Segurança Informática nas Organizações
LEI

Projeto 3: Autenticação

Vasco Marieiro 81332
Diego Trovisco 80228

Introdução:

Este trabalho tem como principal objetivo a exploração de conceitos relacionados com com o estabelecimento de uma comunicação entre dois interlocutores, como por exemplo autenticação e controlo de acesso dos clientes. Tem como objetivo também facilitar a perceção de como os certificados e assinaturas digitais podem ser utilizados nessa mesma comunicação.
Assim, no final deste trabalho, estamos certos de que estaremos mais preparados para trabalhar com ferramentas que explorem essas mesma tecnologias e processos.

O nosso trabalho permite 2 métodos de autenticação.

Através de OTP (One Time Password) e através do cartão de cidadão.
OTP ou senha de utilização única é uma senha que é válida somente uma única vez numa determinada situação, por exemplo, um início de sessão. Este método evita um grande leque de problemas existentes nas senhas tradicionais uma vez que, mesmo que um atacante consiga descobrir a senha utilizada, não poderá ser reutilizada já que esta é descartável.
No nosso caso, para implementarmos OTP começamos por criar um ficheiro (pass.txt) que contém uma key gerada aleatoriamente. Esse mesmo ficheiro tem que estar acessível tanto do lado do cliente como do lado do servidor.

```python
def otp():
    f = open('pass.txt', 'rb')
    key = f.read()
    #print(key)
    totp = TOTP(key, 8, SHA1(), 30, backend=default_backend())
    time_value = time.time()
    totp_value = totp.generate(time_value)
    return totp_value

#OTP lado CLIENTE
```

```python
def otp(w):
    f = open('pass.txt', 'rb')
    key = f.read()
    totp = TOTP(key, 8, SHA1(), 30, backend=default_backend())
    time_value = time.time()
    totp_value = totp.generate(time_value)
    print(totp_value)
    return totp_value==w

#OTP lado SERVIDOR
```

Podemos ver que é gerada uma chave de cada lado tendo em conta, não só o conteúdo do ficheiro como o momento. Assim é sempre garantido a aleatoriedade da password. No lado do servidor, após receber o argumento w (que é a a password gerada no cliente), verifica a igualdade com a palavra chave gerada do seu lado.

```python
aux ='1'
while aux !='0':
    print("Tipo de autenticacao")
    self.aut = input ("Introduza a autenticacao: ")
    if self.aut=="CC":
        w=otp_client.otp()
        self._send({"type":"CC", "val":base64.b64encode(w).decode()})
        aux='0'
    elif self.aut=="OTP":
        w=otp_client.otp()
        self._send({"type":"OTP", "val":base64.b64encode(w).decode()})
        aux='0'
```

```python
def fun_cc(self, message: str) -> bool:
    self.val=base64.b64decode(message["val"])
    s=otp_server.otp(self.val)
    if s is True:
        message={"type":"challenge", "random":self.str}
        self._send(message)
        return True
    else:
        self.transport.close()


def fun_otp(self, message: str) -> bool:
    self.val=base64.b64decode(message["val"])
    print(self.val)
    s=otp_server.otp(self.val)
    if s is True:
        message={"type":'Challenge OK'}
        self._send(message)
        return True
    else:
        print("invalido")
        self.transport.close()
```

Em cima do lado esquerdo é mostrado o lado do cliente, onde após ser selecionado o método de autenticação OTP, do lado do servidor (à direita) vai receber o valor e compará-lo com o gerado do seu lado (como explicado anteriormente)

Se a senha do servidor for igual à senha do cliente, é permitida a comunicação e a troca de ficheiro. Neste caso a mensagem que vai do cliente para o servidor não vai assinada uma vez que já se pressupõe, depois das senhas serem iguais, que é seguro.

