Написать http прокси сервер и простой сканер уязвимости на его основе (часть функционала burp suite без gui). Язык программирования любой, можно использовать любые библиотеки, кроме тех, которые реализуют функционал http прокси (<https://mitmproxy.org> и тп). Пример реализации 1 и 2 пунктов (слегка устаревшие) <https://github.com/john-pentest/fproxy>

<https://github.com/kr/mitm/blob/master/mitm.go>

Обязательно сделать задание в виде докер контейнера (или docker-compose из нескольких контейнеров), в котором прокси слушает на порту 8080, на порту 8000 веб-апи (например,

/requests – список запросов

/requests/id – вывод 1 запроса

/repeat/id – повторная отправка запроса

/scan/id – сканирование запроса)

### 1. Проксирование HTTP запросов – 20 баллов

Должны успешно проксироваться HTTP запросы. Команда curl -x http://127.0.0.1:8080 http://mail.ru (8080 – порт, на котором запущена программа) должна возвращать

<html>

<head><title>301 Moved Permanently</title></head>

<body bgcolor="white">

<center><h1>301 Moved Permanently</h1></center>

<hr><center>nginx/1.14.1</center>

</body>

</html>

На вход прокси приходит запрос вида

GET http://mail.ru/ HTTP/1.1

Host: mail.ru

User-Agent: curl/7.64.1

Accept: \*/\*

Proxy-Connection: Keep-Alive

Необходимо:

- считать хост и порт из первой строчки

- заменить путь на относительный

- удалить заголовок Proxy-Connection

Отправить на считанный хост (mail.ru:80) получившийся запрос

GET / HTTP/1.1

Host: mail.ru

User-Agent: curl/7.64.1

Accept: \*/\*

Перенаправить все, что будет получено в ответ

HTTP/1.1 301 Moved Permanently

Server: nginx/1.14.1

Date: Sat, 12 Sep 2020 08:04:13 GMT

Content-Type: text/html

Content-Length: 185

Connection: close

Location: https://mail.ru/

<html>

<head><title>301 Moved Permanently</title></head>

<body bgcolor="white">

<center><h1>301 Moved Permanently</h1></center>

<hr><center>nginx/1.14.1</center>

</body>

</html>

Убедиться, что

- проксируются все типы запросов (GET, POST, HEAD, OPTIONS)

- проксируются все заголовки

- корректно возвращаются все коды ответов (200, 302, 404)

### 2. Проксирование HTTPS запросов – 20 баллов

Должны успешно проксироваться https запросы. В настройках браузера указать http/https прокси, добавить в ОС корневой сертификат, все сайты должны работать корректно.

Запрос curl -x http://127.0.0.1:8080 https://mail.ru (8080 – порт, на котором запущена программа) должен обрабатываться следующим образом:

- На 8080 порт придет в открытом виде запрос CONNECT <https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/CONNECT>

CONNECT mail.ru:443 HTTP/1.1

Host: mail.ru:443

User-Agent: curl/7.64.1

Proxy-Connection: Keep-Alive

Необходимо считать хост и порт (mail.ru 443) из первой строчки.

Необходимо сразу вернуть ответ (сокет не закрывать, использовать его для последующего зашифрованного соединения)

HTTP/1.0 200 Connection established

После этого curl начнет установку защищенного соединения. Для установки такого соединения необходимо сгенерировать и подписать сертификат для хоста (mail.ru). Команды для генерации корневого сертификата и сертификата хоста <https://github.com/john-pentest/fproxy/blob/master/gen_ca.sh> <https://github.com/john-pentest/fproxy/blob/master/gen_cert.sh>

Необходимо установить защищенное соединение с хостом (mail.ru:443), отправить в него все, что было получено и расшифровано от curl и вернуть ответ.

Убедиться, что получается зайти на сайт mail.ru, авторизоваться и получить список писем

### 3. Повторная отправка проксированных запросов – 20 баллов

Не только проксировать запросы в п.1-2, но и сохранять их вместе с ответом в БД (SQL или NoSQL).

