

Защищаем и настраиваем протоколы TLS/SSL в Windows Server

 atraining.ru/tls-armoring-windows-server

2014-11-26T11:29:47+08:00

```
C:\Users\Ruslan\Desktop\atcmd\x64\Release64\atcmd.exe
Checking system... NT 6.1 or greater; client OS; WinSock 2.2 started; logged as 'Ruslan', all ok
ATcmd 2.1, build 305
(c) 2011-2015 by Ruslan U. Karmanov @ Advanced Training (info@attraining.net)
Check for new versions @ http://www.attraining.ru/soft/
Press ? to get quick help
Autoupdate disabled
ATRAINING-WIN10(config)#tls
ATRAINING-WIN10(config-tls)#pro
ATRAINING-WIN10(config-tls)#protocols
ATRAINING-WIN10(config-tls-protocols)#?

mpuh          Enable Multi-Protocol Unified Hello (obsolete)
onlytls       Enable only TLS - disable all others
pct10         Enable PCT 1.0 protocol (obsolete)
ssl20         Enable SSL 2.0 protocol (obsolete)
ssl30         Enable SSL 3.0 protocol (obsolete)
tls10         Enable TLS 1.0 protocol
tls11         Enable TLS 1.1 protocol
tls12         Enable TLS 1.2 protocol

end           Exit to global configuration mode
exit          Return to previous configuration mode
no            Disable something or return value to default
quit         Quit

ATRAINING-WIN10(config-tls-protocols)#onlytls

Enabling TLS 1.0 (client) globally
Enabling TLS 1.0 (server) globally
Enabling TLS 1.1 (client) globally
Enabling TLS 1.1 (server) globally
Enabling TLS 1.2 (client) globally
Enabling TLS 1.2 (server) globally
Disabling MPUH (client) globally
Disabling MPUH (server) globally
Disabling PCT 1.0 (client) globally
Disabling PCT 1.0 (server) globally
Disabling SSL 2.0 (client) globally
Disabling SSL 2.0 (server) globally
Disabling SSL 3.0 (client) globally
Disabling SSL 3.0 (server) globally
ATRAINING-WIN10(config-tls-protocols)#
```

Привет.

Защита TLS – штука крайне нужная. У неё много аспектов – как безопасность самой хостовой ОС, на которой развёрнут веб-сервер, так и безопасность работающих приложений, криптографические аспекты и многое другое. Я попробую написать про

то, что с моей точки зрения, является важным и не сильно документированным / очевидным. На этом “вода” про то, что с TLS лучше, чем без TLS, а с настроенным TLS лучше, чем с не настроенным TLS, официально объявляется закончившейся.

Стартовые ограничения статьи – не будет рассматриваться тюнинг SSL, т.к. уже много лет как есть TLS. SSL мы будем выключать полностью. Не будет рассматриваться настройка систем младше NT 6.0, т.к. про это можно найти в [предыдущей статье](#), да и те, кто думает о безопасности, наружу Windows Server 2003й и в 2014м году (*это год написания первой версии статьи, если что – в 2017 она расширена*) не выставляет.

Для большинства операций я буду использовать [ATcmd](#) – им проще делать тонкий тюнинг SSL/TLS. Если хотите – можете найти другие методы по выполнению аналогичных действий на своей ОС, это не критично – функционал называется стандартно, и ту же фрагментацию TLS можно настраивать чем удобно.

Несмотря на позиционирование статьи “весь доступный в данный момент функционал”, описываемые меры в полной мере применимы к более ранним версиям Windows Server – например, 2008 R2.

Бронируем TLS на примере Windows Server 2012 R2 / 2016

Поехали.

Версия SSL/TLS на сервере

Первым делом – проведём инвентаризацию того, что умеет поддерживать SCHANNEL в Windows NT. Это будут:

- Очень старый и просто старый варианты SSL – 2.0 и 3.0
- Редкие и давно устаревшие PCT 1.0 и MPUH
- Протоколы TLS 1.0, TLS 1.1, TLS 1.2 и их варианты для работы поверх UDP – DTLS 1.0 и DTLS 1.2

Разберемся по порядку.

Краткая история вопроса – SSL

Протокол SSL был разработан фирмой Netscape, достаточно давно. Я осознанно пропущу исторические подробности, так как они сейчас уже не особо интересны. В его задачи входило следующее:

- Обязательность подтверждения подлинности сервером
- Опциональная проверка подлинности клиента
- Совместная генерация случайного сеансового ключа
- Поддержка различных симметричных алгоритмов для шифрования данных
- Поддержка различных алгоритмов хэширования для реализации проверки целостности через MAC

Первая версия SSL не особо показывалась публике, отсчёт рабочих версий можно начинать с SSL 2.0 (1995й год). Эта версия была первой, которая эксплуатировалась в production, и достаточно оперативно она была доработана до SSL 3.0. Заметим, что хотя это произошло достаточно давно – в 1996 году – стандарт от этого не стал общим и открытым; он оставался стандартом, разработанным конкретной фирмой Netscape, и для его использования в ряде случаев нужна была лицензия. Опубликован IETF он был недавно; RFC 6101 описывает то, что 15 лет уже стандарт де-факто для защиты сессий множества приложений. Но, по сути, с ним уже давно пора прощаться; [TLS существует годы \(с 1999, если быть точнее\)](#). Поддержка TLS 1.0 есть уже во всех, даже устаревших, продуктах, поэтому мы будем в явном виде отключать SSL.

Краткая история вопроса – PCT 1.0 и MPUN

Протокол Private Communications Technology был разработан Microsoft'ом, чтобы улучшить работу SSL 2.0 и, на самом деле, был совместим с оным в плане формата negotiation, работал лучше и безопаснее. Но важнее то, что он подтолкнул разработку TLS. Ведь по сути, на 1995й год (тогда и был предложен [драфт этого протокола на стандартизацию](#)) ситуация была простой – SSL был частной инициативой Netscape, веб нуждался в подобном стандарте, и то, что ещё один вендор начал делать “Такое же, но чуть лучше, и своё” подтолкнуло к пониманию, что если это продолжить, то будет плохо всем – будет пачка разных вариантов от разных вендоров, частично совместимая и закрытая. Это бы затормозило развитие технологий безопасной работы в Интернете, поэтому PCT благополучно похоронили, а через 3 года появился TLS. Поэтому PCT мы будем отключать в явном виде (хоть это и делается автоматически с ядра NT 6.0, но лучше сделать это явно и жестко, чем оставить на самотёк).

Multi-Protocol Unified Hello – ещё одна древняя попытка Microsoft сделать универсальный безопасный протокол типа SSL. Она совсем древняя, поэтому её тоже надо выключать в явном виде – просто чтобы исключить даже попытки её согласовать или предложить.

Краткая история вопроса – TLS 1.0, 1.1 и 1.2

В 1999 году появляется [TLS 1.0](#). Он заменяет собой вендорский SSL 3.0, являясь очень похожим на него, и пишет в заголовок версию {0x03,0x01}, намекая, что он 3.1.

Версия 1.0, по сути, является базовой, хотя, надо отметить, напрямую TLS 1.0 и SSL 3.0 не совместимы; TLS 1.0 “умеет” работать в режиме совместимости с SSL, но именно в режиме, а не “идентично”, как иногда можно прочесть. Например, у SSL 3.0 есть встроенная в протокол неприятность – половина master key будет создаваться из MD5-хэша достаточно предсказуемых данных, что может привести (учитывая текущее положение MD5) к успешной коллизии атаке (в результате станет известна половина бит мастер-ключа, от которого после генерятся

сессионные ключи, а это практически равнозначно компрометации процесса). У TLS 1.0 это поправили, и схема усложнена – PRF-функцией берутся и MD5 и SHA-1 хэши, после чего хог'ятся, что сводит возможность вышеуказанной атаки к нулю. Кстати, из-за этого момента в алгоритме – завязанности ключевой части процесса на MD5, SSL 3.0 не является [FIPS 140-2](#) совместимым и при включении FIPS 140-2 не согласовывается, а вот TLS 1.0 – уже является. Что же ещё, помимо FIPS-совместимости и модификации PRF-функции?

- Убрана поддержка cipher suites с алгоритмом Fortezza
- Клиент обязан поддерживать cipher suite TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA. То есть, обязательна реализация 3DES у клиента.
- В TLS явно и недвусмысленно описали, как именно делается padding в блочных шифрах (т.е. чем “добиваются” блоки некротной длины в CBC-варианте) – это как раз то, из-за чего для борьбы с уязвимостью POODLE (детали про неё можно посмотреть [здесь](#)) нужно отключать поддержку SSLv3.
- В SSL не было явно указано, как защищать pre-master secret. В TLS – формализовали, поэтому разные реализации стали делать это однотипно.
- В TLS гораздо больше возможных alert'ов со стороны сервера, поэтому клиенту “понятнее” и причины отбоя со стороны сервера, и другие проблемы при handshake.
- Поддержка возможности получить корневой сертификат в составе certificate_list с сервера – в SSL можно было получить лишь intermediate.
- В SSL у клиента была возможность отправлять данные некоторое время после отправки Finished. В TLS это убрали.

Поддержка TLS 1.0 сейчас является абсолютным минимумом для защищённых соединений – но со времён 1.0 много воды утекло и многое поменялось – смотрим дальше.

Промежуточными дополнительными расширениями, интересными для нас, будут [RFC 3268](#) от 2002 года, который добавляет поддержку AES (двух основных вариантов – с 128 и 256 битами, третий штатный вариант со 192 как-то не особо прижился) и [RFC 3546](#) от 2003 года, который добавляет пачку extension'ов, включая в себя расширение механизма handshake, добавление SNI (Server Name Indication), согласования размера TLS-фрагмента и расширенной работы с сертификатами и CRL. Его расширяет [RFC 4366](#) от 2006 года, который добавляет ещё расширений – про них упомянем далее.

В 2006 году появляется [TLS 1.1](#). Что он меняет?

Основное в нём – защита от найденных проблем с атаками на блочные шифры, работающие в CBC-режиме.

- Вектор инициализации ранее считался из остатка после CBC-шифрования предыдущего блока данных. Теперь он явный и прямой зависимости от результата предыдущего шифрования не имеет.
- Улучшена обработка padding errors – они теперь вычисляются на уровне “не совпал MAC”, а не “сбой расшифровки”, что улучшает отделение попыток атаки от технических сбоев.
- Улучшена поддержка session resuming – возможности продолжить прерванную сессию без полной переустановки и пересогласования.

Таким образом, TLS 1.1 – не крупный багфикс TLS 1.0, поддержка которого никак особо не перегружает сервера (это я про миф про то, что “мы не держим TLS старше 1.0, потому что он дико грузит серваки”), но закрывает явно существующие уязвимости. Что далее?

В 2008 году появляется [TLS 1.2](#). Он привносит следующие полезные штуки:

- Связка MD5+SHA-1, используемая в предыдущих версиях, и пришедшая на замену одинокому MD5 в семействе SSL, теперь может быть заменена любым хэш-алгоритмом (предпочтительным является представитель семейства SHA-2, в частности SHA-256)
- Поле Verify_data, по сути – PRF (PRF = pseudorandom function = почти KDF, только KDF-функция делает из чего-то ключ, а PRF рандомизирует несколько входных блоков данных в один) от master_secret, MD5 и SHA-1 хэшей данных и строки finished_label (которая может быть или “server_finished” или “client_finished”), теперь мало того, что считается с нужным хэшем, так ещё и перестало быть ограничено 12ю байтами. Т.е. ранее от всего этого брались последние 96 бит, а теперь – столько, сколько надо, что соответственно усиливает надёжность проверки целостности данных.
- Серьёзные изменения в криптографической части. Хэши семейства SHA-2 теперь обязательны к поддержке, а SHA-256 – рекомендуемый минимальный. Убрана поддержка алгоритмов IDEA и DES – с ними теперь TLS-сессию не согласовать, сервер откажется. Обязательным для поддержки стал cipher suite TLS_RSA_WITH_AES_128_CBC_SHA, без его наличия клиента не подключат.
- Множественные доработки в самом алгоритме, коснувшиеся ужесточения порядка действий (отсекая потенциальные атаки, связанные с излишней гибкостью поведения алгоритма ранее), увеличения количества ситуаций, когда сервер высылает клиенту alert с описанием проблемы, а не просто отключает или игнорирует блок данных. Например, теперь клиент, если у него нет сертификатов, а его запросили, не может промолчать, а обязан выслать ответ с пустым списком сертификатов.

В алгоритме также сделана необязательная поддержка SSL 2.0, а впоследствии, в 2011 году, в [RFC 6176](#), SSL 2.0 практически отключается – клиенту явно запрещается посылать hello-запросы с протоколом ниже {0x03, 0x00}, серверу тоже

запрещается их отправлять, а если к серверу они всё же приходят, он будет их игнорировать. Говоря про TLS 1.2 мы будем подразумевать его в варианте “учитывая RFC 6176”.

Фактически, TLS 1.2 – это линейное улучшение криптографических параметров TLS предыдущих версий, явное отсечение устаревших функций плюс ещё большая формализация алгоритма.

После TLS 1.2 развитие TLS также не остановилось – [RFC 5746](#) добавляет новый вариант безопасного пересогласования ключевой информации, а [RFC 5878](#) – новые расширения для новых типов авторизации, ну а [RFC 6066](#) актуализирует эти, ставшие уже стандартными, расширения.

Отдельно можно добавить то, что оригинальные протоколы – SSL, PCT, TLS – были нацелены на защиту TCP-трафика, но параллельно с реализацией TLS появился и DTLS, предназначенный для защиты потока датаграмм юникастового трафика (это UDP, SCTP, DCCP и RTP). В плане функционала версий DTLS 1.0 – это TLS 1.1, а DTLS 1.2 – TLS 1.2. Технически в плане безопасности отличий в нём будет минимум, поэтому отдельно про него писать не будем, упомянём лишь, что “из коробки” он доступен начиная с ядра NT 6.2, а для предыдущего поколения ОС доступен добавляющий его поддержку (только для UDP) в SCHANNEL патч – [KB 2574819](#). Это нужно, например, для работы RDP 8.0, который умеет работать поверх UDP.

Так что материала много – вперёд, к практике.

Отключаем PCT 1.0, MPUN, SSL 2.0 и SSL 3.0

Итак, теперь практика – выключаем совсем SSL 2.0, SSL 3.0, PCT 1.0, MPUN, включаем в явном виде TLS 1.1 и TLS 1.2. С TLS 1.0 ситуация на Ваш выбор – хорошо бы его, конечно, выключить, но надо убедиться, что клиенты это выдержат. В первую очередь это коснётся мобильных клиентов, где (особенно в Android) зоопарк полурботающих реализаций TLS – дело нормальное. Хотя и десктопные не отстают – в той же Filezilla, использующей gnutls, реализацию допилили до полностью RFCшной только летом 2013го года – для примера, у Microsoft это было с Windows Server 2008R2, т.е. с 2009 года. Ну, опенсорс – роль догоняющего-допиливающего-на-бегу-чтобы-хоть-кое-как-работало для них привычная, так что ничего сверхординарного.