Quando é utilizado o CC, o processo OTP também é utilizado uma vez que não compromete a ação. É enviado um challenge tanto do servidor para o cliente (que vai assinado com um certificado previamente distribuído e presente em ambas as máquinas), como do cliente para o servidor (assinado pelo cc), isto para termos a certeza de que as comunicações são entre os intervenientes corretos.

```
elif mtype=="challenge":

    t=message["random"]
    t=sign.sign(t)
    c=cert.accessCert()
    self._send({"type":"challenge",'challengeAss': base64.b64encode(t).decode(), "certificado":base64.b64encode(c).decode()})
    self.chall=True
elif mtype == 'Challenge OK':
    self.chall=True
elif mtype == 'ERROR':
    logger.warning("Got error from server: {}".format(message.get('data', None)))
else:
    logger.warning("Invalid message type")
```

Challenge assinado do lado do cliente: é assinado o challenge e enviado o mesmo para o servidor.
Função SIGN:

```
def sign (text):
    lib = '/usr/local/lib/libpteidpkcs11.so'
    pkcs11 = PyKCS11.PyKCS11Lib()
    pkcs11.load(lib)
    slots = pkcs11.getSlotList()
    for slot in slots:
        print(pkcs11.getTokenInfo(slot))


    all_attr = list(PyKCS11.CKA.keys())
#Filter attributes
    all_attr = [e for e in all_attr if isinstance(e, int)]
    session = pkcs11.openSession(slot)
    for obj in session.findObjects():
# Get object attributes
        attr = session.getAttributeValue(obj, all_attr)
# Create dictionary with attributes
        attr = dict(zip(map(PyKCS11.CKA.get, all_attr), attr))
        print('Label: ', attr['CKA_LABEL'])


    private_key = session.findObjects([(PyKCS11.CKA_CLASS, PyKCS11.CKO_PRIVATE_KEY),(PyKCS11.CKA_LABEL, 'CITIZEN AUTHENTICATION KEY')])[0]
    mechanism = PyKCS11.Mechanism(PyKCS11.CKM_SHA1_RSA_PKCS, None)
    signature = bytes(session.sign(private_key, text, mechanism))
    return signature
```

Função criada para assinar com o cartão de cidadão.
Função accessCert - utilizada para aceder ao certificado:

```python
def accessCert():

    lib = '/usr/local/lib/libpteidpkcs11.so'
    pkcs11 = PyKCS11.PyKCS11Lib()
    pkcs11.load(lib)
    slots = pkcs11.getSlotList()
    public_key = None
    private_key = None
    for slot in slots:

        all_attr = list(PyKCS11.CKA.keys())
        all_attr = [e for e in all_attr if isinstance(e, int)]
        session = pkcs11.openSession(slot)
        for obj in session.findObjects():
            attr = session.getAttributeValue(obj, all_attr)
            attr = dict(zip(map(PyKCS11.CKA.get, all_attr), attr))
            if attr['CKA_CLASS'] is not None and attr['CKA_CERTIFICATE_TYPE'] is not None:
                ca = bytes(attr['CKA_VALUE'])
                ca = x509.load_der_x509_certificate(ca, default_backend())
                test = ca.extensions.get_extension_for_oid(ExtensionOID.BASIC_CONSTRAINTS)
                key_usage = ca.extensions.get_extension_for_oid(ExtensionOID.KEY_USAGE)
                if key_usage.value.digital_signature is True:
                    print(ca)
                    ca=ca.public_bytes(encoding=serialization.Encoding.DER)
                    #z=verificaEC(ca)
                    #print(z)
                    return ca
```

