Computer Systems and Networks Security
2019/2020, 1st SEMESTER

Work-Assignment 1 (v0.9)

Summary

In this work assignment the goal is the development of a secure protocol for a real-time streaming system, allowing that movies encoded in a given format will be transmitted by a server to users, using secure datagram/connectionless channels, that can be unicast, as well as, multicast channels, by configuration. We designate this protocol as SSP (or SECURE STREAMING PROTOCOL).

SSP, regarded as a designed secure streaming channel will provide confidentiality, integrity and authentication properties, protecting the dissemination of media contents from an adversary model and attack-types, as defined in the X.800 security framework. In the system model the streaming server sends the media streams supported by the SSP protocol to a client-proxy, using secure unicast or multicast addressing. The proxy receives the real-time streaming transmissions carrying the protected and encoded datagram packets, delivering the decrypted content as datagram payloads (with mpeg media contents), allowing standard client-media player tool to be used for real time playing. The VLC tool will be used as the reference tool to play the received real-time streams delivered by the proxy.

Complementarily, the development will involve the design and implementation of SHP – Secure Handshake Protocol - an authentication and secure protocol to support client-authentication and access control for requesting contents for dissemination and for the establishment of session keys and secure association parameters, that will be used dynamically by the SSP protocol.

Following the above description, the assignment consists of two complementary deliverables for a total of 20 points:

- *A mandatory deliverable (graded out of 14) consisting in the implementation of the SSP Protocol requirements*
- *An optional delivery (graded out of 6) consisting in the implementation of the SHP protocol requirements,*

In the remaining of the document the SSP and SHP protocols are proposed to be implemented in two phases, as a development methodology: Phase 1 - for the initial approach of the SSP implementation (mandatory) and Phase 2 – in which SSP and SHP will be integrated. In this way students can conclude first the Phase 1 (as a specific mandatory delivery) and Phase 2 (as a complementary and optional delivery).

1. Introduction

The following materials are provided as initial or starting materials for the Work Assignment

- **hjStreamServer.java**: An application supporting the dissemination of media streaming by PtP or Multicast Datagrams Packets, carrying media contents (internally encoded in MPEG) as payloads of UDP datagrams. The server can use destination UNICAST or MULTICAST addresses and UDP ports to send the contents (movies), according to the real-time temporal conditions for remote playing.
- **Movies:** there are two encoded movies provided for streaming tests: **Cars.dat** and **Monsters.dat.**

- **hjUDPProxy.java**: a proxy implementation for UDP proxying, supporting unicast or multicast addressing to receive streaming from the Streaming Server. The proxy uses locally a configuration file: **config.properties**, where addresses and ports (for the reception and for local delivery for final players) can be defined.
- **VLC Tool:** the VLC tool to play the streamed movies is downloadable from **https://www.videolan.org/index.pt.html**

As a starting point, students must test the provided materials, that are ready to support the streaming service, sending the provided movies directly to the VLC tool or indirectly, to the **hjUDPProxy,** which will in turn send to the VLC tool. The dissemination is supported in a "cleartext" channel, with all the frames of the movies sent in clear to the destination.

2. Implementation of Phase 1 – SSP Protocol (MANDATORY)

2.1 Implementation of mySSPStreamServer and mySSPUDPProxy

For the phase 1, you must implement:

- **mySSPStreamServer.java:** implementation developed from the initial hjStreamServer.java, supporting the secure dissemination of the media contents (movies) using SSP – Secure Streaming Protocol. For this objective you must design and implement SSPSockets – a extension of DatagramSockets, implementing the SSP Protocol.
- **mySSPUDPProxy.java:** implementation developed from the initial hjUDPProxy.java, supporting the secure reception of contents streamed from myStreamServer supported by the SSP protocol and delivered as "cleartext" based streams to be played by the VLC Tool. The implementation of myUDPProxy must support delivery endpoints using UDP Datagram Sockets or Multicast Sockets, according to the configuration of the config.properties file, initially used by the provided implementation.

Your implementations must be provided as executable jar files (with all the required classes required to be executed) mySSPStreamServer must be ready to run with the following specification (in shell-command line):

```
$java -jar mySSPStreamServer.jar <movie> <ip> <port> -f <file>
```

<movie> is the movie for streaming that must be in a subdirectory named movies where the mySSPStreamServer runs
The two provided movies can be used: monsters.dat and cars.dat.
<ip> is an IPaddress (that can be a unicast or multicast IP address) where mySSPUDPProxy is waiting for receiving
<port> is the port (UDP port) where mySSPUDPProxy is waiting for receiving
<file> is a configuration file (text file) for the SSP protocol

Examples:

```
$java -jar mySSPStreamServer.jar cars.dat 224.2.2.2 9000 -f SSP_S1.conf
```

myUDPProxy.jar must be ready to run in the following way (shell command-line):

```
$java -jar mySSPUDPProxy.jar -f <file>
```

mySSPUDPProxy.jar must use a file ***config.properties***, as provided in the initial implementation.
<ssp.conf> is the configuration file for the SSP protocol in the **myUDPProxy** endpoint

Example of execution:

```
$java -jar mySSPUDPProxy.jar -f SSP_P1.conf
```

For demonstrations, mySSPStreamServer.jar and muSSPUDPProxy.jar must be able for the use of different configurations in SSP configuration files.

<**SSP_Pi.conf**> <**SSP_Si.conf**> for i= 1,2,3,… etc., for the use or tests with different configurations.

The **ssp** configuration files are text files with the following contents and syntax. For the interpretation of this configuration see the SSP specification in Annex 1.

```
CRYPTO-CIPHERSUITE: <ALG/MODE/PADDING>    // Ciphersuite to be used
MAC1-CIPHERSUITE: <mac1 ciphersuite>      // Mac 1 Ciphersuite
MAC2-CIPHERSUITE: <mac2 ciphersuite>      // Mac 2 Ciphersuiote
IV: <iv> or NULL                          // IV for ciphersuite, depending fro the MODE
SESSION-KEYSIZE: <integer>    // The Session Key Size in bytes
SESSION-KEY: <session-key>    // Key bytes (represented in BASE64 or HEXADECIMAL)
MAC1-KEYSIZE: <integer>       // The Mac1 key size in bytes
MAC1-KEY: <key for MAC 1>     // Mac1 Key (Km1), represented in BASE64 or HEXADECIMAL)
MAC2-KEYSIZE: <integer>       // The Mac2 key size in bytes
MAC2-KEY: <key for MAC 2>     // Mac2 Key (Km2), represented in BASE64 or HEXADECIMAL)
```

Note: All the secrets (keys could be stored in different entries of a protected (encrypted) keystore JCEKS-storetype, protected by a keystroke password and different passwords for the entries. Another solution could be to protect/encrypt the keys of the configuration file with a password-based scheme. However, we suggest that you must have all in the configuration file not protected/encrypted, to facilitate the possibility of quickly defining different cryptographic configurations including different crypto ciphersuites, including MACs (by defining the proper keys, keysizes and IVs, for the purpose of testing the generality and configurability of your implementation.

2.2 SSP specification and implementation

The SSP specification is in the ANNEX 1.

The SSP implementation must be approached by implementing SSPSockets, a generic abstraction implementing the SSP protocol and supporting its configurations (in files SSP.conf) provided for each endpoint.

You are free to design and implement SSPSockets, considering these sockets as extensions of UDPSockets that must be able to be used for unicast or multicast UDP communication. By using your SSPSockets the need for code modification in mySSPStreamingServer or mySSPUDPProxy compared with the original implementations hjSTreamingServer and hjUDPProxy will be minimal. The smaller these changes, the more valuable our implementation will be for evaluation purposes.

3. Implementation of Phase 2 – Final solution integrating SSP and SHP protocols (OPTIONAL)

3.1 Implementation of myFinalStreamingServer and myFinalUDPProcy

For this phase you will implement the following componentsm that must integrate the SSP and SHP Protocols.

- **myFinalStreamingServer.jar**
- **myFinalUDPProxy.jar**

myFinalStreamingServer and myFinalUDPProxy must be ready to run with the following specifications, very similar as the executions in Phase 1 (in shell-command lines):

```
$java mySSPStreamServer <movie> <ip> <port> -f <file>
```

Example:

```
$java myFinalStreamingServer monsters.dat 224.2.2.2 9000 -f server_F1.conf
```

$java mySSPUDPProxy -f <file>

Example:

```
$java mySSPUDPProxy -f proxy_F1.conf
```

As alternative (optional) you can provide your implementations for the streaming server and proxy, as docker containers. In this case you must maintain the possibility to use the configuration files as arguments for the docker execution.

3.2 SHP specification and implementation

The SHP specification and its integration with SSP is in the ANNEX 2.

The SHP implementation must be approached by implementing an additional method for SSPSockets, that encapsulates the SHP protocol and supporting its configurations (in te provided configuration files for each endpoint). Then, the SHP protocol must run when the Proxy calls:

SSPSockets.SHPstarthandshake (<required arguments>)

When the StreamingServer starts, it waits initially to receive the SHP handshake, as specified, and after the handshake phase supported by SHP, the Proxy and the StreamingServer will establish all the keys and needed security associations to continue, supporting the streaming as in Phase 1, supported by SSP.

You must note that all the required parameterizations will be proided by the configuration files (following their specifications). In the directory CONFIGS (PHASE 2) of the provided materials, different configuration files for phase 2 are initially provided together with a generic description of their contents.

It is also relevant to note that the integration of SHP with the SSP protocol only requires the implementation of the SHP, subjacent to the method SHPstarthandshake( ). Then, once again, the differences in your code between the original implementations (hjStreamingServer and hjUDPProxy) or your implementations in Phase 1 and the final implementations in Phase 2 must be minimal, because all the required support for the SHP and SSP protocols will be implemented by SSHSockets( ).

4. Submission of your solution and evaluation criteria

The submission of Phases 1 and 2 will be done by using a SUBMISSION FORM to be announced in a CLIP Message sent to students. The submission form and the dates for the submission are defined (together with the specifications) in:

http://vps726303.ovh.net/srsc2021

See: Work Assignment 1 (Tab WAs).

Indicative reference of evaluation criteria for the Implementation of PHASE 1:

Goals	**Marks (for a total of 14/20 points)**
Correct implementation and operation of **mySSPStreamingServer** and **mySSPUDPProxy**, following all the specifications and supporting media streaming (provided movies) and playing (VLC) correctly and real time conditions: (No Interruptions, No Cuts) as in the initial unprotected implementations	**Until 8**
Minimal changes (Number of different code lines and different code) between the provided implementations for hjStreamingServer and hjUDPProxy and the implemented solutions supporting SSP (with the SSPSOckets)	**Until 2**
Correct operation with the different configurations in the provided configuration files for testing (supporting and passing all the configurations)	**Until 2**
Generic quality of the implementation	**Until 2**

Indicative reference of evaluation criteria for the Implementation of PHASE 2:

Goals	**Marks (for a total of 20/20 points)**
Correct implementation and operation of **myFinalStreamingServer** and **myFinalUDPProxy**, following all the specifications.	**Until 14**
Minimal changes (Number of different code lines and different code) between the provided implementations for hjStreamingServer and hjUDPProxy and the implemented solutions supporting SSP (with the SSPSOckets)	**Until 2**
Correct operation with the different configurations in the provided configuration files for testing (supporting and passing all the configurations)	**Until 2**
Generic quality of the implementation	**Until 2**

Indicative Penalizations:

A penalization of 1 point per day of delay will be considered for work deliverables and submissions after the defined limit-date for submission.

Annex 1

SSP Specification

SSP MESSAGE:

SSP-HEADER || SSP-PAYLOAD

SSP-HEADER = VERSION-INFO || CONTENT-TYPE || PAYLOAD_TYPE || PAYLOAD_SIZE

VERSION-INFO: 2 bytes (8 bits SSP VERSION + 8 bits SHP VERION)

Ex: 1.1 : 0x0101 (hexadecinal) means SSP version 1, SHP version 1

CONTENT-TYPE: 1 byte : 0x01 for SSP (that only have one only content type)

PAYLOAD_TYPE: 1 byte (encoding a standard value):

0x01 : means that the payload corresponds to payload-type 1: a SSP Payload

0x02 : means that the payload corresponds to payload-time 2: a SHP Payload (implemented in Phase 2)

Note) Given the flexibility to define content-types and payload-types, the protocol can be easily extended to support different contents (payloads) and payload-types.

PAYLOAD_SIZE: 4 bytes (or integer): the size (in bytes) of the Payload

Note) The HEADER has a fixed size: 11 Bytes but the payload in SSP can have variable sizes.

SSP-PAYLOAD: $E(K_S, [\ Mp \ || \ MAC1_{Km1}(Mp)\ ]\ ) \ || \ MAC2_{Km2}(C)$

The SSP payload corresponds to a protected message (carrying a media frame) that must be processed according to the payload-type

For the requirements of Phase 1, you must implement only a Payload Type 0x01, that will have the following format for protected messages:

In the SSP-PAYLOAD, the components are:

Mp = [id || nonce || M] // nonce can e implemented as a sequence number or a random nonce per message

M: corresponds to the plaintext payload stream (as used in the initial unprotected implementation of hjStreaming.java and hjProxyjava.

$C = E(K_S, [\ Mp \ || \ MAC_{KM}(Mp)\ ]$

Ks: symmetric session key

Km1: MAC key

Km2: a MAC key for Fast control DoS mitigation

Note) The PAYLOAD SIZE is variable, depending on the Mp size and also for the cryptographic constructions used, that will be parameterized in the configuration files of SSP endpoints (in mySSPStreaingServer and mySSPUDPProxy).

Annex 2

SHP Specification

The SHP specification supports an initial handshake for the Phase 2 implementation, as represented in the following diagram representing the SHP Handshake sequence.

Proxy > StreamingServer:	HANDSHAKE-REQUEST MESSAGE
SetramingServer > Proxy:	AUTHENTICATED-HELLO-CHALLENGE
Proxy >StreamingServer:	RESPONSE-TO-CHALLENGE
StreamingServer> Proxy:	KEY-SA-ESTABLISHMENT
Proxy > StreamingServer:	HANDSHAKE-DONE

After the above handshake, the SteramServer will start the transmission of a requested movie, with all the established secrecy parameters, using the SSP protocol (as it works in PHASE 1).

Specification for SHP Message-Types

Each message exchanged in the SHP specification must be supported by a specific payload-type using the format of the SSP protocol, but with specific SHP payloads, as specified next:

HANDSHAKE-REQUEST-MESSAGE:

It is a cleartext payload, with the following format:

"HELLO"|| proxyID || <requested-movie>||N1|| PWDCS || Epwd (X)

"HELLO": a cleartext string "HELLO"

ProxyID: a Proxy unique identifier

Requested movie: the name of the movie

N1: a secure random value

TS1: A timestamp

PWDCS: Password Encryption Ciphersuite definition of the Password-Based Scheme used by the Proxy

Epwd (X): Password-Based Encrytion of X, using as password the SHA-256 hash value of a password or passphrase corresponding to the proxy

X: SHA1 ("HELLO"|| proxyID || <requested-movie>|| N1)

AUTHENTICATED-HELLO-CHALLENGE:

It is a payload with the following structure:

SIGCRYPTOSUITE|| PublicKeyS || H(N1)|| N2, || Sig(X)

SIGCRYPTOSUITE: The digital signature type used by the SteramingServert

PublicKeyS: A PublicKey of the Server, to be used for the SIGCRYPTOSUTE

H(N1): this is a response to the client challenge N1, returned as a HASH of N1, using SHA-256

N2: this is a new secure random nonce sent by the StreamingServer

Sig(X): a digital signature

X = PublicKeyS || H(N1)|| N2

RESPONSE-TO-CHALLENGE

SIGCRYPTOSUITE || PublicKeyP || N3 || DHpublicC || Sig(X)

SIGCRYPTOSUITE: The digital signature type used by the Proxy

PublicKeyP: PubllicKey of the Proxy, to be used for the SIGCRYPTOSUITE

Sig(X) A digital signature of X

X = H(N2) || N3 || DHpublicC

H(N2) – A hash of the N2 challenge previous sent by the Streaming Server

N3: A new challenge sent by the proxy

DHpubic: A Public DiffieHellman Number gerenated by the client

KEY-SA-ESTABLISHMENT

SIGCRYPTOSUITE || H(N3) || DHpublicS || || {N4 || SSPCIPHERSUITE}$_{\text{PublicKeyP}}$ || Sig(X)

DHpubicS: A Public DG number generated by the Steraming Server

Sig(X) A Digital Signature (using SIGCRYPTOSUITE

X = DHpublicS

N4: A new fresh random sent by the Streaming Server

SSPCIPHERSUITE: The SSPCIPHERSIOTE PARAMETERS for the SSP

{ }$_{\text{PublicKey}}$: Encrypted envelope

HANDSHAKE-DONE

E_K ("FINISHED" || H(N4)

FINISHED: The string "FINISHED"

H( ) : Hash function SHA-256

k: Session Key ACK derived from the DH Agreement

Note: From k and SSPCIPHERSUITE, you must generate the SecretShared Key and MAC Keys that will be used by the SSP protocol for the movie dissemination and reception.