دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)
دانشکده مهندسی کامپیوتر

گزارش پروژه جبرانی امنیت اطلاعات
SSH Chatroom

گردآورنده:
رهام زنده دل نوبری

استاد درس:
دکتر شهریاری

بهمن ۱۴۰۱

کتابخانه Twisted

کتابخانه Twisted یکی از کتابخانه‌هایی است در زبان برنامه‌نویسی Python که با استفاده از آن می‌توان پروتکل‌های ارتباطی گوناگون و پروتکل‌های امنیتی گوناگون را پیاده‌سازی کرد. این کتابخانه می‌تواند پروتکل‌های TLS، SSL و SSH را نیز پیاده‌سازی کند. هدف ما در این پروژه پیاده‌سازی یک Chatroom با استفاده از ساختن جلسه SSH و احراز هویت توسط سرور SSH است.

در ابتدا بایستی سه کتابخانه زیر را نصب کنیم:

```
>>pip install twisted
>>pip install pyOpenSSL
>>pip install service_identity
```

سپس می‌توانیم از کتابخانه Twisted استفاده کنیم و پروتکل SSH را پیاده‌سازی کنیم. هر پروتکل SSH در کتابخانه Twisted به چند چیز نیازمند است:

۱- SSH Factory: نقش AS را در کربروس دارد و پایگاه داده‌ای از کلیدهای عمومی و خارجی خود و کلاینت‌ها را دارد. همچنین لیستی از کاربران و رمزهای آنها را نیز در خود دارد. همچنین این تابع نقش ساخته شده قلمرو سرور و جلسات آن را نیز دارد.

۲- SSH Protocol: پروتکل‌هایی که در جلسه‌های خود آنها را اجرایی می‌کنیم. این پروتکل‌ها می‌توانند توسط خود برنامه‌نویس طراحی شوند و یا از پروتکل‌های پیش‌فرض کتابخانه Twisted می‌توانیم استفاده کنیم.

۳- SSH Avatar: کلاینت‌ها و یوزرها همان SSH Avatar ها هستند که در خود لیستی از جلسات(Session) را دارند. این جلسات با استفاده از SSH Protocol های تعریف شده به اصطلاح Wrap می‌شوند و در واقع تمام عملکردهایی که ما برای پروتکل‌ها تعریف کرده‌ایم، برای جلسه بین سرور و کلاینت نیز قابل استفاده می‌شود. نام دیگر آن ConchUser است.

۴- SSH Realm: قلمرو سرور را بایستی مشخص کنیم. در این قلمرو لیستی از SSH Avatar خواهیم داشت که در واقع همان کلاینت‌های متصل به سرور هستند.

۵- Public and Private RSA Key: برای رمزگذاری جلسه‌های کلاینت‌ها و پیام‌های رد و بدل شده استفاده می‌شود.

مرحله اول

در ابتدا کلیدی را در ترمینال به شکل زیر ایجاد می‌کنیم:

```
C:\Users\asus>ssh-keygen -t rsa -b 4096 -C "aut.ac.ir"
Generating public/private rsa key pair.
Enter file in which to save the key (C:\Users\asus/.ssh/id_rsa):
C:\Users\asus/.ssh/id_rsa already exists.
Overwrite (y/n)? y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in C:\Users\asus/.ssh/id_rsa.
Your public key has been saved in C:\Users\asus/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:dY167ek2fVH/L6ycm4jXFyl0RLqPLBwCQ52w+zWRlrk aut.ac.ir
The key's randomart image is:
+---[RSA 4096]----+
|      oo .   ..  |
|     . .o  + +.  |
|      +   B +..  |
|       + o =.o. .|
|      . S E.o...o|
|       . + =.+oo.|
|         . o.oo+oo|
|          ..+.+* +|
|         ... *=.o+|
+----[SHA256]-----+
```

همانطور که می‌بینیم دستور فوق کلید عمومی و خصوصی مورد نظر را تولید می‌کند. این کلید ۴۰۹۶ بیتی است و با الگوریتم RSA ساخته می‌شود.

