Лабораторная работа №6

Основы TLS и HTTPS

В. Слюсарь

Ноябрь 2022

Цель работы

Освоить основные принципы защищенной передачи данных на примере TLS и HTTPS.

Порядок выполнения работы

Для успешного выполнения работы требуется сначала выполнить лабораторную работу "Настройка http- и веб-сервера".

1. Изучите теоретические сведения.
2. Согласно инструкции, создайте самоподписанный TLS сертификат и используйте его для обеспечения безопасного соединения с вашим веб-сервером.
3. Скопируйте форму отчета о проделанной работе, приведенную в конце данного документа, в текстовый или табличный редактор. Сделайте скриншоты диалога информации о сертификате в веб-браузере. Результаты сохраните на личный диск.
4. Ответьте на контрольные вопросы.
5. Предъявите готовый отчет преподавателю.

Теоретические сведения

Незащищенные и защищенные соединения

Итак, наш веб-сервер запущен, http сервер работает. Теперь достаточно разместить наши сервера на компьютере с публичным IP-адресом, и мы получим вебсайт в сети интернет, ведь так? Каких-то 25 лет назад ответом было бы однозначное "да". Однако в 20-х годах

21-го века всё уже не так просто. Наверняка, все вы видели в адресной строке веб-браузера пиктограмму “замочек”, который иногда бывает перечеркнут. Этот замочек указывает нам, **защищено** ли наше подключение к тому вебсайту, который мы сейчас просматриваем.

Если подключение не защищено, то обмен данными между вами (клиентом) и сервером происходит непосредственно: вы посылаете запрос, который без каких-либо изменений упаковывается и передается серверу с использованием протоколов более низких уровней. Другими словами, если где-то по пути следования пакетов в них кто-нибудь **подсмотрит**, то они смогут прочитать их содержимое (пароли, коды подтверждения, фотографии и т.п.). И это только полбеды! Если ваше соединение с сервером не защищено, то вы не можете быть уверены, что общаетесь с тем, с кем думаете. То есть, некий злоумышленник может **представиться сервером**, и вы никак не сможете отличить его от оригинала.

Для решения первой проблемы (“подглядывания”) применяется **шифрование** данных. Для решения второй (“имперсонации”) — системы на основе **сертификатов и цепочек доверия**.

Симметричное и ассиметричное шифрование

Тема шифрования весьма глубоко изучена, и здесь мы представим только самую необходимую информацию в упрощенном виде.

Для начала рассмотрим такую схему: пусть клиент и сервер каким-то образом (например, при личной встрече) договорились о создании одинакового **ключа шифрования**, с помощью которого можно как зашифровать, так и расшировать сообщение. Примером очень простого алгоритма шифрования с ключем будет “шифрование прибавлением”, где к каждой букве изначального сообщения прибавляется какое-то число (которое мы берем из ключа), превращая ее в другую букву. Тогда, для расшифровки сообщения достаточно (на основании того же ключа) обратно вычесть из каждой быквы нужное число. Реальные алгоритмы симметричного шифрования устроены значительно сложнее, однако принцип их работы сохраняется: они используют один и тот же ключ как для шифрования, так и для расшифровки сообщений (откуда и название — симметричное шифрование).

Однако, такой метод очевидно нам не подходит — мы же не можем устраивать ”личную встречу“ с владельцем каждого вебсайта, на который хотим зайти. Оказывается, существуют алгоритмы шифрования, в которых мы можем передавать зашифрованные сообщения и без этого. Они построены на принципе **открытого и закрытого ключей**[1]. Основной принцип работы таких алгоритмов заключается в следующем: сервер генерирует у себя пару ключей, один позволяет только зашифровывать сообщения, а второй — только расшифровывать. Первый ключ сервер открыто (откуда и название) сообщает всем желающим, а

[1]Если вы когда-нибудь использовали протокол удаленного доступа SSH, то скорее всего уже сталкивались с этими понятиям

второй хранит только у себя. Используя открытый ключ, любой клиент может послать зашифрованное сообщение серверу, которое тот расшифрует своим закрытым ключем.

TLS и HTTPS

Итак, как же все-таки вышеописанное позволит нам решить наши проблемы: подлглядывание и имперсонацию? Также можно заметить, что схемы ассиметричного шифрования позволяют только нам отправлять зашифрованные сообщения серверу, но не наоборот.

