170D WOBC Module K Exam II-A

`reverse`: Man in the Middle

In a secret underground bunker we have uncovered an obelisk of unknown origin. The obelisk has a dialing mechanism capable of sending data to the far reaches of the universe. We've recently discovered that we're not the only ones using the obelisk, there is communication that appears to be transmitting to a point just outside our solar system. Fortunately the communication is using an unsecure version of TLS that we've been able to decrypt. What we've found in the decrypted payload is unsettling. It's clear that the messages are directing an armada towards earth. They're not friendly.

We believe we can avoid extinction by executing a man in the middle attack, modifying the encrypted traffic to send the agressors back from where they came.

You will be provided with a `.pcap` containing one or more packets of encrypted conversation. As well as the decryption key needed to decrypt the payload. To help, you will also be provided with a library to link against to perform the encryption and decryption.

Your task is to write a command line utility (`reverse`) that will decrypt each message in the `.pcap`, reverse the direction commands, re-encrypt, and write the result back out to `out.pcap`. If the payload contains a message we need to print the message to `STDOUT` without the padding. We will then prepend the message with `"Don't "` and reinsert into the `out.pcap` conversation.

The command will be called as follows:

```
reverse <128-bit key> <input file>
```

ex:

```
reverse df290c80bb7f9ecb08d4a6b3030951aa input.pcap
reverse 0dedd2d9229a260285cb491845010efc simple_input.pcap
```

DIcE Rubric

Document	Design Plan	Does the design plan provide a clear general overview of the project?	3%
		Is the design plan easy to understand?	2%
	Test Plan	Are test cases detailed enough to repeat easily?	2%
		Are expected results stated clearly?	2%
		Are requirements adequately covered by test cases?	1%
	Project Writeup	Does the writeup document challenges and successes encountered?	2%
		Does the writeup document any lessons learned?	3%
	Writing	Is the project free of grammatical and spelling errors?	4%
		Is non-code formatting consistent?	1%
	Code Formatting	Does code conform to the Barr-C?	4%
		Are appropriate names chosen to enable code readability?	2%
		Are comments added where appropriate and aid understanding of the logic?	2%
		Is any outside code cited appropriately?	2%
	Total		**30%**
Implement	Version Control	Does the project have the correct name and default branch?	1%
		Were commits broken down into appropriate scopes?	3%
		Are commit messages simple and informative?	1%
	Architecture	Are effective and efficient data structures used?	5%
		Was the code designed and constructed in a modular fashion?	10%
		Were generally sound decisions made with regard to architecture?	10%
		Can the project be compiled with `gcc -Wall` no warnings?	5%
	Testing	Does the program include robust unit tests?	4%
		Do all automated tests pass when run from `make check`?	1%
	Total		**40%**

Execute	Safety	Does the program avoid crashing or infinite loops, even on invalid input?	5%
		Does the program correctly clean up allocated memory?	2%
		Does the program avoid dereferencing incorrectly?	3%
	Parsing	Does the program pass `make all` with no warnings?	5%
	Requirements	Were all inputs parsed correctly and yield the correct output?	5%
		Are all other requirements met?	5%
	Performance	Does the program scale appropriately with input and data?	3%
		Does the program execute in a timely manner?	2%
	Total		**30%**

Area	Requirement
Document	All documentation must be in PDF format unless otherwise specified.
Document	All documentation must be located in a `doc/` folder at the top level of the project.
Document	The design document must be located in `doc/design.pdf`
Document	The test plan must be located in `doc/testplan.pdf`
Document	The project writeup must be located in `doc/writeup.pdf`
Document	All code must conform to Barr-C.
Implement	Project must be stored in the assigned VCS account, under the project name `reverse`.
Implement	No third-party header files/libraries may be used unless signed off by the Program Manager or Instructor.
Implement	Project must use appropriate data types or structures, and relationships between them.
Implement	All automated tests and test code must be located in a `test/` folder at the top level.
Implement	Project must provide appropriate automated unit tests.
Implement	Project must compile without errors or warnings with `gcc`.
Implement	A valid makefile must be provided that correctly compiles the project.
Implement	Implement parsing all Alien Message Payloads explained below.
Implement	The Sequence IDs in the alien packet must be updated to match the reversed directions.
Implement	Reversed Alien Packet Movements must be written into the output `.pcap`.
Implement	Packets of Message type must be prepended with `"Don't "` and writen into the output `.pcap`.
Implement	Packets of message type must be printed to `STDOUT` preceded by the Destination ID.
Execute	Project must run on the class machine.

