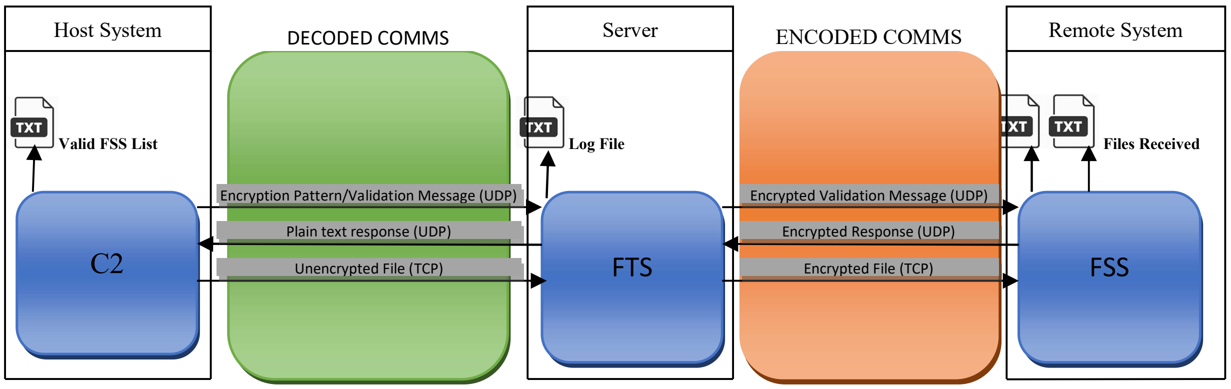
Basic Skill Level Exam

Project Title: File Transfer Service
As of Date: 14 March 2018

OVERVIEW:

Your organization is tasked with creating a File Transfer Service (FTS) that sends an encoded file to a designated File Storage Service (FSS). In order to test your developed FTS, your organization is required to create a simplified version of the command and control (C2) and File Storage Service (FSS) components that can handle the requirements outlined in this project.

SYSTEM DIAGRAM:



GRADING RUBRIC:

GRADING RUBRIC			
Documentation			
User Manual (4%)	Is the user manual easy to understand?	1%	12%
	Does the user manual follow a logical sequence?	0.5%	
	Are the sections in the user manual Cleary defined and laid out?	0.5%	
	Does the document define all aspects of user interaction?	1%	
	Does the user manual provide a general overview of the project?	1%	
Test Report (4%)	Is the test report easy to understand?	1%	
	Does the test report document all test cases and their results?	1%	
	Does the test report provide detailed steps to repeat test?	1%	
	Does the test report provide screen shots where appropriate?	1%	
Project Challenges (%2)	Did the individual address any challenges or concerns with the project?	2%	
Grammar (%2)	Are all documents free of grammatical errors?	1%	
	Are all documents free of spelling errors?	1%	
Implementation and construction			
Source Code (25%)	Were comments descriptive, concise, and aid code readability?	4%	
	Was source code not written by the exam taker properly cited i.e. source location (to include web address or page number and book title), date taken, and author?	3%	
	Was memory managed appropriately: use of malloc/calloc/realloc to create memory, free to release memory, and free of memory leaks when ran with valgrind?	7%	