مرحله دوم

سپس هر کدام از بخش‌های مورد نیاز برای پروتکل SSH پیاده‌سازی می‌کنیم.

```
class SSHProtocol(recvline.HistoricRecvLine):
    def __init__(self, user):
        self.user = user
```

در ابتدا کلاس SSHProtocol را پیاده‌سازی می‌کنیم. در این کلاس بایستی سه بخش اصلی را توسعه دهیم:

۱. connectionMade: که در آن توانسته‌ایم یک جلسه با کلاینت ایجاد کنیم و ارتباط SSH برقرار است.
۲. connectionLost: که در آن به هر دلیلی ارتباط سرور با کلاینت قطع شده است.
۳. dataReceived یا lineReceived: که در آن داده‌ای از سمت کلاینت به ما که سرور هستیم رسیده است.

هر کدام از بخش‌های فوق به ترتیب در کدهای زیر نوشته شده‌اند:

```
def connectionMade(self):
        recvline.HistoricRecvLine.connectionMade(self)
        self.terminal.write("Welcome to the SSH Chatroom!")
        self.terminal.nextLine()
        self.command_help()
        self.showPrompt()
        self.user.realm.clients[self.user.username] = self
        with open('logfile.log', 'a') as f:
                        f.write(f"Client[{self.user.username.decode('utf-8')}]
connectection success at {datetime.now()}\n")
```

```
def connectionLost(self, reason):
        self.user.realm.clients.pop(self.user.username)
        if len(self.user.realm.clients) > 0:
            for client in self.user.realm.clients:
                self.user.realm.clients[client].terminal.write(f"Client[{self.use
r.username.decode('utf-8')}] disconnected!")
                self.user.realm.clients[client].terminal.nextLine()
                self.user.realm.clients[client].showPrompt()
        with open('logfile.log', 'a') as f:
            f.write(f"Client[{self.user.username.decode('utf-8')}] disconnected at
{datetime.now()}\n")
```

```
def lineReceived(self, line):
        line = line.strip()
        if line:
            print(line)
            with open('logfile.log', 'a') as f:
```

```
                f.write(f"Client[{self.user.username.decode('utf-8')}]: {line} at
{datetime.now()}\n")
            cmdAndArgs = line.split()
            cmd = cmdAndArgs[0]
            args = cmdAndArgs[1:]
            func = self.getCommandFunc(cmd)
            if func:
                try:
                    func(*args)
                except Exception as e:
                    self.terminal.write(f"Error: {e}")
                    self.terminal.nextLine()
            else:
                self.terminal.write("No such command!")
                self.terminal.nextLine()
        self.showPrompt()
```

در قسمت گرفتن داده از سمت کلاینت، به عنوان سرور منتظر دستوری هستیم و انتظار داریم یکی از تابع‌هایی که با نام‌هایی که با command_ شروع می‌شوند، صدا زنده شده باشد. با دستور getCommandFunc می‌توانیم دستور داده شده توسط کلاینت را تشخیص دهیم و تابع مربوط به آن را اجرا کنیم.

توابع پیاده‌سازی شده در پروتکل عبارتند از:

```
def command_help(self):
        """Showing the available commands based on the function names starting
with command_"""
        publicMethods = filter(
            lambda funcname: funcname.startswith('command_'), dir(self))
        commands = [cmd.replace('command_', '', 1) for cmd in publicMethods]
        self.terminal.write(f"Commands: {' '.join(commands)}")
        self.terminal.nextLine()
```

دستورات قابل اجرا را براساس توابعی که با command_ شروع می‌شوند شناسایی می‌کنیم و آنها را برای کلاینت چاپ می‌کنیم. برای ارسال اطلاعات از دستور self.terminal.write() استفاده می‌کنیم. این دستور در جلسه مشترک بین سرور و کلاینت شروع به نوشتن اطلاعات می‌کند.

```
 def command_echo(self, *args):
        self.terminal.write(b' '.join(args))
        self.terminal.nextLine()
```

