pySockSSL

Release 0.2

trichimtrich

May 21, 2020

CONTENTS

It is similar to

- Burpsuite
- mitmproxy
- Charles Proxy
- Fiddler
- and many more ...

This package is a simple implementation of SOCKSv4 and SOCKSv5 in Twisted framework.

Including CLI tool + API to:

- Generate and control context for self-signed rootCA + dummy CA for any domain
- Run SOCKSv4 + SOCKSv5 server on any platform
- Support user + pass authentication
- Intercept and process data for TCP/TLS stream

CHAPTER

ONE

CONTENTS

1.1 Installation

Via pypi

```
pip3 install pysockssl
```

Or from source code

```
git clone https://github.com/trichimtrich/pysockssl
cd pysockssl
python3 setup.py install
```

1.2 Usage

You can use this package as standalone cli tool, or import it in your project to do other tasks like capture packet or filter them with your rules.

1.2.1 With CLI

```
$ sockssl --help
Usage: sockssl [OPTIONS] COMMAND [ARGS]...

Options:
--help  Show this message and exit.

Commands:
genca  Generate root CA
run    Run a standalone SOCKS server
```

- Generate rootCA with ON and CN

```
sockssl genca rootCA.crt rootCA.key -org mycompany -cn myCA
```

- Run SOCKSv4 / SOCKSv5 server (no TLS mitm)

```
sockssl run v4
sockssl run v5
```

- Run SOCKSv4 / SOCKSv5 server with TLS mitm

```
sockssl run v4 -c rootCA.crt -k rootCA.key -h 0.0.0.0 -p 9999
sockssl run v5 -c rootCA.crt -k rootCA.key -h 0.0.0.0 -p 9999
```

- Run SOCKSv4 / SOCKSv5 server with TLS mitm + authentication

```
sockssl run v4 -c rootCA.crt -k rootCA.key -h 0.0.0.0 -p 9999 -u user1 -u user2
sockssl run v5 -c rootCA.crt -k rootCA.key -h 0.0.0.0 -p 9999 -u user1 pass1 -u user2␣
↪pass2
```

- Don't forget to trust *rootCA.crt* in you client if you want to see data in TLS stream

1.2.2 With API

- Run a standalone SOCKSv4 server

```
from sockssl.service import SockService
from sockssl.protocol import SOCKSv4
from sockssl import log

# not necessary, but for debug only
log.init(log.DEBUG)

HOST = '0.0.0.0'
PORT = 9999

svc = SockService()
svc.set_host_port(HOST, PORT)
svc.set_protocol(SOCKSv4)

svc.serve_forever()
```

- Capture TLS stream with SOCKSv5

```
from sockssl.certstore import CertStore
from sockssl.service import SockService
from sockssl.protocol import SOCKSv5, ISOCKS
from sockssl import log

# not necessary, but for debug only
log.init(log.ERROR)

HOST = '0.0.0.0'
PORT = 9999

class MySOCKS(SOCKSv5, ISOCKS):
    def _addr(self, addr):
        return '{}:{}:{}'.format(addr.type, addr.host, addr.port)

    def on_connect(self):
        print('Client {} has entered'.format(self._addr(self.addr_client)))

    def on_disconnect(self):
        print('Client {} disconnected'.format(self._addr(self.addr_client)))

    def on_socks_established(self):
        print('Client {} created tunnel with {}'.format(self._addr(self.addr_client),
```

(continues on next page)

(continued from previous page)

```
                                                    self._addr(self.addr_client)))

    def on_recv_client(self, data):
        print('Client {:24} ---> Server {:24}: {:4} bytes: {}'.format(
            self._addr(self.addr_client),
            self._addr(self.addr_server),
            len(data),
            data[:16]
        ))

        return data

    def on_recv_server(self, data):
        print('Client {:24} <--- Server {:24}: {:4} bytes: {}'.format(
            self._addr(self.addr_client),
            self._addr(self.addr_server),
            len(data),
            data[:16]
        ))

        return data

cs = CertStore()
# generate root ca
cs.gen_root_ca(org='myON', cn='myCN')
# save to file, dont forget to trust myroot.crt in client
cs.dump_root_cert('myroot.crt')
cs.dump_root_key('myroot.key')

svc = SockService()
svc.set_host_port(HOST, PORT)
svc.set_cert_store(cs)
svc.set_protocol(MySOCKS)

svc.serve_forever()
```

- You can change the data stream before send to server or back to client
- Other examples can check on `/examples` directory

1.3 API Documentation

This package provides a high-level interface to manage, generate certificates and run a SOCKS service in Twisted framework. The following modules are defined:

1.3.1 CertStore

class sockssl.certstore.**CertStore**(*root_cert=None*, *root_key=None*)
Manage Root Certificate Authority, and generate Dummy Certificate

__init__(*root_cert=None*, *root_key=None*)
Create a CertStore instance

Parameters

- **root_cert** (*OpenSSL.crypto.X509, optional*) – rootCA certificate - x509 instance. Defaults to None.
- **root_key** (*OpenSSL.crypto.PKey, optional*) – rootCA private key - PKey instance. Defaults to None.

dummy_ctx(*sni*)
Get SSL context of dummy ceritficate from SNI list. Use for passing to twisted StartTLS

Parameters **sni** (*List[str]*) – List of domain name, ips

Returns twisted.internet._sslverify.OpenSSLCertificateOptions

dump_root_cert(*filename*, *format='PEM'*)
Dump rootCA certificate to file

Parameters

- **filename** (*str*) – path to certificate file
- **format** (*str, optional*) – format of certificate, support PEM/DER. Defaults to "PEM".

dump_root_key(*filename*, *format='PEM'*)
Dump rootCA private key to file

Parameters

- **filename** (*str*) – path to private key file
- **format** (*str, optional*) – format of private key, support PEM/DER. Defaults to "PEM".

gen_root_ca(*org*, *cn*, *exp=94608000*, *key_size=2048*)
Generate rootCA certificate + privatekey and store in root_key, root_cert

Parameters

- **org** (*str*) – Organization name
- **cn** (*str*) – Common Name
- **exp** (*int, optional*) – Expiration time in second. Defaults to 94608000 == 3 years
- **key_size** (*int, optional*) – RSA key size (1024/2048/...). Defaults to 2048.

load_root_cert(*filename*, *format='PEM'*)
Load rootCA certificate from file

Parameters

- **filename** (*str*) – path to certificate file
- **format** (*str, optional*) – format of certificate, support PEM/DER. Defaults to "PEM".

load_root_key(*filename*, *format='PEM'*)
Load rootCA private key from file

Parameters

- **filename** (*str*) – path to private key file
- **format** (*str, optional*) – format of private key, support PEM/DER. Defaults to "PEM".

root_cert = None
OpenSSL.crypto.x509 (certificate) instance of rootCA

root_ctx()
Get SSL context of rootCA ceritficate. Use for passing to twisted StartTLS

Returns twisted.internet._sslverify.OpenSSLCertificateOptions

root_key = None
OpenSSL.crypto.PKey (private key) instance of rootCA

sockssl.certstore.**find_client_hello**(*packet*)
Parse ClientHello Packet of TLS 1.2

Parameters **packet** (*bytes*) – stream of packet

Returns TlsClientHello

sockssl.certstore.**is_sni**(*obj*)
Check if an object is SNI instance

Parameters **obj** (*object*) –

Returns bool

1.3.2 Service

class sockssl.service.**SockService**(*host=None*, *port=None*, *cert_store=None*, *protocol=None*, *users=None*, *data=None*)
Service class to manage certstore and serve your socks protocol

__init__(*host=None*, *port=None*, *cert_store=None*, *protocol=None*, *users=None*, *data=None*)
Create a SockService instance

