verificação se fez necessária pelo tamanho do corpo de certificados que foram coletados para a verificação.

Para a validação dos certificados, foi executado o script listado no Código xyz.

Listing 3.5 – SEPARAR.py

```
from subprocess import *
from glob import glob

def erros():
    erros = {0 : 'ok',
        1 : 'ok',
        2 : 'unable to get issuer certificate',
        3 : 'unable to get certificate CRL',
        4 : 'unable to decrypt certificate\'s signature',
        5 : 'unable to decrypt CRL\'s signature',
        6 : 'unable to decode issuer public key',
        7 : 'certificate signature failure',
        8 : 'CRL signature failure',
        9 : 'certificate is not yet valid',
        10 : 'certificate has expired',
        11 : 'CRL is not yet valid',
        12 : 'CRL has expired',
        13 : 'format error in certificate\'s notBefore field'
           ,
        14 : 'format error in certificate\'s notAfter field',
        15 : 'format error in CRL\'s lastUpdate field',
        16 : 'format error in CRL\'s nextUpdate field',
        17 : 'out of memory',
        18 : 'self signed certificate',
        19 : 'self signed certificate in certificate chain',
        20 : 'unable to get local issuer certificate',
        21 : 'unable to verify the first certificate',
        22 : 'certificate chain too long',
        23 : 'certificate revoked',
        24 : 'invalid CA certificate',
        25 : 'path length constraint exceeded',
        26 : 'unsupported certificate purpose',
        27 : 'certificate not trusted',
        28 : 'certificate rejected',
```

```
34          29 : 'subject issuer mismatch',
            30 : 'authority and subject key identifier mismatch',
36          31 : 'authority and issuer serial number mismatch',
            32 : 'key usage does not include certificate signing'
               ,
38          50 : 'application verification failure'}
    return erros
40
def find_subject(certificado):
42      comando_um = ['openssl', 'x509', '-in', certificado, '-
            text']

44      process = Popen(comando_um, stdout=PIPE)
        [response, error] = process.communicate()
46      process.wait()

48      tokens = response.split('O=')

50      token = ""
        token = tokens[2]
52      micro = token.split(',')
        subject = micro[0]
54      sub = subject.split('/')
        return str(sub[0])
56
def find_issuer(certificado):
58      comando_um = ['openssl', 'x509', '-in', certificado, '-
            text']

60      process = Popen(comando_um, stdout=PIPE)
        [response, error] = process.communicate()
62      process.wait()

64      tokens = response.split('O=')

66      token = ""
        token = tokens[1]
68      micro = token.split(',')
        issuer = micro[0]
```

```
70      iss = issuer.split('/')
        return str(iss[0])
72
    def separador_c():
74      certificados = []
        c = ""
76      arquivo = open('lista.dupla','r')
        for line in arquivo:
78          c = str(line.split(',')[0])
            certificados.append(str(c))
80      return certificados

82  def separador_r():
        roots = []
84      r = ""
        arquivo = open('lista.dupla','r')
86      for line in arquivo:
            r = str(line.split(',')[1])
88          roots.append(str(r.replace("\n","")))
        return roots
90
    def verifica_c(certificado):
92      args = ['openssl', 'verify', certificado]

94      process = Popen(args, stdout=PIPE)
        [response, error] = process.communicate()
96      process.wait()

98      tokens = response.split()

100     issuer = find_issuer(certificado)
        company = '-'
102
        foundError = False
104     errorCode = 1

106     for token in tokens:
            if foundError:
108             errorCode = int(token)
```

```
            foundError = False

        if token == 'error':
            foundError = True

    error_list = erros()

    message = ""
    message = error_list[int(errorCode)]

    dados = (errorCode, message, certificado, issuer)

    return dados

def verifica_rc(certificado, root):
    args = ['openssl', 'verify', '-CAfile', root, certificado
        ]

    process = Popen(args, stdout=PIPE)
    [response, error] = process.communicate()
    process.wait()

    issuer = find_issuer(certificado)
    company = '-'

    tokens = response.split()

    foundError = False
    errorCode = 1

    for token in tokens:
        if foundError:
            errorCode = int(token)
            foundError = False

        if token == 'error':
            foundError = True

    error_list = erros()
```

```
148     message = ""
        message = error_list[int(errorCode)]
150
        if errorCode == 1:
152         errorCode = 0

154     dados = (errorCode, message, certificado, issuer)

156     return dados

158 if __name__ == '__main__':
        certificado = separador_c()
160     root = separador_r()

162     for c, r in zip(certificado, root):
            if (r == "-"):
164             dados = verifica_c(c)
                print '%d,%s,%s,%s' % dados
166         else:
                dados = verifica_rc(c, r)
168             print '%d,%s,%s,%s' % dados
```

O 3.5 foi então o responsável pela verificação dos certificados. Primeiro ele gera getou dois vetores pareados usando o arquivo gerado pelo 3.4 e cuja ordem é foid a mesma da lista gerada por tal script. Uma vez determinados, esses vetores pareados foram verificados utilizando comandos OpenSSL listados abaixo.

