nginx: включение OCSP и HSTS

Технология OCSP

Как узнать, является ли сертификат доверенным? Сделать это можно единственным способом – спросить об этом у самого поставщика, т.е. у удостоверяющего центра, который хранит всю информацию, связанную с выпущенным сертификатом.

При подключении к серверу, клиент должен проверить действительность сертификата сервера по списку отозванных сертификатов – CRL, или по протоколу интерактивного статуса сертификата – OCSP. Проблема CRL заключается в том, что списки могут вырасти до огромных размеров и скачивание может затянуться на вечность.

С помощью OCSP (Online Certificate Status Protocol) веб-браузеры могут проверить достоверность SSL-сертификата. Реализуется это при помощи технологии OCSP Stapling (сшивание OCSP). В этом случае веб-сервер загружает копию ответа удостоверяющего центра, которая затем передается непосредственно в браузер.

Фактически, ответчики OCSP управляются удостоверяющим центром, недоступность которого для браузера приведёт к ошибке, если ответ не будет получен своевременно. Это уменьшает безопасность, позволяя атакующему наводнить запросами ответчик OCSP, чтобы отключить проверку.

Решение заключается в том, чтобы разрешить серверу отправлять в процессе рукопожатия TLS запись OCSP из кэша, так чтобы не затрагивать ответчика OCSP. Этот механизм избавляет клиента от необходимости связываться с ответчиком OCSP и называется **сшиванием OCSP**.

Сервер посылает ответ OCSP из кэша только если клиент его запрашивает, сообщая в CLIENT HELLO о поддержке расширения TLS

status_request.

Большинство серверов сохраняют в кэш OCSP-ответы на 48 часов. Через регулярные интервалы времени сервер будет подключаться к ответчику OCSP удостоверяющего центра, чтобы получить свежую запись OCSP. Расположение ответчика OCSP берётся из подписанного сертификата, из поля Authority Information Access – доступ к информации о подлинности.

Метод OCSP Stapling помогает быстро и безопасно установить достоверность SSL-сертификата. Последовательность проверки достоверности сертификата по технологии OCSP Stapling состоит из следующих шагов:

- Шаг 1. Веб-сервер, на котором находится вебсайт, защищенный SSL, отправляет запрос удостоверяющему центру. В ответе от УЦ приходит статус сертификата, а также подписанный timestamp (временная метка). Подписание метки позволяет гарантировать то, что она не будет каким-либо образом изменена веб-сервером.
- Шаг 2. Браузер посетителя подключается к серверу. В этот момент сервер привязывает временную метку, полученную от УЦ, к SSL-сертификату.
- Шаг 3. Браузер проверяет временную метку. Она подписана поставщиком сертификата, а значит, ей можно доверять.
- Шаг 4. Если SSL-сертификат является доверенным, то в таком случае браузер откроет страницу. В противном случае пользователь получит сообщение об ошибке.

Данный подход позволяет снять нагрузку с удостоверяющих центров и перенести ее на веб-хостинги. Как результат, SSL-соединения устанавливаются быстрее, что позволяет защитить конфиденциальную информацию пользователей от попадания в руки злоумышленников.

Благодаря OCSP Stapling достигаются сразу несколько целей:

- Гарантируется безопасность и конфиденциальность данных пользователей

- Пользователи быстрее загружают защищенный контент, поскольку браузерам не нужно совершать многочисленные запросы
- Сохраняется пропускная способность на стороне клиента, что является преимуществом для мобильных пользователей
- Рост доверия и удовлетворенности клиентов за счет увеличения скорости доставки защищенного контента

Требования

Необходим nginx версии не ниже 1.3.7. Также нужно создать исключение в пакетном фильтре, чтобы разрешить веб-серверу совершать исходящие подключения к вышестоящим серверам OCSP.

В настройки ssl.conf добавляем (докум. nginx):

```
## Включаем OCSP-ответы, тем самым уменьшая время загрузки страниц у пользователей
ssl_stapling on;
ssl_stapling_verify on;
resolver 8.8.8.8 8.8.4.4 valid=300s;
resolver_timeout 5s;
ssl_trusted_certificate     /etc/letsencrypt/live/tst-amo.net.ua/chain.pem;
```

OCSP Must-Staple для Let’s Encrypt / certbot

Если пользуетесь для HTTPS-сертификатов бесплатным Let’s Encrypt, то там ситуация будет ещё проще. Вы можете что вручную добавлять в строку запроса сертификата параметр --must-staple, например в cron:

```
certbot renew --quiet --must-staple --allow-subset-of-names
```

что добавить в файл cli.ini (или в конкретный conf-файл в каталоге /renewal) строчку:

```
must-staple = True
```

Разве что учитывайте, что файл cli.ini будет перекрывать собой настройки для конкретных сертификатов – т.е. если на одной системе у вас пачка доменов, и certbot регулярно пробегает и обновляет сертификаты, то заданное в cli.ini будет перекрывать всё остальное. С “айтишной” точки зрения это непривычно – общая настройка “для всей системы” перекрывает более специфичные частные. Однако тут логика чуть другая – файл cli.ini – это “то, что по умолчанию добавлять как будто бы админ это руками к каждому запросу дописывает”. Так что прописывайте OCSP Must-Staple в cli.ini только если это точно надо для всех сертификатов на данной системе, если же нет – то добавьте только в нужные.