Do lado do servidor é recebido o challenge assinado e vai ser verificada a assinatura (se realmente correspondem a quem se espera ou não):

```python
def data_received(self, data: bytes) -> None:
    if self.state == STATE_OPEN and self.chall==True:
        print(data)
        sign=data[-256:]
        y=cert.verificaEC(self.cert)
        if y== True:
            data = data[:len(data)-256]
            print(sign)
            if self.tipo=="AES":
                data=aes.decrypt(data, self.b, self.h, self.private_key)
            if self.tipo=="CHACHA20":
                data=chacha20.decrypt(data, self.h, self.private_key)
            print(data)
        else:
            print("Nao confiavel")
    elif self.state == STATE_OPEN and self.chall==False:
        pass
    elif self.state ==STATE_DATA:
        print(data)
        if self.tipo=="AES":
            data=aes.decrypt(data, self.b, self.h, self.private_key)
        if self.tipo=="CHACHA20":
            data=chacha20.decrypt(data, self.h, self.private_key)
```

Função verificaEC: vai verificar o certificado do CC, se este é válido ou não.

```
def verificaEC(ca):
    try:
        if ca.not_valid_before < datetime.datetime.utcnow() < ca.not_valid_after:
            cn = ca.issuer.get_attributes_for_oid(NameOID.COMMON_NAME)
            issuerid = cn[0].value[-4:]
            c= open("/home/user/Desktop/teste/guiao06/PTEID/pem/EC de Aute ticacao do Cartao de Cidadao "+issuerid+".pem", "rb")
            print(c)
            issuer_public_key=x509.load_pem_x509_certificate(c.read(), default_backend())
            #print(issuer_public_key)
            cert_to_check = ca
            issuer_public_key.public_key().verify(cert_to_check.signature,cert_to_check.tbs_certificate_bytes,padding.PKCS1v15(),cert_to_check.signature_ha
    except InvalidSignature:
        print("falhou")
    print("passou")
    y=verificaCA(issuer_public_key)
    return y

def verificaCA(ca):
    try:
        if ca.not_valid_before < datetime.datetime.utcnow() < ca.not_valid_after:
            cn = ca.issuer.get_attributes_for_oid(NameOID.COMMON_NAME)
            issuerid = cn[0].value[-3:]
            c= open("/home/user/Desktop/teste/guiao06/PTEID/pem/Cartao de Cidadao "+issuerid+".pem", "rb")
            print(c)
            issuer_public_key=x509.load_pem_x509_certificate(c.read(), default_backend())
            #print(issuer_public_key)
            cert_to_check = ca
            issuer_public_key.public_key().verify(cert_to_check.signature,cert_to_check.tbs_certificate_bytes,padding.PKCS1v15(),cert_to_check.signature_ha
    except InvalidSignature:
        print("falhou")
    print("passou")
    x=verificaADO(issuer_public_key)
    return x

def verificaADO(ca):
    try:
        if ca.not_valid_before < datetime.datetime.utcnow() < ca.not_valid_after:
            c= open("/home/user/Desktop/teste/guiao06/PTEID/pem/ecraizestado.pem", "rb")
            print(c)
            issuer_public_key=x509.load_pem_x509_certificate(c.read(), default_backend())
            #print(issuer_public_key)
            cert_to_check = ca
            issuer_public_key.public_key().verify(cert_to_check.signature,cert_to_check.tbs_certificate_bytes,padding.PKCS1v15(),cert_to_check.signature_ha
    except InvalidSignature:
        print("falhou")
    print("passou")
    return True
```

Do lado do servidor todas as mensagens vão assinadas:

```
def _send(self, message: str) -> None:
    """
    Effectively encodes and sends a message
    :param message:
    :return:
    """
    print(message)

    logger.debug("Send: {}".format(message))

    message_b = (json.dumps(message) + '\r\n').encode()

    if self.state==STATE_CONNECT:
        print("Assinar")
        message_b+=self.sign_message(message_b)
    self.transport.write(message_b)
```

```
def sign_message(self, message):
    private_key=rsa.load_pem("private_key.pem","vasco")
    signature = private_key.sign(message,padding.PSS(mgf=padding.MGF1(hashes.SHA256()),salt_length=padding.PSS.MAX_LENGTH),hashes.SHA256())
    return signature
```

