مهندسی اینترنت: تمرین عملی HTTP **سیّد سبحان عابدی 9731707**

سوال 1)

جواب

زمانی که یک سرویس روی شبکه داخل و پشت NAT اجرا می‌شود، به صورت پیشفرض از خارج شبکه داخلی نمی‌توان به آن دسترسی داشت مگر اینکه Port Forwarding انجام شود. ولی اگر فرض کنیم در شبکه داخلی از NAT هم استفاده نکرده باشیم، میتوانیم برای Edge Router این شبکه داخلی یکسری قوانین مسیریابی قراردهیم که ارتباط یک سرویس را (یک پورت از یک میزبان به خصوص) از شبکه خارجی قطع کند.

سوال 2)

در یک اتصال TLS، Server همواره باید برای کاربر Certificate خود را ارسال کند ولی Server همچنین این امکان را دارد که از کاربر نیز یک Certificate درخواست کند و اگر کاربر پیام no digital certificate alert را برای Server برگرداند، Server می‌تواند اتصال را برقرار نکند و SSL/TLS Handshake شکست می‌خورد. از این رو با استفاده از لایه SSL/TLS می‌توان برای امن کردن سرویس و سنجش هویت کاربران استفاده کرد.

عملیات TLS Termination توسط یک Reverse Proxy انجام می‌شود. به این صورت که کاربر یک اتصال برمبنای TLS با این Proxy برقرار می‌کند و به این صورت Proxy هویت کاربر را تایید می‌کند و یک اتصال امن با او برقرار می‌کند. سپس این Proxy پیام‌های رمزنگاری شده‌ای را که از سمت کاربر دریافت می‌کند، رمزگشایی می‌کند و پیام‌های نهایی که بر اساس HTTP ساده (یا پروتکل های ارتباطاتی ساده دیگر) هست را برای Server روی شبکه محلی ارسال می‌کند و نتیجه‌ای که از Server دریافت می‌کند را براساس کلیدهای ارتباطاتی کاربر رمزگذاری می‌کند و پیام رمزگذاری شده بر اساس پروتکل HTTPS را برای کاربر ارسال می‌کند.

سوال 3)

برای این بخش پروژه httpbin برروی docker و با پورت 8081 بالا آورده شده و یک Reverse Proxy نوشته شده با Python پیام‌ها را از روی پورت 80 دریافت می‌کند و در صورت صحت Authentication پیام را برای پورت 8081 ارسال می‌کند و پاسخ را به کاربر برمی‌گرداند.

Http Server ایجاد شده، به صورت پیشفرض بر روی پورت 80 گوش می‌کند و پیام‌ها را برای آدرس 127.0.0.1:8081 برمی‌گرداند ولی در هنگام صدازدن برنامه Python می‌توان با استفاده از پارامترهای port -- و hostname-- این دو مقدار پیشفرض را تغییر داد.

برای تولید JWT ها در این برنامه از کتابخانه jwcrypto استفاده شده. متاسفانه tokenهایی که توسط این کتابخانه و سایت jwt.io با اطلاعات کاملا یکسان ایجاد می‌شدند، متفاوت بودن. بخش Hash مربوط به Header یکسان هست ولی Hashای که برای Payload/Claims ایجاد می‌کنند اندکی متفاوت هست که باعث می‌شد tokenهای ایجاد شده توسط این وبسایت متفاوت از token مورد انتظار Reverse Proxy باشد و باعث Authentication Failure می‌شود. برای همین در هربار اجرای برنامه یک کلید متقارن 256 بیتی ایجاد می‌کند و بر اساس آن JWT از پیش نوشته شده در برنامه (Header و Claimها از قبل داخل برنامه مشخص شده‌اند) امضاء می‌شود. برای اینکه امتحان برنامه راحت باشد، برنامه چند دستور curl با JWT امضا شده خروجی می‌دهد که می‌توان از آنها برای آزمایش برنامه استفاده کرد. همچنین یک exp claim به گونه‌ای تنظیم می‌شود که این JWT پس از 2 دقیقه منقضی شود و پس از آن پیام 401 برگردانده می‌شود. همچنین برای امنیت بیشتر کلیدهای JWT رمزنگاری می‌شوند.