	Were variable names clear, concise and descriptive in regards to their use?	4%	48%
	Were all variables used?	4%	
	Were secure coding practices used?	3%	
Architecture 8%	Were effective and efficient data structures used?	2%	
	Did the program match the design specification?	2%	
	Was adequate input validation conducted?	2%	
	Was the code structured in a modular fashion?	2%	
Testing Methodology 5%	Was a test plan integrated into the design of the project?	1%	
	Was thorough testing conducted?	2%	
	Were all tests in the test report documentation repeatable?	1%	
	Did all tests produce the output as described in the documentation?	1%	
Version Control 5%	Was one branch for each feature constructed?	1%	
	Were tags created as releasable version of code were successfully integrated into the master branch?	1%	
	Was one branch per a feature used and merged to master as completed?	1%	
	No work was conducted in the master branch?	2%	
Modularity 5%	Were header and source files used to create modular code?	2%	
	Was code logically separated into multiple .h and .c files?	2%	
	Did the use of structures support the modularity of the code?	1%	
	Execution		
Builds (10%)	When make is invoked does it compile without warnings when using –Wall command line option?	3%	40%
	When make is invoked does it build with the flags –Wall and –Werror?	6%	
	When make is invoked are all files placed in the specified directory?	1%	
Executes (10%)	Is the program fault tolerant when given bad input?	5%	
	Does input return the expected output?	2%	
	Does invalid input cause unexpected results and/or an error/crash?	3%	
Requirements (20%)	Are all requirements met?	6%	
	Was a multi-threaded server in C created as defined by the specification?	4%	
	Was a Command and Control element (C2) and File Storage Service (FSS) created as specified in the document?	5%	
	Is the transfer of a file from the C2 to the FTS and FTS to FSS supported as described in the document?	3%	
	Was encoding supported as specified in the document?	2%	
TOTAL	**Total**	**Earned**	

FUNCTIONAL REQUIREMENTS:

REQUIREMENT		GRADING AREA
FTS	Create a multi-threaded server in C called FTS	IMPLMENTATION
FTS	Support both TCP and UDP connections in your FTS	EXECUTION
FTS	Log all session information to include: C2's IP Address and port, the FSS IP and port, encode pattern, name of the file received from the C2, size of the file received from the C2, and the size of the file transferred to the FSS.	EXECUTION
FTS	Support the specified communication protocol associated with this project in the FTS.	IMPLMENTATION
FTS	Ensure the FTS applies the encode pattern specified by the C2's request packet before sending data to the corresponding FSS.	EXECUTION
FTS	Support the ability to receive files from multiple C2s at the same time via TCP.	EXECUTION
FTS	Support the ability to transmit files from the FTS to multiple FSS at the same time.	EXECUTION
C2	Create a C2 program with Python3 as described in this document.	IMPLMENTATION

C2	Support TCP and UDP network communications in the C2.	IMPLEMENTATION
C2	Support the specified communication protocol associated with this project in the C2.	IMPLEMENTATION
C2, FTS, FSS	Support the ability to transfer a file from the C2 to the FSS via the FTS.	EXECUTION
FSS	Create the FSS program with Python3 as described in this document.	IMPLEMENTATION
FSS	Support TCP and UDP network communications in the FSS.	EXECUTION
FSS	Support the specified communication protocol associated with this FSS in this projects documentation.	IMPLEMENTATION
C2, FSS, FTS	Support the encode options specified in the FSS, C2, and FTS specified in this document.	EXECUTION
FTS	Encode data transmitted from the FTS to the FSS.	EXECUTION
FTS	Decode data transmitted from the FTS to the C2.	EXECUTION
DOC	Submit a paper called "Project Challenges" that describes any challenges encountered, design issues and/or security flaws associated with this project and /or design issues	DOCUMENTATION
DOC	Submit a user's guide called "User's Guide" describing the initialization and execution of the file transfer system. An individual should be able to follow this user's guide to properly execute every aspect of the file transfer system.	DOCUMENTATION
DOC	Submit a paper titled "Project Testing" describing the testing methodology and practices used and their results (ensure that images are used to show test results when appropriate).	DOCUMENTATION
FTS	Build from a submitted Makefile on Linux.	EXECUTION

REQUIREMENTS:

REQUIREMENT		GRADING AREA
C2	The C2 must support both a prompted input and command line options.	EXECUTION
C2	The C2 must support a connection timeout feature.	IMPLEMENTATION
FTS	If a connection times out before a file transfer has been completed the FTS must attempt to reconnect to the FSS and finish transmitting the file.	IMPLEMENTATION
FSS	The FSS must support both command line arguments and a prompted input if no arguments are supplied.	IMPLEMENTATION
FSS	The FSS must decrypt files received before writing them to disk.	EXECUTION
FSS	The FSS must support a connection timeout feature.	IMPLEMENTATION
C2, FSS, FTS	All code must be commented using descriptive comments.	IMPLEMENTATION
C2, FSS, FTS	All output must be descriptive and meaningful to a user.	EXECUTION
C2, FSS, FTS	All code must be modular.	CONSTRUCTION
DOC	All testing methodology and test results must be documented in a detail manner in the Project Testing document.	DOCUMENTATION
DOC	All test must be repeatable via the information in the "Project Testing" document.	DOCUMENTATION
C2, FSS, FTS	All code must be tested.	CONSTRUCTION
GEN	No commits to the master branch, only merges	CONSTRUCTION
C2, FSS, FTS	Conduct Input validation on all user input	IMPLEMENTATION

CONSTRAINTS:

CONSTRAINT	GRADING AREA
The submitted makefile must supports the following: a. "cleanAll" – Removes object files and binaries b. "clean" – Removes all object Files c. "debug" – Builds the program with debugging symbols d. "build" – Builds the program without debug information and removes object files e. "buildAll" – Builds the program with debug information and leaves object files	IMPLEMENTATION
Repository submission directory structure must match the following: Top Level Directory /---SRC /---|---Header Files /---|---Source Files /---|---Test Harness /---|-----|---Test Files /---BIN /-----|---IPAddWhiteList /-----|---FSS.py /-----|---FTS /-----|---C2.py /---Documentation /---|----User's Guide /---|----Design Document /---|----Project Testing /---|----Project Challenges	CONSTRUCTION
All code reuse must be cited in the manner prescribed below: AUTHOR NAME: WEB ADDRESS or PUBLICATION: DATE YOU PULLED IT:	IMPLEMENTATION
The C2 must transfer a file to the user specified FSS in its entirety.	EXECUTION
The C2 must maintain a client list consisting of known FSS ports and IP addresses.	CONSTRUCTION
The C2 program cannot crash.	EXECUTION
The FSS program cannot crash.	EXECUTION
The FTS program cannot crash.	EXECUTION
No memory leaks can be found in any program.	IMPLEMENTATION

CONSIDERATIONS DURING REVIEW:

DEMONSTRATED ABILITY TO:	Grading Area
Multi-threading in C.	IMPLEMENTATION
Effective use of locking mechanisms in regards to multithreading.	IMPLEMENTATION
Effective use of appropriate data structures.	IMPLEMENTATION
Effective use of bitwise operations.	IMPLEMENTATION
Proper File I/O in C and Python.	EXECUTION
Appropriate use of control flow structures such as if, if-else, else and loops in C and python.	IMPLEMENTATION
Use of dynamic memory in C.	IMPLEMENTATION
Appropriate project structure and code modularity in C and Python.	CONSTRUCTION
Appropriate use of return values to determine specific errors that occur.	CONSTRUCTION
Network programming in both C and Python.	IMPLEMENTATION

SYSTEM ARCHITECTURE:

The file transfer system consists of three programming components the C2, the FTS, and FSS, as well as a user defined communication protocol, and encode/decode functionality.

OBJECTIVES:

1. Create a Python program that can be used interactively through prompted messages or via command line arguments to receive required input. This Python program will act as a C2 element that sends and receives communications over transmission control protocol (TCP) and user datagram protocol (UDP) and follows internet protocol version 4 (IPv4) specifications. The C2 element should maintain a file containing all known file storage system IPv4 addresses and listening ports that the user can select.

2. Create a user defined protocol to handle the communications between the C2 and the file storage service over UDP.

3. Create a multi-threaded C program that serves as a file transfer service that communicates over the UDP port range of 16000-17000 and TCP port range of 17000-18000 that will spawn a new thread for each connection established. The file transfer service must maintain a continuous operational status and maintain control over each thread created. The file transfer service will maintain an encoded log of the connections established as well as the total number of data bytes transmitted during the connection. The file transfer service will encode/decode communication as defined by received encoded pattern received from the C2. Communication between the C2 and the file transfer service will be uuencoded. Communication between the file transfer service and the file storage server will be encoded.