Parameters

- **host** (*Optional[str], optional*) – Interface as ip or hostname. Defaults to None.
- **port** (*Optional[int], optional*) – Port in integer. Defaults to None.
- **cert_store** (*Optional[CertStore], optional*) – Instance of CertStore Class. Defaults to None.
- **protocol** (*Any, optional*) – Class of protocol you want to serve. Defaults to None.
- **users** (*Any, optional*) – Auth users data of protocol. List[str] for SOCKSv4, List[Tuple[str, str]] for SOCKSv5. Defaults to None.
- **data** (*Any, optional*) – Global data for that protocol (like auth data), will pass to SockFactory. Defaults to None.

serve_forever()
Listen TCP and run reactor forever

set_cert_store(*cert_store=None*)

Set CertStore instance to intercept TLS traffic. Set to None if you don't want to do TLS mitm.

Parameters **cert_store** (*Optional[CertStore], optional*) – Instance of CertStore Class. Defaults to None.

set_data(*data=None*)
Set global data share between connection

Parameters **data** (*Any, optional*) – Global data for that protocol (like auth data), will pass to SockFactory. Defaults to None.

set_host_port(*host*, *port*)
Set listen interface and port for service

Parameters

- **host** (*str*) – Interface as ip or hostname
- **port** (*int*) – Port in integer

set_protocol(*protocol*, *users=None*)
Set protocol and users data for service to serve, usually SOCKv4 or SOCKSv5.

Parameters

- **protocol** (*Any*) – Class of protocol you want to serve
- **users** (*Any, optional*) – Auth users data of protocol. List[str] for SOCKSv4, List[Tuple[str, str]] for SOCKSv5. Defaults to None.

1.3.3 Factory

class sockssl.factory.**SockFactory**(*protocol*, *users=None*, *cert_store=None*, *data=None*)
Twisted framework Protocol Factory: store global context and produce protocol handler for each connection

__init__(*protocol*, *users=None*, *cert_store=None*, *data=None*)
Create a SockFactory instance

Parameters

- **protocol** (*Any*) – Class of protocol. SOCKSv4, SOCKSv5, ... or your class
- **users** (*Any, optional*) – Auth users data of protocol. Defaults to None.
- **cert_store** (*Optional[CertStore], optional*) – Instance of CertStore to do TLS mitm. Defaults to None.
- **data** (*Any, optional*) – Global data variable share between connection. Defaults to None.

1.3.4 Protocols

class sockssl.protocol.**SOCKSv4**(*logging=None*, *reactor=<twisted.internet.epollreactor.EPollReactor object>*)
Implementation of SOCKSv4 protocol compatiables with ISOCKS interface

class sockssl.protocol.**SOCKSv5**(*reactor=<twisted.internet.epollreactor.EPollReactor object>*)
Implementation of SOCKSv5 protocol compatiables with ISOCKS interface

1.3.5 Interface

class sockssl.protocol.isocks.**ISOCKS**
Interface to interact with SOCKS protocol

addr_client: <InterfaceClass twisted.internet.interfaces.IAddress> = None
Client information

addr_server: <InterfaceClass twisted.internet.interfaces.IAddress> = None
Server information

factory: <InterfaceClass twisted.internet.interfaces.IProtocolFactory> = None
Factory instance of current protocol connection

on_connect()
Trigger when a client connected

on_disconnect()
Trigger when client disconnected

on_recv_client(*data*)
Process data sent from client to server

Parameters **data** (*bytes*) – Data from client

Returns *bytes* – Data will be sent to server

on_recv_server(*data*)
Process data sent back from server to client

Parameters **data** (*bytes*) – Data got from server

Returns *bytes* – Data will be sent back to client

on_socks_established()
Trigger when a SOCKS tunnel established

on_socks_failed()
Trigger when a SOCKS connection failed to establish

1.4 More

Please check out the source code at Github

Any bug or issue please raise via git tracker or email me at trichimtrich@gmail.com

There are also examples in the pySockSSL repository that may help you getting started.

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX

Symbols

A

C

D

F

G

I

L

M

O

R

S