دستور فوق نوشته‌ای را به عنوان ورودی می‌گیرد و در ترمینال جلسه چاپ می‌کند.

```
def command_send(self, *args):
        """Sending a message to all the connected clients in our realm."""
                self.user.realm.clients[args[0]].terminal.write(f"Message   from
Client[{self.user.username.decode('utf-8')}] : ")
        self.user.realm.clients[args[0]].terminal.write(b' '.join(args[1:]))
```

```
        self.user.realm.clients[args[0]].terminal.nextLine()
        self.user.realm.clients[args[0]].showPrompt()
```

دستور فوق پیامی را برای تمامی کلاینت‌هایی که در SSHRealm هستند ارسال می‌کند. (چون ما درون یک Chatroom هستیم.

```
def command_quit(self):
        self.terminal.write("Goodbye!")
        self.terminal.nextLine()
        self.terminal.loseConnection()
        self.user.realm.clients.pop(self.user.username)
        if len(self.user.realm.clients) > 0:
            for client in self.user.realm.clients:
                self.user.realm.clients[client].terminal.write(f"Client[{self.use
r.username.decode('utf-8')}] disconnected!")
                self.user.realm.clients[client].terminal.nextLine()
                self.user.realm.clients[client].showPrompt()
        with open('logfile.log', 'a') as f:
           f.write(f"Client[{self.user.username.decode('utf-8')}] disconnected at
{datetime.now()}\n")
```

دستور فوق زمانی اجرا می‌شود که کلاینت دستور quit را برای سرور فرستاده باشد.

```
def command_clear(self):
        self.terminal.reset()
```

نوشته های ترمینال جلسه را پاک می‌کند.

```
@implementer(ISession)
class SSHAvatar(avatar.ConchUser):
@implementer(portal.IRealm)
class SSHRealm(object):
```

دو کلاس فوق را نیز در کد تعریف می‌کنیم و آماده می‌شویم که سرور را اجرا کنیم.

نیاز است که SSH Factory و پرتال احراز هویت آن را در شروع برنامه اجرا کنیم:

```
sshFactory = factory.SSHFactory()
    sshFactory.portal = portal.Portal(SSHRealm())

    users = {'roham': b'123', 'parham': b'456'}
    sshFactory.portal.registerChecker(
        checkers.InMemoryUsernamePasswordDatabaseDontUse(**users))
```

همچنین یک دیکشنری از کاربرها و رمز آنها را برای پایگاه داده‌ای SSH Factory برای احراز هویت کلاینت ها می‌سازیم.

```
 pubKey, privKey = getRSAKeys()
    sshFactory.publicKeys = {b'ssh-rsa': pubKey}
    sshFactory.privateKeys = {b'ssh-rsa': privKey}
```

سپس کلیدها را می‌گیریم و آماده رمزگذاری می‌شویم.

```
reactor.listenTCP(22222, sshFactory)
    reactor.run()
```

reactor ها نقش یک حلقه بینهایت یا چرخه زندگی را دارد و تا زمانی که سرور به مشکل بر نخورد همواره برپا خواهد بود.

مرحله سوم

در این مرحله اقدام به تست کد نوشته شده می‌کنیم:

ابتدا سرور را اجرا می‌کنیم.

```
PS D:\Uni\Information Security\SSH Chatroom> & C:/Users/asus/AppData/Local/Programs/Python/Python39/python.exe "d:/Uni/Information Security/SSH Chatroom/sshserver.py"
C:\Users\asus\AppData\Local\Programs\Python\Python39\lib\site-packages\twisted\conch\ssh\transport.py:97: CryptographyDeprecationWarning: Blowfish has been deprecated
  b"blowfish-cbc": (algorithms.Blowfish, 16, modes.CBC),
C:\Users\asus\AppData\Local\Programs\Python\Python39\lib\site-packages\twisted\conch\ssh\transport.py:101: CryptographyDeprecationWarning: CAST5 has been deprecated
  b"cast128-cbc": (algorithms.CAST5, 16, modes.CBC),
C:\Users\asus\AppData\Local\Programs\Python\Python39\lib\site-packages\twisted\conch\ssh\transport.py:106: CryptographyDeprecationWarning: Blowfish has been deprecated
  b"blowfish-ctr": (algorithms.Blowfish, 16, modes.CTR),
C:\Users\asus\AppData\Local\Programs\Python\Python39\lib\site-packages\twisted\conch\ssh\transport.py:107: CryptographyDeprecationWarning: CAST5 has been deprecated
  b"cast128-ctr": (algorithms.CAST5, 16, modes.CTR),
```

