United Front 2 – Hybrid Storage UDP Covert Channel

User Guide

Our channel transmits UDP datagrams from a client executable file written in C# to a server written in python. The client accepts user input in the form of a string. The client then converts each individual character of the input string into chars. The client translates each character to its ASCII equivalent and stores it in a char array. The client creates a key that is a random hexadecimal character between 0 and 15. This value is added to the value that is used for the length of the datagram generated by the client that is filled with pseudorandom ASCII characters. The key is inserted into this datagram at the location specified by this formula:

$$index = (7 + int(((len(message))/5))) \% len(message)$$

The client sends each character of the original message as a separate datagram to the server. The server and the client both know how to calculate the location of the key embedded within the message. The server first notes the length of the datagram it received. Using the formula, it locates the key and subtracts the value of the hexadecimal character represented by the key from the total length of the datagram. This value is the deciphered character sent by the client, represented as an ASCII equivalent value. As each message is received by the server, this process converts the ciphertext messages into plaintext and presents them to the user.

Example:

~Original Message~		~Over the Wire~	
Plaintext	Key	Cipertext	
I (73)	3	(76)	L
L (76)	a	(86)	V
o (111)	0	(111)	o
v (118)	3	(121)	y
e (101)	9	(110)	n
M (77)	1	(78)	N
y (121)	8	(129)	?
D (68)	5	(73)	I
o (111)	f	(126)	~
G (103)	5	(108)	l

Plaintext: "ILoveMyDog"
Ciphertext: "LVoynN?I~l"

We considered multiple things for OPSEC for our covert channel. We looked at the traffic of the target network and decided to use UDP for a few reasons. UDP represented about 20% of the total traffic, meaning our data would not change this percentage considering the total amount of UDP packets contained in the packet capture file. We also wanted our UDP traffic to blend in with all the other UDP traffic. To accomplish this, we generated “fluff,” random ASCII characters intended to disguise our traffic as part of the network. Our traffic looks indistinguishable to the human eye in comparison with UDP traffic already present in the network, and our datagram lengths are similar as well. While our server and client currently send messages very slowly, optimizing our channel would increase the bandwidth without impacting OPSEC. UDP traffic typically is used to transfer lots of data quickly, so increasing our speed of transmission would not be unusual. UDP traffic also only has a data field since the data is handled at a higher level in the network layer. This means neither human nor machine can interpret what the data in the data field is to be used for, disguising its intentions very well since only an application layer can understand it.

Installation:

- Download both the client and server packages.

- If not installed, install python3 to the server machine.

- If not installed, install or acquire a program capable of compiling C# code.

Server use:

- Modify the server.py file server IP and client IP variables to reflect the network topology to be used. These are located near the top of the file and are marked with comments.

- Run the .py file.

Client use:

- Open the client.cs file and modify the server IP and client IP to reflect the network topology to be used. These are located near the top of the file and are marked with comments.

- Compile the client.cs file to an executable.

- Run the client.exe.

- Type in the message to be sent to the server.

- Press enter.