Оказывается, что совместив методы симметричного и ассиметричного шифрования, мы можем огранизовать **защищенный канал** связи между клиентом и сервером. Делается это следующим образом:

1. Клиент подключается к серверу в незащищенном режиме, и получает от него открытый ключ (**сертификат**)
2. Клиент генерирует случайный ключ для симметричного шифрования (**сессионный ключ**)
3. Клиент шифрует свой сессионный ключ с помощью сертификата и отправляет его серверу
4. Сервер расшифровывает полученное сообщение, получив таким образом сессионный ключ
5. Обе стороны получили сессионный ключ, и переходят к общению на основе симметричного шифрования

Таким образом решается проблема подслушивания. Проблема же имперсонации решается весьма забавным образом. Тот самый сертификат, который мы получили на шаге 1, содержит помимо открытого ключа еще и информацию о том, кем является владелец этого ключа. К примеру, сертификат сервера miet.ru содержит информацию наподобие:

```
я сервер miet.ru, и мой ключ abcabc123123
```

Далее следует логичный вопрос: *а с какого перепугу мне верить что это и правда сервер miet.ru?* Ответ на этот вопрос такой: в сертификате содержится еще одна важная строка:

```
подлинность моего сертификата подтверждает сертификат GlobalSign
```

Ну это уже совсем глупость? А вот нет, не глупость. Сертификаты объединяются в т.н. **цепочки доверия**, где сертификат отсылает к родительскому и так далее и так далее, пока цепочка не достигнет **корневого сертификата**. А вот эти корневые сертификаты вшиты в ваш веб-браузер и операционную систему, и составляют основание цепочки доверия.

Подытожим: при первом подключении сервер присылает вам свой сертификат, в котором содержится его открытый ключ, а также информация о подлинности. Вы используете этот открытый ключ для передачи сессионного ключа, и далее общаетесь по зашифрованному каналу. Только что мы с вами в весьма упрощенном виде описали схему работы протокола **TLS** (англ. transport layer security — протокол защиты транспортного уровня), который пришел на смену очень похожего алгоритма SSL (англ. Secure Sockets Layer — уровень защищённых сокетов). Если по полученному защищенному каналу передаются сообщения протокола HTTP, то такой вот "HTTP поверх TLS" называют **HTTPS**. Именно протокол HTTPS используется сегодня практически для всего общения между веб-браузером и различными веб-серверами в сети интернет. Если же по какой-то причине общение происходит по простому HTTP (без S), то вы увидите как в адресной строке появится пиктограмма перечеркнутого замочка, который мы упоминали в начале.

Практическая часть: получение самоподписанного сертификата и защита соединения с веб-сервером

Для получения "настоящего" сертификата нужно иметь публичный IP-адрес, поэтому в рамках данной лабораторной работы мы этого сделать не сможем. Однако мы можем сгенерировать самоподписанный (то есть с цепочкой доверия, которая заканчивается на нас самих) сертификат и использовать его для защиты соединения с веб-сервером nginx.

1. Распакуйте архив `openssl-1.0.2u-x64_86-win64.zip`, который приложен к лабораторной
2. Запустите терминал (cmd.exe), и перейдите в каталог, в котором лежит распакованный файл `openssl.exe`
3. Создайте в этом же каталоге файл с именем `ssl.conf` со следующим содержимым:

   ```
   [req]
   distinguished_name = req_distinguished_name

   [req_distinguished_name]
   ```

4. Сгенерируйте сертификат и приватный ключ, выполнив в терминале команду

   ```
   openssl req -x509 -nodes -days 365 -newkey rsa:2048
   -keyout privatekey.key -out certificate.crt -config ssl.conf
   -subj "/CN=localhost"
   ```

В случае успешного выполнения команды, вы увидите распечатку подобную приведенной на Рис.1.

