HTTPS com TLS em Modo RSA

Rudá Moura
2016

Agenda

- Introdução
- SSL/TLS
- Certificados
- Protocolo HTTPS
- FPS
- Conclusão

Introdução

- HTTPS = HTTP Secure = HTTP no topo de SSL/TLS. Porta TCP 443. URI https://…

- Idéia da Netscape (dos anos 90) para prover segurança ao canal de comunicação do HTTP.

- Tecnologias: asymmetric Public Key Infrastructure (PKI) system = Chaves Pública/Privada + Infraestrutura de Certificados + Algoritmos de Criptografia.

SSL/TLS

- O SSL foi desenvolvido nos anos 90 para possibilitar transações bancárias e comerciais na Web.

- Autores: Netscape, MasterCard, Bank of America, MDI & Silicon Graphics.

- ~~SSLv1~~, SSLv2 (1994), SSLv3 (1999). IETF TLS (2001).

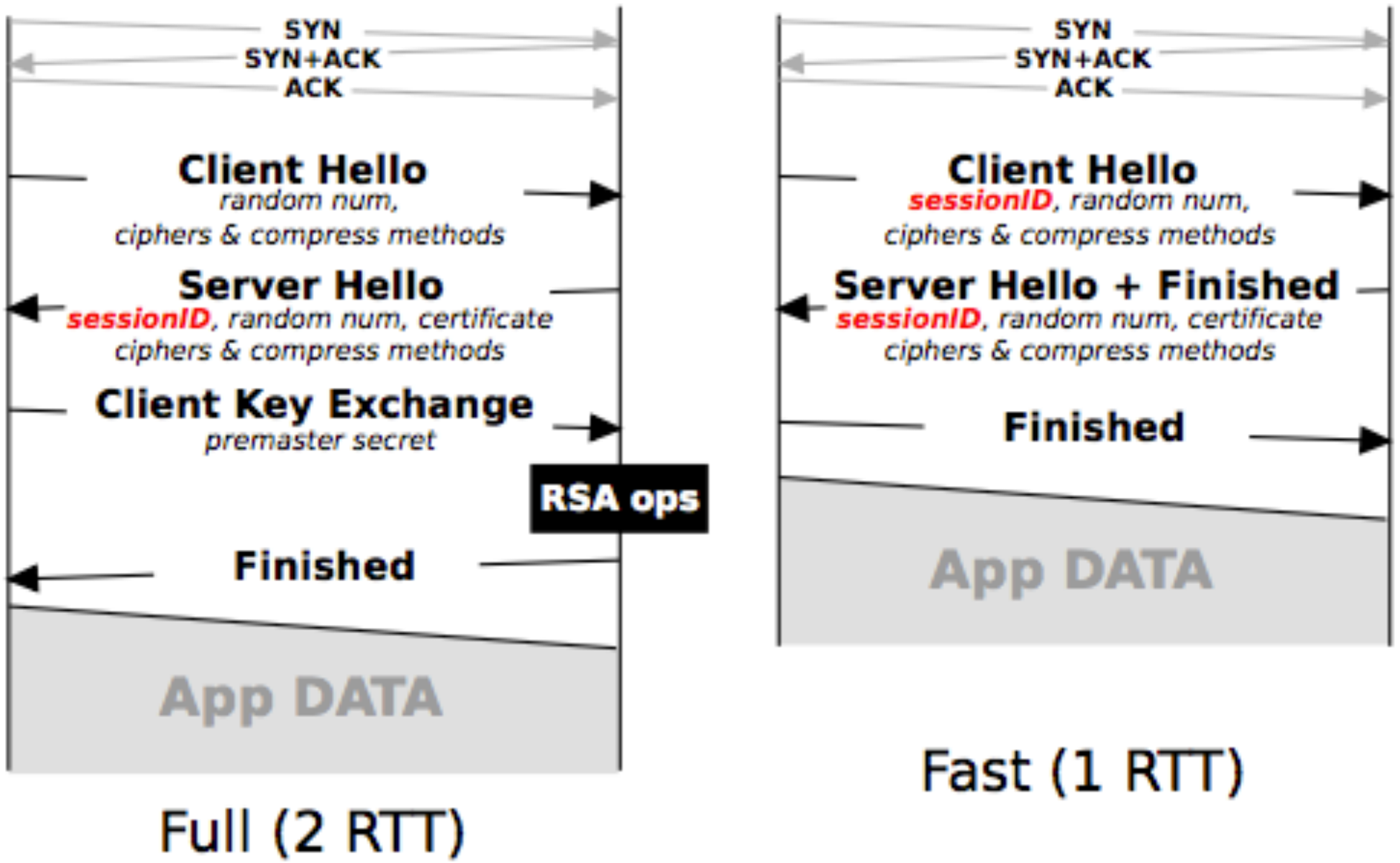
HTTPS: Inicialização

- Cliente (HTTP/TLS) abre conexão com o Servidor e envia o *ClientHello* para iniciar o handshake do TLS. Negocia cifras e compressão.