Проверка OCSP из консоли:

```
# openssl s_client -connect tst-amo.net.ua:443 -tls1 -tlsextdebug -status

CONNECTED(00000003)
TLS server extension "renegotiation info" (id=65281), len=1
0001 - <SPACES/NULS>
TLS server extension "EC point formats" (id=11), len=4
0000 - 03 00 01 02 ....
TLS server extension "session ticket" (id=35), len=0
TLS server extension "status request" (id=5), len=0
TLS server extension "heartbeat" (id=15), len=1
0000 - 01 .
depth=2 O = Digital Signature Trust Co., CN = DST Root CA X3
verify return:1
depth=1 C = US, O = Let's Encrypt, CN = Let's Encrypt Authority X3
verify return:1
depth=0 CN = tst-amo.net.ua
verify return:1
OCSP response:
======================================
OCSP Response Data:
    OCSP Response Status: successful (0x0)
    Response Type: Basic OCSP Response
    Version: 1 (0x0)
```

```
    Responder Id: C = US, O = Let's Encrypt, CN = Let's
Encrypt Authority X3
    Produced At: Jul 21 09:37:00 2019 GMT
    Responses:
    Certificate ID:
      Hash Algorithm: sha1
                                  Issuer     Name     Hash:
7EE66AE7729AB3FCF8A220646C16A12D6071085D
                                   Issuer     Key     Hash:
A84A6A63047DDDBAE6D139B7A64565EFF3A8ECA1
      Serial Number: 034CBD9A46BC406FA10ED89DDBB09119A6BF
Cert Status: good
This Update: Jul 21 09:00:00 2019 GMT
Next Update: Jul 28 09:00:00 2019 GMT
....
```

Если технология выключена или неработает, то будет вывод, типа:

```
OCSP response: no response sent
```

Для более полной проверки идем на ssllabs. Если включена поддержка OCSP то будет примерно такая картина:

OCSP stapling	**Yes**

При *первом* тестировании может показать, что **OCSP stapling No**, это стандартное поведение nginx’a. При первом запросе nginx отвечает без ocsp_stapling и асинхронно пытается его получить. Если ему это удается, то в следующий раз он отдаёт ocsp response.

Включаем HSTS

Для рейтинга **A+** нужно, что бы был включен HSTS, для этого добавляем в ssl.conf:

```
## On HSTS
add_header  Strict-Transport-Security  "max-age=15768000;
includeSubDomains; preload";
```

Strict Transport Security (HSTS)	**Yes** max-age=15768000; includeSubDomains; preload

Проверяем используя curl:

```
# curl -s -D- https://tst-amo.net.ua | grep -i Strict
Strict-Transport-Security:          max-age=31536000;
includeSubDomains; preload
```

Что такое HSTS?

HSTS (HTTP Strict Transport Security) – это механизм защиты от даунгрейд-атак на TLS, указывающий браузеру всегда использовать TLS для сайтов с соответствующими политиками. Стандарт описан в RFC6797, а политики бывают двух видов:

Динамические

Политика применяется из HTTP-заголовка Strict-Transport-Security при первом заходе на сайт по HTTPS, в нём указан срок действия и применимость к субдоменам:

```
Strict-Transport-Security:          max-age=15768000;
includeSubDomains;
```

Статические

Статические политики захардкожены в браузер и для некоторых сайтов включает привязку к вышестоящему CA, выпустевшему сертификат (например: google.com, paypal.com или torproject.org). Причем она может действовать только когда сайт открыт через TLS, разрешая незащищённое соединение, но блокируя MitM с подменой сертификата.

Список из Chromium используют все популярные браузеры (Firefox, Safari и IE 11+Edge) и добавить в него сайт может любой желающий, если веб-сервер отдаёт заголовок Strict-Transport-Security со сроком действия от двух лет и ключевым словом **preload** в конце:

```
Strict-Transport-Security:          max-age=63072000;
```

includeSubDomains; preload

Данный механизм преследует достаточно простую цель – сделать так, чтобы ресурс, доступный по HTTPS, был бы доступен исключительно по HTTPS, без возможности “сваливания” в обычный HTTP. Т.е. если у вас есть сайт вида **https://www.atraining.ru/** , то предполагается, что всё взаимодействие клиента идёт только по HTTPS, даже если каким-то образом клиенту будет подсунута ссылка на http-версию (обычно в целях перехвата данных).