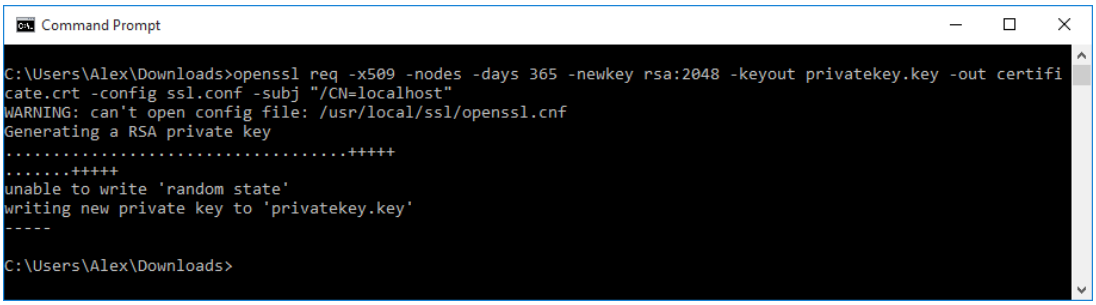

Рис. 1: Сертификат и ключ сгенерировались

5. В конфигурационном файле nginx (созданный вами файл в папке conf.d) модифицируйте директиву `server` следующим образом:

```
server {
    listen 127.0.0.1:443 ssl;
    server_name localhost;

    ssl_certificate C:\ПУТЬ\К\ФАЙЛУ\certificate.crt;
    ssl_certificate_key C:\ПУТЬ\К\ФАЙЛУ\privatekey.key;

    ...
}
```

6. Перейдите в терминале в папку nginx и выполните команду `nginx -s reload` для обновления конфигурации nginx

7. Откройте в веб-браузере страницу `https://localhost/`. Если вы всё сделали правильно, то вы увидите страницу приветствия nginx, но сначала браузер может предупредить вас о том, что сертификат небезопасен (мы это и так знаем). Попав на страницу, вы должны увидеть питкограмму замочка, возможно с восклицательным знаком как на Рис.2;

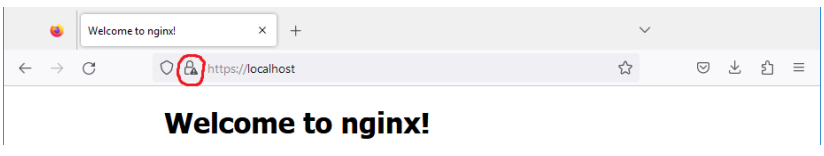

Рис. 2: Соединение защищено, но сертификат не прошел проверку

8. Нажмите на пиктограмму замочка и откройте диалог информации о сертификате. В разных браузерах эти кнопки называются по-разному. Найдите страницу/диалог, похожие на Рис.3. Включите скриншот в отчет.

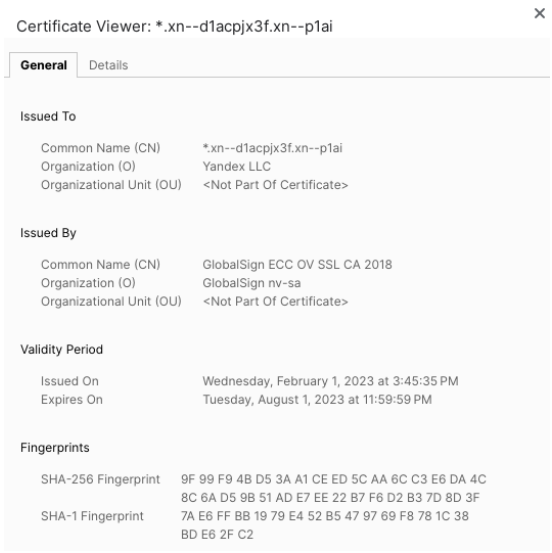

Рис. 3: Информация о сертификате ya.ru

Контрольные вопросы

1. Какие, по вашему мнению, есть *недостатки* у HTTPS по сравнению с HTTP?

2. Существует атака на протокол TLS, которая заключается в том, что вы (злоумышленник) “подсовываете” свой TLS сертификат клиенту в тот момент, когда он ожидает его от сервера. Как называется такая атака? Как с ней бороться?

3. Зачем в TLS используется симетричное шифрование? Почему нельзя использовать ассиметричное шифрование для всех сообщений?

Отчет к лабораторной работе №6

Задание №1

Сгенерируйте TLS сертификат и приватный ключ согласно приведенной инструкции. Прикрепите к отчету скриншот информации о вашем сертификате, а также ответы на вопрос:

- Как называются файл, содержащий сертификат? А файл, содержащий приватный ключ?
- Почему nginx нужно знать исертификат, и приватный ключ?
- Как вы поняли, что ваш веб-сервер работает по протоколу https?