Comando	***Descrição***
$ cmdVerify	*$ openssl verify certificado.crt*
$ cmdCAverify	*$ openssl verify -CAfile certificado.root certificado.crt*

Tabela 5 – Tabela de Comandos OpenSSL na Verificação

Através do último script (3.5) foi possível gerar um arquivo .csv (comma separated value) com o nome de cada certificado e os resultados de suas avaliações automatizadas. Cada resultado corresponde a um único erro relacionado ao certificado, acusado pela verificação

do OpenSSL e extraído automaticamente.

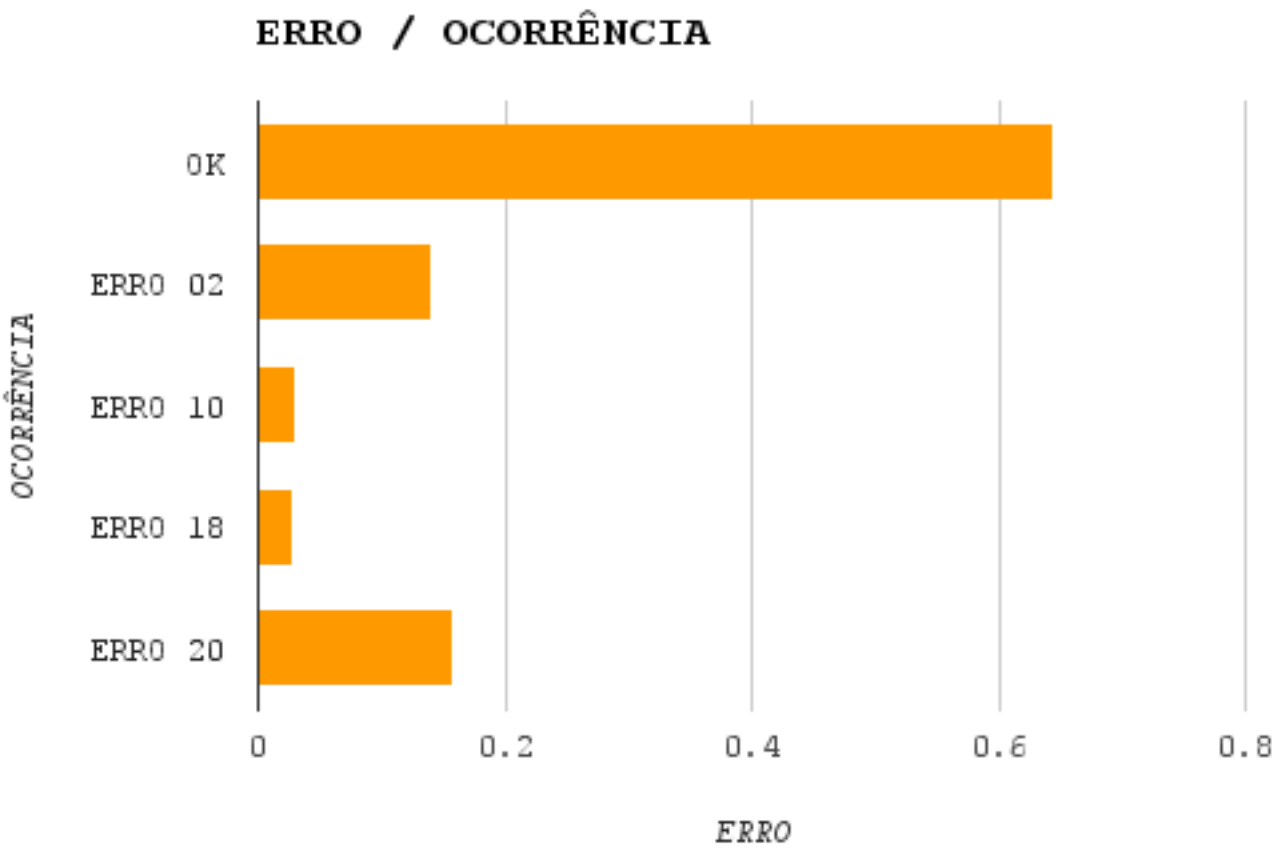
O resultado do script do Código xyz apresenta„ além do número do erro e de informações relativas ao certificado, qual a natureza do erro apontado, de acordo com a lista de retornos do comando verify do OpenSSL, que podem ser um resultado de verificação positiva ou um dentre os trinta e dois erros previstos de serem encontrados. Estes erros são listados e descritos no Anexo XYZ.