Area	Requirement
Execute	Project must not crash or get stuck in an infinite loop, even on invalid input.
Execute	Project must be invoked as `reverse` from the top level directory of the repository.

Suggested Implementation

1. Read the key and input `.pcap` filename from the command line.
2. Parse the binary `.pcap` file and for each TLS section extract the encrypted message.
3. Pass the encrypted message to the library.
4. Parse the returned decrypted contents.
    - If the packet is Message type print Destination ID and the message to `STDOUT`.
        1. Make the first character lowercase and prepend the message in frame with `"Don't "`.
        2. Update the padding
    - If the packet is a Movement type reverse the movement operation.
5. Pass rebuilt frame to library for re-encryption.
6. Write the re-encrypted and re-sequenced conversation to `out.pcap`.

NOTE The ordering in the `.pcap` is not important, just make sure that the Sequence IDs are updated appropriately.

TIP For simplicity the TCP packets will be in order, however the Alien Packets might be out of sequence.

`.pcap` Encoding Format

Offsets	Octet	0								1								2								3							
File Offset	**Header Offset**	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
`.pcap` **File Header**																																	
0	0	**File Type ID**																															
4	4	**Major Version**																**Minor Version**															

8	8	GMT Offset					
12	12	Accuracy Delta					
16	16	**Maximum Length of a Capture**					
20	20	**Link Layer Type**					
`.pcap` **Packet Header**							
24	0	UNIX Epoch					
28	4	µs from Epoch					
32	8	**Length of Data Captured**					
36	12	Untruncated Packet Length					
Ethernet Frame							
40	0	Destination MAC					
44	4	Destination MAC				Source MAC	
48	8	Source MAC					
52	12	**Ethernet Type**					
IPv4 Header							
54	0	**Version**	**IHL**	DSCP	ECN	**Total Length**	
58	4	Identification				Flags	Fragment Offset
62	8	Time to Live		**Protocol**		Header Checksum	
66	8	Source IP Address					
70	12	Destination IP Address					
TCP Header							
74	0	Source Port				**Destination Port**	

78	4	**Sequence Number**											
82	8	Acknowledgement Number											
86	12	**Data Offset**	Reserved	`NS`	`CWR`	`ECN`	`URG`	`ACK`	`PSH`	`RST`	`SYN`	`FIN`	Window Size
90	16	Checksum											Urgent Pointer
TLS Header : When TLS Content Type equals Application Data (23 or 0x17)													
94	0	**Content Type (0x17)**											
95	1	Version											**Length**
99…	5…	**Encrypted Message…**											

TIP Except for the extra credit, Content Types other than 0x17 can be ignored and passed directly into the output `.pcap`.

NOTE For simplicity `.pcap` will only ever contain one TCP session and will only include packets from one sender.

Encrypted Message Format

Message will be encrypted with AES128 using the CBC block method. The cipher suite used has the IANA name TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256. Encryption will be covered later in the course so it’s not expected that you’re able to decrypt and encrypt the message. A library will be provided to do this for you. However you will need to correctly pull the payload out of the decrypted message.

For reference, the encrypted message will be structured like so:

Offsets	**Octet**	0								1							
Packet Offset	**Header Offset**	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	Opaque IV (Initialization Vector)															
2…	2…	Payload… (The Alien Packet, Padded to 16 bit boundaries)															

n	n	Opaque MAC (SHA256 hash)
n+2	n+2	Opaque MAC (SHA256 hash)

Decrypted Message Format

Alien Packet Header

Consistent header for all decrypted alien packets.

