0.1 Protocols

We define two protocols that calculate the number of rounds and the time it takes Alice to exfiltrate a file from Bob's machine that she illegally controls, using n covert channels while we assume that Eve can read all the n channels but can block b out of these n channels where $b \leq (n-1)$.

From here on we will refer to Alice as the Server, Bob as Client and Eve as watcher.

Round: Each iteration over the available n channels is defined as a round.

Through out this protocol, to guarantee data authentication and integrity we used HMAC, it is a particular type of "Message Authentication Code" that uses a "Cryptographic Hash Function" with a "Secrete Key". With our protocol, to calculate the HMAC for the entire file we used the cryptographic function SHA-512 with a key K while to calculate the HMAC for the chunks of the file we used the cryptographic function SHA-1 with a key K'.

Assumption: Since the client and the server are controlled by the same person we assume the hashing keys and the cryptographic functions that are used to calculate the $MAC(key, Message)$ can be safely applied to both the client and the server.

Summary of Protocol #1:
We compute the HMAC $T = MAC(K, M)$ of the entire file. After that, we divide the file into chunks and compute the HMAC $t_i = MAC'(K', m_i)$ for each chunk of the file. First, we create a packet of type #1 containing the calculated HMAC $T' = MAC(K, M')$ where M' consists of all the attributes inside the packet, the form of type#1 packet is $(T', andOtherAttributes)$, it will broadcast over all the available n channels. Then, we construct packets of type #2 that are formed of $(ti, andOtherAttributes)$ and send each packet in parallel over the n available channels. Repeat until all packets are sent. Meanwhile, at the beginning, the server will try to verify one of the type #1 packets, by computing the HMAC of the extracted attributes of each packet and then compare the calculated HMAC with the received HMAC T'. After that, for each type #2 packet received, it will compute $t_m = MAC'(K', m_i)$ and compare it with t_i. If they are equal, the server will accept the packet otherwise it will drop it. And when it receives the entire packets, it will compute $T_M = MAC(K, M_{received})$ of the entire file and compare it with T. If they are equal it will accept the entire file otherwise it will drop it.

Summary of Protocol #2:
We compute the HMAC $T = MAC(K, M)$ of the entire file. After that, we divide the file into chunks and compute the HMAC $t_i = MAC'(K', m_i)$ for each chunk of the file. First, we create a packet of type #1 containing the calculated HMAC $T' = MAC(K, M')$ where M' consists of all the attributes inside the packet, the form of type#1 packet is $(T', andOtherAttributes)$, it will broadcast over all the available n channels. Then, for each chunk of the file, we construct a packet of type #2 that contains $(ti, andOther$ $Attributes)$. The client picks the first n packets and send a different packet over each of the n channels, then the client waits for an ACK from the server that contains the packets which were delivered correctly, the server will deduct the correctly received packets and will retransmit the remaining packets, this will be repeated until all the n packets are received correctly and then the client will move to the next n packets and so on. Meanwhile, the server at the beginning will try to verify one of the type #1 packets, by computing the HMAC of the attributes inside each packet and then

compare the calculated HMAC with the received HMAC T'. After that for each packet of type #2 received the server will compute the HMAC $t_m = MAC'(K', m_i)$ and compare it with its corresponding t_i. If they are equal, the client will accept the packet and add it to the list of correctly received packets otherwise the packet will be dropped, when the server is done from verifying the n packets it will broadcast an ACK over all the n channels to the client containing the list of sequence numbers of the packets that were correctly received asking it to resend the rest of the packets and repeat until it correctly receives the entire n packets. So, when the server receives the complete file it will compute $T_M = MAC(K, M_{received})$ of the entire file and compare it with T that was sent in a packet of type #1. If they are equal it will accept the file otherwise it will drop it.

For Both Protocols: T protects the file as a whole, whereas each t_i protects a chunk of the file and tells us which chunk was corrupted. Tagging the index along with the chunk takes care of an adversary permuting the chunks. This protocol will work even by assuming that the adversary can corrupt the channels adaptively. i.e, at each run, the corrupted channels are chosen before Bob sends stuff. In other words, even if the watcher decides to corrupt always the same channels the protocol will converge since, after each run, some packets are duplicated or sent over a different channel.

0.1.1 Protocol #1 :

Assumes that the adversary can block an unknown number of channels b out of n channels where $b < n$. s is the packet size used.

Suppose we have a file of size f that the server needs to exfiltrate from the client's machine. we divide this file into p packets where

$$p = \frac{f}{s}$$

Protocol #1 in general, will calculate the HMAC of the entire file and then form a packet containing:

- The HMAC of the complete packet calculated from the attributes::
  1. Packet type.
  2. Number of packets to be sent.
  3. Length of the file name.
  4. The file name that the client will send.
  5. HMAC of the entire file.
- Packet type.
- The file name that the client is going to send.
- The length of the file name.
- The number of packets to send.
- The HMAC of the entire file.