Запустим ATcmd и выключим-включим нужное – для этого зайдём в контекст **tls**, потом в субконтекст **protocols** и выполним удобную команду **onlytls**:

```
C:\Users\Ruslan\Desktop\atcmd\x64\Release64\atcmd.exe
Checking system... NT 6.1 or greater; client OS; WinSock 2.2 started; logged as 'Ruslan', all ok
ATcmd 2.1, build 305
(c) 2011-2015 by Ruslan U. Karmanov @ Advanced Training (info@atraining.net)
Check for new versions @ http://www.atraining.ru/soft/
Press ? to get quick help
Autoupdate disabled
ATRAINING-WIN10(config)#tls
ATRAINING-WIN10(config-tls)#pro
ATRAINING-WIN10(config-tls)#protocols
ATRAINING-WIN10(config-tls-protocols)#?

  mpuh          Enable Multi-Protocol Unified Hello (obsolete)
  onlytls       Enable only TLS - disable all others
  pct10         Enable PCT 1.0 protocol (obsolete)
  ssl20         Enable SSL 2.0 protocol (obsolete)
  ssl30         Enable SSL 3.0 protocol (obsolete)
  tls10         Enable TLS 1.0 protocol
  tls11         Enable TLS 1.1 protocol
  tls12         Enable TLS 1.2 protocol

  end           Exit to global configuration mode
  exit          Return to previous configuration mode
  no            Disable something or return value to default
  quit          Quit

ATRAINING-WIN10(config-tls-protocols)#onlytls

  Enabling TLS 1.0 (client) globally
  Enabling TLS 1.0 (server) globally
  Enabling TLS 1.1 (client) globally
  Enabling TLS 1.1 (server) globally
  Enabling TLS 1.2 (client) globally
  Enabling TLS 1.2 (server) globally
  Disabling MPUH (client) globally
  Disabling MPUH (server) globally
  Disabling PCT 1.0 (client) globally
  Disabling PCT 1.0 (server) globally
  Disabling SSL 2.0 (client) globally
  Disabling SSL 2.0 (server) globally
  Disabling SSL 3.0 (client) globally
  Disabling SSL 3.0 (server) globally
ATRAINING-WIN10(config-tls-protocols)#
```

[Выключаем все SSL, MPUH и PCT и явно включаем TLS](#)

[\(кликните для увеличения до 609 px на 660 px\)](#)

Посмотрим на результаты, выполнив команду **show tls**:


```
C:\Users\Ruslan\Desktop\atcmd\x64\Release64\atcmd.exe
ATRAINING-WIN10(config)#sh tls

TLS Renegotiation
  From this host: Any type of TLS renegotiation enabled
  To this host: Any type of TLS renegotiation enabled
Client authentication trust mode: Machine trust
Kernel mode SSL: Disabled
Intermediate certificates download: From local store only
Send RFC 5746 SCSU (0x00 0xFF) in TLS options: Implicitly disabled
Send SSL Close Notify: Implicitly disabled
Issuer cache
  Size: 100 sessions
  Timeout: 600 sec
SSL/TLS-sessions cache
  Maximum elements: 50000 sessions
  Timeout (as a server): 72000 sec
  Timeout (as a client): 72000 sec
SSL certificate mapping methods: Support all
Send trusted issuers CIL to connecting clients: Enabled

Fragmented TLS handshake message size limit
  As a client: Fragmented messages are not processed
  As a server with non-auth client: 16384 bytes
  As a server with SSL auth client: 32768 bytes
Extra Record processing: Opt-in mode
SCHANNEL Log level: Errors

Protocols settings
PCT 1.0 usage for outgoing sessions is Disabled; accepting for incoming is Disabled
MPUH for outgoing sessions is Disabled; accepting for incoming is Disabled
SSL 2.0 for outgoing sessions Disabled; accepting for incoming is Disabled
SSL 3.0 for outgoing sessions Disabled; accepting for incoming is Disabled
TLS 1.0 for outgoing sessions Enabled; accepting for incoming is Enabled
TLS 1.1 for outgoing sessions Enabled; accepting for incoming is Enabled
TLS 1.2 for outgoing sessions Enabled; accepting for incoming is Enabled

ATRAINING-WIN10(config)#_
```

[Смотрим состояние протоколов SSL, MPUH, PCT и TLS на хосте](#)
([кликните для увеличения до 742 px на 478 px](#))

Вполне ОК. Теперь надо тюнить выживший TLS.

Пересогласование TLS

В TLS есть интересные дополнительные механизмы, которые нуждаются в настройке. Первый из них – это пересогласование. Суть достаточно простая – во всех протоколах, которые обеспечивают конфиденциальность данных, принято время от времени менять ключи, которые используются для этой задачи. Смена их может быть разной – как частичной (например, в IPsec без PFS – когда из одного ключевого материала периодически “нарезаются” разные ключи), так и полной (когда новый ключевой материал генерится каждый раз заново). Идея в том, что эта операция – смены ключей – крайне важна и её проведение практически всегда связано с нагрузкой на CPU. В классическом TLS пересогласование мог “заказать” клиент фактически в любой момент, что и использовалось для возможной атаки (т.е. можно подключиться к серверу и заставить его малыми усилиями со своей стороны постоянно и неограниченно делать сложные вычислительные задачи).

Безопасное пересогласование TLS описывается в стандарте [RFC 5746](#) и решает проблему с этой возможной уязвимостью.

По сути, Windows-хост может работать в плане этого RFC в двух режимах – в режиме совместимости (т.е. допускать и небезопасное, “классическое” пересогласование TLS), и в “безопасном”, допуская только пересогласование в

новом формате. По умолчанию, работа идёт в режиме совместимости. Это не всегда полезно, поэтому надо знать, как включать только безопасное пересогласование TLS.

Кроме того, в данном стандарте описывается специальный workaround для старых серверов, которые не поддерживают безопасное пересогласование – таким серверам отправляется специальный псевдо-cipher-suite, называемый Signaling Cipher Suite Value (SCSV), с кодом 0x00FF. Устаревший сервер проигнорирует этот неизвестный cipher suite, а клиент догадается, что сервер старый и не умеет новый безопасный renegotiation_info.

Вообще, если уж совсем закручивать гайки, то пересогласование надо выключать и, если уж оставлять, то только при TLS 1.2, потому что даже безопасное пересогласование при версиях TLS до 1.2й предоставляет шанс DoS на сервер. Но учитывайте – достаточно много софта этот вариант (выключение пересогласования) не переживёт. Поэтому мы включим только безопасный вариант пересогласования и явно выключим поддержку SCSV. Если она останется, то наш TLS будет посылать “заглушку” вместо запроса возможности безопасного пересогласования.

Т.е. суть наших настроек сейчас проста – мы предполагаем, что все сервера умеют работать с RFC 5746 (стандарт от 2010 года, так что времени уже прошло достаточно). Отмечу, что подобные настройки надо сделать и на серверах, которые иницируют TLS-подключение на другие хосты (например, почтовых), ведь в этом случае они выступают как клиенты.

Включаем поддержку TLS Renegotiation Indication Extension

В том же контексте **tls** у нас есть команда **renego**, которая по запросу покажет, что она умеет:

```
C:\Users\Ruslan\Desktop\atcmd\x64\Release64\atcmd.exe
Check for new versions @ http://www.atraining.ru/soft/
Press ? to get quick help
Autoupdate disabled
ATRAINING-WIN10<config>#tls
ATRAINING-WIN10<config-tls>#?

  cache          Enter TLS cache subcontext
  cs             Enter TLS cipher suites subcontext
  frag          Enter TLS fragmentation subcontext
  protocols      Enter TLS protocols subcontext

  clienttrust    Client authentication trust mode
  close-notify   Send SSL Close Notify
  intermediate   Intermediate certificate download source
  kernelmode     Enable SSL Kernel Mode
  loglevel       Set SCHANNEL log level
  renegot        Set TLS renegotiation mode
  scsv           Send SCSV (0x00 0xFF) in TLS options
  sendca         Send root CA CTL to client
  sendextra      How to send and receive TLS Frag flag

  end            Exit to global configuration mode
  exit           Return to previous configuration mode
  no             Disable something or return value to default
  quit           Quit

ATRAINING-WIN10<config-tls>#renego ?

  all            Enable all types of TLS renegotiation
  secure         Enable only secure TLS renegotiation
  none           Disable all types of TLS renegotiation
```

[Варианты настройки пересогласования TLS](#)

([кликните для увеличения до 614 px на 456 px](#))

Выберем безопасный вариант, указав **renego secure**:

```
C:\Users\Ruslan\Desktop\atcmd\x64\Release64\atcmd.exe
Checking system... NT 6.1 or greater; client OS; WinSock 2.2 sta
rted; logged as 'Ruslan', all ok
ATcmd 2.1, build 305
(c) 2011-2015 by Ruslan U. Karmanov @ Advanced Training <info@at
raining.net>
Check for new versions @ http://www.atraining.ru/soft/
Press ? to get quick help
Autoupdate disabled
ATRaining-WIN10<config>#tls
ATRaining-WIN10<config-tls>#renego secure
    Enabling only secure TLS renegotiation... done
ATRaining-WIN10<config-tls>#_
```

[Включаем только безопасное пересогласование TLS](#)
(кликните для увеличения до 533 px на 394 px)

Явно выключаем SCSV

Ну и явно выключим SCSV:

```
C:\Users\Ruslan\Desktop\atcmd\x64\Release64\atcmd.exe
Checking system... NT 6.1 or greater; client OS; WinSock 2.2 started; logged as 'Ruslan', all ok
ATcmd 2.1, build 305
(c) 2011-2015 by Ruslan V. Karmanov @ Advanced Training (info@advancedtraining.net)
Check for new versions @ http://www.atraining.ru/soft/
Press ? to get quick help
Autoupdate disabled
ATRAINING-WIN10(config)#tls
ATRAINING-WIN10(config-tls)#no scsv
    Don't send SCSU pseudo-cipher-suite on connection
ATRAINING-WIN10(config-tls)#
```

[Выключаем SCSV в TLS](#)

[\(кликните для увеличения до 523 px на 477 px\)](#)

С этим вроде всё. Понятное дело, если хочется супер-защиты, то надо **renego none**, если не TLS 1.2.

Дополнительно, из практики, добавлю, что не всё так однозначно в плане поддержки безопасного пересогласования TLS – допустим, сервера Office 365 не полностью поддерживают данный стандарт, поэтому если Вы включите только безопасное пересогласование TLS, то можете получить проблемы с “облачным” Lync 2013 – выглядеть эти проблемы будут как невозможность подключиться к демонстрации слайдов или whiteboard, в локальных логах Вы увидите событие 36888 от SCHANNEL, в котором будет сообщаться об ошибке TLS с кодом 40 и The Windows SChannel error state is 1207. Так что если видите что-то подобное – попробуйте восстановить настройки, опять включив “любое пересогласование”, а не только безопасное.

Фиксируем только нужные cipher suites

Итак, при согласовании TLS один из ключевых моментов – это выбор комплекта допустимых cipher suites. Вот как выглядит полный список поддерживаемых в CNG наборов:

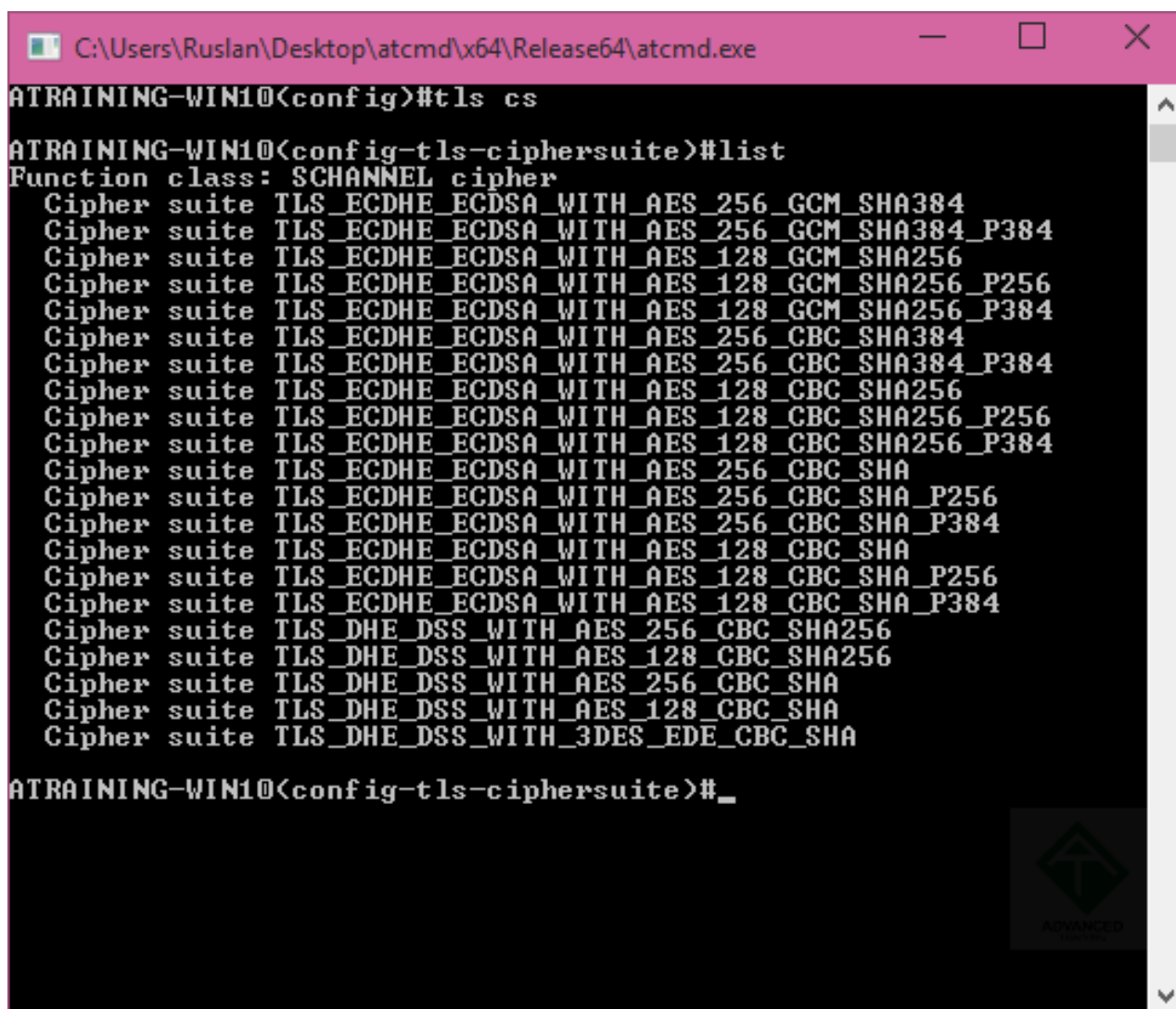
```
TLS_RSA_WITH_AES_128_CBC_SHA TLS_RSA_WITH_AES_256_CBC_SHA
TLS_RSA_WITH_RC4_128_SHA TLS_RSA_WITH_3DES_EDE_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA_P256
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA_P384
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA_P521
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA_P256
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA_P384
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA_P521
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA_P256
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA_P384
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA_P521
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA_P256
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA_P384
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA_P521 TLS_DHE_DSS_WITH_AES_128_CBC_SHA
TLS_DHE_DSS_WITH_AES_256_CBC_SHA TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA
TLS_RSA_WITH_RC4_128_MD5 SSL CK_RC4_128_WITH_MD5
SSL CK_DES_192_EDE3_CBC_WITH_MD5 TLS_RSA_WITH_NULL_SHA
TLS_RSA_WITH_NULL_MD5 TLS_RSA_WITH_AES_128_CBC_SHA256
TLS_RSA_WITH_AES_256_CBC_SHA256 TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256_P256
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256_P384
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256_P521
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384_P256
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384_P384
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384_P521
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256_P256
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256_P384
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256_P521
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384_P384
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384_P521
TLS_DHE_DSS_WITH_AES_128_CBC_SHA256 TLS_DHE_DSS_WITH_AES_256_CBC_SHA256
TLS_RSA_WITH_NULL_SHA256 TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256_P256
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256_P384
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256_P521
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384_P384
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384_P521
```

Внушительно. Надо сокращать. Первым делом – убираем все те, кто содержит в себе:

- SSL_ – варианты явно устарели и должны быть удалены, нет никакого смысла предъявлять их при согласовании что серверу, что клиенту.
- TLS_RSA_ – во-первых при их согласовании не получится Forward Secrecy, во-вторых возможна потенциальная уязвимость FREAK, базирующаяся на использовании старых “экспортных” RSA.
- Варианты с NULL – это когда согласуется “SSL без шифрования” – нам категорически не подходят.

- DES / 3DES / RC4 – данные варианты на фоне ощутимо более быстрого и надёжного AES, который к тому же на современных процессорах аппаратно ускоряется, не интересны.
- MD5 / SHA-1 – даже Windows XP и Windows Server 2003 с нужным патчем поддерживает хэши семейства SHA-2, поэтому нет смысла предлагать и поддерживать старые варианты, тем более, что выбор MD5, допустим, выглядит странно – учитывая, что даже в TLS 1.0 хэш был MD5+SHA-1, явный выбор одинокого MD5 – удивителен.