کد برنامه:

```
1.  #!/usr/bin/env python3
2.  import argparse
3.  import json
4.  import requests
5.  import sys
6.  import time
7.  from http.server import BaseHTTPRequestHandler, HTTPServer
8.  from socketserver import ThreadingMixIn
9.
10. from jwcrypto import jwt, jwk
11.
12. hostname = '127.0.0.1:8081'
13. sym_key = ''
14.
15.
16. class ProxyHTTPRequestHandler(BaseHTTPRequestHandler):
17.     protocol_version = 'HTTP/1.1'
18.
19.     def send_request(self, request_method, body=True):
20.         sent = False
21.         try:
22.             url = 'http://{}{}'.format(hostname, self.path)
23.             request_body = None
24.             content_len = int(self.headers.get('content-length', 0))
25.             request_body = self.rfile.read(content_len)
26.             print()
27.             if self.check_authentication():
28.                 self.headers.replace_header("Host", hostname)
29.                 print("Request Header")
30.                 print(self.headers)
31.                 resp = requests.request(request_method, url, data=request_body, headers=self.headers,
    verify=False)
32.                 sent = True
33.
34.                 self.log_request(resp.status_code)
35.                 self.send_response_only(resp.status_code, None)
36.                 self.send_resp_headers(resp)
37.
38.                 if body:
39.                     self.wfile.write(resp.content)
40.             else:
41.                 self.send_error(401, 'Unauthorized')
42.                 sent = True
43.             return
44.         finally:
45.             if not sent:
46.                 self.send_error(404, 'error trying to proxy')
47.
48.     def do_HEAD(self):
49.         self.send_request('HEAD', body=False)
50.
51.     def do_GET(self, body=True):
52.         self.send_request('GET', body)
53.
54.     def do_POST(self, body=True):
55.         self.send_request('POST', body)
56.
57.     def do_DELETE(self, body=True):
58.         self.send_request('DELETE', body)
59.
60.     def do_PUT(self, body=True):
61.         self.send_request('PUT', body)
62.
63.     def do_PATCH(self, body=True):
64.         self.send_request('PATCH', body)
65.
66.     def check_authentication(self):
67.         auth = self.headers.get('Authorization', None)
68.         if auth is None:
69.             print("No Authentication")
70.             return False
71.
72.         auth_list = auth.split(' ')
```

```
        if len(auth_list) != 2 or auth_list[0] != 'Bearer':
            print("Bad Authorization Header")
            return False
        try:
            encrypted_token = jwt.JWT(key=sym_key, jwt=auth_list[1])
            simple_token = jwt.JWT(key=sym_key, jwt=encrypted_token.claims)
        except Exception as e:
            print("Bad JWT")
            return False
        if simple_token.serialize() != encrypted_token.claims:
            return False
        del self.headers['Authorization']
        try:
            exp_time = json.loads(simple_token.claims)['exp']
        except KeyError:
            print("No Expiry Time!")
            return False

        if exp_time < int(time.time()):
            print("Token Is Expired")
            return False

        return True

    def send_resp_headers(self, resp):
        respheaders = resp.headers
        print('Response Header')
          for key in respheaders:
              if key not in ['Content-Encoding', 'Transfer-Encoding', 'content-encoding', 'transfer-
    encoding',
                             'content-length', 'Content-Length']:
                  print(key, respheaders[key])
                  self.send_header(key, respheaders[key])
          self.send_header('Content-Length', str(len(resp.content)))
          self.end_headers()


def parse_args(argv=None):
    if argv is None:
        argv = sys.argv[1:]
    parser = argparse.ArgumentParser(description='Proxy HTTP requests')
    parser.add_argument('--port', dest='port', type=int, default=80,
                        help='serve HTTP requests on specified port (default: 80)')
    parser.add_argument('--hostname', dest='hostname', type=str, default='127.0.0.1:8081',
                        help='hostname to be processd (default: 127.0.0.1:8081)')
    args = parser.parse_args(argv)
    return args


class ThreadedHTTPServer(ThreadingMixIn, HTTPServer):
    """Handle requests in a separate thread."""


def main(argv=None):
    if argv is None:
        argv = sys.argv[1:]
    global hostname, sym_key
    args = parse_args(argv)
    hostname = args.hostname
    sym_key = jwk.JWK.generate(kty='oct', size=256)
    cur_time = int(time.time())
    token = jwt.JWT(header={'alg': 'HS256'},
                    claims={'iss': '9731707', 'sub': '1234567890', 'aud': '9733048', 'exp':
    cur_time+40})
    token.make_signed_token(sym_key)

    eToken = jwt.JWT(header={"alg": "A256KW", "enc": "A256CBC-HS512"},
                     claims=token.serialize())
    eToken.make_encrypted_token(sym_key)
    print('http server is starting on {} port {}...'.format(args.hostname, args.port))
    server_address = ('127.0.0.1', args.port)
    httpd = ThreadedHTTPServer(server_address, ProxyHTTPRequestHandler)
    print('http server is running as reverse proxy\n')
    print('Try with curl:')
    port = args.port
    if port == 80:
        port = ''
```

```
148.      else:
149.          port = ':' + str(port)
150.      print('curl --proto HTTP --head -H "accept: application/json" -H "Authorization: Bearer {}"
    127.0.0.1{}'.
151.            format(eToken.serialize(), port))
152.      print('curl --proto HTTP -i -X GET -H "accept: application/json" -H "Authorization: Bearer {}"
    127.0.0.1{}/get'.
153.            format(eToken.serialize(), port))
154.      print('curl --proto HTTP -i -X POST -H "accept: application/json" -H "Authorization: Bearer {}"
    127.0.0.1{}/post'.
155.            format(eToken.serialize(), port))
156.      print('curl --proto HTTP -i -X DELETE -H "accept: application/json" -H "Authorization: Bearer {}"
    127.0.0.1{}/delete'.
157.            format(eToken.serialize(), port))
158.      print('curl --proto HTTP -i -X PUT -H "accept: application/json" -H "Authorization: Bearer {}"
    127.0.0.1{}/put'.
159.            format(eToken.serialize(), port))
160.      print('curl --proto HTTP -i -X PATCH -H "accept: application/json" -H "Authorization: Bearer {}"
    127.0.0.1{}/patch'.
161.            format(eToken.serialize(), port))
162.      print()
163.      httpd.serve_forever()
164.
165.
166.  if __name__ == '__main__':
167.      main()
168.
```