همانطور که قابل مشاهده است، اخطارهایی مبتنی بر باطل شدن برخی روش‌های رمزنگاری را مشاهده می‌کنیم.

حال اقدام به اتصال شدن به سرور می‌کنیم. با دستور زیر می‌توانیم از ترمینال به سرور متصل شویم:

```
PS D:\Uni\Information Security\SSH Chatroom> ssh roham@localhost -p 22222
roham@localhost's password:
```

همانطور که مشاهده می‌کنیم برای کاربر roham، سرور از ما درخواست رمز کرده است. رمز کاربر roham مقدار ۱۲۳ است. رمز را وارد می‌کنیم و به جلسه SSH وارد می‌شویم:

```
>>> Welcome to the SSH Chatroom!
Commands: clear echo help quit send
$
```

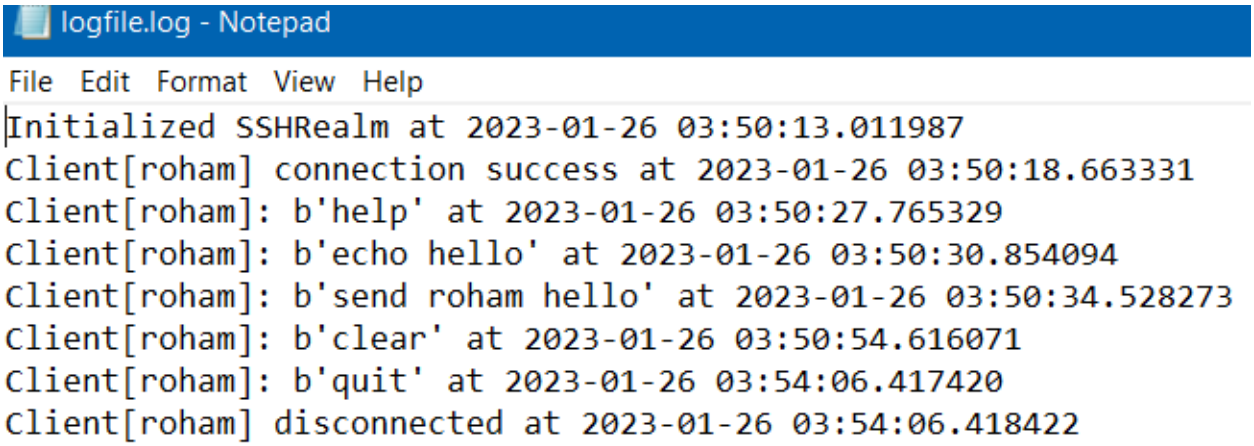
سرور برای ما پیام Welcome می‌فرستد و با دستور command_help، تمامی دستورهای ممکن را چاپ می‌کند.

حال می‌توانیم از دستورهای لیست شده هر کدام را تست کنیم:

```
>>> Welcome to the SSH Chatroom!
Commands: clear echo help quit send
$ help
Commands: clear echo help quit send
$ echo hello
hello
$ send roham hello
Message from Client[roham] : hello
$
```

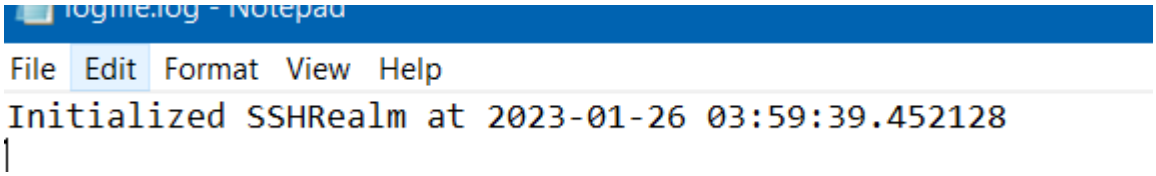
دستور clear ترمینال را خالی می‌کند و دستور quit نیز از ترمینال SSH Chatroom خارج می‌شود.