4. Create a Python program to act as a file storage service that receives encoded C2 communications using an encode pattern from the options specified below from a file transfer service (for the purposes of this assignment the developer will specify an encode pattern comprised of a minimum of two encoding options each formatted: **[option]:[bytes];etc…**, i.e. **^20:32;ror8:80**), decrypts the C2 communications received from the file transfer service, encodes the decrypted validation message using the same encode, and sends the encoded message back to the C2 server via the file transfer service. After the encoded connection is established, the file storage service will receive and store the received file after it has been decrypted. (For the purposes of this assignment file recall does not need to be coded)

C2 (Written in Python):

The C2 component of the file transfer system can be called either from the command line with command line arguments or interactively. Once all input is received and validated, it initiates a connection over a UDP port. The C2 maintains a list of all IPv4 and port numbers of the valid file storage services. The C2 will establish a connection to the file transfer service through one of the known port numbers in the file transfer service range. C2 must be able to support a timeout or no connection made on a port and move to the next port. If no connection can be made after the known port range is exhausted, the user is notified and the C2 component exits. If the C2 was executed using the interactive/menu option, the user is provided the option to attempt connection to the file transfer service again or exit. The initial UDP message contains the destination IPv4 address of the data file server, the port that the storage service is listening on for messages, the encode pattern that the file transfer service will use to encode/decode the information being transferred, and the pass phrase to be used in encoding the information being transferred. After confirming that the file storage service possesses the appropriate decryption method, the C2 then sends an uuencoded file over a TCP connection to the file transfer service for encoding before being transmitted to the file storage service.

If C2 cannot verify the pass phrase, an error message is displayed to the user and the file transfer is terminated.

USER DEFINED COMMUNICATION PROTOCOL (Enabled in C and Python):

The user defined communication protocol will allow for the establishment of a communication path through the file transfer service for the C2 and file storage system. The user defined communication protocol will follow the UDP header. The structure of the user defined communication protocol will be controlled based on the value found in the packet type byte. The valid values for packet types are:

1) 0x00 – used to indicate that the packet will be used to request that a connection be established with a specified file storage service (Request Packet Type: C2 to File Transfer).

2) 0x01 – used to indicate that the packet will be used to initiate a connection from the file transfer service to a specified file storage service. (Initiate Packet Type: File Transfer Service to File Storage Service)

3) 0x02 – used to indicate that the packet is a response from a file storage service. (Response Packet Type: File Storage Service Response to File Transfer Service.)

4) 0x03 – used to indicate that the packet is a validation message from the file transfer service. (Validate Packet Type: File Transfer Service Response to C2)

Diagrams for each of the packet types for the user defined communication protocol with the intended use cases are pictured below:

OFFSET	0								1								2									3						
Octet	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	UDP HEADER																															
8	PACKET TYPE								■																							
9	FILE STORAGE SERVICE IPv4 ADDRESS																															
13	DESTINATION PORT																ENCODE PATTERN BYTE LENGTH															
17	ENCODE PATTERN																															
VARIES	DECODED INITIALIZATION MESSAGE																															

Figure 1: User defined communication protocol (request packet type; C2->FTS)

OFFSET	0								1								2								3							
Octet	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	UDP HEADER																															
8	PACKET TYPE								■																							
9	ENCODED INITIALIZATION MESSAGE																															

Figure 2: User defined communication protocol (initiate packet type; FTS->FSS)

OFFSET	0								1								2								3							
Octet	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	UDP HEADER																															
8	PACKET TYPE								PORT TO ESTABLISH TCP CONNECTION																■							
11	ENCODED VALIDATION MESSAGE																															

Figure 3: User defined communication protocol (response packet type; FSS->FTS)