- Servidor oferece SessionID, Cifras certificadas e métodos de compressão.

- Cliente inicia a troca de chaves.

- Quando o handshake terminar, o cliente pode enviar o request HTTP como *application data* do TLS.



Fonte: The Cost of the “S” in HTTPS

Certificados

- Server Name (identidade).
- Certificate Authority (quem assinou).
- Public Key (do Servidor).
- SSL warnings, certificados expirados e certificados auto-assinados.

HTTPS: Identificação

- Uma URI conhecida é utilizada para iniciar o HTTPS e um Certificado é recebido. O que fazer com o certificado é problema do cliente.

- O cliente procura por um subjectAltName extension of type dNSName. Caso contrário, e utilizado o campo Common Name. Se usar IP, então iPAddress subjectAltName deve estar presente.

- Cliente faz o match da identidade, valida o certificado com a assinatura da certificadora.

- Nota: Servidor não se importa com a identidade do cliente.

Website	Internet Explorer	Google Chrome	Firefox	Safari	Opera
www.facebook.com	RC4-SHA	RC4-SHA	RC4-SHA	RC4-SHA	RC4-SHA
www.twitter.com	RC4-SHA	RC4-SHA	RC4-SHA	RC4-SHA	RC4-SHA
www.yahoo.com	AES128-SHA	CAMELLIA256-SHA	CAMELLIA256-SHA	AES128-SHA	AES256-SHA
www.google.com	**ECDHE-RSA-AES128-SHA**	**ECDHE-RSA-RC4-SHA**	**ECDHE-RSA-RC4-SHA**	**ECDHE-RSA-RC4-SHA**	RC4-SHA
login.live.com	AES128-SHA	AES128-SHA	AES128-SHA	AES128-SHA	AES128-SHA
www.aol.com	RC4-SHA	RC4-SHA	RC4-SHA	RC4-SHA	RC4-SHA
www.apple.com	AES256-SHA	AES256-SHA	AES256-SHA	AES256-SHA	AES256-SHA
commerce.paltalk.com	RC4-SHA	RC4-SHA	RC4-SHA	RC4-SHA	RC4-SHA

Fonte: https://news.netcraft.com/archives/2013/06/25/ssl-intercepted-today-decrypted-tomorrow.html

PFS

- Problema: Pode-se guardar o tráfego SSL de uma comunicação, com a esperança (no futuro) de se obter a chave privada e ler os dados contidos.

- There is a defence against this, known as perfect forward secrecy (PFS). When PFS is used, the compromise of an SSL site's private key does not necessarily reveal the secrets of past private communication; **connections to SSL sites which use PFS have a per-session key which is not revealed if the long-term private key is compromised**. The security of PFS depends on both parties discarding the shared secret after the transaction is complete (or after a reasonable period to allow for session resumption).

- Perfect forward secrecy was invented in 1992, pre-dating the SSL protocol by two years, and consequently one might reasonably have expected that SSL would have made operational use of PFS from the outset. Nevertheless, almost twenty years later, PFS usage is not used by the majority of SSL sites.