Для такого удаления неплохо подходит новый контекст [ATcmd](#), который называется **tls ciphersuites**. В нём будет удобная команда **clean**, которая позволит прямо “на ходу”, без перезагрузки хоста и переприменения групповых политик, удалять из списка активных cipher suites нужные по какому-либо критерию. Например, так будет выглядеть удаление из списка согласуемых всех cipher suites, которые содержат в названии “DES”:



```

C:\Users\Ruslan\Desktop\atcmd\x64\Release64\atcmd.exe
ATRAINING-WIN10<config>#tls cs
ATRAINING-WIN10<config-tls-ciphersuite>#list
Function class: SCHANNEL cipher
Cipher suite TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
Cipher suite TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384_P384
Cipher suite TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
Cipher suite TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256_P256
Cipher suite TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256_P384
Cipher suite TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
Cipher suite TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384_P384
Cipher suite TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
Cipher suite TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256_P256
Cipher suite TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256_P384
Cipher suite TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
Cipher suite TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA_P256
Cipher suite TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA_P384
Cipher suite TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
Cipher suite TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA_P256
Cipher suite TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA_P384
Cipher suite TLS_DHE_DSS_WITH_AES_256_CBC_SHA256
Cipher suite TLS_DHE_DSS_WITH_AES_128_CBC_SHA256
Cipher suite TLS_DHE_DSS_WITH_AES_256_CBC_SHA
Cipher suite TLS_DHE_DSS_WITH_AES_128_CBC_SHA
Cipher suite TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA
ATRAINING-WIN10<config-tls-ciphersuite>#_
  
```

[Удаление из списка согласовываемых TLS cipher suites по определённому критерию \(кликните для увеличения до 533 px на 453 px\)](#)

```
C:\Users\Ruslan\Desktop\atcmd\x64\Release64\atcmd.exe
ATRAINING-WIN10(config-tls-ciphersuite)#clean DES
Searching 'DES':
Founded cipher suite TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
Founded cipher suite TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384_P384
Founded cipher suite TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
Founded cipher suite TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256_P256
Founded cipher suite TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256_P384
Founded cipher suite TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
Founded cipher suite TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384_P384
Founded cipher suite TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
Founded cipher suite TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256_P256
Founded cipher suite TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256_P384
Founded cipher suite TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
Founded cipher suite TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA_P256
Founded cipher suite TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA_P384
Founded cipher suite TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
Founded cipher suite TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA_P256
Founded cipher suite TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA_P384
Founded cipher suite TLS_DHE_DSS_WITH_AES_256_CBC_SHA256
Founded cipher suite TLS_DHE_DSS_WITH_AES_128_CBC_SHA256
Founded cipher suite TLS_DHE_DSS_WITH_AES_256_CBC_SHA
Founded cipher suite TLS_DHE_DSS_WITH_AES_128_CBC_SHA
Founded cipher suite TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA - will be removed
Removing CHANNEL cipher suite 'TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA'... done

Now:
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384_P384
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256_P256
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256_P384
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256_P256
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256_P384
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA_P256
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA_P384
```

Удаление из списка согласовываемых TLS cipher suites по определённому критерию - результат

(кликните для увеличения до 633 px на 535 px)

Если что-то удалили зря – есть команда **add**, которая добавляет в верх списка (т.е. самым первым на согласование) указанный cipher suite.

Так вот, продолжим. Потенциальный список сразу сокращается:

```
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA_P256
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA_P384
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA_P521
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA_P256
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA_P384
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA_P521
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA_P256
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA_P384
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA_P521
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA_P256
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA_P384
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA_P521
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256_P256
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256_P384
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256_P521
```


TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384_P256
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384_P384
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384_P521
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256_P256
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256_P384
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256_P521
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384_P384
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384_P521
TLS_DHE_DSS_WITH_AES_128_CBC_SHA256 TLS_DHE_DSS_WITH_AES_256_CBC_SHA256
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256_P256
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256_P384
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256_P521
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384_P384
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384_P521

Не забудем в пылу борьбы, что для согласования TLS 1.0 надо обязательно оставить TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA, а TLS 1.2 – TLS_RSA_WITH_AES_128_CBC_SHA. Т.е. в зависимости от того, включим ли мы TLS 1.0, нам придётся добавить первый из упомянутых либо нет. Список все равно внушительный, поэтому при его дальнейшем урезании будем исходить из следующих соображений.

1. ECDHE лучше, чем DHE, благодаря дополнительной “степени защиты” в виде усложнения алгоритма генерации ключевого материала (при помощи эллиптических кривых). Плюс ECDHE быстрее. Генерация ключевого материала – по сути, самое важное, т.к. лобовая атака на AES – крайне маловероятна, а вот на предсказуемость генерации ключа – гораздо более реалистична. Мы не можем явно выбрать группу DH, а то бы, конечно, по меньшей мере отказались от групп 1,2,5 и выбрали бы 16, ну или 14-15, но нам тут такой выбор не предоставляется, поэтому, чтобы форсировать группы на 19-20ю, мы выберем ECDHE.
2. AES-128 надо предпочесть AES-256 в исключительно редком варианте, когда криптографические вычисления (в силу огромного потока трафика) нагружают CPU так, что это является критичным. Хотя в таком случае лучше подумать о более серьёзном процессоре, ну или балансировке нагрузки, но тут как получится уж – а мы пока остановимся на логике “если уж поддерживается AES, то AES-256”.
3. Выбор хэша из SHA-2 (SHA-256 или SHA-384 или SHA-512) – по сути, меньше всего влияющее на безопасность трафика дело, т.к. что на SHA-256, что на более старшие хэши, атаки именно на фальсификацию “на лету” пока особо не придумано. Нам важнее, чтобы быть более защищённым в ситуации “кто-то полностью заснифил сессию и в оффлайне хочет расшифровать её содержимое”, а в таком сценарии хэш особо не интересен. Поэтому я бы предложил остановиться на SHA-512, исключительно по причине сомнительности траты дополнительных вычислительных ресурсов на “старшие” варианты хэширования.

4. AES-GCM лучше, чем AES-CBC, но он бывает только в TLS 1.2, исключительно. Поэтому оставлять его как единственный вариант – правильно лишь при абсолютной уверенности, что все хосты, взаимодействующие по TLS, поддерживают 1.2 со всеми “новшествами”.

Переберём список в соответствии с этой логикой и получим следующий вариант:

- Всё с TLS_ECDHE_ECDSA_* , с AES-256, в порядке убывания стойкости хэшей/EC-групп
- Всё с TLS_ECDHE_RSA_* , с AES-256, в порядке убывания стойкости хэшей/EC-групп

Если есть одинаковые варианты, различающиеся только по AES, то вариант с AES-GCM выше.

Плюс, понятное дело, обязательный минимальный для TLS 1.2 вариант TLS_RSA_WITH_AES_128_CBC_SHA (если надо TLS 1.0, то ещё и TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA). Их наличие, конечно, не очень хорошо, но согласование пойдёт со “старших” cipher suites, поэтому данная ситуация будет все равно гораздо лучше, чем дефолтная.

Действуем – если используете ATcmd, то можете править командами **add**, **clean** и видеть результаты сразу, на ходу. Если нужно задать в масштабах домена – неплохо будет сделать отдельную политику типа “Hisec TLS” – надо будет зайти в настройку групповых политик, выбрать там **Computer Configuration -> Administrative Templates -> Network -> SSL Configuration Settings -> SSL Cipher Suite Order** и указать нужные cipher suites – как положено по инструкции, через запятую и без пробелов, начиная с самых стойких. Строка ограничена 1023 символами, но нам их, после секвестра списка, вполне хватит.

Список можно подсократить, исходя из логики “используется только современный браузер, все субварианты указывать нет смысла, если поддерживает SHA-2, то там и SHA-256, и SHA-384 сразу точно есть”. В итоге он может выглядеть, допустим, так:

TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384_P521,
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384_P521,
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384_P521, TLS_RSA_WITH_AES_128_CBC_SHA

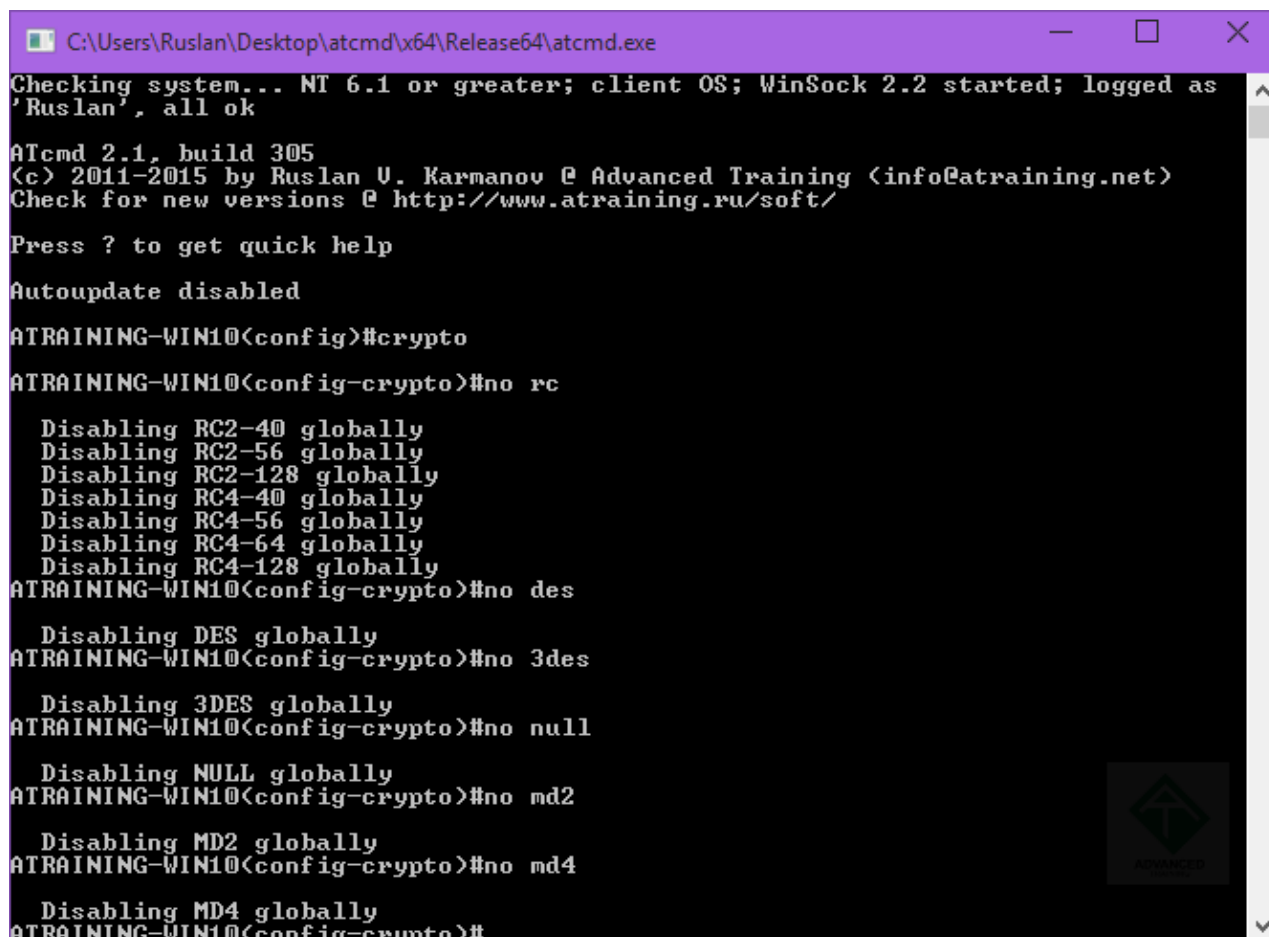
Далее – жёстко убираем даже малейшую возможность использования слабых криптоалгоритмов.

Блокируем небезопасные криптоалгоритмы

В Windows поддерживается множества криптоалгоритмов, которые на данный момент уже совсем не нужны. Например, есть поддержка хэшей MD2 и MD4 – они использовались в ранних реализациях стека IPsec (который для Windows 2000 разрабатывала компания Cisco, поэтому он с первой же версии работал, а не как иногда бывает), а сейчас встречаются разве что в древних сертификатах Verisign.

Или классический DES, который в силу роста вычислительных мощностей, да и того, что в ключе из 64 бит только 56 являются уникальными, уже не безопасен, т.к. его bruteforce доступен коммерческим заказчикам. Или замечательные по стойкости шифры RC2 на 40 и 56 бит, или RC4 (который уже вполне официально, с февраля 2015 года, в TLS поддерживать не надо – см. [RFC 7465](https://tools.ietf.org/html/rfc7465)).

В общем, мы в явном виде “убьём” в системе следующие криптоалгоритмы: NULL, RC2, RC4, DES, 3DES, MD2, MD4. На всякий случай, так сказать.

A screenshot of a Windows command prompt window titled "C:\Users\Ruslan\Desktop\atcmd\x64\Release64\atcmd.exe". The window shows the output of the ATcmd application. It starts with a system check, then displays version information and a copyright notice for Ruslan U. Karmanov. The user enters the command "crypto" to enter the cryptographic configuration menu. Subsequently, the user enters "no rc" to disable RC algorithms, "no des" to disable DES, "no 3des" to disable 3DES, "no null" to disable NULL, "no md2" to disable MD2, and "no md4" to disable MD4. The output for each command shows the specific algorithms being disabled globally.

```
C:\Users\Ruslan\Desktop\atcmd\x64\Release64\atcmd.exe
Checking system... NT 6.1 or greater; client OS; WinSock 2.2 started; logged as
'Ruslan', all ok

ATcmd 2.1, build 305
(c) 2011-2015 by Ruslan U. Karmanov @ Advanced Training (info@atraining.net)
Check for new versions @ http://www.atraining.ru/soft/

Press ? to get quick help
Autoupdate disabled

ATRAINING-WIN10(config)#crypto
ATRAINING-WIN10(config-crypto)#no rc

  Disabling RC2-40 globally
  Disabling RC2-56 globally
  Disabling RC2-128 globally
  Disabling RC4-40 globally
  Disabling RC4-56 globally
  Disabling RC4-64 globally
  Disabling RC4-128 globally
ATRAINING-WIN10(config-crypto)#no des

  Disabling DES globally
ATRAINING-WIN10(config-crypto)#no 3des

  Disabling 3DES globally
ATRAINING-WIN10(config-crypto)#no null

  Disabling NULL globally
ATRAINING-WIN10(config-crypto)#no md2

  Disabling MD2 globally
ATRAINING-WIN10(config-crypto)#no md4

  Disabling MD4 globally
ATRAINING-WIN10(config-crypto)#
```

[Отключаем небезопасные криптоалгоритмы в Windows Server](#)
(кликните для увеличения до 665 px на 488 px)

Задаём минимальное количество бит для DH

Для создания поверх недоверенной среды передачи данных совместно используемых сессионных бит обычно используется алгоритм Диффи-Хеллмана. Чем больше бит стартово создаётся – тем потенциально безопаснее сессия. Впрочем, не всегда – но данная тема выходит за рамки настройки TLS в Windows Server. Для нас будет важно, чтобы TLS-сессии не согласовывались, используя малые количества бит в DH-обмене – например, 512 или 768.

Мы выставим минимальным значением 2048 бит (это соответствует 14й группе DH и отсечёт согласование 1й, 2й и 5й групп). Это несложно и делается через ATcmd – нужно зайти в контекст **crypto** и ввести команду **dh-minbits 2048**:

```
C:\Users\Administrator\Desktop\atcmd-311\atcmd.exe
HOST1(config)#crypto
HOST1(config-crypto)#dh-m
HOST1(config-crypto)#dh-minbits 2048

Setting minimum DH exchange bits to 2048

HOST1(config-crypto)#exit

HOST1(config)#sh crypto
Hash functions status
MD2: On MD4: On MD5: On
SHA-1: On SHA-2/256: On SHA-2/384: On SHA-2/512: On

Ciphers status
DES: On
3DES: On
AES-128: On
AES-192: On
AES-256: On
RC2-40/128: On
RC2-56/56: On
RC2-56/128: On
RC2-128/128: On
RC4-40/128: On
RC4-56/128: On
RC4-64/128: On
RC4-128/128: On
NULL cipher suite: On

FIPS-140 mode
In registry (become active after reboot): Disabled
In registry (obsolete from NT 6.0): Undefined
Active now: Disabled

Key exchange status
Diffie-Hellman: On (with minimum length of 2048 bits)
PKCS: On
ECDH: On

HOST1(config)#_
```

[Настраиваем минимальное число бит у DH-генерации в Windows Server](#)
(кликните для увеличения до 976 px на 754 px)

Управляем серверным комплектом ECC

Криптография с использованием эллиптических кривых поддерживается в Windows Server начиная с NT 6.0 / CNG. Основным плюсом, на момент начала поддержки, была бОльшая скорость при сопоставимой стойкости. То есть никто не спорит, что вы до сих пор можете чувствовать себя достаточно защищённым, если будете использовать RSA с ключами по 4096 бит или даже больше – просто ECC имеет сопоставимую стойкость при ощутимо меньшем затраченном процессорном времени и трафике обмена.

Начиная с Windows 10 / Windows Server 2016, вы можете штатно управлять тем, какие именно эллиптические кривые могут быть использованы в SSL/TLS-cipher suites на вашей системе. Первым делом можно посмотреть, какие ECC поддерживаются на конкретном хосте в данный момент (их список расширяем, поэтому может меняться) – командой `certutil -DisplayEccCurve`:

```
Windows PowerShell
PS C:\Users\Ruslan> certutil -DisplayEccCurve
Microsoft SSL Protocol Provider:
-----
Curve Name          Curve OID          Public Key Length  CurveType          EccCurveFlags
-----
curve25519          1.2.840.10045.3.1.7 255              29                 0xa
nistP256            1.3.132.0.34       256              23                 0x7
nistP384            1.3.132.0.34       384              24                 0x7
brainpoolP256r1     1.3.36.3.3.2.8.1.1.7 256              26                 0x7
brainpoolP384r1     1.3.36.3.3.2.8.1.1.11 384              27                 0x7
brainpoolP512r1     1.3.36.3.3.2.8.1.1.13 512              28                 0x7
nistP192            1.2.840.10045.3.1.1 192              19                 0x7
nistP224            1.3.132.0.33       224              21                 0x7
nistP521            1.3.132.0.35       521              25                 0x7
secP160k1           1.3.132.0.9         160              15                 0x7
secP160r1           1.3.132.0.8         160              16                 0x7
secP160r2           1.3.132.0.30        160              17                 0x7
secP192k1           1.3.132.0.31        192              18                 0x7
secP192r1           1.2.840.10045.3.1.1 192              19                 0x7
secP224k1           1.3.132.0.32        224              20                 0x7
secP224r1           1.3.132.0.33        224              21                 0x7
secP256k1           1.3.132.0.10        256              22                 0x7
secP256r1           1.2.840.10045.3.1.7 256              23                 0x7
secP384r1           1.3.132.0.34       384              24                 0x7
secP521r1           1.3.132.0.35       521              25                 0x7

CNG Curves:
-----
Curve Name          Curve OID          Public Key Length
-----
brainpoolP160r1     1.3.36.3.3.2.8.1.1.1 160
brainpoolP160t1     1.3.36.3.3.2.8.1.1.2 160
brainpoolP192r1     1.3.36.3.3.2.8.1.1.3 192
brainpoolP192t1     1.3.36.3.3.2.8.1.1.4 192
brainpoolP224r1     1.3.36.3.3.2.8.1.1.5 224
brainpoolP224t1     1.3.36.3.3.2.8.1.1.6 224
brainpoolP256r1     1.3.36.3.3.2.8.1.1.7 256
brainpoolP256t1     1.3.36.3.3.2.8.1.1.8 256
brainpoolP320r1     1.3.36.3.3.2.8.1.1.9 320
brainpoolP320t1     1.3.36.3.3.2.8.1.1.10 320
brainpoolP384r1     1.3.36.3.3.2.8.1.1.11 384
brainpoolP384t1     1.3.36.3.3.2.8.1.1.12 384
brainpoolP512r1     1.3.36.3.3.2.8.1.1.13 512
brainpoolP512t1     1.3.36.3.3.2.8.1.1.14 512
curve25519          1.2.840.10045.3.1.7 255
ec192wapi           1.2.156.1.1235.1.1.2.1 192
nistP192            1.2.840.10045.3.1.1 192
nistP224            1.3.132.0.33       224
nistP256            1.2.840.10045.3.1.7 256
nistP384            1.3.132.0.34       384
nistP521            1.3.132.0.35       521
numsP256t1          1.3.132.0.35       256
numsP384t1          1.3.132.0.35       384
numsP512t1          1.3.132.0.35       512
secP160k1           1.3.132.0.9         160
secP160r1           1.3.132.0.8         160
secP160r2           1.3.132.0.30        160
secP192k1           1.3.132.0.31        192
secP192r1           1.2.840.10045.3.1.1 192
secP224k1           1.3.132.0.32        224
secP224r1           1.3.132.0.33        224
secP256k1           1.3.132.0.10        256
secP256r1           1.2.840.10045.3.1.7 256
secP384r1           1.3.132.0.34       384
secP521r1           1.3.132.0.35       521
wtls7               1.3.132.0.30        160
wtls9               2.23.43.1.4.9       160
wtls12              1.3.132.0.33        224
x962P192v1          1.2.840.10045.3.1.1 192
x962P192v2          1.2.840.10045.3.1.2 192
x962P192v3          1.2.840.10045.3.1.3 192
x962P239v1          1.2.840.10045.3.1.4 239
x962P239v2          1.2.840.10045.3.1.5 239
x962P239v3          1.2.840.10045.3.1.6 239
```

[Список поддерживаемых в Windows Server эллиптических кривых](#)
(кликните для увеличения до 856 px на 937 px)

Список большой, но по факту настройкой “по умолчанию” будет являться комплект из быстрой 25519, и двух NIST’овских – P256 и P384.

```
C:\Users\Ruslan\Desktop\atcmd\x64\Release64\atcmd.exe
ATRAINING-WIN10(config)#sh tls

TLS Renegotiation
  From this host: Any type of TLS renegotiation enabled
  To this host: Any type of TLS renegotiation enabled
Client authentication trust mode: Machine trust
Kernel mode SSL: Disabled
Intermediate certificates download: From local store only
Send RFC 5746 SCSV (0x00 0xFF) in TLS options: Implicitly disabled
Send SSL Close Notify: Implicitly disabled
Issuer cache
  Size: 100 sessions
  Timeout: 600 sec
SSL/TLS-sessions cache
  Maximum elements: 10000 sessions
  Timeout (as a server): 10 hours
  Timeout (as a client): 10 hours
SSL certificate mapping methods: Support all
Send trusted issuers CTL to connecting clients: Enabled

Fragmented TLS handshake message size limit
  As a client: 32768 bytes
  As a server with non-auth client: 16384 bytes
  As a server with SSL auth client: 32768 bytes
Extra Record processing: Opt-in mode
SCHANNEL Log level: Errors

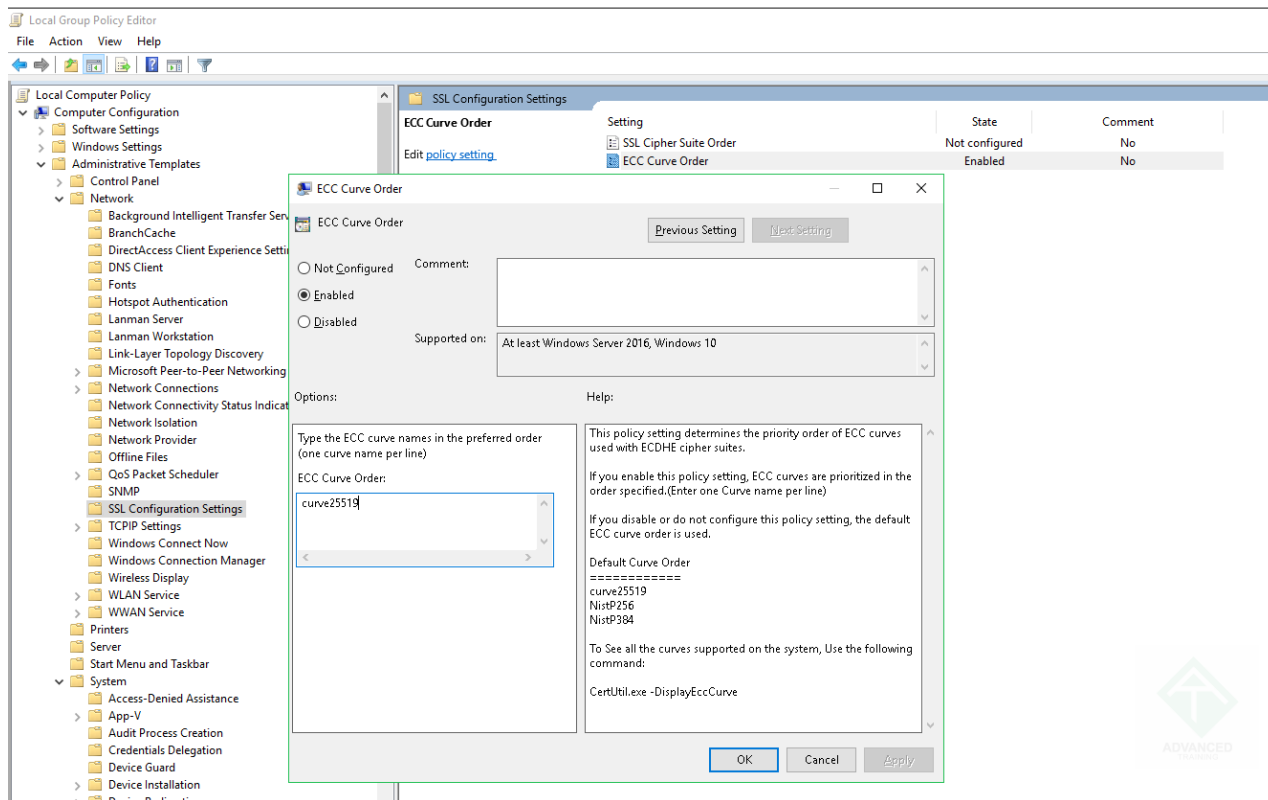
Protocols settings
  PCT 1.0 usage for outgoing sessions is On; accepting for incoming is On
  MPUH for outgoing sessions is On; accepting for incoming is On
  SSL 2.0 for outgoing sessions Off; accepting for incoming is On
  SSL 3.0 for outgoing sessions On; accepting for incoming is On
  TLS 1.0 for outgoing sessions On; accepting for incoming is On
  TLS 1.1 for outgoing sessions On; accepting for incoming is On
  TLS 1.2 for outgoing sessions On; accepting for incoming is On

ECC Curves: curve25519,NistP256,NistP384
ATRAINING-WIN10(config)#_
```

[Включенные по умолчанию в Windows Server эллиптические кривые](#)

[\(кликните для увеличения до 976 px на 722 px\)](#)

Этот вариант подойдёт в подавляющем большинстве случаев – но если хотите, можете, например, выкинуть “предположительно скомпрометированные NIST’овские кривые с малым числом бит” (см. [доклад Daniel J. Bernstein и Tanja Lange](#) , и оставить только **curve25519**. Это делается штатно, через group policy:



[Изменяем список используемых в Windows Server эллиптических кривых \(кликните для увеличения до 1330 px на 832 px\)](#)

Учтите только, что в WDK явно указано, что применяться на 100% эта настройка будет только после перезагрузки системы: *Starting in Windows 10, CNG no longer follows every update to the cryptography configuration. Certain changes, like adding a new default provider or changing the preference order of algorithm providers, may require a reboot.*

Настраиваем фрагментацию TLS

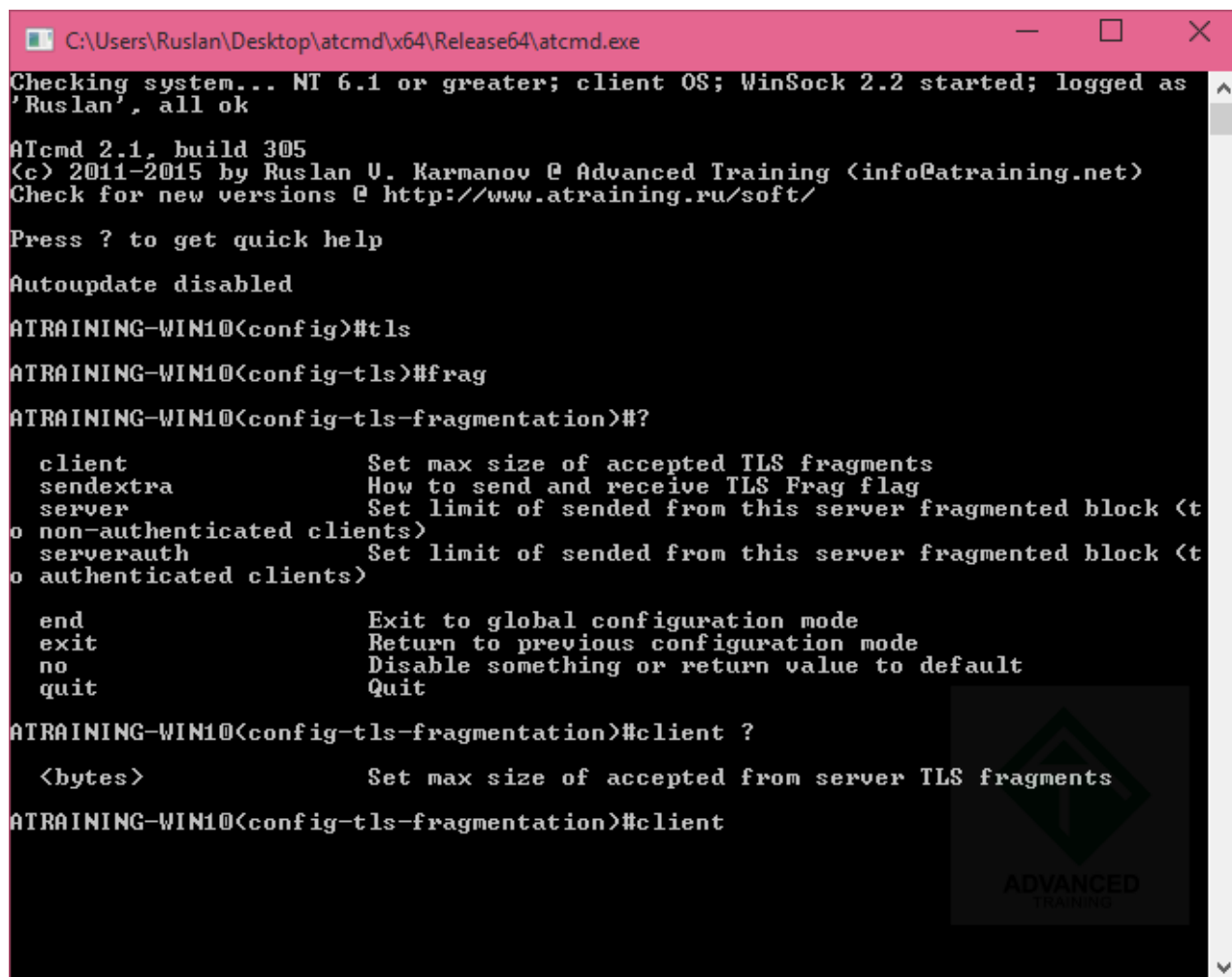
При обмене TLS передаёт данные сообщениями (message). Максимальный размер данных сообщений ограничен в TLS 1.0 – и равен 16.384 байт. Если какое-либо сообщение (не данные, а всё, включая заголовки) больше этого числа, то включается механизм фрагментации TLS (не путайте с IP-фрагментацией). Данный механизм изначально поддерживается в NT 6.1 (Windows 7 и Windows Server 2008 R2) – для остальных систем нужен патч [KB 2541763](#).

Проблема в том, что у разных серверов данное значение может быть разным – да и у клиентов (особенно мобильных) – тоже. На практике встречаются значения от 8192 байт (половинный стандарт, такое было на WinMobile) до 32768 байт (некоторые web-сервера). Поэтому нам нужно заранее подготовиться к этому, разрешив обработку фрагментов крупного размера. Это повлияет на многое – простейший пример; сервер считает, что максимальный размер фрагмента – 32K, и отдаёт в процессе TLS-handshake'a пачку сертификатов (он же имеет право ознакомить клиента с цепочкой оных, чтобы упростить ему проверку подлинности предъявляемого своего серверного сертификата), которая в сумме весит 20K. А клиент принимает только до 16K. Результат – отбой сессии на фазе установления.

Что мы можем настроить? Три значения – максимальный размер TLS-фрагмента, который мы отправляем, если мы:

- TLS-клиент (параметр **client**)
- TLS-сервер для клиента, который не аутентифицировался при помощи предъявления x.509 – сертификата (параметр **server**)
- TLS-сервер для клиента, который аутентифицировался при помощи предъявления x.509 – сертификата (параметр **serverauth**)

Делается это несложно:

A screenshot of a Windows command prompt window titled "C:\Users\Ruslan\Desktop\atcmd\x64\Release64\atcmd.exe". The window shows the ATcmd 2.1 interface. It starts with a system check, then displays version information and a link to the website. The user enters the command "tls" to enter the TLS configuration mode. Then, they enter "frag" to enter the fragmentation configuration mode. The prompt shows a list of options: "client" (Set max size of accepted TLS fragments), "sendextra" (How to send and receive TLS Frag flag), "server" (Set limit of sended from this server fragmented block <to non-authenticated clients>), and "serverauth" (Set limit of sended from this server fragmented block <to authenticated clients>). The user enters "client ?" and the prompt shows the option "<bytes>" (Set max size of accepted from server TLS fragments). The user then enters "client".

```
C:\Users\Ruslan\Desktop\atcmd\x64\Release64\atcmd.exe
Checking system... NT 6.1 or greater; client OS; WinSock 2.2 started; logged as 'Ruslan', all ok
ATcmd 2.1, build 305
(c) 2011-2015 by Ruslan U. Karmanov @ Advanced Training <info@atraining.net>
Check for new versions @ http://www.atraining.ru/soft/

Press ? to get quick help
Autoupdate disabled
ATRAINING-WIN10(config)#tls
ATRAINING-WIN10(config-tls)#frag
ATRAINING-WIN10(config-tls-fragmentation)#?

  client          Set max size of accepted TLS fragments
  sendextra       How to send and receive TLS Frag flag
  server          Set limit of sended from this server fragmented block <to
  non-authenticated clients>
  serverauth      Set limit of sended from this server fragmented block <to
  authenticated clients>

  end             Exit to global configuration mode
  exit            Return to previous configuration mode
  no              Disable something or return value to default
  quit            Quit

ATRAINING-WIN10(config-tls-fragmentation)#client ?

  <bytes>         Set max size of accepted from server TLS fragments

ATRAINING-WIN10(config-tls-fragmentation)#client
```

[Настройка TLS-фрагментации в Windows Server](#) ([кликните для увеличения до 661 px на 522 px](#))

Замечу также, что данные значения имеют максимумы (соответственно, в порядке указания, 32K, 16K, 32K – т.е. фрагменты максимально могут быть до 32K, за исключением случая подключения неаутентифицирующегося клиента) и связаны следующей логикой – для определения максимального размера фрагмента в последнем из сценариев (подключение клиента, подтверждающего свою подлинность сертификатом) берётся максимальное значение из пары **server** / **serverauth**.

Установим всё в максимальные значения, чтобы принимать все возможные фрагменты, и продолжим.

Анти-BEAST или принудительная TLS-фрагментация

В 2012 году для обработки специфичной атаки (тот самый BEAST) фирма Microsoft выпустила специальный патч, который добавлял дополнительное управление фрагментацией (уже не размером фрагмента, а самой логикой работы оной) в SCHANNEL. В принципе, атака BEAST отражается просто – надо не использовать TLS 1.0 и SSL 3.0, а использовать только TLS 1.1 и выше, но это легко сказать, но трудно реализовать в современном интернете, где куча опенсорсных подделок годами работают с устаревшими и уязвимыми реализациями протоколов и стандартов, исходя из отговорок вида *“раз вообще хоть кое-как работает, то работает идеально, это же СПО”*. Поэтому да, если бы все системы, подключающиеся по TLS, были бы хотя бы Windows 7 или Windows Server 2008 R2, можно было бы просто включить только TLS 1.2 и автоматически решить огромное число проблем – но увы, приходится делать это иначе.

После установки данного патча (это [KB 2638806](#) или, если сразу для всех платформ и подробнее – [MS12-006 / CVE-2011-3389](#)) у нас появляется возможность выбрать из нескольких вариантов. Первый и используемый по-умолчанию вариант – это “В случае, если другая сторона подтверждает отправкой флага при установке SSL/TLS-сессии, что тоже установила этот патч и настроена его использовать, фрагментировать handshake так, чтобы избежать использования атаки BEAST на блочные шифры”. Т.е. говоря проще, если данный патч везде установлен, всё происходит само. Два других варианта – это принудительно фрагментировать handshake всегда или никогда.

Мы бы, конечно, выставили “всегда”, но всё же это повлечёт много проблем с совместимостью, да и не забываем, что все эти пляски интересны только если у нас исключительно TLS 1.0, на старших версиях этой проблемы уже нет. Поэтому выставим в явном виде дефолтный режим:

```
C:\Users\Ruslan\Desktop\atcmd\x64\Release64\atcmd.exe

ATRAINING-WIN10<config>#tls frag
ATRAINING-WIN10<config-tls-fragmentation>#?

client          Set max size of accepted TLS fragments
sendextra       How to send and receive TLS Frag flag
server          Set limit of send from this server fragmen
ted block <to non-authenticated clients>
serverauth      Set limit of send from this server fragmen
ted block <to authenticated clients>

end             Exit to global configuration mode
exit            Return to previous configuration mode
no             Disable something or return value to default

quit           Quit

ATRAINING-WIN10<config-tls-fragmentation>#sendextra ?

optin           Fragmented TLS handshake only if requested
always         Always use anti-BEAST fragmentation logic
none           Disable anti-BEAST fragmentation logic

ATRAINING-WIN10<config-tls-fragmentation>#sendextra optin

Now SSL/TLS handshake fragmentation optional

ATRAINING-WIN10<config-tls-fragmentation>#
```

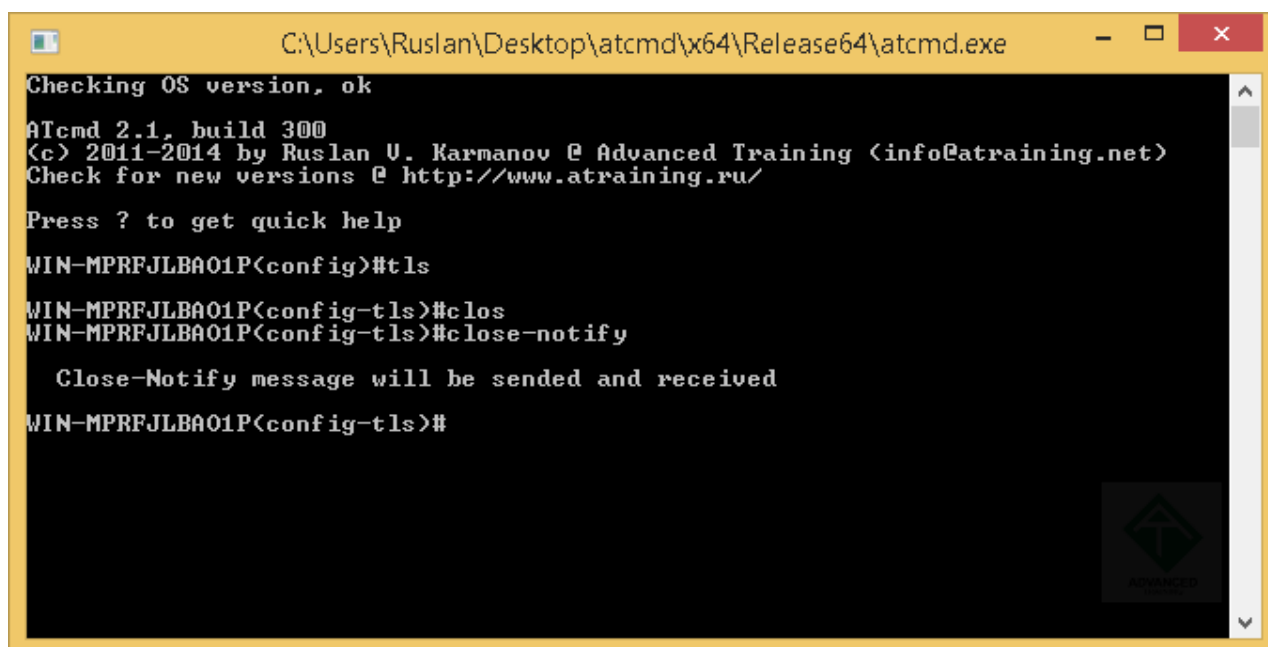
[Выставляем TLS SendExtraRecord в opt in](#)
([кликните для увеличения до 564 px на 394 px](#))

Теперь далее.

SSL Close-Notify

Когда клиент закрывает TLS-сессию, есть два варианта поведения со стороны сервера – полноценно закрыть её и сэкономить силы, “прикрыв на время”, чтобы в случае переподключения того же клиента продолжить работу с ним, не выполняя “с нуля” всю криптографическую и математически интенсивную работу – генерацию ключевого материала и подобное. Это поведение можно настроить, влияя на параметр SSL Close-Notify.

По умолчанию включён “облегчённый” режим – мы включим полновесный, так как это закрывает целый класс атак на truncation – игры с манипуляцией длинами ответов и запросов.



```
C:\Users\Ruslan\Desktop\atcmd\x64\Release64\atcmd.exe
Checking OS version, ok
ATcmd 2.1, build 300
(c) 2011-2014 by Ruslan U. Karmanov @ Advanced Training (info@atraining.net)
Check for new versions @ http://www.atraining.ru/
Press ? to get quick help
WIN-MPRFJLBA01P(config)#tls
WIN-MPRFJLBA01P(config-tls)#clos
WIN-MPRFJLBA01P(config-tls)#close-notify
    Close-Notify message will be sended and received
WIN-MPRFJLBA01P(config-tls)#
```

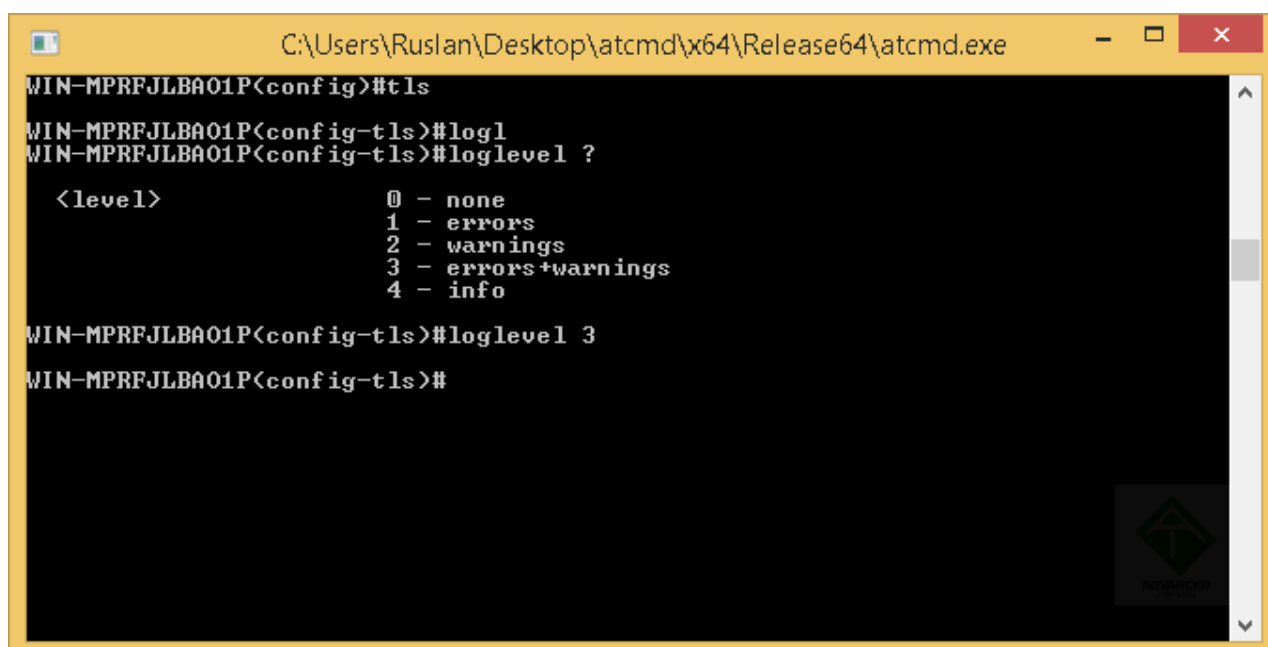
[Обработка Close-Notify в TLS](#)

[\(кликните для увеличения до 677 px на 343 px\)](#)

Обращу внимание, что включение этого режима, по сути, обозначает так же и то, что мы, как сервер, будем не только обрабатывать клиентский Close-Notify, но и будем отправлять клиенту явное уведомление о том, что закрываем сессию и новая будет создаваться “с нуля”.

Журналирование TLS

Подсистема SCHANNEL, которая отвечает в ОС на базе Windows за встроенную реализацию TLS, может сообщать о своих ошибках и проблемах в отдельный журнал. Для отслеживания результатов наших настроек, да и вообще, чтобы не пропустить критичные ошибки, настроим это журналирование:



```
C:\Users\Ruslan\Desktop\atcmd\x64\Release64\atcmd.exe
WIN-MPRFJLBA01P(config)#tls
WIN-MPRFJLBA01P(config-tls)#logl
WIN-MPRFJLBA01P(config-tls)#loglevel ?
    <level>
        0 - none
        1 - errors
        2 - warnings
        3 - errors+warnings
        4 - info
WIN-MPRFJLBA01P(config-tls)#loglevel 3
WIN-MPRFJLBA01P(config-tls)#
```

[Настройка журналирования TLS](#)

[\(кликните для увеличения до 677 px на 343 px\)](#)

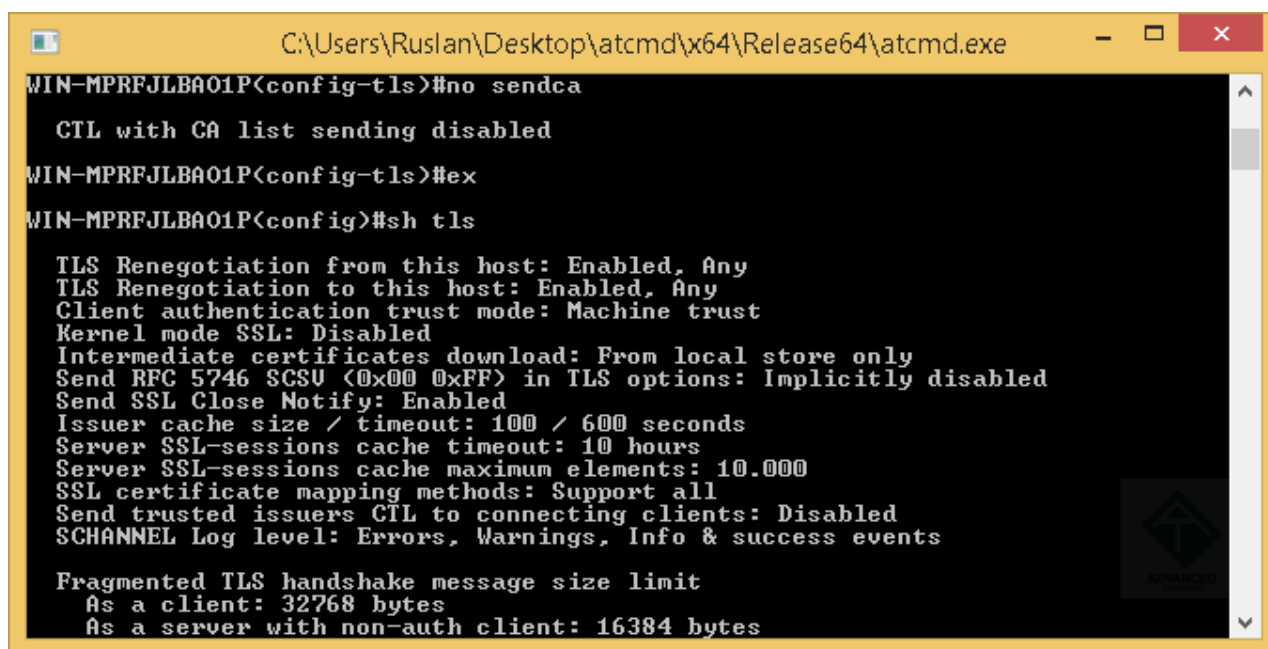
Обратите внимание – анализ ошибок SCHANNEL – очень хороший способ “вычищения” потенциальных неприятностей до того, как они станут реально мешать работе. Во многих случаях SCHANNEL не может согласовать что-то, молча переходит на упрощённый/ослабленный режим и работает дальше, а администратор про это просто не знает, предполагая, что раз ОС новая и все патчи установлены, то защищённость высокая. Журналирование SCHANNEL поможет наглядно увидеть, всё ли так хорошо, как кажется. :)

Отправка клиенту списка доверенных СА в CTL-формате

Когда клиент подключается к серверу, тот может предоставить ему список тех issuers, кому сервер доверяет. Это и логично – в случае, если клиент имеет широкие возможности по аутентификации при помощи x.509-сертификатов, данный список поможет автоматически уменьшить список потенциально пригодных для подтверждения своей подлинности сертификатов.

Однако, начиная с NT 6.2, данная практика считается небезопасной и отправка этого списка отключена по-умолчанию. Это, кстати, ещё и ускоряет установление соединения.

