sctPT - An SCTP-Based Pluggable Transport in Python[1]

Alexander Mages[2]

Overview of Functionality

At the highest level, this client takes TCP traffic via SOCKS, proxies it over SCTP[3] to the server, which subsequently proxies it outward in original TCP. The reverse is also true, when the server receives TCP traffic, it is proxied upstream to the client via SCTP, which is further proxied across loopback to the client application via SOCKS. The program has a couple use cases. I will explain each one in non-exhaustive detail below.

1. Client usage: Pluggable Transport using the Tor Browser
    a. The program looks to environmental variables defined by the Tor Browser to decide on configuration parameters
    b. It starts an SCTP socket and a SOCKS over TCP socket. After this, the TCP socket listens and begins to accept connections and the SCTP socket attempts to connect to the server
    c. After both connections are established, data received through SOCKS is sent downstream via SCTP and any data received via SCTP is forwarded upstream through SOCKS
2. Server usage: As a Tor Bridge supporting this transport
    a. The program looks to environmental variables defined by Tor to decide on configuration parameters, as well as the orport
    b. It starts a socket running SCTP and a socket running TCP
    c. Both sockets begin listening and accepting connections
    d. Once both endpoints are connected, data begins proxying
    e. Any data received through SCTP is sent downstream via TCP and any data received via TCP is sent upstream through SCTP
3. Tor-less usage: Currently implemented
    a. The above overviews still apply, but without the use of pyptlib
    b. This functionality is currently implemented for testing purposes

---

[1] https://github.com/Alexander-Mages/sctPT
[2] Advised by Eugene Vasserman
[3] https://en.wikipedia.org/wiki/Stream_Control_Transmission_Protocol

i. To switch to one of the former use cases, uncomment code as described by various comments throughout the following files:
    1. sctPT.py
    2. Server.py
    3. Client.py

Dependencies and Respective Justifications

- pyptlib_python3[4]; a Python 3 fork of pyptlib[5]
    - This library abstracts away a large part of the configuration process, and it handles interfacing with the Tor Browser
    - In this program it is used to gather information from the browser, decide whether it is a client or a server, and report success/failure to the browser after execution
- multiprocessing[6]
    - Standard library with an API very similar to Threading
    - Instead of using thread based parallelism, it uses subprocesses. The rationale for using this library over Threading is the significant performance boost.
        - It achieves this by effectively bypassing the Global Interpreter Lock (GIL) through using multiple processes
- pysocks[7]
    - This library is used for all SOCKS related networking. It enables SOCKS over TCP to be used with an almost identical API to a TCP or SCTP socket object.
- socket
    - Socket is a standard library that is the basis of networking in this program
    - SCTP sockets are created using C/POSIX-like socket options
        - socket.socket(socket.AF_INET, socket.SOCK_STREAM, socket.IPPROTO_SCTP)
        - “IPPROTO_SCTP” or protocol #132
- logging
    - Standard library used to implement logging functionality

[4] https://github.com/Alexander-Mages/pyptlib_python3
[5] https://pyptlib.readthedocs.io/en/master/
[6] https://docs.python.org/3/library/multiprocessing.html
[7] https://pypi.org/project/PySocks/

    - By default, it is set to verbose. By modifying the argparse default in sctPT.py for verbosity this can be changed.
- argparse
    - Standard library used to parse command line arguments

Limitations

- Not tested with the Tor Browser
    - All testing has been done using telnet and ncat's SCTP functionality
        - ncat --sctp 1.1.1.1 6000
        - telnet 1.1.1.1 9050
- There is no data obfuscation (other than base64 encoding) and the data is very much cleartext
- No native Windows support for SCTP
    - There are libraries that could solve this problem but I have not used them nor know if they are reliable or functional
    - I have written a partial SCTP stack in Scapy (client only). It is very much a prototype, but it is functional. Scapy is supported in Windows and as far as I know does not require native support due to the low level nature of the interface.
        - https://github.com/Alexander-Mages/Py-SCTP-Stack
    - https://github.com/dreibh/sctplib
- Connections not built around Tor standards, some features missing
    - No ExtORPORT functionality
    - The connections are not built in accordance with any spec, it was programmed around my current testing tools
    - Along with many other things that I am most likely unaware of
- Networking might not be done in accordance with best practices
    - No reconnect functionality; if a connection dies, the program must be restarted
    - SCTP is used in TCP-like manner; does not implement many of SCTP's more advanced features
    - Server/client accept() vs. connect() decisions are largely arbitrary
    - There is no buffer used, the socket handles all queued data
- PySocks and pyptlib are third-party libraries
    - pysocks is maintained, but pyptlib seems old and had to be "manually" ported to Python 3

    - Primarily used python-modernize[8]
- Not extensively tested, but proven functional on multiple machines
- Not tested on the open internet, only on a local network

Instructions for Running

1. Ensure SCTP is permitted and supported on both the client and the server
2. Ensure dependencies are installed
3. export PYTHONPATH=$PYTHONPATH:~/sctPT/
    a. Importing custom library has been unstable for me, this generally solves the issue
4. Run “python3 sctPT.py --server” on the server
5. Run “telnet 127.0.0.1 9000” on the server
6. Run “python3 sctPT.py” on the client
7. On the client, run “telnet 127.0.0.1 9050”
8. After all of this, the endpoints should be connected, and data can be sent back and forth

Additional Notes

- Concept derived from Tor wiki PT ideas page:
    - https://gitlab.torproject.org/legacy/trac/-/wikis/doc/PluggableTransports/ideas
- SOCKS4 and SOCKS5 are supported. SOCKS5 is default.
- Code structure influenced by obfsproxy[9]

[8] https://python-modernize.readthedocs.io/en/latest/
[9] https://github.com/Yawning/obfsproxy