Offsets	**Octet**	0								1								2								3							
Packet Offset	**Header Offset**	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	0	Version				Type				Total Length																							
4	4	Source ID																Destination ID															
8	8	Sequence ID																															
12…	12…	Payload…																															
n	n	Padding to 16bit boundary																															

Field	Type	Description
Version	uint4	Version of packet format. This document and the programs written for it are in support of version 1.
Type	uint4	Type of payload, either 0 = Message or 1 = Movement
Total Length	uint24	Length of the packet in octets/bytes, including the fixed header.
Source ID	uint16	This source ID we believe to be the alien homeworld, this will always be 1.

Destination ID	uint16	Each squadron in the armada has it’s own ID. This is the ID of the squadron to whom the packet is being sent.
Sequence ID	uint32	A monotonically increasing ID for a given sender/receiver pair.

NOTE Don’t forget to update the Sequence ID when the movements are reversed.

Alien Message Payload

Type = 0

Offsets	**Octet**	0								1								2								3							
Packet Offset	**Payload Offset**	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
12…	0…	Message…																															

Field	**Type**	**Description**
Message	char[]	ASCII-encoded string. ***NOT*** NULL-terminated.

When a message payload is encountered the message should be printed to `STDOUT` preceded by the Destination ID. The message should be rewritten into the frame with `"Don't "` prepended to the message with the first character lowercased.

WARNING Don’t forget to readjust the padding to take into account the change in message length. (see rfc5246 for padding info)

Example

Input

```
Message : "Arm torpeedoes."
```

Output

```
STDOUT  : "Arm torpedoes"
Message : "Don't arm torpedoes."
```

Alien Movement Payload

Type = 1

Offsets	**Octet**	0								1								2								3							
Packet Offset	**Payload Offset**	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
12	0	Galactic Azimuth																															
16	4	Galactic Azimuth (cont.)																															
20	8	Galactic Inclination																															
24	12	Galactic Inclination (cont.)																															
28	16	Distance																															
32	20	Speed																															

Field	Type	Description
Galactic Azimuth	double	A IEEE 754 double-precision decimal (aka binary64), measured in degrees.
Galactic Inclination	double	A IEEE 754 double-precision decimal (aka binary64), measured in degrees.
Distance	float	A IEEE 754 single-precision decimal (aka binary32), measured in meters.
Speed	float	A IEEE 754 single-precision decimal (aka binary32), measured in meters/second.

Reversing Movement Payload

Add 180 degrees both the Azimuth and Inclination. Azimuth and Inclination are bounded to [0 to 360) degrees (zero inclusive, 360 exclusive). Azimuth and Inclination are relative to true North and Horizon, not the local reference of the target. Distance and Speed should be kept the same.

WARNING Alien movements might not be in order in the input `.pcap`. Make sure to verify the sequence IDs in the Alien Packet Header.

Example

Input

```
Galactic Azimuth      : 120.0
Galactic Inclination  : 330.0
Distance              : 123.0
Speed                 : 50.0
```

Output

```
Galactic Azimuth      : 300.0
Galactic Inclination  : 150.0
Distance              : 123.0
Speed                 : 50.0
```

Provided Materials

The starter kit includes the following materials:

- Encryption/Decryption libarary with `README.md`.
- Various example input `.pcap` files, with corresponding decryption keys.
- Corresponding example decrypted `.pcap` files. Technically invalid, but show decrypted ciphertext in place.
- Corresponding example output `.pcap` files with re-encrypted TLS.

- `generator` executable can be used to generate an input `.pcap`

Suggested Extra Credit

Area	Feature	+
Document	Write a `man(1)` page to document the program.	+2
Implement	Update the TCP sequence numbers to reflect the new packet sizes as well.	+2
Implement	Update the SHA256 MAC in the TLS frame (see RFC7366).	+3
Implement	Update checksum in the TCP frame (see RFC793).	+5