Мы, чтобы не экспериментировать с умолчаниями, в явном виде выключим отставку этого списка, предполагая, что подключающиеся к нам клиенты уже заранее знают, какой сертификат они нам хотят предъявить, и дополнительная помощь в таком виде им просто не требуется.



```
C:\Users\Ruslan\Desktop\atcmd\x64\Release64\atcmd.exe
WIN-MPRFJLBA01P(config-tls)#no sendca
CTL with CA list sending disabled
WIN-MPRFJLBA01P(config-tls)#ex
WIN-MPRFJLBA01P(config)#sh tls

TLS Renegotiation from this host: Enabled, Any
TLS Renegotiation to this host: Enabled, Any
Client authentication trust mode: Machine trust
Kernel mode SSL: Disabled
Intermediate certificates download: From local store only
Send RFC 5746 SCSV (0x00 0xFF) in TLS options: Implicitly disabled
Send SSL Close Notify: Enabled
Issuer cache size / timeout: 100 / 600 seconds
Server SSL-sessions cache timeout: 10 hours
Server SSL-sessions cache maximum elements: 10.000
SSL certificate mapping methods: Support all
Send trusted issuers CTL to connecting clients: Disabled
SCHANNEL Log level: Errors, Warnings, Info & success events

Fragmented TLS handshake message size limit
As a client: 32768 bytes
As a server with non-auth client: 16384 bytes
```

[Отключение отправки списка issuer CA в виде CTL клиенту](#)

[\(кликните для увеличения до 677 px на 343 px\)](#)

Логика проверки x.509-сертификата клиента

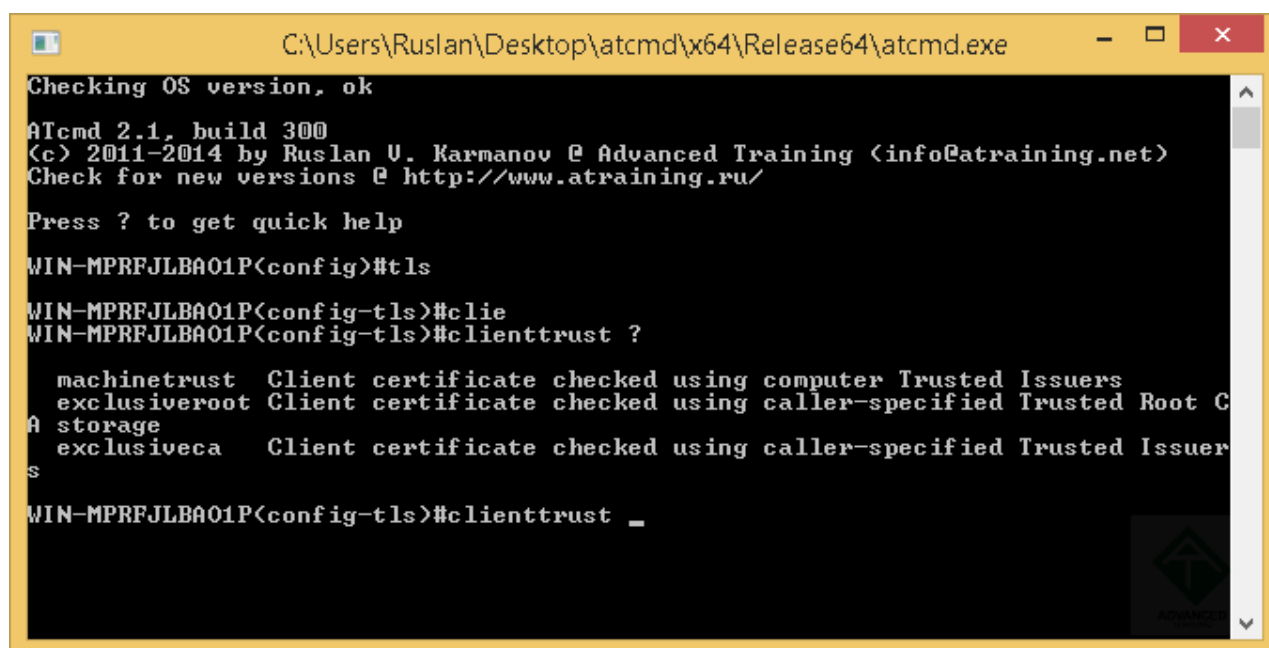
Обычно этот момент не очень акцентируется в логике установки SSL/TLS-соединения – подразумевается что-то типа “ну, клиент должен предоставить нормальный такой сертификат”, не уточняя, что точно имеется в виду под этим. Давайте разберёмся.

Понятно, что сертификат клиента должен быть валидным на базовом уровне – т.е. правильного формата, не просроченный, не потерявший целостность, обладающий всеми обязательными полями. Но в остальном критерии “подходящий” могут быть разными. Мы можем влиять на логику проверки, выбирая один из трёх вариантов:

- Проверка цепочки доверия сертификата клиента считается удачной, если цепочка закончилась на сертификате, входящем в Trusted Issuers (например, IIS-серверу предоставили CTL-файл с ними в явном виде) (это параметр `machinetrust`)
- Проверка цепочки доверия сертификата клиента считается удачной, если цепочка закончилась на сертификате, входящем в Trusted Issuers и являющимся корневым (т.е. самоподписанным) (это параметр `exclusiveroot`)
- Проверка цепочки доверия сертификата клиента считается удачной, если цепочка закончилась на сертификате, входящем или в Trusted Issuers, или в локальное хранилище сертификатов (это параметр `exclusiveca`)

Что выбрать в конкретном варианте – нужно смотреть для каждой ситуации отдельно, но например вариант `exclusiveroot` будет увеличивать время процессинга, игнорируя то, что сертификат клиента уже признан валидным, т.к. подписан валидным intermediate CA, но будет добавлять безопасности, т.к. валидация intermediate CA, по сути, будет постоянной и принудительной.

Вы сможете повлиять на это командой **clienttrust** в контексте **tls**:



```
C:\Users\Ruslan\Desktop\atcmd\x64\Release64\atcmd.exe
Checking OS version, ok
ATcmd 2.1, build 300
(c) 2011-2014 by Ruslan U. Karmanov @ Advanced Training <info@atraining.net>
Check for new versions @ http://www.atraining.ru/
Press ? to get quick help
WIN-MPRFJLBA01P<config>#tls
WIN-MPRFJLBA01P<config-tls>#clie
WIN-MPRFJLBA01P<config-tls>#clienttrust ?
    machinetrust Client certificate checked using computer Trusted Issuers
    exclusiveroot Client certificate checked using caller-specified Trusted Root C
    A storage
    exclusiveca Client certificate checked using caller-specified Trusted Issuer
    s
WIN-MPRFJLBA01P<config-tls>#clienttrust _
```

[Логика проверки сертификата клиента](#)
([кликните для увеличения до 677 px на 343 px](#))

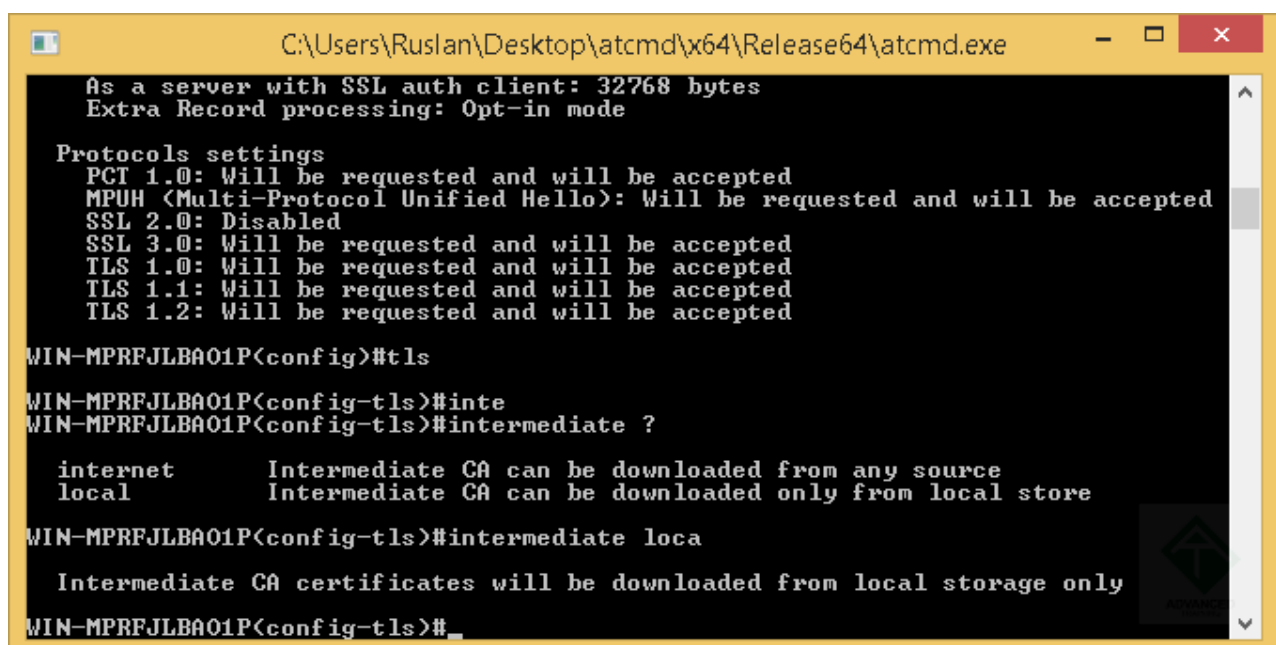
Не забудьте, что этот параметр влияет только на ситуации, когда клиент подтверждает свою подлинность сертификатом. И это необязательно лобовой сценарий “Кто-то с токеном удалённо подключается к корпоративному portalу” – это, например, 802.1x в случае, когда машинам раздали сертификаты. Это wifi, когда клиенты с сертификатами и EAP-TLS / PEAP. Будьте осторожны с тюнингом таких параметров, и первоначально точно выясните, на что это повлияет на данном конкретном сервере.

Проверка промежуточных сертификатов через Интернет

Когда клиент предъявляет свой сертификат, мы проверяем всю цепочку доверия, пока не натолкнёмся или на ошибку, или на явно доверенный сертификат. Однако, не все родительские сертификаты могут быть у нас (сервера) локально – возможно, что проверка цепочки клиента повлечёт за собой необходимость загрузить сертификат (или несколько) по указанному URL'у.

Это действие не всегда нужно или полезно – например, публичному серверу при такой логике работы можно сделать интересную проблему, заставляя его пачками пытаться загружать сертификаты с удалённого и медленного сервера, создавая множественные “потихоньку открывающиеся” SSL/TLS-соединения. В ряде же ситуаций – допустим, когда к серверу должны подключаться только свои, корпоративные клиенты, предъявляя свои, местные сертификаты, вопрос о том, надо ли серверу обращаться в Интернет, чтобы узнать про их подлинность, вообще исключается.

Мы сделаем настройку как раз под такой сценарий – выключим возможность подгрузки intermediate CA certificates из Интернета.



```
C:\Users\Ruslan\Desktop\atcmd\x64\Release64\atcmd.exe
As a server with SSL auth client: 32768 bytes
Extra Record processing: Opt-in mode

Protocols settings
PCT 1.0: Will be requested and will be accepted
MPUH <Multi-Protocol Unified Hello>: Will be requested and will be accepted
SSL 2.0: Disabled
SSL 3.0: Will be requested and will be accepted
TLS 1.0: Will be requested and will be accepted
TLS 1.1: Will be requested and will be accepted
TLS 1.2: Will be requested and will be accepted

WIN-MPRFJLBA01P<config>#tls
WIN-MPRFJLBA01P<config-tls>#inte
WIN-MPRFJLBA01P<config-tls>#intermediate ?

internet      Intermediate CA can be downloaded from any source
local         Intermediate CA can be downloaded only from local store

WIN-MPRFJLBA01P<config-tls>#intermediate loca
Intermediate CA certificates will be downloaded from local storage only
WIN-MPRFJLBA01P<config-tls>#_
```


[Загружаем сертификаты промежуточных СА только из локального хранилища \(кликните для увеличения до 677 px на 343 px\)](#)

Теперь про дополнительный механизм безопасности HTTPS, включаемый на уровне веб-сервера.

Механизм HSTS – HTTP Strict Transport Security

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”)

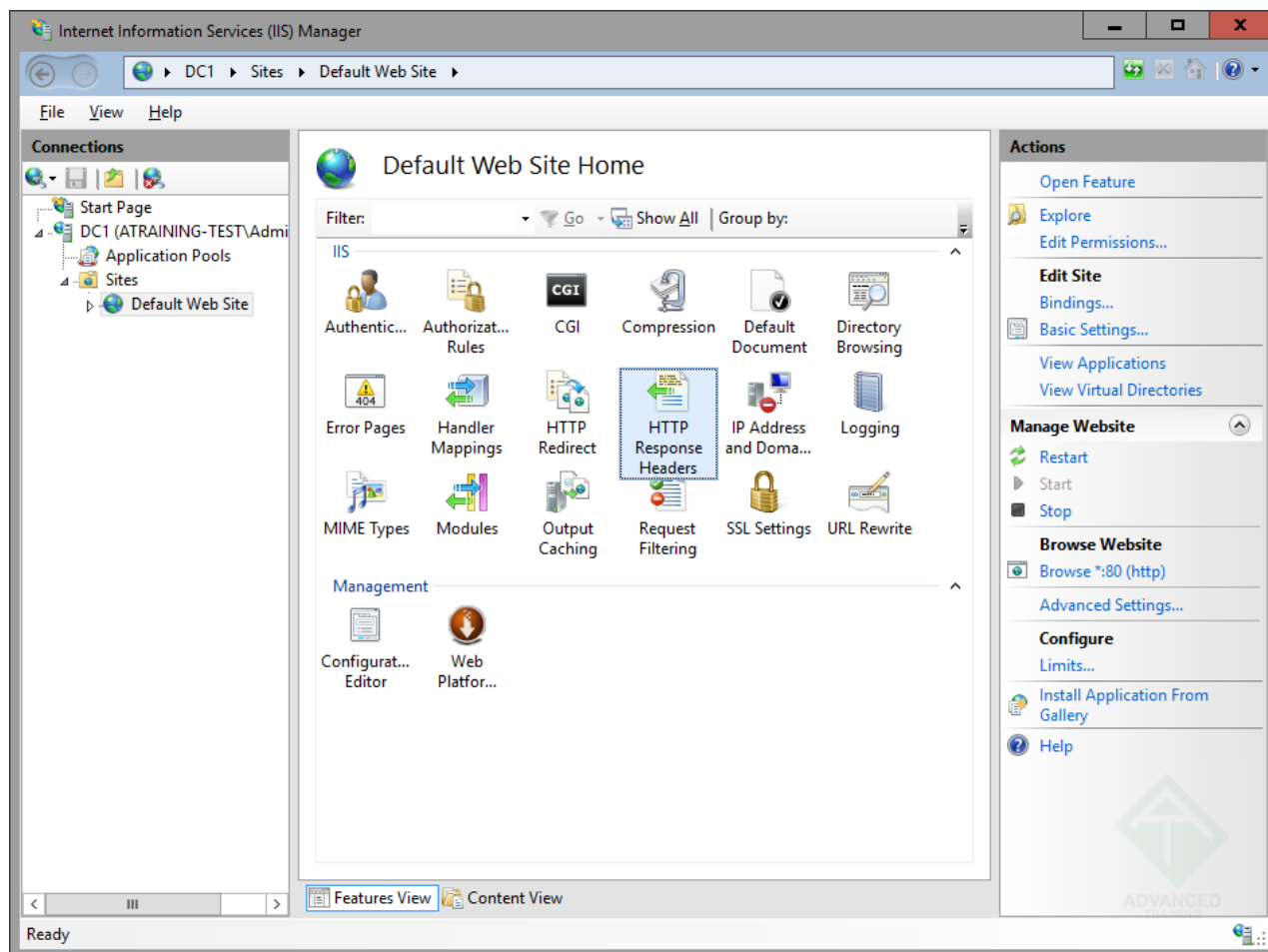
Клиент:

- 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.

Проблемой HSTS будет то, что самое первое обращение к серверу будет не знать про этот механизм, ну и то, что злонамеренные товарищи могут “срезать” этот доп.заголовок в ответе. Но – никто и не говорит, что это 100%е решение, таких решений вообще в природе нет. Частично вопрос проблемы первичного обращения снимается заданием pre-loaded списков сайтов в некоторых новых браузерах, частично – фактическим отключением HTTP для защищаемых ресурсов (весьма хороший метод). Главное, что любая атака, подразумевающая “downgrade” уже установленной HTTPS-сессии натолкнётся на то, что при включённом и согласованном HSTS браузер сам “на лету” исправит HTTP-ссылки на HTTPS и продолжит работу.