Quando a mensagem é recebida do lado do cliente, a assinatura irá ser verificada pela função verfi:

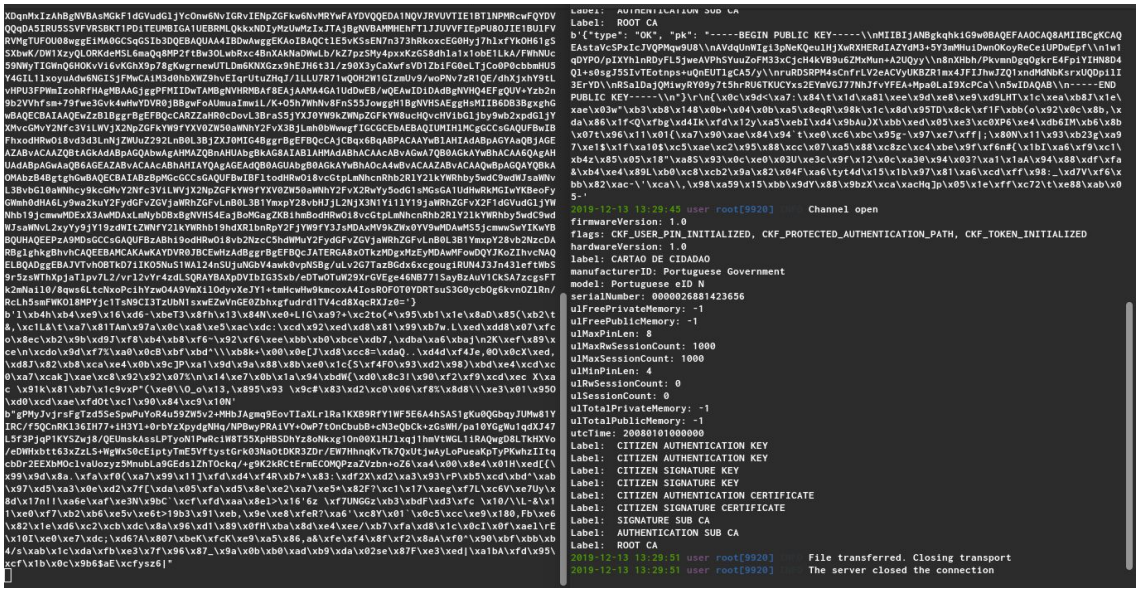
```python
def verfi(self, sign, message):
    with open("certificate.pem", "rb") as f:
        t=x509.load_pem_x509_certificate(f.read(),default_backend())

    if t.not_valid_before < datetime.datetime.utcnow() < t.not_valid_after:
        try:
            t.public_key().verify(t.signature, t.tbs_certificate_bytes, padding.PKCS1v15(),t.signature_hash_algorithm)
        except InvalidSignature:
            return False
    else:
        return False
    try:
        t.public_key().verify(sign,message,padding.PSS(mgf=padding.MGF1(hashes.SHA256()),salt_length=padding.PSS.MAX_LENGTH),hashes.SHA256())
    except: InvalidSignature
    return True
```

O certificado utilizado para assinar as mensagens do servidor é previamente distribuído e é verificada a sua validade.

Após a verificação da identidade, e se correr tudo conforme as verificações, a ligação não é interrompida e a mensagem do cliente para o servidor pode ser enviada seguramente, conforme o tipo de cifra escolhida.

Workflow de todo o processo



Servidor Cliente

Conclusão

A realização deste trabalho permitiu-nos colocar em prática todo o conhecimento adquirido durantes as aulas teóricas e práticas. Permitiu-nos perceber melhor como é utilizada a assinatura digital bem como funciona o controlo de acesso entre dois interlocutores.
Foi um trabalho desafiante, contudo são estes mesmos desafios que nos vão permitir ser melhores profissionais da área, bem como melhores pessoas.