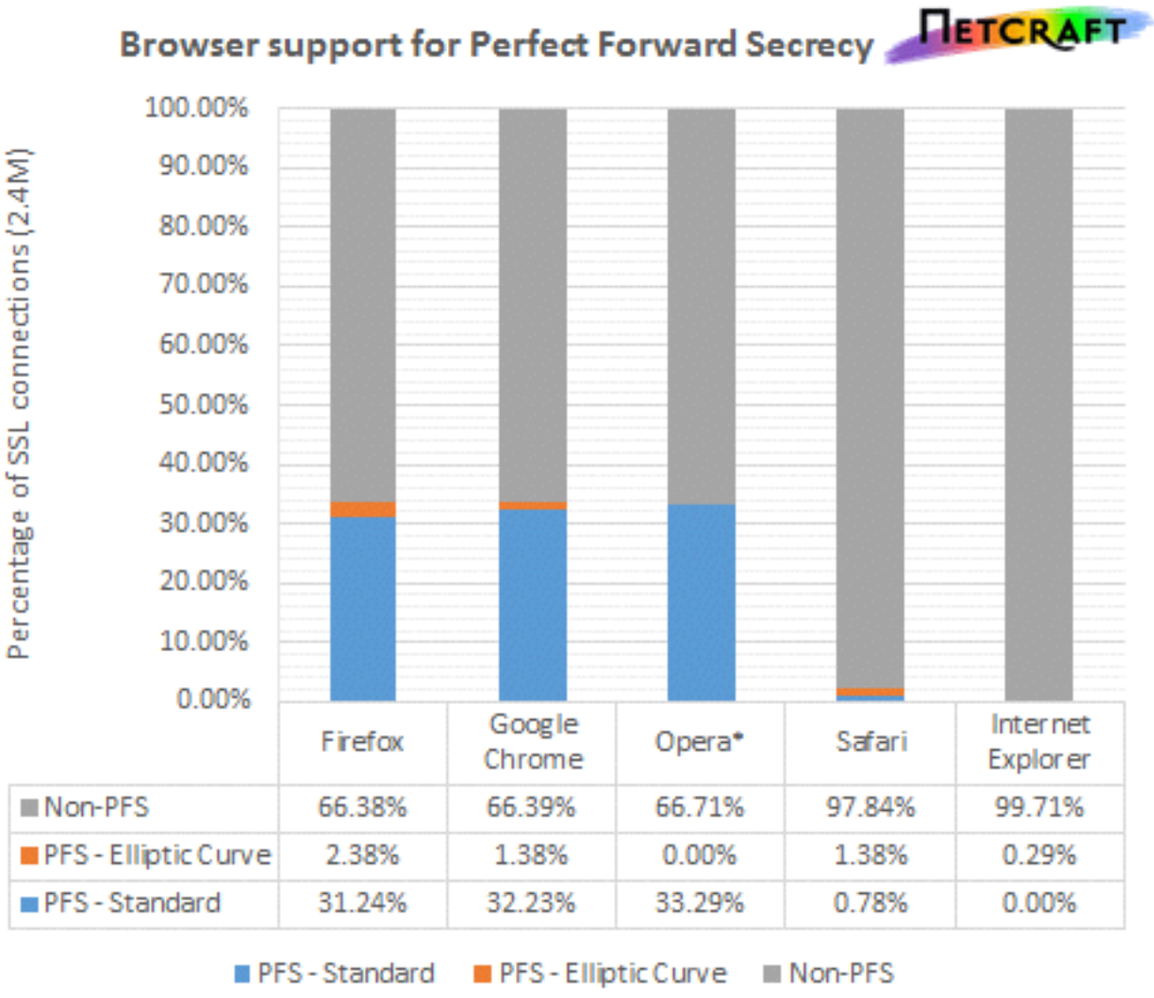
PFS (cont.)

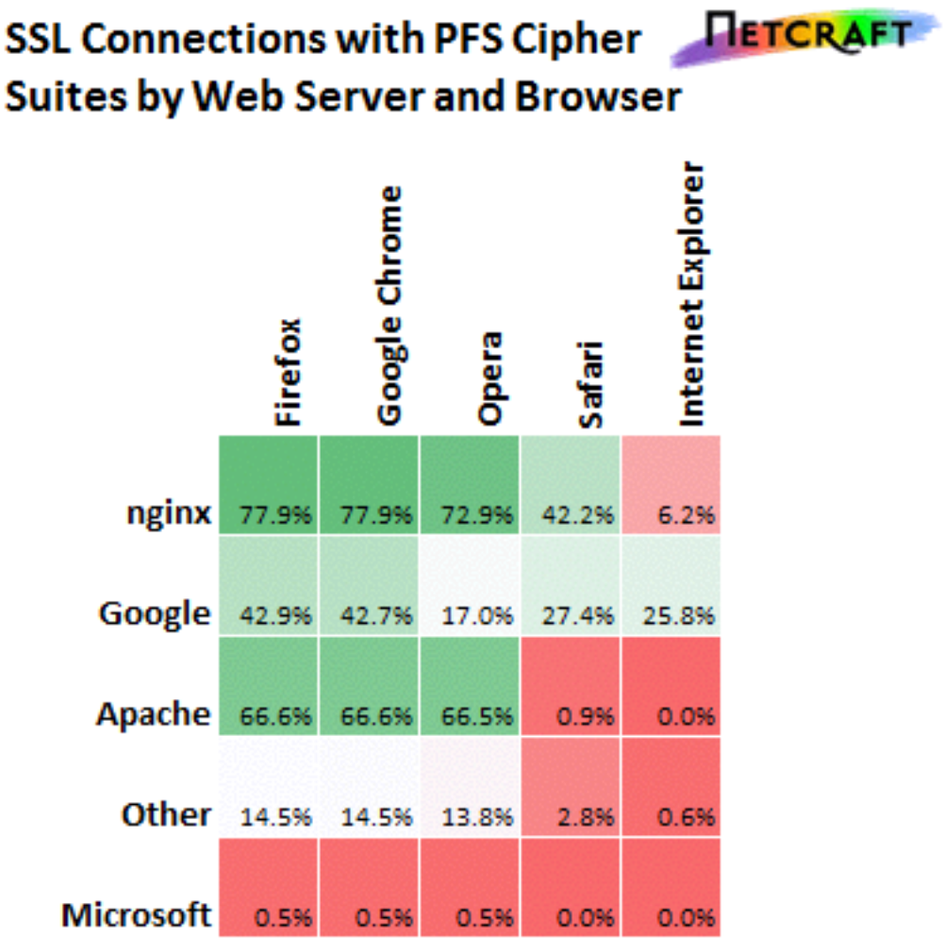
- The cipher suite selected for the SSL connection depends on an agreement between the browser and the SSL site.