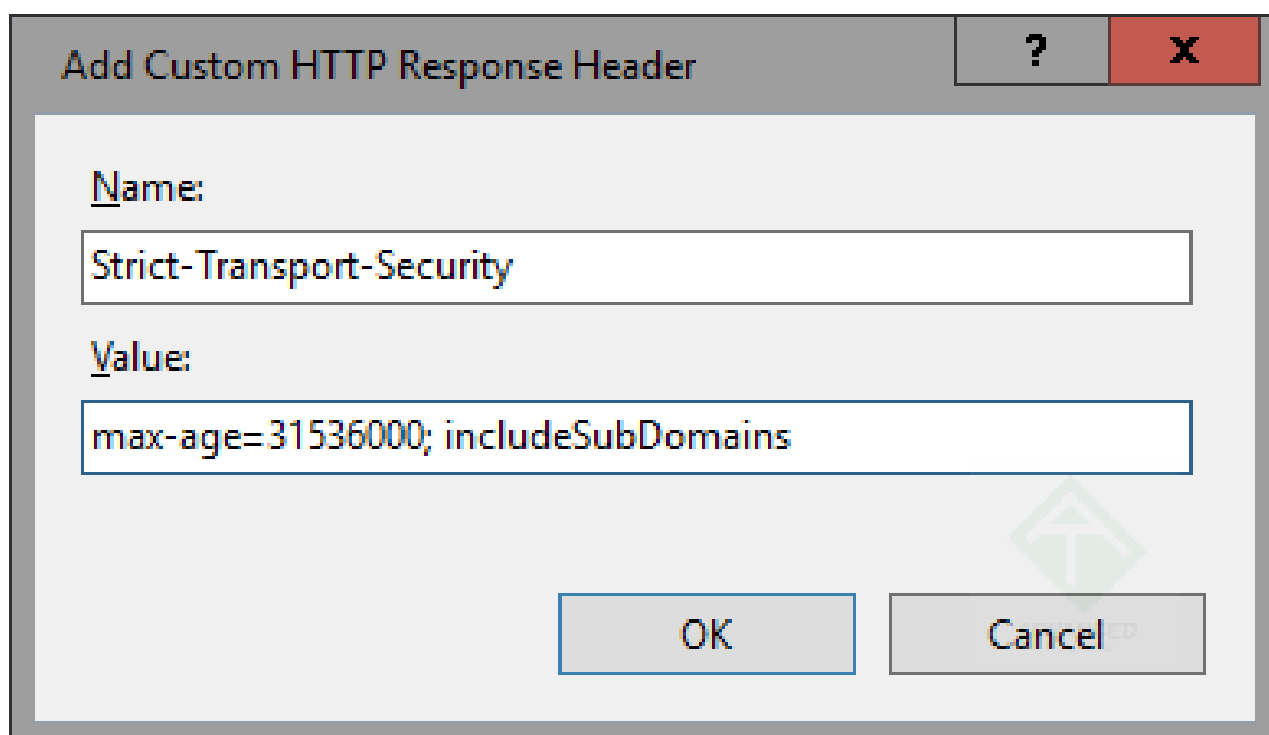
Как включать? Зайдём в консоль управления IIS и выберем там HTTP Response Headers для нужного нам сайта:



[Настраиваем HTTP Strict Transport Security для IIS](#)

([кликните для увеличения до 946 px на 707 px](#))

Задаём заголовок HSTS и указываем ему два параметра:

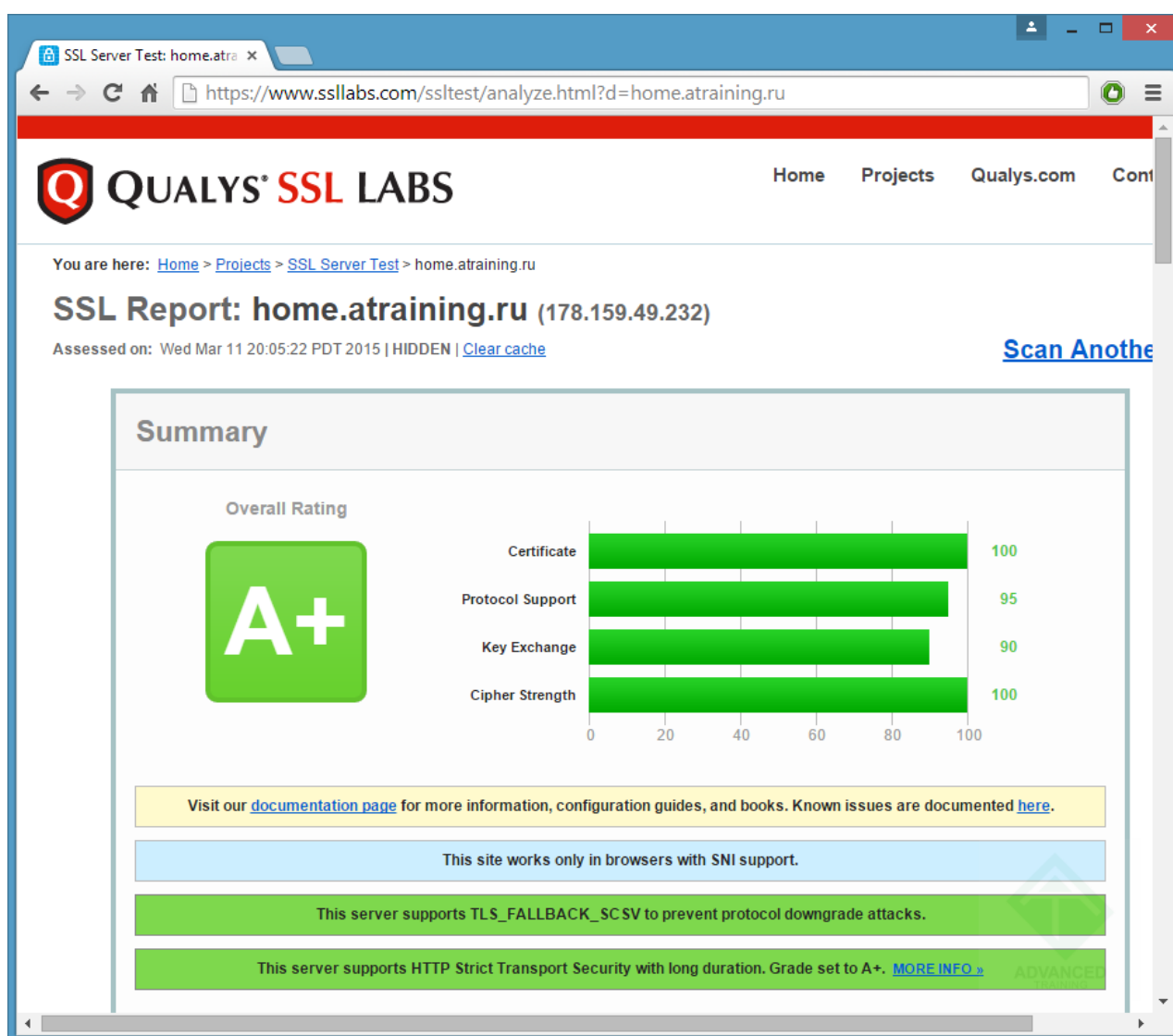


[Включаем HTTP Strict Transport Security для IIS](#)

[\(кликните для увеличения до 389 px на 224 px\)](#)

Первый – **max-age** – указывает, сколько времени в секундах, после получения распознанного ответа от сервера с данным заголовком, получателю (user agent'у, в нашем случае – браузеру) надо относить сервер к категории Known HSTS hosts и работать с ним в соответствии с этим. Второй – **includeSubDomains** – будет указывать, что в случае, если внутри HTTPS-сессии будет получена ссылка на ресурс, находящийся на субдомене относительно текущего (т.е. идёт работа с example.com, и в ответе приходит ссылка на CSS-файл с ресурса cdn.example.com), необходимо действовать в той же логике, т.е. предотвращать уход на HTTP.

Результат глазами увидеть будет трудно, но различные средства анализа защищённости увидят, что сервер поддерживает HSTS:



[Результат - работающая на IIS технология HSTS](#)

[\(кликните для увеличения до 933 px на 817 px\)](#)

Данная технология поддерживается всеми современными браузерами – за исключением, увы, IE – только его 11я версия, притом работающая на Windows 10, читает данный заголовок и следует вышеуказанной логике. Ну, будем надеяться, что

это скоро поправят – добавление этого функционала не кажется каким-то сверхсложным.

Если что – данный механизм детально описан в [RFC 6797](#), но, в общем-то, про весь его функционал мы уже рассказали.

Улучшение работы HSTS – механизм preloading

Для того, чтобы ещё эффективнее и безопаснее работать с сайтами, которые точно доступны только по HTTPS, существует механизм preloading. Для его активации нужны следующие условия:

1. Добавить в HTTP-заголовок Strict-Transport-Security слово preload;
2. Указать max-age не менее 18 недель (max-age должен быть 10886400 секунд и более);
3. Отдавать данный заголовок при обращении к домену 2го уровня, а не к субдомену (т.е. для **www.atraining.ru** этот механизм включить не получится – только для **atraining.ru**);

В результате заголовок будет выглядеть как-то так:

max-age=63072000; includeSubDomains; preload

После выполнения этих условий надо добавить свой сайт в форму на сайте <httpspreload.org>:

Enter a domain for the HSTS preload list:

atrainiing.ru

Check status and eligibility


Status: atraining.ru is not preloaded.
Eligibility: atraining.ru is eligible for the HSTS preload list.

Submit

☒ I am the site owner of atraining.ru or have their permission to preload HSTS.
(If this is not the case, atraining.ru may be sending the HSTS preload directive by accident. Please [contact hstspreload@chromium.org](#) to let us know.)

☒ I understand that preloading atraining.ru through this form will prevent **all subdomains and nested subdomains** from being accessed without a valid HTTPS certificate:
*.atrainiing.ru
..atrainiing.ru
...

Submit atraining.ru to the HSTS preload list



Настраиваем HTTP Strict Transport Security preloading

([кликните для увеличения до 956 px на 757 px](#))

и тогда у поддерживающих этот список браузеров даже первое обращение к вашему сайту будет уже защищённым и выполняться исключительно по HTTPS.

Хотелось бы обратить внимание на важную деталь – чтобы пройти данный тест надо, чтобы обращения по HTTP на корневой домен сразу же с кодом 301 редиректились на HTTPS-вариант. То есть HTTP-вариант должен по первому запросу отдавать заголовок Strict-Transport-Security и код 301 – не на какой-то другой сайт, а на идентичный URL, только с HTTPS. Если например редирект идёт с **http://atrainiing.ru** сразу на **https://www.atrainiing.ru**, то прелоадинг не сработает – он реализуется только для доменов второго уровня.

Механизм НРКР – HTTP Public Key Pinning

Задача этого механизма, описанного в [RFC 7469](#) – “подстраховать” того, кто инициирует TLS-сессию с сервером, дав возможность дополнительной проверки сертификата. НРКР позволяет добавить в ответ веб-сервера доп.заголовков, в котором указывается, какими хэшами может обладать предоставленный сертификат.

Важно то, что это не является гарантией безопасности – в случае, если клиент первый раз обращается к ресурсу X, и существует промежуточный участник, который может подменить сертификат, этот же участник может подменить и хэш в заголовке Public-Key-Pins. Поэтому нужно чётко представлять себе задачи данного механизма – он нужен, чтобы разово обратившись к HTTPS-ресурсу, браузер закешировал бы у себя “список легальных хэшей”, а потом сверял бы при последующих обращениях хэши открытых ключей, которые ему присылает сервер, с этим списком.

Заголовок будет достаточно прост и похож на HSTS. В нём могут быть следующие параметры:

- **max-age** – указывает на время, на которое информация из этого заголовка кэшируется браузером клиента;
- **includeSubDomains** – так же как и в случае с HSTS говорит о том, учитывать ли информацию из заголовка при обращении к субдоменам;
- **report-uri** – задаёт URI, по которому браузер может (если умеет и настроен соответствующим образом) [сигнализировать о сбое проверки](#);
- **pin-sha256** – задаёт SHA-2/256 хэш открытого ключа;

Из этих параметров обязательными являются **max-age** и **pin-sha256**. Теперь чуть подробнее про каждый.

HTTP Public Key Pinning – параметр max-age

Это – время кэширования информации о “возможных хэшах открытых ключей предоставленных сервером сертификатов”. Измеряется в секундах, рекомендованное значение – неделя и больше. Если значение будет меньше, ряд тестов будут говорить о “слишком слабом значении max-age у НРКР” – это не будет ошибкой, но возможно будет влиять на “общий балл”. Несмотря на то, что это кэширование, зависимости “чем больше время, тем быстрее доступ” нет, так как сервер отдаёт данный заголовок клиенту каждый раз.

HTTP Public Key Pinning – параметр pin-sha256

Данный параметр, по сути, основной – в нём указываются SHA-2/256-хэши “легальных” открытых ключей (пока только SHA-2/256, но в дальнейшем, возможно, будут поддерживаться и другие алгоритмы). Этих параметров должно быть как минимум два, и вот по какой причине – при обращении к серверу клиент получает от него цепочку сертификатов, после чего хэширует открытые ключи и ищет их в НРКР-списке. Если хотя бы один из предоставленной цепочки нашёлся – как корневой, так и промежуточный или удостоверяющий конкретный сервер – то всё ОК; но механизм ещё и подстраховывается от ситуации “что-то случилось и надо резко сменить сертификат” и для полноценной работы требует, чтобы в заголовке НРКР был хотя бы 1 хэш, не соответствующий ни одному из предъявляемых сертификатов.

Зачем это нужно?

Представьте ситуацию, что вы сформировали заголовок HPKP, вписав туда пару хэш-значений (например, корневого сертификата и сертификата сервера) и выставив max-age на неделю. Сервер отдаёт этот заголовок клиенту. Клиент добросовестно проверяет соответствие TLS-сертификатов и хэшей, фиксирует, что всё ОК, и кэширует у себя в браузере данный заголовок. Время пошло.

На следующий день сертификат сервера компрометируется и его отзывают. Например выясняется, что кто-то скопировал его с сервера вместе с закрытым ключом. Или компрометируется промежуточный CA, выдавший этот сертификат. Или, что ещё похуже – корневой CA. Ну или сертификат просто закончил срок действия (по хэшу ключа-то это не предугадать никак). В любом случае сертификат надо отзывать и заменять. Это делается, но у клиентов-то закэшировано, что “предоставляемый данным сайтом X комплект сертификатов обязательно должен содержать сертификаты с хэшами N1 и N2”. И новый – хороший и не-отозванный сертификат – клиенты забраковывают с криками “нас пытаются обмануть, мы точно знаем, какой должен быть хэш хотя бы у одного сертификата из цепочки!”.

Ситуация откровенно дурацкая – поэтому, чтобы подстраховаться, вам изначально предлагается добавить в заголовок хотя бы 1 (можно и больше) хэш сертификата, который сейчас не в цепочке, предъявляемой сервером. Например, другого корневого CA. Или выдающего. Такой параметр pin-sha256 будет называться backup key и для корректной работы HPKP такой нужен хотя бы один.

Поэтому минимальный правильный комплект pin-sha256 – это два значения хэша; одно соответствующее одному из хэшей открытых ключей какого-либо из сертификатов, предъявляемых сервером, и одно не-соответствующее.

Посчитать pin-sha256 для вашего сайта несложно по [ВОТ ЭТОЙ ВОТ ССЫЛКЕ](#) – достаточно ввести полный URL и система покажет вам полный список хэшей для всех полученных от HTTPS-сервера сертификатов:

Home > Tools > HPKP Hash Generator

Create your HPKP hash

Hash

Here is your PKP hash for *.advancedtraining.ru: pin-sha256="yGZ5I90Sz/A0BXIAPdjjJaWbyRqw+B2eN1f5QFk5vq9w="

Here is your PKP hash for StartCom Class 2 IV Server CA: pin-sha256="fVZ1UIYWgPZvUe9THLBBeg1pNN7LF4HI373EsQhUAdE="

Here is your PKP hash for StartCom Certification Authority: pin-sha256="5C8kvU039KouVrI52D0eZ5Gf4Onjo4Khs8tmyTIV3nU="



[Вычисление pin-sha256 для HTTPS-сайта](#)
(кликните для увеличения до 1161 px на 406 px)

HTTP Public Key Pinning – параметр includeSubDomains

Здесь всё, как в HSTS – “распространять ли действие информации из заголовка на субдомены”. Если вы используете wildcard-сертификат, то этот вариант обычно удобен – можно на корневом домене объявить заголовок, и всякие субдомены уже обойдутся без него – когда с ними будет устанавливаться HTTPS-сессия, браузер будет учитывать информацию “основного” домена и не будет тратиться лишний трафик на то, чтобы каждый субдомен явно сообщал “да, у меня такой же сертификат с такими же хэшами”.