No próprio arquivo .csv resultante foi utilizada a função SUMIF(), nativa da ferramenta OpenOffice Math, que adiciona termos ao somatório quando estes atendem a condição especificada dentro de seus parâmetros.

4 Resultados

Através da verificação automática feita utilizando a ferramenta OpenSSL, em conjunto com *scrypts*, foram alcançados resultados iniciais. Esses resultados retornaram uma tabela, cujos valores incluem se as verificações encontraram algum erro e qual foi o erro encontrado, além de identificarem em quais certificados foram encontrados cada um dos erros.

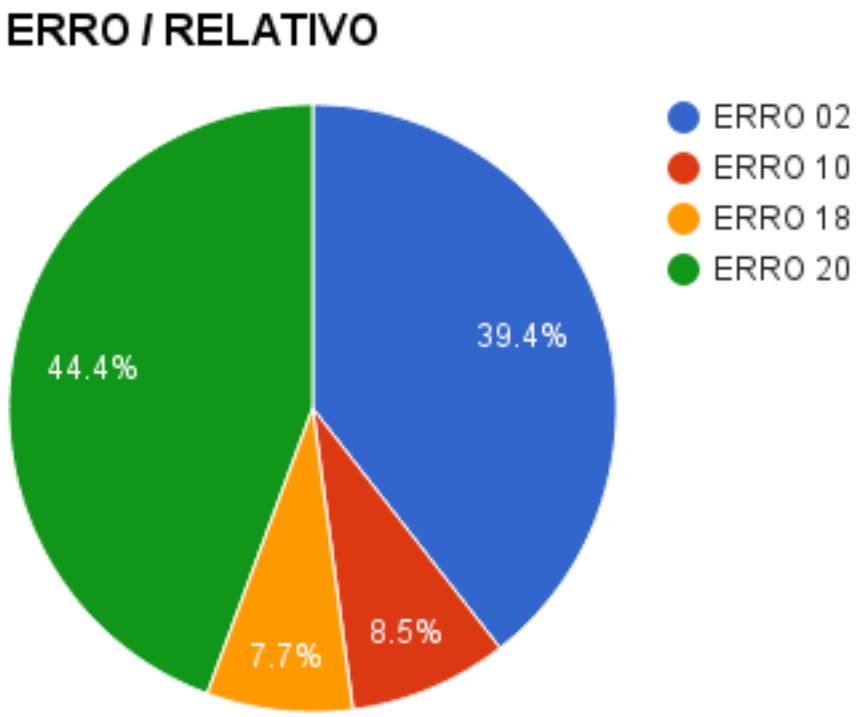
Figura 9 – Taxa de Ocorrência dos Erros



Os números com que os erros se repetiram foram um dos dados extraídos da tabela, que na etapa inicial deste trabalho é baseada na verificação de apenas 799 (setecentos e noventa e nove) certificados digitais. Vale notar, também, que destes certificados verificados, foram considerados certificados de pessoa jurídica em sites de qualquer domínio.

Os erros constatados se apresentaram na seguinte proporção:

Figura 10 – Taxa de Erros Relativos



4.1 Sem Erro

Os certificados que não apresentaram erros na verificação representam aproximadamente 64,45% (sessenta e quatro por cento) dos certificados verificados no primeiro momento do trabalho. Esses certificados estão, de acordo com a verificação feita pela ferramenta OpenSSL, de acordo com as exigências dos PKCSs que adotam o padrão x509 como formato final.

4.2 Erro 02 - Unable to get issuer Cetificate

Na documentação da ferramenta OpenSSL, o erro 02 (dois) identifica que a verificação retornou a mensagem: Incapaz de conseguir o certificado do emissor. Esse erro se apresenta quando a cadeia de certificados não está completa na listagem disponível no certificado analisado. O erro 02 (dois) se apresentou em aproximadamente 14% (quatorze por cento) dos certificados verificados.