Реализуется это с двух сторон, сервером и клиентом. Сервер:

- Добавляет в HTTP-ответ специальный заголовок Strict-Transport-Security, в котором говорит, что надо включить данный механизм.
- Указывает в параметрах время, которое будет действовать данная директива (т.е. что-то вида “вот с текущего момента и ещё целый год, ты сюда ходи только по https, если увидишь ссылку на этот домен, но с http – поправь **до** отправки запроса”)

Клиент:

- Кэширует в браузере эту информацию на указанное время и проводит при каждом обращении анализ заголовка и следование указаниям со стороны сервера.
- Если видит проблему с подключением – например, недоверенный сертификат – сразу отказывается устанавливать соединение (т.е. действует жестче, чем обычно).

Звучит достаточно просто – но, по факту, данный механизм отсекает целую пачку потенциальных проблем вида “после установки HTTPS-сессии как-то удалось перейти обратно на HTTP”, например атаку SSL-stripping MItM (делается утилитой sslstrip), или кражу cookies через утилиту Firesheep.

Протестировать HSTS можно на сайте [hstspreload](), там же

добавить себя в статический список.

Имейте в виду, что как только вы установите заголовок STS и отправите свой домен в список предварительной загрузки HSTS, его невозможно оттуда удалить. Это одностороннее решение сделать ваши домены доступными через HTTPS. Вы берёте на себя обязанность перед пользователями поддерживать HTTPS версию сайта, включая поддержку вашего доверенного сертификата.

Дополнительная защита сервера с помощью HTTP headers

Добавим строки в наш ssl.conf

```
# XSS Protection for Nginx web server
add_header X-Frame-Options SAMEORIGIN;
add_header X-XSS-Protection "1; mode=block";
add_header X-Content-Type-Options nosniff;
add_header X-Robots-Tag none;
```

X-Frame-Options в заголовке HTTP-ответа может использоваться для указания того, разрешено ли браузеру открывать страницу в фрейме или iframe.

Это предотвратит внесение содержимого сайта в другие сайты.

Вы пытались внедрить Google.com на свой сайт в качестве рамки? Вы не можете, потому что он защищен, и вы тоже можете защититься.

Для **X-Frame-Options** есть три параметра:

1. SAMEORIGIN: позволяет загрузку контента в frame/iframe только если фрейм и страница, его загружающая, расположены на одном домене.
2. DENY: этот параметр предотвратит отображение страницы в фрейме или iframe (например, Roundcube перестанет показывать содержимое писем).
3. ALLOW-FROM URI: этот параметр позволяет отображать страницу только по указанному источнику.

X-XSS-Protection

Заголовок X-XSS-Protection может предотвратить некоторые **XSS**-атаки («межсайтовый скриптинг»), он совместим с IE 8+, Chrome, Opera, Safari и Android.

Google, Facebook, Github используют этот заголовок, и большинство консультантов по предупреждению проникновений порекомендуют Вам его использовать.

X-Content-Type-Options

Можно предотвратить атаки с использованием подмены **MIME** типов, добавив этот заголовок ответа HTTP. Заголовок содержит инструкции по определению типа файла и не допускает сниффинг контента. При конфигурации потребуется добавить только один параметр: “nosniff”.

X-Robots-Tag – управления индексированием и показом:

noindex Не показывать эту страницу, а также ссылку “Сохраненная копия” в результатах поиска.
nofollow Не выполнять переход по ссылкам на этой странице.
none Аналогично метатегам noindex, nofollow.

Content-Security-Policy

Для RoundCube CSP имел такую запись:

```
add_header Content-Security-Policy "default-src 'self'; script-src 'self' 'unsafe-eval' 'unsafe-inline'; style-src 'self' 'unsafe-inline'; img-src 'self'; frame-src 'self'; connect-src 'self'; frame-ancestors 'self'; base-uri 'self'; form-action 'self'";
```

Проверка X-HEADERS

http://vladimir-stupin.blogspot.com/2017/02/ocsp-nginx.html
https://www.atraining.ru/tls-armoring-windows-server/#hsts
https://forum.netgate.com/topic/129063/ocsp-must-staple-nginx-

configuration
https://content-security-policy.com/

HSTS в Nginx или получаем оценку A+ от SSL Labs

https://habr.com/ru/post/320164/
https://certbot.eff.org/docs/using.html#where-are-my-certificates
https://www.howtoforge.com/tutorial/nginx-with-letsencrypt-ciphersuite/
https://serverfault.com/questions/874936/adding-hsts-to-nginx-config
Использование HTTP заголовков для предупреждения уязвимостей
https://developers.google.com/search/reference/robots_meta_tag?hl=ru