- Diffie-Hellman key exchange and variants of it are used to negotiate a per-session shared secret key between two parties without ever transmitting the key itself.

- The per-session key can be discarded after the session has terminated leading to the ephemeral property which PFS relies upon.

- Variants = Elliptic curve-based DHE key exchange despite being faster is supported by fewer SSL sites than conventional DHE.





Fonte: https://news.netcraft.com/archives/2013/06/25/ssl-intercepted-today-decrypted-tomorrow.html

Handshake

1 Client Hello

2 Server Hello,
Certificate,
Server Hello Done

3 Client Key Exchange,
Change Cipher Spec,
Finished

4 Change Cipher Spec,
Finished

5 **Application Data**

6 **Application Data**

ClearText, **CipherText**

Client Hello

- **Version** versão do protocolo que o cliente quer falar. Ex.: TLS 1.0 (0x0301).
- **Random** contém 4 bytes que formam uma data UTC (GMT Unix Time) + 28 bytes aleatórios.
- **Session ID** é um token que pode ser utilizado para um *handshake* mais rápido. Aqui ele é *null* (não existente previamente).
- **Cipher Suites** é a lista de conjuntos de algoritmos de criptografia que o cliente suportar. Ex.: *TLS_RSA_WITH_RC4_128_MD5, TLS_RSA_WITH_AES_128_CBC_SHA, TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA.*
- **Compression Methods** é a lista de algoritmos de compressão que o cliente suporta. Pode ser *null* (nenhum).
- **Extensions** são características adicionais, por exemplo, *server_name* é o nome do servidor que estamos querendo alcançar. Outros: *Extended Master Secret*, *elliptic_curvers*, *signature_algorithms, etc.*

Server Hello

- **Version** versão do protocolo que servidor concordou. Ex.: TLS 1.0 (0x0301).
- **Random** contém 4 bytes que formam uma data UTC (GMT Unix Time) + 28 bytes aleatórios.
- **Session ID** é um token (32 bytes) que o cliente pode utilizar para um *handshake* rápido.
- **Cipher Suite** é o conjunto de algoritmos de criptografia que o servidor escolheu, da lista oferecida pelo cliente. Ex.: *TLS_RSA_WITH_RC4_128_MD5*
- **Compression Method** é o método de compressão que o servidor escolheu, da lista que o cliente enviou. Pode ser *null* (nenhum).
- **Extensions** são características adicionais que o servidor concordou em usar, da lista que o cliente ofereceu.

Server Certificate + Done

- **Certificates** contém a lista de certificados que o servidor apresenta para o cliente. Ex.: Certificados X.509v3.

- Cada certificado deve conter uma chave. Ex.: RSA (RSA Public Key), DH_RSA (Diffie-Hellman key), etc.

- **Certificate Status** contém a resposta do *Online Certificate Status Protocol* (OCSP).

- **Server Hello Done** marca o fim do Server Hello.

Client Key Exchange

- O cliente seleciona entre *RSA* ou *Diffie Hellman* e envia uma chave como parâmetro.

- No caso de RSA, o cliente deve criar uma *Pre-Master Secret (PMS),* cifrar (usando a chave pública RSA) e enviar para o servidor, para que ambos conheçam.