4.3 Erro 10 - Certificate has Expired

Na documentação da ferramenta OpenSSL, o erro 10 (dez) identifica que a verificação retornou a mensagem: O certificado expirou. Esse erro se apresenta se, e somente se, o certificado estiver sendo utilizado além de sua validade original. Um certificado digital tem um período de validade especificado no momento em que ele é emitido, e uma vez que esse prazo se cumpriu, ele deixa de ser reconhecido como válido. O erro 10 (dez) se apresentou em aproximadamente 3% (três por cento) dos certificados verificados.

4.4 Erro 18 - Self Signed Certificate

Na documentação da ferramenta OpenSSL, o erro 18 (dezoito) identifica que a verificação retornou a mensagem: Certificado auto-assinado. Neste caso em específico, duas condições foram encontradas irregulares no certificado. A primeira e mais evidente é que o alvo da certificação e o emissor do certificado são a mesma entidade; a segunda que se apresenta é que o certificado não se encontra na lista de certificados confiáveis, o que significa que ele não é uma autoridade certificadora raiz. O erro 18 (dezoito) se apresentou em aproximadamente 2,75% (dois vírgula setenta e cinco por cento) dos certificados verificados.

4.5 Erro 20 - Unable to get Local Issuer Certificate

Na documentação da ferramenta OpenSSL, o erro 20 (vinte) identifica que a verificação retornou a mensagem: Incapaz de acessar o emissor local do certificado. Um erro que ocorre quando o certificado do emissor não pôde ser encontrado. Nesse caso o certificado do emissor é inicialmente procurado através da referência presente no certificado em análise, e quando o acesso não oferece um retorno positivo, a ferramente verificadora procura o certificado localmente, analisando o certificado do emissor passado na verificação, retornando esse erro caso ainda haja incoerências na verificação do certificado. O erro 20 (vinte) representa 15,75% (quinze vírgula setenta e cinco por cento) dos certificados verificados.

5 Conclusão

Os resultados apresentam um número notável de exceções de segurança, sendo disparadas pela verificação realizada pelo *OpenSSL*. Estas, encontradas em ambiente controlado de estudo, mostram que existem falhas em certificados obtidos de um ambiente real, um dado preocupante, visto que todos os certificados apresentados deveriam ser seguros. Essas exceções podem se tornar um risco de falhas maiores de segurança, em ambiente real, resultando em riscos na comunicação, que supostamente deveria ser segura.

Um número percentual de exceções pode ser notado no corpo de certificados estudados, entretanto não é possível afirmar uma única causa. Foi observado que um grande número de certificados apresentaram o *Erro 20* ao serem analisados e isso serve como um forte indicador de que a origem dos problemas pode estar na *cadeia de assinaturas digitais* que se faz presente em todo certificado digital. Essa cadeia, que deveria ser confiável, pode não estar funcionando de forma ideal, e suas razões devem ser investigadas.

Outras *exceções* foram retornadas em menor número durante as verificações, o que pode indicar erros isolados. A ocorrência desses erros também é preocupante, pois essas exceções também podem acarretar em problemas de segurança, que podem se caracterizar como falhas dependendo de como forem explorados. Isso indica que as exceções de segurança existem, e podem ser encontradas em certificados em uso, um fator alarmante, uma vez que o uso de certificados digitais é cada vez mais um recurso explorado como forma de aumentar a segurança nas transações realizadas por vias de comunicação digitais.

Referências

CRYPTO, M. *https://en.wikipedia.org/wiki/FILE:DES-main-network.png*. [S.l.]: Matt Crypto, 2005. Citado 2 vezes nas páginas 15 e 29.

ITI. *www.iti.gov.br/icp-brasil/estrutura*. [S.l.]: Instituto Nacional de Tecnologia da Informação, 2015. Citado 2 vezes nas páginas 15 e 34.

RSA, L. Pkcs #8 private key information syntax standard. Redwood City, CA, 1993. Citado na página 29.

RSA, L. Pkcs #10 certification request syntax standard. Bedford, MA, 2000. Citado na página 31.

RSA, L. Pkcs #1 rsa cryptography standard. Cambridge, MA, 2012. Citado 2 vezes nas páginas 29 e 30.

SCHNEIER, B. *Applied cryptography protocols, algorithms, and source code in C*. [S.l.]: Wiley, 1996. ISBN 9780471128458. Citado 3 vezes nas páginas 25, 26 e 30.

STALLINGS, W. *Cryptography and Network Security: Principles and Practice*. [S.l.]: Prentice Hall, 2011. ISBN 9780136097044. Citado 6 vezes nas páginas 15, 26, 29, 30, 31 e 32.

YONG, M. K. *http://www.mkyong.com/java/jsoup-send-search-query-to-google/*. [S.l.]: mkyong, 2015. Citado na página 39.

Anexos