فایل logfile.log را مشاهده می‌کنیم و خواهیم داشت:



دستوری که برای ما اهمیت دارد، دستور send است. با استفاده از این دستور می‌توانیم به کلاینتی که در قلمرو سرور هست و نام آن را می‌دانیم پیامی بدهیم.

سرور را از اول اجرا می‌کنیم و این دفعه دو کلاینت را هم‌زمان به آن متصل می‌کنیم. خواهیم داشت:



در ابتدا سرور را اجرا کرده‌ایم.

```
Connection to localhost closed.
PS D:\Uni\Information Security\SSH Chatroom> ssh roham@localhost -p 22222
roham@localhost's password:
```

```
>>> Welcome to the SSH Chatroom!
Commands: clear echo help quit send
$
```

سپس کلاینت roham را لاگین کرده و به سرور متصل می‌شویم.

خواهیم داشت:

logfile.log - Notepad

File Edit Format View Help

```
Initialized SSHRealm at 2023-01-26 03:59:39.452128
Client[roham] connection success at 2023-01-26 04:00:34.725436
```

حال کلاینت parham را نیز لاگین می‌کنیم و به سرور متصل می‌شویم.

```
Connection to localhost closed.
PS D:\Uni\Information Security\SSH Chatroom> ssh parham@localhost -p 22222
parham@localhost's password:
```

```
>>> Welcome to the SSH Chatroom!
Commands: clear echo help quit send
$
```

خواهیم داشت:

File Edit Format View Help

```
Initialized SSHRealm at 2023-01-26 03:59:39.452128
Client[roham] connection success at 2023-01-26 04:00:34.725436
Client[parham] connection success at 2023-01-26 04:02:36.818425
```

حال پیامی را از کلاینت roham به کلاینت parham می‌فرستیم:

```
>>> Welcome to the SSH Chatroom!
Commands: clear echo help quit send
$ send parham hello good sir
$
```

```
>>> Welcome to the SSH Chatroom!
Commands: clear echo help quit send
$ Message from Client[roham] : hello good sir
```

پیامی را نیز از سمت کلاینت parham برای کلاینت roham ارسال می‌کنیم:

```
>>> Welcome to the SSH Chatroom!
Commands: clear echo help quit send
$ Message from Client[roham] : hello good sir
send rohamtings!
$
```

```
>>> Welcome to the SSH Chatroom!
Commands: clear echo help quit send
$ send parham hello good sir
$ Message from Client[parham] : greetings!
```

همانطور که مشاهده می‌کنیم، سرور SSH Chatroom به درستی پیام‌ها را ارسال می‌کند.

در فایل logfile.log داریم:

logfile.log - Notepad

File Edit Format View Help

```
Initialized SSHRealm at 2023-01-26 03:59:39.452128
Client[roham] connection success at 2023-01-26 04:00:34.725436
Client[parham] connection success at 2023-01-26 04:02:36.818425
Client[roham]: b'send parham hello good sir' at 2023-01-26 04:12:39.278211
Client[parham]: b'send roham greetings!' at 2023-01-26 04:14:56.898264
```

در نهایت نیز هر کدام از کلاینت‌ها تصمیم به quit کردن و در نتیجه disconnect شدن می‌گیرند:

```
>>> Welcome to the SSH Chatroom!
Commands: clear echo help quit send
$ Message from Client[roham] : hello good sir
send rohamtings!
$ Client[roham] disconnected!
$
```

اگر با هر کلاینت quit کنیم، بقیه کلاینت‌های حاضر در Chatroom از قطعی اتصال این کاربر اطلاع خواهند یافت.

در نهایت جفت کلاینت‌ها quit می‌کنند و داریم: