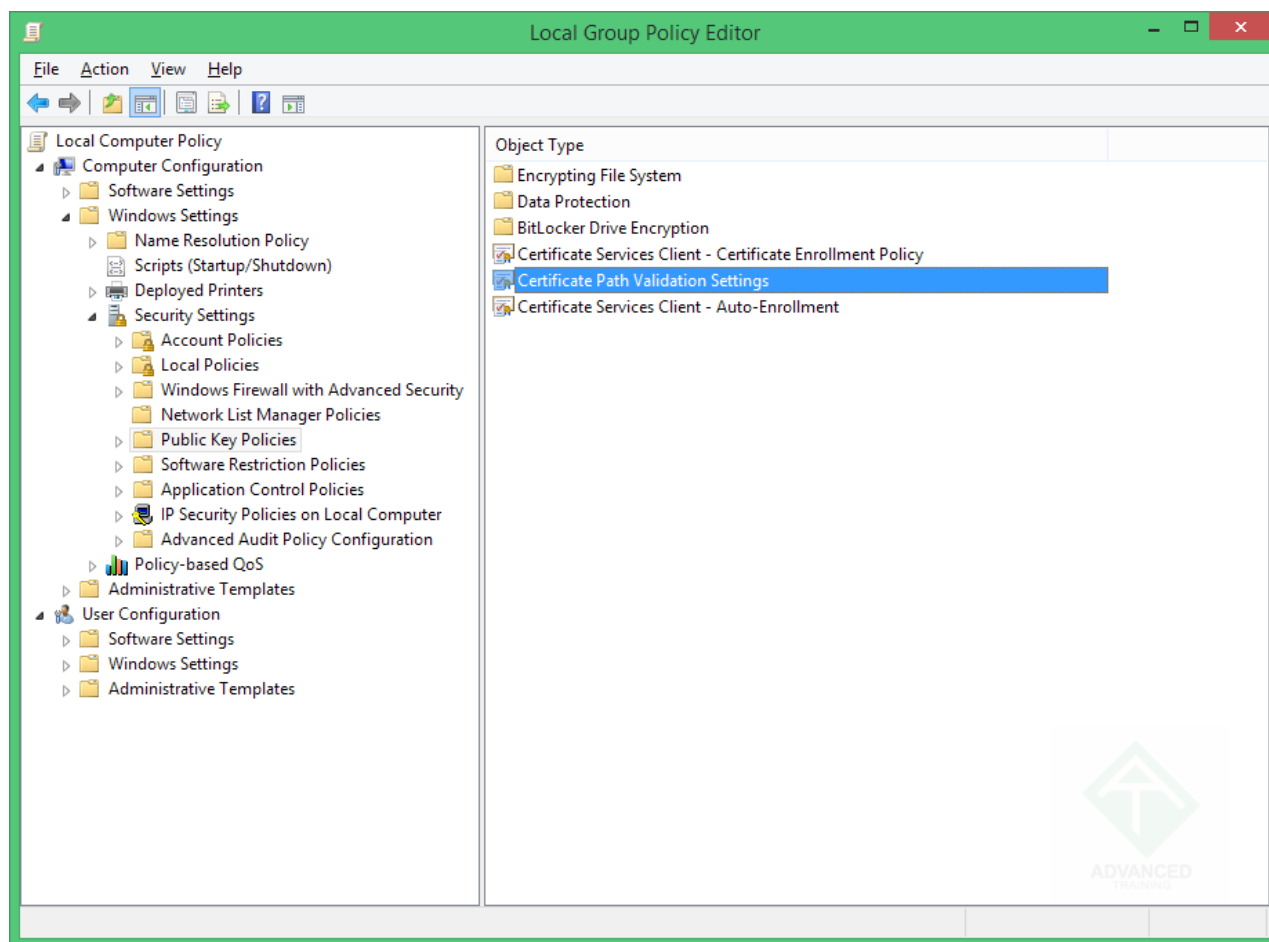


# OCSP и связанные с ним технологии - OCSP Stapling и OCSP Must-Staple

 atraining.ru/ocsp-settings-tuning-stapling

2018-04-13T23:11:47+08:00



Привет.

OCSP (Online Certificate Status Protocol) – полезная технология, делающая работу с PKI-системами более быстрой.

Несмотря на то что данная технология древняя, и поддерживается всеми крупными СА более десяти лет, в большинстве применений речь далее чем о “ну, те, у кого мы сертификаты берём, OCSP-сервер подняли и он работает” не идёт.

Однако и у самого OCSP есть настройки, и у связанных с этим протоколом технологий – тоже.

Немного углубимся в тему – от вас потребуется примерное представление о PKI (лучше [подробное](#), но не обязательно) и желание сделать что-то лучше (это уже обязательно).

# Настраиваем и оптимизируем OCSP и связанные с ним технологии

---

Приступим.

## Кратко про OCSP

---

У сертификатов x.509, как и у любых сущностей, есть жизненный цикл. Они рождаются, живут, и умирают. Всё, как у людей.

Помимо фактической смерти сертификата от старости (в фиксированное время) у него бывает преждевременная кончина – она называется “отзывом”. Нужен этот механизм для того, чтобы резко прекратить использовать сертификат – по причине, например, компрометации ключа, либо изменения сущности, которую подтверждает сертификат, или какой-либо ещё неустранимой проблемы, после которой использование сертификата – как связки “криптографический материал плюс информация о связанной с ним сущности” – становится бессмысленным. Сертификат вроде как жив и срок годности ещё не завершился, но использовать его можно разве что в справочных целях вида “вот смотрите, когда-то это было сертификатом”. У людей это реализуется через каминг-аут или фразу “кстати я веган, отлично себя чувствую”.

Соответственно, возникает задача – как по сертификату определить, что его отозвали. Так как сертификат подписан, то редактировать его нельзя – следовательно, нельзя разместить прямо в нём какую-то информацию об отзыве в момент самого отзыва. Значит надо будет изначально предусмотреть в сертификате справочное поле, говорящее о внешней системе, обладающей информацией об отзыве. СА много, поднять СА может каждый, а значит информация эта будет специфична для каждого конкретного сертификата. Сертификат у нас идентифицируется уникальным id, по сути серийным номером, а раз так, то этой информации – id сертификата плюс данные о конкретном СА – хватит, чтобы проверить информацию об отзыве.

Как это сделать?

Первая и самая простая идея – формировать специальный документ в формате x.509v2 – называемый списком отозванных сертификатов (CRL). Это простой файл с линейным перечнем “кого отозвали”. Проблема в том что файл этот, выходит, постоянно растёт, и если у СА много сертификатов, то CRL-файл шустро увеличивается. Кроме того, ведь клиентам его надо обновлять по достаточно плотному расписанию, и чем чаще – тем лучше; нельзя же предположить, что через минуту не отзовут какой-либо сертификат, которым будет подписано пришедшее через пару минут письмо. Следовательно у этого файла появляется и явно прописанный “срок годности” – то время, через которое его точно надо обновить (скажем, неделя), плюс появляется задача “как можно чаще обновлять, чтобы оперативно отреагировать на свежееотозванный сертификат”.

Перекачивать весь CRL-файл грустно, поэтому практически сразу же были придуманы две технологии:

- Публиковать отдельные CRL по типам сертификатов – скажем, крупный CA может вести один CRL про отозванные сертификаты веб-серверов, а другой – про отозванные сертификаты электронной почты;
- Публиковать т.н. “дельта-CRL” – файлы, содержащие только сертификаты, отозванные после определённой даты. По задумке создателей механизма, клиентское ПО должно это всё творчески анализировать и, скажем, раз в месяц подкачивать “основной” CRL, а раз в день – “текущую дельту”.

Можно было также наделать много выдающих CA, чтобы “распределить” по их CDP – CRL Distribution Point’ам отозванных, но это уже совсем хардкор. Сама идея “разделить один огромный файл на части, чтобы перекачивать было проще” – рабочая, тут никто не спорит (впрочем, является явным “костылём”, т.к. не исправляет причину, а лишь временно отодвигает проблему роста CRL).

По итогам оба этих варианта не сильно исправили ситуацию, ну а с дельта-CRL некоторое клиентское ПО так работать и не научилось.

Всё скатывалось к тому что, приняв письмо размером в пять килобайт, подписанное цифровой подписью кого-то крупного – типа Verisign – надо подкачать здоровенный основной CRL и пачку дельта-CRL к нему, всё это собрать в единый массив и проверить подпись у письма, чтобы узнать, имеет ли смысл его читать, или нет. И все равно, даже если это старательно делать, то ситуацию “отозвали сертификат пару часов назад” можно упустить и принять поддельное сообщение за корректно подписанное.

При этом ситуация усугубляется тем, что информацию о старых отозванных сертификатах удалять нельзя. Почему?

Представьте себе архив электронной почты за, скажем, пять лет. Если хранить только информацию об отзыве за последний год, а более старую вычищать из CRL, то анализ архива за прошедшие 2-3 года превратится в сложную задачу. Если среди полученных 2 года назад писем будет подписанное отозванным на момент получения сертификатом, то оно будет ошибочно воспринято как “нормальное”. Ведь информация о том, что на момент получения этого письма его подпись была выполнена уже отозванным сертификатом – отсутствует.

Что делать? Очевидно – нужен сервис, лишённый всех этих вопросов по части “болезней роста”. Им и стала реализация протокола Online Certificate Status Protocol, или OCSP.

OCSP реализуется через веб-сервис, умеющий получить запрос “отозван ли вот этот конкретный сертификат” и ответить “да” или “нет”. Особенно полезно, отметим, то, что ответы OCSP представляют из себя отдельные объекты, подписанные сервером и обладающие сроком годности – а значит они могут передаваться между

системами, кэшироваться и обновляться, следуя какой-либо нужной в специфичной ситуации логике. Адрес OCSP-сервиса не заменяет собой CRL в сертификате – он является дополнительным методом проверки валидности, указываемым в AIA-расширении.

Теперь кратко про то, как будут реализованы компоненты OCSP на практике.

## OCSP-сервер

---

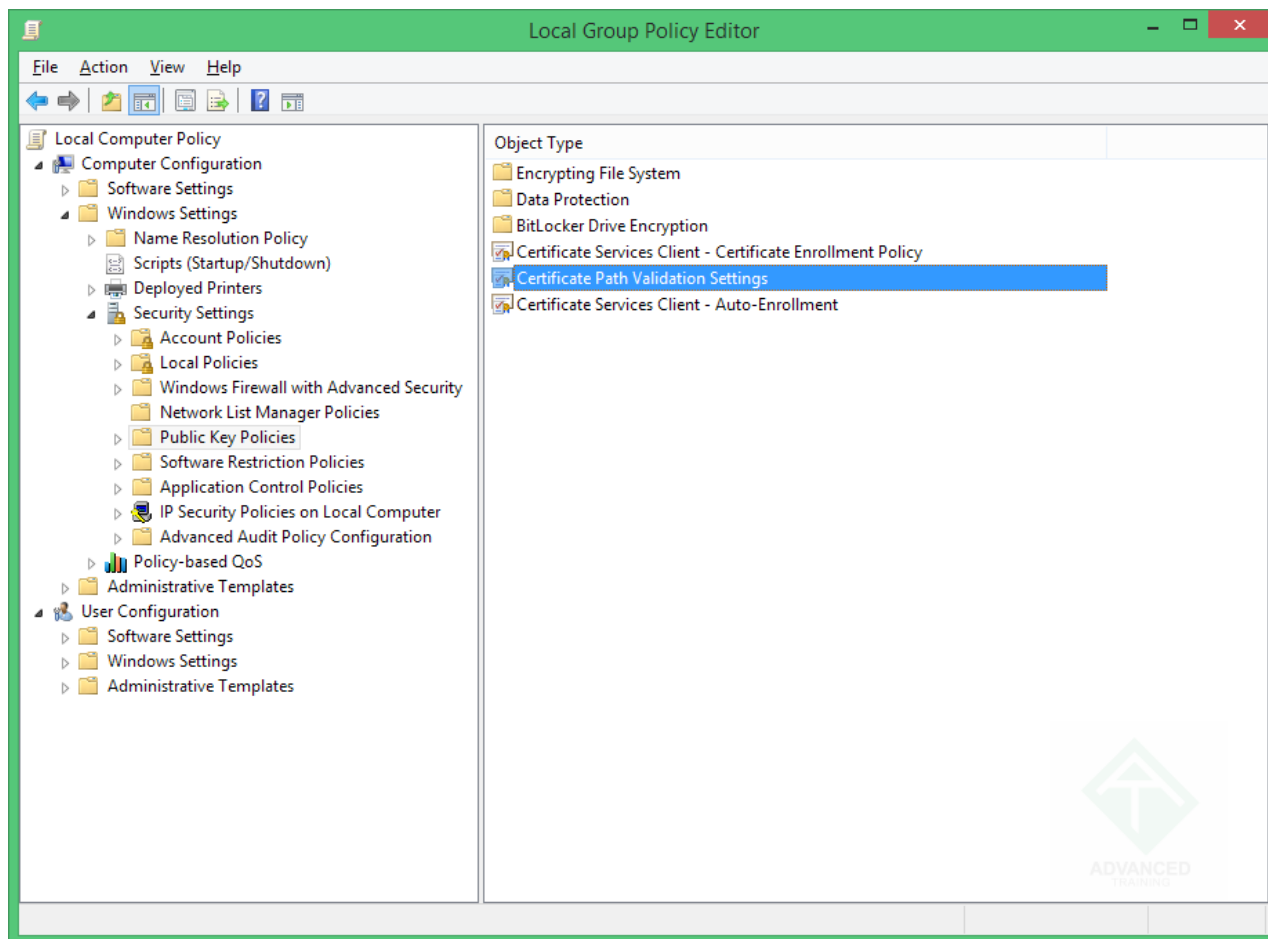
Поднять свой OCSP-сервер для локальной инфраструктуры PKI нетрудно – он будет встроен в, например, ОС Windows Server начиная с NT 6.0 и будет называться OCSP Responder. Про его настройку [и так подробно говорится на курсах](#), но вкратце там всё очень просто – вы обеспечиваете OCSP Responder'у доступ к свежему CRL-файлу нужного CA (или файлАМ, если их несколько), выдаёте сервису специфичный OCSP-сертификат и сервис функционирует. По сути он получает запрос клиента, ищет id сертификата в локальном CRL-файле и отвечает “да” или “нет”, тривиально снимая с клиента задачу по загрузке полного CRL'а, сбору того из кусочков delta-CRL'ов и всякого подобного.

Оптимизировать там, опять же, особенно нечего – этот сервис должен быть постоянно доступен и очень быстр, что вообще характерно для современных сервисов, так что перейдём к клиентской стороне.

## OCSP-клиент

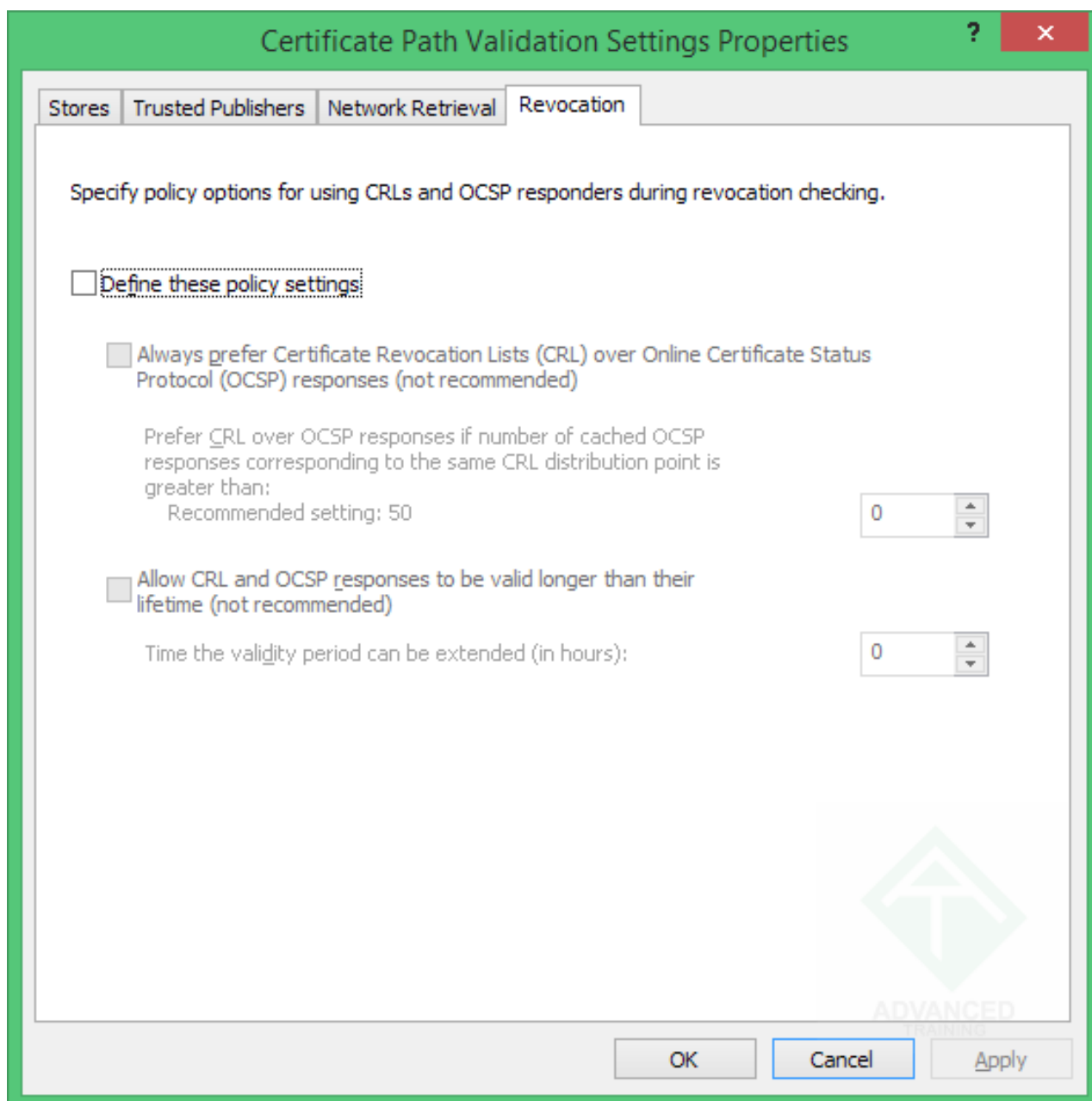
---

Поддержка анализа и обработки OCSP-записей в поле AIA у x.509v3-сертификатов появляется в Windows Vista; там же появляется и возможность управлять работой OCSP. Сделана она штатно, через механизм объектов групповой политики:



[Настройка проверки сертификатов на отзыв в Windows через механизм Group Policy. \(кликните для увеличения до 921 px на 678 px\)](#)

Нас будет интересовать блок настроек про выбор между OCSP и CRL (когда для конкретного сертификата доступны оба) и модификацию времени валидности данных об отзыве:



### [Настройки OCSP и CRL-проверки сертификатов в Windows](#) (кликните для увеличения до 561 px на 563 px)

Шучу, не будет. Тонкая настройка “как именно выбирать между OCSP и CRL” нужна крайне редко – например, в сценарии “в локальной сети высокий уровень безопасности и используется IPsec transport mode для защиты определённых видов трафика” можно будет “заставлять” обращаться к закешированному CRL (который заранее через ту же Group Policy раздаётся), экономя время на установку SA. Увеличение срока валидности OCSP/CRL сверх указанного также пригодится очень редко – например у мобильного устройства, перемещающегося между имеющими частичный доступ к Интернету региональными объектами одного крупного предприятия.

Основное у клиента – это “Умеет ли читать OCSP-поле” / “Умеет ли обрабатывать множественный ответ OCSP-responder’а”. Сейчас это, в общем, почти у всех.

Вроде всё хорошо – задачу с растущими CRL'ами решили. Но технологии не стоят на месте и на базе OCSP развиваются более интересные возможности.

Первая по важности из них – это OCSP Stapling.

## OCSP Stapling

---

Прикрепление OCSP-ответа – технология, развивающая применение OCSP и имеющая очевидные плюсы.

Ключевой – скорость. Вместо того, чтобы запросить у сервера сертификат, проверить его по всем параметрам (валидный по формату, действительный, обладает всеми нужными полями и использует понятные для хоста криптоалгоритмы), а потом сходить к OCSP responder'у, чтобы узнать про “отозван или нет”, OCSP Stapling предлагает “прикреплять” к ответу TLS-сервера и сертификат и OCSP-ответ от responder'а про этот сертификат. Это экономит время клиента – ему не надо устанавливать сессию до OCSP responder'а, плюс делает систему чуть безопаснее – OCSP responder не может собирать информацию “кто и с каких адресов подключается к серверам, которым мы выдали наши сертификаты” – он видит лишь факт того, что сервер многократно спрашивает про свой собственный сертификат.

Проблема со скоростью OCSP также завязана на то, что в первой реализации – [RFC 6066](#) – OCSP разрешал в ответе только информацию про один сертификат. То есть клиент, даже получив от сервера полную цепочку сертификатов до какого-то trusted – ну или не от сервера, имея часть из сертификатов из этой цепочки локально – все равно делал несколько (обычно 3-4) запросов про каждый из них отдельно. В обновлённой спецификации, [RFC 6961](#), которая “Multiple Certificate Status Request Extension”, OCSP responder может выдать информацию сразу про несколько сертификатов в одном ответе – но вопрос, насколько широко реализован обновлённый OCSP со стороны различного клиентского ПО (в частности, браузеров). Поэтому OCSP Stapling потенциально может выиграть ещё больше времени, экономя ещё и на количестве запросов.

В дополнение, OCSP Stapling потенциально делает установку соединения более надёжной – потому что если вы подключились к веб-серверу, и работает OCSP Stapling, то информацию об отзыве вы получите от этого же сервера, а не подключаясь к стороннему сервису (который может не работать). Сервер кэширует валидный OCSP-ответ, поэтому в случае кратковременных перебоев OCSP responder'а, ответственного за сертификат X, обладающий этим сертификатом X сервер будет отправлять клиентам валидный OCSP-ответ, а если бы клиенты обращались напрямую к responder'у, то они бы получали отказ в обслуживании после достаточно длительного тайм-аута.

OCSP Stapling, кстати, не просто “айтишная мелкая штука, ускоряющая работу”, а вполне серьёзная технология, упоминающаяся как нужная к реализации в [разделе 3.4.1.2 стандарта NIST SP 800-52r1](#).

Надо использовать. Как?

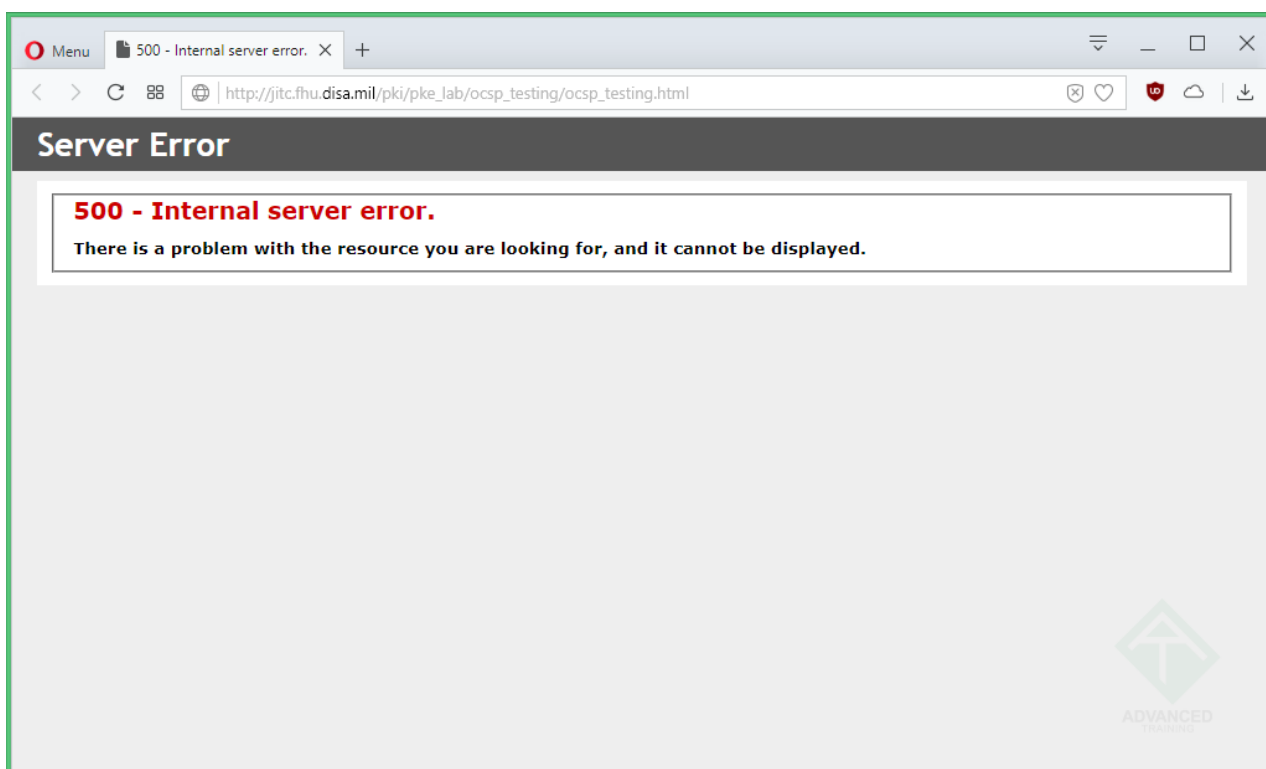
## OCSP Stapling в IIS

---

Никак.

OCSP Stapling поддерживается начиная с NT 6.0 и детали его реализации в IIS, судя по всему, IIS унесёт с собой в могилу. Известно лишь то, что вопросы о “*когда вы доделаете поддержку Multiple Certificate Status Request Extension?*” подвисали в воздухе на форумах MSDN ещё в 2017 году. Что, в принципе, можно считать достаточным для принятия решения не особо использовать IIS для таких штук.

Что интересно, “неуправляемая” реализация OCSP Stapling в Microsoft IIS прошла все нужные сертификации американского Минобороны, о чём сообщается на [специальной странице](#). Текущее состояние этой страницы довершает картину:



[Тут должна быть цитата Лаврова про Д.Б., но останемся в рамках приличий \(кликните для увеличения до 1026 px на 615 px\)](#)

## OCSP Stapling в nginx

---

В варианте с nginx всё куда как лучше.

Включается OCSP Stapling, выключенный по умолчанию, вот так:

```
ssl_stapling on;
```



Чтобы он (механизм OCSP Stapling) работал, нужна информация о “родительском” сертификате и прямое указание на DNS-сервер, через который будут разрешаться FQDN’ы из AIA-расширения.

Существует также возможность перекрыть адрес OCSP-сервера для конкретного сертификата – в разделе конфигурации про нужный сервер можно задать команду:

```
ssl_stapling_responder http://url;
```

Это позволяет делать схемы вида “ферма nginx-серверов обращается к специальному серверу, кэширующему с запасом по времени OCSP-ответы и очень часто их проверяющему”, делая работу фермы более быстрой, снимая с nginx’ов несвойственные им задачи по сверхбыстрому рефрешу OCSP-ответов и улучшая надёжность всей системы. Грубо говоря, можно поставить специальный узел, который будет раз в 30 секунд бегать за OCSP-ответом про используемый сертификат (или сертификаты), и практически мгновенно реагировать на ситуацию “отозвали”, а также замаскирует реальные адреса серверов, которым нужна эта информация.

Дополнительной мерой безопасности будет проверка OCSP-ответа:

```
ssl_stapling_verify on;
```

Это подстрахует от ситуации “нам отдали некорректно подписанный ответ и мы его, не читая, прикрепили к ответу для клиента, и клиент проверил его и понял, что он сбойный”.

В случае правильной настройки внешние средства проверки будут определять поддержку OCSP Stapling примерно так:

https://dev.ssllabs.com/ssltest/analyze.html?d=www.atraining.ru

DROWN	No, server keys and hostname not seen elsewhere with SSLv2 (1) For a better understanding of this test, please read <a href="#">this longer explanation</a> (2) Key usage data kindly provided by the <a href="#">Censys</a> network search engine; original DROWN website <a href="#">here</a> (3) Censys data is only indicative of possible key and certificate reuse; possibly out-of-date and not complete
Secure Renegotiation	Supported
Secure Client-Initiated Renegotiation	No
Insecure Client-Initiated Renegotiation	No
BEAST attack	Not mitigated server-side ( <a href="#">more info</a> ) TLS 1.0: @xc814
POODLE (SSLv3)	No, SSL 3 not supported ( <a href="#">more info</a> )
POODLE (TLS)	No ( <a href="#">more info</a> )
Downgrade attack prevention	Yes, TLS_FALLBACK_SCSV supported ( <a href="#">more info</a> )
SSL/TLS compression	No
RC4	No
Heartbeat (extension)	No
Heartbleed (vulnerability)	No ( <a href="#">more info</a> )
Ticketbleed (vulnerability)	No ( <a href="#">more info</a> )
OpenSSL CCS vuln. (CVE-2014-0224)	No ( <a href="#">more info</a> )
OpenSSL Padding Oracle vuln. (CVE-2016-2107)	No ( <a href="#">more info</a> )
ROBOT (vulnerability)	No ( <a href="#">more info</a> )
Forward Secrecy	Yes (with most browsers) ROBUST ( <a href="#">more info</a> )
ALPN	Yes h2 http/1.1
NPN	Yes h2 http/1.1
Session resumption (caching)	Yes
Session resumption (tickets)	Yes
OCSP stapling	Yes
Strict Transport Security (HSTS)	Yes max-age=31536000; includeSubdomains; preload
HSTS Preloading	Chrome Edge Firefox IE Yes max-age: 2592000 includeSubDomains: true report-uri: https://www.atraining.net/report-uri/ pin-sha256: h6801m+28v3zbgkRHqg8L29Egztzh89C1SyUCOQmQJ= pin-sha256: lnsM2T/O9/U84sJFdnrpsFp3awZJ+ZZbYpCWWhGloaHl= pin-sha256: Vjs84z+80w/Nsr1YKqWQboSiR63WwWxhMN+eWys=
Public Key Pinning (HPKP)	
Public Key Pinning Report-Only	No
Public Key Pinning (Static)	No ( <a href="#">more info</a> )

[Поддержка OCSP Stapling обнаружена у сервера www.atraining.ru](#)  
([кликните для увеличения до 1345 px на 997 px](#))

Что там у клиентской стороны?

## OCSP Stapling со стороны клиентов

Основная задача для OCSP Stapling – это, конечно, упрощение жизни веб-браузеров. Большинство из них на этот момент (апрель 2018 года) давно уже имеют поддержку OCSP Stapling.

Проверить свой браузер можно, например, [по ссылке на браузерный тест SSLabs](#). Браузер с поддержкой OCSP Stapling будет выдавать что-то такое:

Protocol Details	
Server Name Indication (SNI)	Yes
Secure Renegotiation	Yes
TLS compression	No
Session tickets	Yes
OCSP stapling	Yes
Signature algorithms	SHA256/ECDSA, RSA_PSS_SHA256, SHA256/RSA, SHA384/ECDSA, RSA_PSS_SHA384, SHA384/RSA, RSA_PSS_SHA512, SHA512/RSA, SHA1/RSA
Named Groups	tls_grease_4a4a, x25519, secp256r1, secp384r1
Next Protocol Negotiation	No
Application Layer Protocol Negotiation	Yes h2 http/1.1
SSL 2 handshake compatibility	No

[Поддержка OCSP Stapling обнаружена у браузера Opera, которым я пользуюсь \(кликните для увеличения до 963 px на 386 px\)](#)

В некоторых браузерах можно будет управлять этой возможностью – т.е. говорить явно “если с ответом сервера идёт какая-то доп.инфа, называемая OCSP Response, то читай/игнорируй её”. В Firefox, например, так:

```
security.ssl.enable_ocsp_stapling = true/false
```

Вообще, как только речь идёт не о связке “веб-сервер – браузер”, лучше всего в явном виде убедиться, что OCSP Stapling поддерживается системой. К примеру, почтовый сервис Postfix не поддерживает OCSP вообще, исходя из логики “клиенту надо, пусть он и проверяет”. В плане почтового edge-сервера это, кстати, даже логично.

Далее, если перейти от ПО к компонентам ПО, то тоже будут особенности реализации. Например, возможность управления OCSP Stapling будет встроена в Windows’овскую реализацию Kerberos – что удивительно, т.к. в системе глобально этой возможности нет.

Вы сможете настроить следующее – использовать ли прикреплённый Kerberos-сервером ответ с OCSP-информацией, либо игнорировать и следовать к явно указанному в сертификате OCSP-серверу. Как понятно, всё это возможно только в случае, когда контроллер домена работает на NT 6.0 и выше, и имеет сертификат со специфичным OID’ом про Kerberos.

```
HKLM\SYSTEM\CurrentControlSet\Control\Lsa\Kerberos\Parameters\
```

В этом ключе будет параметр **RequestOCSP**, как обычно типа DWORD32, который можно установить в нуль и тогда клиент будет всегда сам лично ходить проверять сертификат DC на отзыв через OCSP responder. Это, по идее, безопаснее – но клиентам сразу же нужен быстрый доступ во внешние сети на моменте инициализации подключения к DC, то есть на фазе загрузки – а это не всегда так просто, особенно если детали доступа во внешние сети как раз и настраиваются через Group Policy, которая загружается уже после того, как тикет от Kerberos получен.

Так что обычно единица в этом ключе и упрощает настройки, и ускоряет работу Kerberos – но, опять-таки – именно когда у вас “усиленный” вариант, с дополнительной валидацией DC через x.509v3.

Далее – технология OCSP Must-Staple.

## OCSP Must-Staple

---

OCSP Must-Staple (см. [RFC 7633](#)) – технология, нужная для дальнейшей оптимизации работы с TLS-серверами. Идея достаточно проста – в сертификат добавляется OID, который указывает на то, что к этому сертификату всегда должен

идти “прикреплённый” OCSP-ответ. Браузер, таким образом, запрашивая сертификат сразу же получает сигнал о том, что сервер должен к этому ответу приложить OCSP – или, если не приложит, то это ленивый сервер и с ним надо разорвать связь. То есть клиентский браузер чётко знает – “я сам ходить за OCSP даже пробовать не буду – мне должны прислать валидный OCSP-ответ”. Что, как понятно, упрощает алгоритм подключения и исключает задержку на “решаем, так OCSP-ответ нам пришлют или самим сходить надо будет”.

Ещё раз – именно **должны**. Ваш сервер, предъявляя такой сертификат, подписывается под то, что к TLS-ответу будет прикреплён OCSP.

Добавить OCSP Must-Staple в сертификат несложно – это OID 1.3.6.1.5.5.7.1.24, со значением 5. Сделать это надо на фазе создания CSR, т.е. запроса на сертификат. В случае OpenSSL это будет выглядеть так – в `openssl.conf` в раздел “параметры запроса” надо будет добавить вот такую строчку:

```
tlsfeature = status_request
```

(в случае старых версий OpenSSL, до 1.1.0, надо будет явно написать OID, т.к. там название `status_request` ещё не известно – и строчка будет `1.3.6.1.5.5.7.1.24 = DER:30:03:02:01:05`).

Я, чтобы не портить “главный” `openssl.conf`, обычно делаю специальный вариант файла, используемого именно для определённого типа запросов – вот такой, например, используется для генерации сертификата для `www.atraining.ru`:

```
[ req ] default_bits = 4096 default_keyfile = тут путь до файла с закрытым
ключом/private.key distinguished_name = req_distinguished_name encrypt_key
= no prompt = no req_extensions = req_v3_web
[ req_distinguished_name ] countryName = CN stateOrProvinceName = Hong Kong
localityName = Hong Kong organizationName = Advanced Training commonName =
www.atraining.ru
[ req_v3_web ] basicConstraints = CA:FALSE keyUsage = nonRepudiation,
digitalSignature, keyEncipherment subjectAltName = @alt_names
1.3.6.1.5.5.7.1.24 = DER:30:03:02:01:05
[ alt_names ] DNS.1 = www.atraining.ru DNS.2 = atraining.ru
```

Почему отдельный файл с настройками, а не исправление глобального `openssl.conf`? Причина проста – расширение OCSP Must-Staple необязательно нужно для всех сертификатов, создаваемых на этой конкретной системе. К примеру, часть почтовых систем – те же `postfix` и `dovecot` – не хотят брать на себя работу по облегчению проверки сертификатов со стороны клиента, и добавление OCSP Must-Staple будет, фактически, нарушением логики их работы – они будут показывать сертификат, где написано “я, сервер, обязан к этому сертификату приложить зашифрованный OCSP-ответ”, но прикладывать не будут, т.к. не умеют и не планируют уметь.

Поэтому то, что хорошо для сертификатов веб-сервера – необязательно хорошо для любого сертификата вообще.

В приведённом примере, как видно, я объявляю раздел `req_v3_web`, куда вписываю “добавлять OID от OCSP Must-Staple” в формате, пригодном для любых версий OpenSSL.

Использовать этот конфигурационный файл можно примерно так:

```
openssl genrsa -out atraining-ru.key 4096 openssl req -new -sha512 -key  
atrainig-ru.key -out atraining-ru.csr -config atraining-ru.conf
```

Т.е. первой строчкой создаём ключевую пару RSA на 4096 бит, второй – делаем из открытого ключа запрос на сертификат, учитывая конфигурацию в файле `atrainig-ru.conf` (приведён выше).

После этот CSR можно использовать для запроса сертификата у любого CA. Варианты запроса через веб, где надо открыть CSR в блокноте и закопипастить в окошко, рассматривать не будем из-за тривиальности – рассмотрим в качестве примера распространённый сервис Let’s Encrypt.

## 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`, 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.

Выглядеть это в результате будет примерно так:

## Certificate #1: RSA 4096 bits (SHA256withRSA)



### Server Key and Certificate #1

Subject	www.atraining.ru Fingerprint SHA256: 38cb985201c72826aa43814303f132445d5555efd08ed270ec116e73bac86aee Pin SHA256: eQuL6R3TKjWTKiR8pKUGECJ+ruu3QC8QbgVyzmnXq8o=
Common names	www.atraining.ru
Alternative names	atraining.ru www.atraining.ru
Serial Number	03311957e45eacc3b386dd82b20186ba1d19
Valid from	Sun, 08 Apr 2018 18:45:52 UTC
Valid until	Sat, 07 Jul 2018 18:45:52 UTC (expires in 2 months and 27 days)
Key	RSA 4096 bits (e 65537)
Weak key (Debian)	No
Issuer	Let's Encrypt Authority X3 AIA: http://cert.int-x3.letsencrypt.org/
Signature algorithm	SHA256withRSA
Extended Validation	No
Certificate Transparency	Yes (certificate)
OCSP Must Staple	Supported
Revocation information	OCSP OCSP: http://ocsp.int-x3.letsencrypt.org
Revocation status	Good (not revoked)
DNS CAA	Yes policy host: atraining.ru issuewild: digicert.com flags:0 issuewild: globalsign.com flags:0 iodef: https://www.atraining.ru/report-uri flags:0 iodef: mailto:info@atraining.ru flags:0 issue: comodoca.com flags:0 issue: digicert.com flags:0 issue: globalsign.com flags:0 issue: letsencrypt.org flags:0 issuewild: comodoca.com flags:0
Trusted	Yes Mozilla Apple Android Java Windows



### [Поддержка OCSP Must-Staple со стороны сервера](#) (кликните для увеличения до 1004 px на 879 px)

Никаких специальных действий именно в настройках сервера – не нужно.

Расширение OCSP Must-Staple – это просто атрибут в сертификате, чтобы клиент точно знал, что “я всегда присылаю OCSP вместе с сертификатом” и упрощал логику выбора поведения.

## OCSP Expect-Staple

Творческий товарищ [Scott Helme](#) в 2017 году предложил ввести новый HTTP security header – **Expect-Staple**. Суть проста – внутри HTTP-информации поместить поле, в котором можно было бы сказать браузеру, куда жаловаться, если наличие OCSP Stapling'a заявлено, но по факту отсутствует.

Заголовок сделан полностью по аналогии с HSTS, про который подробно есть в [статье про настройку TLS](#). Выглядеть он будет например так:

```
max-age=31536000; report-uri="https://www.atraining.ru/report-uri/expect-staple"; includeSubDomains; preload
```


Это, как понятно, само тело заголовка – а его добавление в случае nginx будет выглядеть, например, так:

```
add_header Expect-Staple 'max-age=31536000; report-  
uri="https://www.atraining.ru/report-uri/expect-staple"; includeSubDomains;  
preload';
```

Формат вполне прост – указывается `report-uri`, куда слать JSON-крики о помощи, `preload` говорит о том, что “надо бы нас добавить в гуглолист хороших HTTPS-сайтов” (про это также есть в [статье про настройку TLS](#)), `max-age` указывает срок (хотя бы год, чтобы добавили), а `includeSubDomains` – что мы не будем тратить трафик на указание этого же заголовка в HTTP-ответах всех субдоменов. В моём случае заголовок отдаётся у `atraining.ru`, но не отдаётся у служебных `cdn.ATRaining.ru` и `s.ATRaining.ru`.

В идеальном случае после добавления этого заголовка и повторного обхода [hstspreload.org](https://hstspreload.org) запись про ваш домен будет доступна подключающимся самый первый раз браузерам и они сразу же будут и подключаться по TLS, и ждать OCSP-ответа, и ругаться по указанному в `Expect-Staple` `report-uri` про то, что такого ответа нет (про настройку `report-uri` можно прочитать [здесь](#)). Фактически, первичное подключение станет быстрее и безопаснее. [Вот здесь](#) можно посмотреть на процесс реализации Expect-Staple в Chromium.

Выглядит добавленный заголовок `Expect-Staple` примерно так:

Security Report Summary	
	Site: <a href="https://www.atraining.ru/">https://www.atraining.ru/</a>
	IP Address: 178.159.49.230
	Report Time: 13 Apr 2018 14:26:41 UTC
	Headers: <div> <span>✔ Strict-Transport-Security</span> <span>✔ X-Content-Type-Options</span> <span>✔ Content-Security-Policy</span> <span>✔ X-XSS-Protection</span> <span>✔ X-Frame-Options</span> <span>✔ Referrer-Policy</span> </div>

Raw Headers	
HTTP/1.1	200 OK
Server	nginx
Date	Fri, 13 Apr 2018 14:26:44 GMT
Content-Type	text/html; charset=UTF-8
Transfer-Encoding	chunked
Connection	keep-alive
Vary	Accept-Encoding
Set-Cookie	ADVANCEDTRAINING=MGS1IeAdnsF0h36lffk6FdWFr5PTNLWA9ISqUzOn15jKD; path=/; domain=www.atraining.ru; secure; HttpOnly
Pragma	no-cache
Link	<https://www.atraining.ru/wp-json/>; rel="https://api.w.org/"
Link	<https://www.atraining.ru/>; rel=shortlink
Link	</wp-content/plugins/disqus-comment-system/public/js/comment_count.js?ver=3.0.15>; rel=preload; as=script
Cache-Control	max-age=0, no-cache, no-store
Strict-Transport-Security	max-age=31536000; includeSubdomains; preload
Public-Key-Pins	max-age=2592000; pin-sha256="h6801m+z8v3zbgkRHpq6L29Esgfzhj89C1SyUCOQmqU="; pin-sha256="InsM2T/O9/j84sjFdnrpsFp3awZj+ZZbYpCWWhGloaHI="; pin-sha256="Vjs8r4z+80wjNcr1YKepWQboSIRi63WswXhIMN+eWys="; report-uri="https://www.atraining.net/report-uri/"; includeSubDomains
X-Content-Type-Options	nosniff
Content-Security-Policy	upgrade-insecure-requests; report-uri https://www.atraining.ru/report-uri/
X-XSS-Protection	1; mode=block
X-Frame-Options	SAMEORIGIN
Referrer-Policy	no-referrer-when-downgrade
Expect-CT	max-age=0, report-uri=https://www.atraining.ru/report-uri/
Expect-Staple	max-age=31536000; report-uri="https://www.atraining.ru/report-uri/"; includeSubDomains; preload

[Поддержка HTTP-заголовка Expect-Staple со стороны сервера](#)  
(кликните для увеличения до 1211 px на 968 px)

Что ж, теперь можно про всякое дополнительное.

## Вспомогательные вопросы при работе с OCSP

Действия, нужные со стороны сервера и клиентов – примерно понятны. Что пригодится админу?