OFFSET	0								1								2								3							
Octet	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	UDP HEADER																															
8	PACKET TYPE								PORT TO ESTABLISH TCP CONNECTION																							
9	UN ENCODED VALIDATION MESSAGE																															

Figure 4: User defined communication protocol (validate packet type; FTS->C2)

FILE TRANSFER SERVICE (Written in C):

The file transfer service component will ensure that it is continuously running, available, and is able to simultaneously facilitate communication between multiple C2 components and multiple file storage services. It will enable the calling C2 component to validate that the requested file storage service is able to encode and decode data as defined in the requested encode pattern. The file transfer service must maintain an encoded log with a record of each session attempted or established and the total number of bytes of data passed across the session, not the total number of bytes transmitted (i.e. number of bytes passed across the UDP initialization session, the size of the file transmitted across a TCP session, etc). The file transfer service will ensure encoded communications are maintained from the file transfer service to the file storage service. The file transfer service will establish a TCP session to forward received data, after it is encoded by the file transfer service, from the originating C2 component to the requested file storage service.

ENODE/DECODE FUNCTIONALITY:

The file transfer service and file storage service within the file transfer system will be able to encode and decode messages based on the encode pattern specified by the C2 and the encoding options defined in this document. The file transfer service is required to handle any encode pattern provided by the C2. Each file storage system is only designed to handle one set encode pattern. Each encode pattern must contain at least two encoding options to be valid.

The encoding options that the file transfer service and file storage service must be able to support are:

SYMBOL	OPERATOR	DESCRIPTION
^	Bitwise XOR	True, if A or B are true but not both.
~	Bitwise Not	Toggle bits off that are on, and turn the ones that were off on.
ror	Rotate right	Shift bits right and take any bits that would fall off move them to the highest bit.
rol	Rotate left	Shift bits left and take any bits that would fall off move them to the low order bit.

The encoding pattern is created by putting a series of encoding options together separated by a semicolon (;). Each individual encoding pattern set is formatted as the option (symbol and value if required) followed by the colon (:) separator and the number of bytes to perform the action on. The encoding pattern is used to encode or decode every packet upon receipt or transmission by the file transfer service and the file storage service. Some example encoding patterns are:

1) ^2:40;rol1:120
2) ~:20;ror3:40
3) ror8:80;^16:160.

FILE STORAGE SERVICE (Written in Python):

The file storage service receives encoded communications from the file transfer service. The file storage service will receive an initialization communication from the file transfer service on the UDP port defined when the file storage service was initialized. Upon receipt of the initialization communication, the file storage service will decrypt the packet using the encode pattern it has been set up to be able to encode and decode. After the data has been decoded, the decoded message will be encoded with the same encode pattern and transmitted back to the file transfer service along with the TCP port that the file transfer service should use to establish the connection for file transfer. The file storage service will listen for a communication across the specified TCP port. As the file storage service receives the file being transmitted, it is decrypted and saved to a log.

GENERAL SESSION LAYOUT:

1) User executes the C2 component and provides the required information either via the command line or through prompts.

2) C2 transmits the following to the file transfer service over UDP: a request packet type, the IPv4 address of the file storage service, the port number to communicate with the file storage service, the length of the encode pattern in (bytes), the encode pattern, and a non-encoded message to use as an initialization message.

3) The file transfer service receives the communication initialization request over UDP from the C2, encodes the initialization message using the encode pattern and sends the following to the file storage service over UDP specified by the IPv4 address and port number found in the initialization packet: initiate packet type and encoded initialization message.

4) The file storage service receives the initiate request over UDP, decrypts the encoded initialization message, encodes the decrypted initialization message, and transmits the following back to the file transfer service over UDP: a response packet type, encoded validation message.