This packet will broadcast over the all n channels at the beginning of the exfiltration process to notify the server that a file will be sent now. Meanwhile, the server will start receiving these packets. For the first packet received it will, extract the attributes, calculate the HMAC of the extracted attributes and then compare it with the received HMAC of the complete packet if they verify it will create a file, save other attributes for later comparisons and drop all other packets of the same type. Otherwise, it will drop the packet and move on to the next one. Then the client divides the complete file into chunks. For each chunk, we form a packet p_i containing:

- The HMAC of the complete packet calculated from the attributes:
  1. Packet type
  2. Sequence number.
  3. Chunk of data from the file.
- Packet type.

- Sequence Number.
- Chunk of data from the file.

We will broadcast each p_i packet over all the n channels in parallel. E.g, packet 1 will be sent over n channels simultaneously, then packet 2 will be sent also over all the available channel and so on, until we send all the p packets. Meanwhile, the server is receiving these packets, it extracts the data inside them, then calculate the HMAC of the attributes:

- Packet type.
- Sequence Number.
- Chunk of data from the file.

The server will compare the calculated HMAC with the HMAC that was extracted from the packet, if they are equal it will drop all other packets that have the same sequence number otherwise it will drop it and then check the next packet and so on until it will have the complete file. After having the complete file it will calculate the HMAC of the entire file and compare it with the HMAC that was saved from the packet of type #1.

We define two types of packets in protocol #1 that will be explained later in this section. The first step is done when the client broadcasts a packet of type #1 (Table: 1)

We call this packet the "File Transfer Request", it is broadcast over all the n channels at the beginning. Its job is to prepare the server to receive a new file. When the server receives this packet it will:

- Extract the attributes:
    1. OP Code (Type of the packet).
    2. Number of Packets.
    3. Length of the File Name.
    4. File Name.
    5. Hash of the File.
- Calculate the HMAC of the extracted attributes.
- Compare the calculated HMAC with the complete HMAC that is received within the packet.
- If they are equal it will:
    1. Create a file with the name "File Name".
    2. Save the complete HMAC of the file.
    3. Save the number of packets of the entire file.
    4. Drop all other packets of the same type.
- If they are not equal it will move to the next packet.

This packet is broadcast only one time at the beginning of the transmission of each file and the server will accept the first packet where the HMAC verifies and it will drop the rest.

Then we have packet of type 2 (Table: 2).

We call this packet the "Data packet", each data packet is also broadcast over all the n channels. Its job is to send the data (*chunks*) of the file. When the server receives this packet it will:

- Calculate the HMAC of the extracted attributes.
    1. OP Code (Type of the packet).
    2. Sequence Number.
    3. Chunk of Data from the File.
- Compare the calculated HMAC with the received HMAC.
- If the HMACs are equal it will accept the packet and drop all other packets with the same sequence number.
- If the HMACs are not equal it will drop the packet and move to the next one.

Complete HMAC	Op Code = 1	Number of Packets	Length of the File Name	File Name	HMAC of the file

Table 1: Type #1 packet of protocol #1.

HMAC	Op Code = 2	sequence no.	Chunk of Data from the File

Table 2: Type #2 packet of protocol #1.

Note: The final "Data packet" will be padded with (char = 0) by the client in case it is not equal to the defined packet size, and then it will be stripped by the server when it is received.
The server will keep reading these "Data Packets" until the total number of packets accepted equals the number of packets attribute that was received in the "File Transfer Request" packet. Then it will calculate the HMAC of the entire file received and compare it with the HMAC that was received in the "File Transfer Request" packet, if they are equal it will accept the file otherwise, it will drop it. After that, it will get ready to receive the next file.

This protocol guarantees the delivery of the complete file even in the presence of a watcher who is trying to modify the packets between the client and the server. Although, this protocol does not use acknowledgment packets from the server to specify which packets were correctly delivered or no. It has high costs in terms of bandwidth and number of rounds as each packet is being broadcast over every channel each time.

0.1.2 Protocol #2 :

Suppose we have n channels out of which the watcher can corrupt at most an unknown number of channels b where $b \leq (n-1)$. Further, Suppose we have a file of size f that the server needs to exfiltrate from the client's machine. we divide this file into p packets where

$$p = \frac{f}{s}$$

Protocol #2 in general, will calculate the HMAC of the entire file and then form a packet containing:

- The HMAC of the complete packet calculated from the attributes:
  1. Packet type.
  2. Number of packets.
  3. Length of the file name.
  4. The file name that the client will send.
  5. HMAC of the entire file.
- Packet type
- The file name that the client is going to send.
- The length of the file name.
- The number of packets to send.
- The HMAC of the complete file.

This packet will broadcast over the all n channels at the beginning of the exfiltration process to notify the server that a file will be sent now. Meanwhile, the server will start receiving these packets. For the first packet received it will, extract the attributes, calculate the HMAC of the extracted attributes and then compare it with the received HMAC. If they verify it will create a file, save other attributes for later comparisons and drop all other packets of the same type. Otherwise, it will drop the packet and move on to the next one. After that, the client will divide the complete file into chunks and for each chunk, we form a packet p_i containing:

- The HMAC of each chunk of the file calculated from the attributes:
    1. Packet type
    2. Sequence number.
    3. chunk of data from the file.
- Packet type
- Sequence Number.
- Chunk of data from the file.