### Как определить время кэширования OCSP-ответа

Вы можете вручную посмотреть прикрепленный OCSP-ответ и увидеть срок годности. Например так:

```
echo QUIT | openssl s_client -connect www.atraining.ru:443 -tls1_2 -status 2> /dev/null | grep -A 17 'OCSP response:' | grep -B 17 'Next Update'
```

Параметры команды достаточно тривиальны – посылается запрос на [www.atraining.ru:443](https://www.atraining.ru:443), используется [TLS 1.2](#), отправка **QUIT** в самом начале нужна, чтобы соединение после получения информации сразу же прервалось, а **grep**’ом мы просто выводим только нужный кусок из рулона текста.

## Надо ли предварительно подготавливать OCSP-ответы



В различных источниках бытует мнение о том, что OCSP – плохая технология, потому что самый первый клиент будет подключаться, а OCSP-ответа нет. Поэтому предлагаются схемы типа “давайте после старта веб-сервиса сами к себе обратимся, симитировав самого первого клиента, чтобы этот вопрос был снят”.

Идея неплохая, но есть мнение, что проще ускорить процесс кэширования OCSP-ответа, а также грамотно настроить тайм-ауты. Тайм-ауты OCSP – это время, за которое веб-сервер, получив запрос от клиента на OCSP Stapling, должен сбегать к OCSP responder’у и принести этот ответ. Соответственно, вопросы быстродействия сети, скорости разрешения DNS-имён и поиска рабочего OCSP responder’а из нескольких – весьма актуальны. Не бойтесь выставить тайм-ауты на 10 или даже 15 секунд – это не приведёт к замедлению работы с клиентом, это приведёт к снижению до нуля числа отбоев по причине “извини, братишка, не успел OCSP тебе передать – вот тебе TLS-ответ формата “уж как могу””.

В nginx существует возможность брать OCSP Stapling из файла – это делается через команду `ssl_stapling_file` и предполагает некий фоновый скрипт, который регулярно (это несложно, зная срок годности OCSP-ответа) ходит наружу и перезаписывает этот файл в случае получения удачного ответа. Применять ли этот метод – решать вам; в больших развёртываниях зачастую проще сделать отдельные системы, прекэширующие все OCSP-ответы для нужных сертификатов и очень быстро доступные для линейки проксирующих и выполняющих SSL termination серверов.

Теперь чуток про безопасность OCSP.

## Вопросы безопасности OCSP

---

Казалось бы – удивительное дело; речь ведь идёт о технологии, которая в подавляющем большинстве случаев ускоряет проверку отзыва сертификата. Вроде как никаких дополнительных вопросов безопасности здесь быть не должно. Однако они есть, и их надо учитывать.

## Концептуальная проблема зависимости от третьей стороны

---

В прекрасном мире высокотехнологичного будущего PKI занимает особое место – помимо ореола чистой науки и математики, PKI базируется на истинной независимости от сторонних участников в вопросе проверки подлинности. В самом деле, вы можете проверить сертификат на валидность лишь обладая нужным ПО, реализующим хорошо известные криптоалгоритмы. Каждая проверка – что повторное вычисление хэша, что проверка цифровой подписи – упрётся в строгую математику, которая скажет, что “если хэши разные – то речь точно о разных входных данных”, или “если то, что обработано открытым ключом, не сходится добитово с тем, что было обработано закрытым – значит, это не пара ключей”.

Проверка на отзыв сертификата базировалась на том, что у вас есть файл со списком отозванных сертификатов. Файл подписан СА, которому вы доверяете. Вы проверяете наличие нужного сертификата в этом файле как вам хочется и когда вам хочется.

В случае же OCSP мы получаем посредника, который не показывает нам оригинальный CRL-файл. Он получает от нас запросы и отвечает, подтверждая лишь свою подлинность. Ответ OCSP responder'a не содержит подтверждения подлинности ответа – лишь то, что ответ получен в определённое время и не был прочитан/модифицирован по пути от вас, запрашивающего, до него, отвечающего.

## Проблема разглашения информации при OCSP-запросе

---

Вы отчитываетесь “какой я сертификат сейчас обрабатываю”, отправляя его идентификатор на OCSP-сервер. Он ведёт логи и имеет список “в какое время с какого адреса какой сертификат проверяли” – что, при доступности базы СА “какой сертификат с каким id я выдал какому домену”, даёт отличную картинку “вот на какие домены ходят с этого сетевого адреса”. При приближении числа SSL/TLS-сайтов к 100% – картинка становится всё более чёткой. Да, а вкупе с тем, что для внутренних систем предприятия – от электронной почты до RDP-серверов – тоже принято запрашивать “нормальные сертификаты от внешних PKI” – то картинка прекрасно дополняется журналированием “на какие внутренние сервера, опубликованные снаружи, ходит”.

Да, и чем более подробно информация о сертификатах дробится – например, “а чё мелочиться, на каждый FQDN по сертификату запросим, бесплатные же” – тем картинка ещё детальнее, потому что вообще однозначная связь “сертификат с серийником N = домену [www.example.com](http://www.example.com)”.

Работаете через VPN? Не беда; вы навряд ли пользуетесь PPTP, а значит попадаете на проверку на отзыв сертификата сервера, и делаете это исключительно разово при установке соединения. Ну, а обращаясь к локальным ресурсам уже после того, как VPN заработал – даже если по адресу вида <https://192.168.0.1/> – вы все равно, получая сертификат, автоматически проверяете его через соединение, которое используется для работы с Интернет, а значит все равно показываете наружу “какой внутренний сервер я открываю и когда”. Вся информация опять-таки прозрачно сводится воедино, т.е. запросы выходят из-под одного адреса.

Как понятно, в серьёзных решениях по части безопасности вопрос маскировки OCSP-запросов и связанных с ними DNS-запросов – действительно актуален. И сбрасывать со счетов свою собственную, хорошо работающую PKI-инфраструктуру – рано, несмотря на крики про “да есть бесплатные вроде работающие в принципе нормальные сервисы”. Есть, но вопросы их монетизации – как всегда, под покровом

лозунгов про Свободу и Открытость – достаточно очевидны. И “в принципе нормальные” – это не “отличные” и даже не хорошие – а “и так сойдёт”. Что далеко не во всех случаях пригодно для применения.

## Резюме

---

Правильная настройка работы OCSP и связанных с этой технологией механизмов может улучшить безопасность и увеличить скорость работы HTTPS-сайтов. Так как сейчас практически весь Интернет состоит именно из них – то не забудьте об этой “мелочи”.

Удач!