Запросы необходимо сохранять в распаршеном виде (можно использовать любые библиотеки). Необходимо парсить:

* HTTP метод (GET/POST/PUT/HEAD)
* Путь и GET параметры
* Заголовки, при этом отдельно парсить Cookie
* Тело запроса, в случае application/x-www-form-urlencoded отдельно распасить POST параметры

Ответы необходимо сохранять также в распаршеном виде

* Не забыть про gzip и другие методы сжатия! (можно либо расшифровывать их, либо изменять заголовки на стороне прокси)

Пример распаршенного запроса:

POST /path1/path2?x=123&y=asd HTTP/1.1

Host: example.org

Header: value

Cookie: cookie1=1; cookie2=qwe;

z = zxc

{

"method": "POST",

"path": "/path1/path2",

"get\_params": {

"x": 123,

"y": "qwe"

},

"headers": {

"Host": "example.org",

"Header": "value"

},

"cookies": {

"cookie1": 1,

"cookie2": "qwe"

},

"post\_params": {

"z": "zxc"

}

}

Пример распаршенного ответа:

HTTP/1.1 200 OK

Server: nginx/1.14.1

Header: value

<html>...

{

"code": 200,

"message": "OK",

"headers": {

"Server": "nginx/1.14.1",

"Header": "value"

},

"body": "<html>..."

}

Убедиться, что получается записать запрос на авторизацию на сайте mail.ru и заново его отправить

### 4. Сканер уязвимости – 20 баллов

В зависимости от варианта реализовать проверку уязвимости для запросов в БД из п.3. Можно реализовать предложенную проверку, или придумать свою

**Вариант 1.** Command injection – во все GET/POST/Сookie/HTTP заголовки попробовать подставить по очереди

;cat /etc/passwd;

|cat /etc/passwd|

`cat /etc/passwd`

В ответе искать результат выполнения команды (строчку "root:"), если нашелся, писать, что данный GET/POST параметр уязвим

Нарыжный Алексей

Проклашкина Софья

Бочарова Татьяна

Кригер Станислав

Мироненко Даниил

Денисенко Илья

Карганов Андрей

Койбаев Тамерлан

**Вариант 2.** SQL injection – во все GET/POST/Сookie/HTTP заголовки параметры попробовать подставить по очереди

'

"

Посмотреть изменится ли HTTP код и длина ответа, если что-то из этого изменилось, писать, что данный GET/POST параметр уязвим

Угаров Руслан

Савватеев Андрей

Светлаков Владимир

Куренков Дмитрий

Селянинов Дмитрий

Агеева Ксения

Музалев Олег

Галанин Константин

**Вариант 3.** XXE - если в запросе есть XML (строчка <?xml ...), вместо него вставить

<!DOCTYPE foo [

<!ELEMENT foo ANY >

<!ENTITY xxe SYSTEM "file:///etc/passwd" >]>

<foo>&xxe;</foo>

поискать в ответе строчку "root:", если нашлась, писать, что запрос уязвим

Волкова Юлия

Злобин Данил

Кумова Ксения

Сафронов Константин

Тикунова Полина

Сычев Николай

Санин Фёдор

Стремин Валентин

**Вариант 4.** XSS - во все GET/POST параметры попробовать подставить по очереди

vulnerable'"><img src onerror=alert()>

В ответе искать эту же строчку, если нашлась, писать, что данный GET/POST параметр уязвим

Омарова Алия

Семянова Елизавета

Андриянов Илья

Демченко Лев

Новиков Александр

Танев Николай

Руденко Александр

Ягмуров Игорь

**Вариант 5.** Dirbuster – заменить путь (/path/...) по очереди на каждую строчку из словаря https://github.com/maurosoria/dirsearch/blob/master/db/dicc.txt

в качестве результата вывести все файлы, которые вернули не 404 код ответа

Ходырев Роман

Рассмагин Александр

Кошенков Дмитрий

Мамоу Асман

Ховен Ольга

Земляков Алексей

Черепнин Михаил

Балюк Андрей

**Вариант 6.** Param-miner – добавить к запросу по очереди каждый GET параметр из словаря https://github.com/PortSwigger/param-miner/blob/master/resources/params со случайным значением (?param=shefuisehfuishe)

искать указанное случайное значение в ответе, если нашлось, вывести название скрытого параметра

Шишков Иван

Жуков Георгий

Чупраков Сергей

Четверов Кирилл

Карпихин Глеб

Беляк Софья

Расулов Арсен

Ашуров Георгий