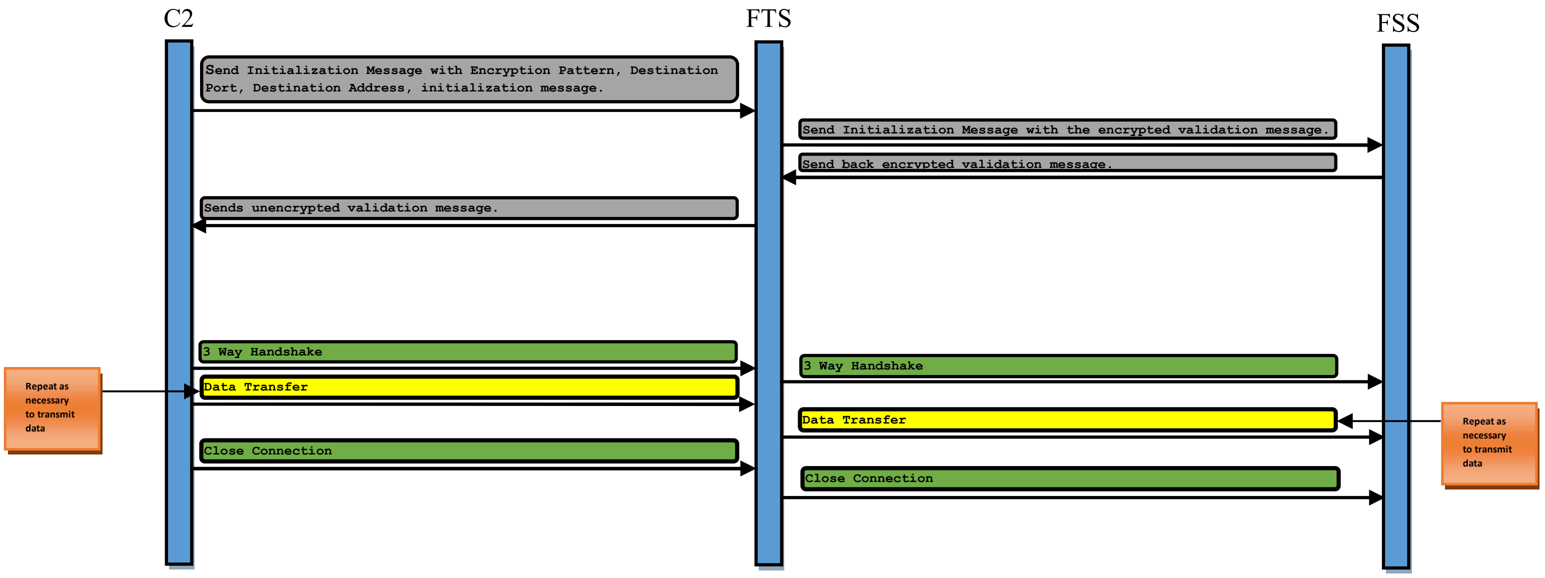
5) The file transfer service receives the validation message from the file storage service across UDP then decrypts the encoded validation message and transmits the following to the C2 over UDP: response packet type and non-encoded validation message.

6) C2 will receive the validation message back from the file transfer service over UDP and validates that the validation message transmitted is equal to the validation message received. If they are different, execution terminates.

7) If the messages are the same, the C2 connects to the file transfer service via TCP and starts sending the file to the file transfer service.

8) The file transfer service accepts the TCP connection, starts receiving the non-encoded file, encodes the file via the established encoding pattern, establishes a TCP connection to the file storage service on the port the file storage service indicated, and transmits the file.

▨	**UDP Packet**
🟩	**TCP Packet**
🟨	**Data Transfer**


C2
FTS
FSS
Send Initialization Message with Encryption Pattern, Destination Port, Destination Address, initialization message.
Send Initialization Message with the encrypted validation message.
Send back encrypted validation message.
Sends unencrypted validation message.
3 Way Handshake
Data Transfer
Close Connection
3 Way Handshake
Data Transfer
Close Connection
Repeat as necessary to transmit data
Repeat as necessary to transmit data

Session Initialization and File Transfer Diagram Example