- Cliente e Servidor derivam a *Master Secret* a partir da Pre-Master Secret.

- NOTA: *Forward Secrecy* só é possível com *Diffie Hellman*.

Pre-Master Secret

- A *PMS* contém 48 bytes: 2 bytes da versão do protocolo TLS (ex.: 0x03, 0x01) + 46 bytes aleatórios.

- Este número é gerado pelo cliente e é cifrado utilizando a chave pública do certificado do servidor ou a chave temporária provida em troca de mensagem.

- A *PMS* **cifrada** é enviada ao servidor, que utiliza a chave privada para conhecer a PMS e derivar outras chaves.

- O PKCS 1.5 pede que a *PMS* deva conter preenchimentos (pads) aleatórios do tamanho exato do módulo (1024 bits/ 128 bytes).

Master Secret

- **master_secret** = PRF(pre_master_secret, “master secret”, ClientHello.random+ServerHello.random) com 48 bytes.

- **PRF** Função geradora de números pseudo-aleatórios, definida no padrão TLS.

- **pre_master_secret** = 2 bytes da versão TLS do cliente + 46 bytes aleatórios gerados.

- “**master secret**” é uma string literal, em que os bytes são utilizados pela PRF.

- Cliente e Servidor conhecem a Master Secret.

Pseudo Random Function

- **PRF**(secret,label,seed) = P_MD5(*S1*,label+seed) ⊕ P_SHA-1(*S2*,label+seed).

- *S1* contém a primeira metade de *secret.*

- *S2* contém a segunda metade de *secret*.

Key calculation

- De **key_block**=PRF(master_secret, “key expansion”,server_random+client_random) derivam-se:
  - client_write_MAC_secret[hash_size]
  - server_write_MAC_secret[hash_size]
  - client_write_key[key_material_length]
  - server_write_key[key_material_length]
  - client_write_IV[IV_size]
  - server_write_IV[IV_size]

Change Cipher Spec

- É uma mensagem simples que o cliente e o servidor trocam entre si, para sinalizar a mudança de estratégias de cifras.

- Ambos concordaram em utilizar um método de criptografia simétrico, um HMAC e as respectivas chaves.

Finished

- O cliente guarda todas as mensagens que trocou até então em *handshake_messages*:
- Calcula-se PRF(master_secret, "client finished", MD5(handshake_messages)+SHA-1(handshake_messages)).
- Utiliza-se os 12 primeiros bytes resultantes em *verify_data*. De *verify_data* tira-se o hash dele, com algoritmo de hash negociado (Ex.: HMAC_MD5).
- Todos esses bytes (verify_data+hash) são cifrados, utilizado o algoritmo simétrico negociado (Ex.: RC4) e enviados para o servidor.
- O Servidor efetua as mesmas operações, *inclusive utiliza a mensagem decifrada de Finished do cliente* como sua mensagem de *Finished*.

Conclusão

- HTTPS oferece ótima segurança ao canal de comunicação HTTP. Utiliza tecnologias de criptografia que hoje são comuns.

- Desafio: Como otimizá-lo sem perder a segurança? Pode ser vulnerável a análise de tráfego, MITM e comprometimento de chaves.

- Obstáculos: Complexidade e custo de certificados. Clientes com SSL obsoleto (inseguro) e algoritmos de criptografia fracos.

Referências

- HTTPS: [https://tools.ietf.org/html/rfc2818]; [https://www.cs.cmu.edu/~dnaylor/CostOfTheS.pdf]

- SSL/TLS: [https://wiki.wireshark.org/SSL]

- Traffic Analysis: [https://www.petsymposium.org/2014/papers/Miller.pdf]

- Certificados: [https://www.eff.org/deeplinks/2014/11/certificate-authority-encrypt-entire-web]

- PFS: [https://news.netcraft.com/archives/2013/06/25/ssl-intercepted-today-decrypted-tomorrow.html]

- Links foram visitados em Setembro de 2016.

Referências

- The First Few Milliseconds of an HTTPS Connection. http://www.moserware.com/2009/06/first-few-milliseconds-of-https.html.

- Wireshark:SSL. https://wiki.wireshark.org/SSL.

- The Transport Layer Security (TLS) Protocol. Version 1.2. https://tools.ietf.org/html/rfc5246.

- Debugging SSL/TLS Connections. https://docs.oracle.com/javase/7/docs/technotes/guides/security/jsse/ReadDebug.html.

- Links visitados em Setembro de 2016.