Protocol #2 will choose the first n packets and then send each packet p_i over a different channel. E.g, over channel #1, packet #1 will be sent, over channel #2, packet #2 will be sent and so on. The packets will be sent in parallel (At the same time). Meanwhile, the server is receiving the packets, it extracts the data inside each packet, then calculate the HMAC of the attributes:

- Packet type.
- Sequence Number.
- Data of the chunk.

Then it will compare the calculated HMAC with the HMAC that was received within the packet. If the HMACs verify it will accept the packet and add it to the list of accepted packets. Otherwise, it will drop it. After inspecting the n packets the server will use an ACK packet containing:

- The HMAC of the complete packet calculated from the attributes::
    1. Packet type
    2. Sequence numbers of the accepted packets.
- Packet type.
- Sequence Numbers of the accepted packets.

This Ack packet will be broadcast over all the n channels to inform the client about the accepted packets then the client will retransmit the packets that were not correctly delivered. This will be repeated until all the n packets are successfully received by the server and then the client will start sending the next n packets and the same process will be repeated till all the p packets are correctly received by the server. When all the p packets are received, the server will calculate HMAC of the entire file and then compare it with the HMAC that was received in the first broadcast packet, if they verify, it will accept the file. Otherwise, it will drop it.

We define three types of packets in protocol #2 that will be explained later in this section. The first step is done when the client broadcasts a packet of type 1 (Table: 3).

As with protocol #1, we call this packet the ”File Transfer Request”, it will broadcast over all the n channels in the beginning. Its job is to prepare the server to receive a new file. When the server receives this packet it will:

- Extract the attributes:
    1. OP Code (Type of packet).
    2. Number of Packets.
    3. Length of the File Name.
    4. File Name.
    5. HMAC of the Data.
- Calculate the HMAC of the extracted attributes.

Complete HMAC	Op Code = 1	Number of Packets	Length of the File Name	File Name	HMAC of the entire file

Table 3: Type #1 packet of protocol #2.

- Compare the calculated HMAC with the complete HMAC that is received within the packet.
- If they are equal it will:
  1. Create a file with the name ”File Name”.
  2. Save the complete HMAC of the file.
  3. Save the number of packets of the entire file.
  4. Drop all other packets of the same type.
  5. Prepare to receive the chunks of the file.
- If they are not equal it will move to the next packet.

This packet is broadcast only one time at the beginning of the transmission of each file and the server will accept the first packet of these where the hashes verify and then it will drop the rest.

Then we have packet of type 2 (Table: 4).

We call this packet the ”Data packet”, each data packet is sent over one available channel. Its job is to send the data of the file. When the server receives this packet it will:

- Calculate the HMAC of the attributes:
  1. Packet type.
  2. Sequence Number.
  3. Chunk of data from the file.
- Compare the calculated HMAC with the received HMAC.
- If the HMACs are equal it will accept the packet and add the packet sequence number to a list of accepted packets.
- If the HMACs are not equal it will drop the packet.

Note: The final ”Data packet” will be padded with (char = 0) by the client in case it is not equal to the defined packet size, and then it will be stripped by the server when it is received.
The server will keep reading these ”Data Packets” until the total number of packets successfully received equals the number of packets received in the ”File Transfer Request” packet. Then it will calculate the HMAC of the entire file received and compare it with the HMAC that was received in the ”File Transfer Request” packet, if they are equal it will accept the file otherwise, it will drop it. Then it will get ready to receive the next file.

Then we have packet of type 3 (Table: 5).

We call this packet the ”Data Packet Acknowledgment”, This packet type is broadcast to the client from the server after inspecting the n packets that were received. Its job is to notify the client which packets were correctly received. When the client receives this packet type it will:

- Extract the attributes:
  1. Op Code (Packet type).
  2. sequence numbers.
- Calculate the HMAC of the extracted attributes.

Complete HMAC	Op Code = 2	sequence no.	Chunk of data from the file.

Table 4: Type #2 packet of protocol #2.

HMAC	Op Code = 3	sequence no. of the received packets

Table 5: Type #3 packet of protocol #2.

- Compare the calculated HMAC with the received HMAC.
- If they are equal it will:
  1. Extract the sequence numbers of the received packets.
  2. Deduct these sequence numbers from the pool of the available packets to be sent.
  3. Drop the other packets of the same type.
  4. Send the remaining packets over the n channels again.
- If they are not equal it will:
  1. drop the packet.
  2. start analyzing the next packet.

This ACK packet is broadcast over all the channels, we can afford that since the size of these packets is relatively small.

This protocol guarantees the delivery of the file even in the presence of a watcher who is trying to modify the packets between the client and the server. It has low costs in terms of bandwidth and number of rounds in comparison with protocol #1 as each packet is sent only one time over a channel and will only be retransmitted in case of a failed delivery. Although this protocol uses ACKs packets. These packets are relatively small in comparison of the data packets so in a way we conserved the low bandwidth.