HTTP Public Key Pinning – параметр report-URI

Данный параметр нужен для указания браузеру, куда стучать в случае обнаружения нарушений. Это полезный пункт – в сценарии вида *“Наш сотрудник работал-работал с нашим корпоративным сайтом из дома, а потом пришёл в Плохую Фирму и попытался открыть наш сайт из их корпоративной сети – а тут-то ему местный Плохой Шлюз ловко подставил фальшивый сертификат. Но у нашего сотрудника был заэкширован заголовок HPKP с правильным хэшем и браузер сотрудника разоблачил подделку и быстро настучал Куда Надо”* всё будет автоматизировано. Данные отправляются в виде POST-запроса в [JSON](#), подробнее про это есть в статье [про настройку отправки отчётности механизма HTTP Public Key Pinning и других](#).

Пример заголовка, используемого у нас на сайте www.atraining.ru:

```
max-age=31536000; pin-sha256=""; pin-sha256=""; pin-sha256=""; report-URI="https://www.atraining.ru/report-uri/hpkp"; includeSubDomains
```

(хэши я не указываю, т.к. они могут меняться – вы можете посмотреть фактический ответ сервера в любое время).

Также заметим, что вместо заголовка Public-Key-Pins может в начале внедрения технологии использоваться Public-Key-Pins-Report-Only – он действует идентично, разве что при нарушении только ругается, а не запрещает доступ к сайту. Удобно для тестирования.

Механизм DNS CAA – DNS Certification Authority Authorization

Этот механизм также нужен для доп.проверки “а хороший ли нам сертификат предъявляет сервер?” – только в отличие от HPKP проверка делается не через тот же веб-сервер, а через DNS. Это увеличивает уровень надёжности – злоумышленнику придётся получить контроль над DNS-зоной, чтобы подделать CAA-запись – ну или подменить запись “на лету”, что в случае с DNSSEC станет практически невозможным.

Данный механизм появился в 2013м году в [RFC 6844](#), который до сих пор Proposed Standard [уже с сентября 2017 года является обязательным для проверки](#) – но в общем-то ничего не мешает использовать данную технологию, так как есть сервисы, умеющие проверять эту запись и делать на основании этого выводы.

Замечу, что для использования DNS CAA вам надо будет развернуть сторонний DNS-сервер (например, ISC BIND), потому что DNS, встроенный в Windows Server, не умеет обрабатывать CAA-записи. Даже если вы внесёте их в файл DNS-зоны вручную.

Формат CAA-записи достаточно прост – в нашем домене **atraining.ru** CAA-запись выглядит так:

```
atraining.ru CAA 0 issue "letsencrypt.org" atraining.ru CAA 0 iodef  
"https://www.atraining.ru/report-uri/caa" atraining.ru CAA 0 iodef  
"mailto:info@atraining.ru"
```

Что же обозначают эти параметры?

DNS CAA – параметр issue

Параметр issue и следующее за ним значение говорят следующее – *“мы подтверждаем, что для подтверждения подлинности указанной сущности (в нашем варианте это домен **atraining.ru**) может быть предъявлен сертификат от издателя **letsencrypt.org**, либо кого-то, наделённым правом действовать от лица **letsencrypt.org**”*.

То есть если кто-то попытается “на лету” подделать наш сертификат, даже подставив выданный доверенным центром сертификации, проверка через DNS CAA покажет, что сертификат может и валидный по всем критериям, да вот выдан не тем, кем надо. Это достаточно эффективно уберёт атаки вида *“к нашему ресурсу подключаются из внутрикорпоративной сети, где развёрнут какой-то проксирующий сервер, типа старого TMG 2010, который “на лету” генерит доверенные сертификаты для всех запросов”* – ведь при этом сценарии сертификат, с точки зрения внутридоменной машины, валидный.

Плюс СА, которому придёт запрос на выдачу сертификата для домена, проверит CAA-записи и скажет “увы, я не имею права выдавать этому домену”, что дополнительно отсекает возможности для подделки (правда, если СА умеет и хочет это делать, что спорно).

Существует также вариант параметра **issuewild**, который записывается идентично, но расширяет сферу применения до *“мы подтверждаем, что для указанной сущности признаётся подходящим сертификат, необязательно явно указывающий на наш FQDN – подойдёт и вариант, например, ***.atraining.ru**”*.

DNS CAA – параметр iodef

Параметр iodef – это стандартный, описанный в [RFC 5070](#) “способ автоматической связи” – например, URL-адрес страницы или e-mail. Нужен для того, чтобы проверяющая со стороны клиента подсистема могла (если умеет) как-то отсигнализировать “проверка не удалась”. Про вариант с report-uri есть [отдельная статья про настройку отчётности механизма DNS CAA и других](#).

Нуль после САА

Это – флаговое битовое поле, длиной с байт, обычно оно заполнено нулями, но может быть задано значение 128 (т.е. включен старший бит) – в этом случае значение будет помечено как критическое и должно обрабатываться с той же логикой, что и critical extensions в x.509v3-сертификате. Начиная с сентября 2017 вы можете выставить 128 у всех issue, чтобы абсолютно точно зафиксировать ситуацию “СА, никогда не обрабатывайте запросы на выдачу нашему домену сертификатов, что бы там не предъявлял запрашивающий, если не находите себя в САА-ответе”.

Выглядит рабочий DNS CAA в результатах тестирования примерно так:

Revocation status	Good (not revoked)
	Yes
DNS CAA	issue: startssl.com iodef: mailto:info@straining.ru
Trusted	Yes

[Работающий DNS CAA для домена](#)
([кликните для увеличения до 1138 px на 161 px](#))

Теперь посмотрим про кэширование – им тоже нужно управлять, чтобы, например, защититься от избыточной нагрузки в случае часто переподключающихся клиентов.

Кэшируем SSL/TLS-сессии – используя Session ID

Задача кэширования SSL/TLS достаточно древняя. Очевидно, что криптографические операции по совместной генерации ключевого материала и обмена оным – достаточно серьёзно напрягают процессор. Очевидно, что во всей задаче установки и поддержания работы SSL/TLS-сессии таких задач много, и разных – некоторые из них могут достаточно легко быть ускорены оборудованием (допустим, шифрование AES), а некоторые всё же придётся делать программно.

Поэтому есть смысл как-то упростить жизнь клиента, который – в силу обрыва связи, или какой-то ещё причины – повторно подключается к серверу. Как-то сделать так, чтобы ему не надо было “с нуля” инициировать сессию. Самый простой вариант – это ввести некий идентификатор сессии, Session ID.

Реализовано это будет так – когда сервер будет отправлять SERVER HELLO, он будет добавлять туда случайно сгенеренный идентификатор сессии. Клиент, если поддерживает данный механизм, сохраняет этот идентификатор, и после может добавить его в CLIENT HELLO, когда будет устанавливать новую сессию. Сервер, как понятно, хранит информацию только успешно установленных сессий, и записывает доп.данные клиента – например, адрес, из-под которого клиент подключается. Если всё ОК, то сессия продолжается, и клиента подключают по сокращённому сценарию согласования – весь этот механизм описан подробно и

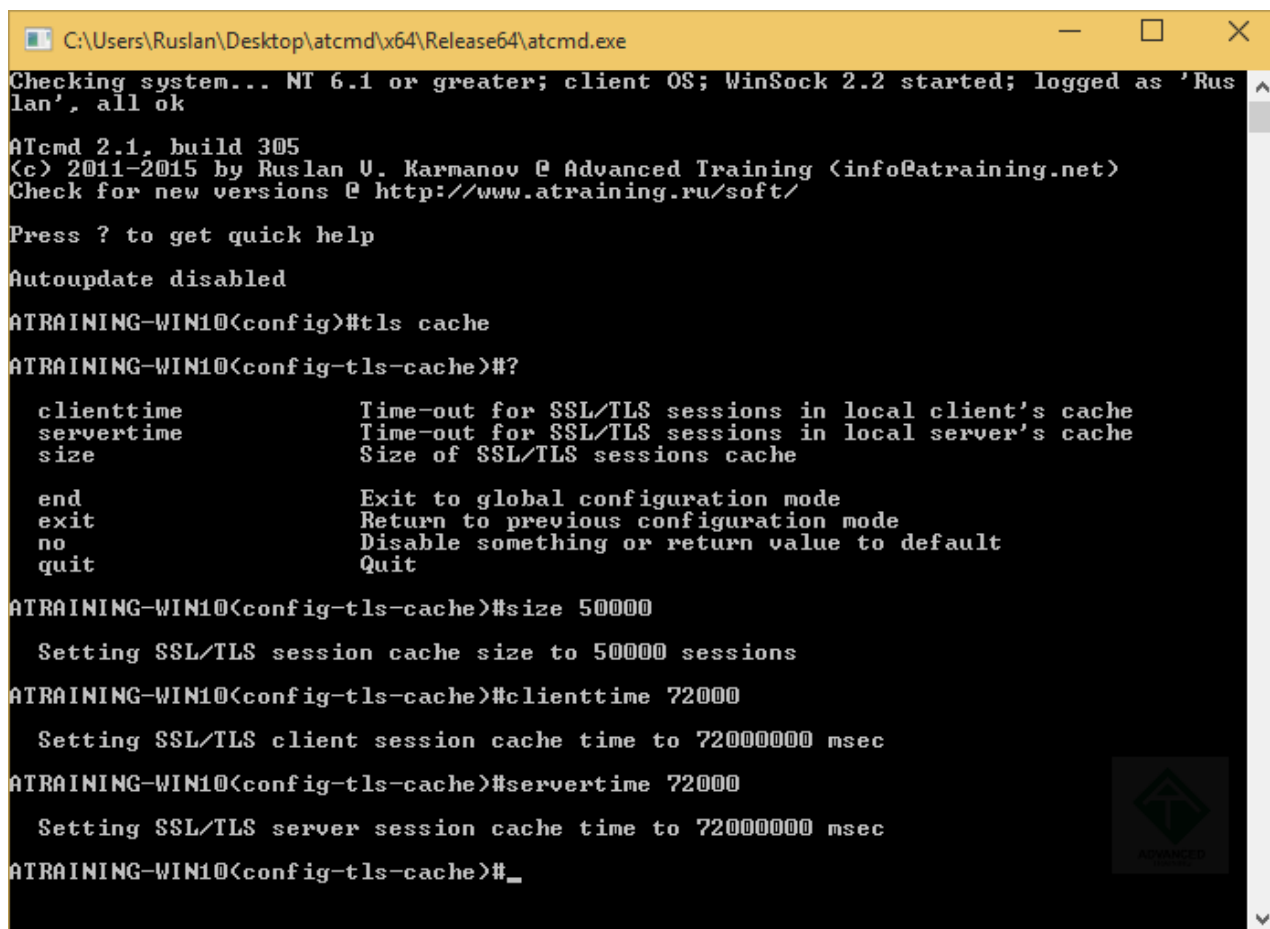
пошагово в [RFC 5246](#). Этот механизм поддерживается начиная с Windows Server 2003 – раньше его надо было дополнительно включать, а теперь он включен в SCHANNEL по умолчанию.

Что же будет кэшироваться? В кэше SCHANNEL будут находиться следующие элементы сессии:

- Master secret – т.е. тот криптографический материал, ради которого был весь стартовый DH/ECDH обмен.
- Список согласованных cipher suite'ов.
- Загруженные сертификаты (в случае серверного кэша – сертификаты, которые предъявил клиент, в случае клиентского – серверные).

Как понятно, возможность не согласовывать/загружать перечисленное выше хорошо ускорит процесс переподключения. Поэтому механизм этот полезный, а нам остаётся только настроить его.

Для этого мы откроем **ATcmd**, зайдём в контекст **tls cache** и выставим там три параметра – максимальный размер кэша Session ID, и тайм-аут одиночных элементов.



```
C:\Users\Ruslan\Desktop\atcmd\x64\Release64\atcmd.exe
Checking system... NT 6.1 or greater; client OS; WinSock 2.2 started; logged as 'Ruslan', all ok

ATcmd 2.1, build 305
(c) 2011-2015 by Ruslan V. Karmanov @ Advanced Training (info@atraining.net)
Check for new versions @ http://www.atraining.ru/soft/

Press ? to get quick help
Autoupdate disabled
ATRAINING-WIN10(config)#tls cache
ATRAINING-WIN10(config-tls-cache)#?

  clienttime      Time-out for SSL/TLS sessions in local client's cache
  servertime      Time-out for SSL/TLS sessions in local server's cache
  size            Size of SSL/TLS sessions cache

  end             Exit to global configuration mode
  exit            Return to previous configuration mode
  no              Disable something or return value to default
  quit           Quit

ATRAINING-WIN10(config-tls-cache)#size 50000
  Setting SSL/TLS session cache size to 50000 sessions
ATRAINING-WIN10(config-tls-cache)#clienttime 72000
  Setting SSL/TLS client session cache time to 720000000 msec
ATRAINING-WIN10(config-tls-cache)#servertime 72000
  Setting SSL/TLS server session cache time to 720000000 msec
ATRAINING-WIN10(config-tls-cache)#_
```

[Настройка TLS Session ID в Windows Server](#) (кликните для увеличения до 696 px на 506 px)

Понятное дело, если ожидается ещё более масштабная нагрузка – параметр “размер кэша” можно увеличить, ну и поиграть со значениями тайм-аутов.

Кэшируем SSL/TLS-сессии – используя Session Tickets

Логика работы данной технологии будет совершенно иной, чем в предыдущем варианте. В случае включения со стороны сервера данного механизма, который полностью называется Transport Layer Security (TLS) Session Resumption without Server-Side State (см. [RFC 5077](#)), на сервере создаётся специальный мастер-ключ, которым шифруются комплекты элементов сессии – почти как в предыдущем варианте – но храниться они будут у клиентов, а не на сервере. Т.е. Session Tickets – это когда сервер шифрует неизвестным клиенту ключом данные о взаимно установленной сессии, и отдаёт клиенту на хранение, а после клиент предъявляет их, сервер успешно расшифровывает, и тогда может продолжать работу.

Звучит изящно, но проблема в том, что это снижает безопасность, т.к. ключ один, и в случае его компрометации будут потенциально уязвимы все Session Tickets, выданные за время действия ключа. Впрочем, это касается лишь ситуации “кто-то захватил сервер и взял ключ и после захватил кэш клиента, где лежит зашифрованный этим ключом сессионный ключ, и тогда расшифровал ранее записанную сессию”. Она теоретически возможна, но на практике проще тогда, если есть возможность захватить и сервер и клиента, просто допросить их с пристрастием.

Включить этот механизм (я про тикеты, а не про допрос) несложно.

Создайте в безопасном месте сервера папку, где будет храниться ключ. После – запустите командлет **New-TlsSessionTicketKey**, указав ему в качестве параметра путь хранения файла с ключом. Например так:

New-TlsSessionTicketKey -Path “C:\TLSKeys\mail-atraining-ru.config”

Командлет попросит Вас придумать пароль – задайте действительно сложный, знаков в 40-50, тем более, что вам этот пароль потребуется лишь пару раз. После успешного выполнения командлета, определите, под какой учётной записью работает Application Pool того сайта, которому вы хотите разрешить использовать эту технологию кэширования, и включите, опять указав пароль при выполнении командлета:

Enable-TlsSessionTicketKey -Path “C:\TLSKeys\mail-atraining-ru.config” -ServiceAccountName “System”

Проверить факт разрешения доступа указанной учётной записи к файлу с ключом просто – зайдите в каталог

%SYSTEMDRIVE%\ProgramData\Microsoft\Crypto\TlsSessionTicketKeys и там будут подкаталоги с SID’ами учётных записей, которым можно работать с TLS Session Ticket Key.

В общем-то всё, теперь механизм работает – если захотите поменять ключ, то отключите командлетом **Disable-TlsSessionTicketKey**, а после создайте новый ключ и опять включите.

Не забудьте проверить, установлен ли патч [KB 3109853](#) – он разрешает некоторые проблемы совместимости с не-Windows системами.

OCSP и проверка сертификатов HTTPS-клиента

OCSP – это полезный способ увеличения оперативности проверки сертификата на отзыв; вместо того, чтобы скачивать и регулярно обновлять CRL-файлы, можно напрямую спросить у специального хоста “отозван ли сертификат с id=12345?”. Это удобно и обычно быстрее (когда сертификатов мало, а вопросов про них много, проще разово скачать CRL). Механизм OCSP Stapling позволяет ещё сильнее улучшить быстродействие – HTTPS-сервер кэширует ответы от OCSP-сервера (подделать их он не может, они подписаны) и передаёт клиенту по запросу. Т.е. клиенту самому, подключаясь к узлу, не надо, скачав сертификат, выяснять кто его выдал, где этот “кто” живёт и где у него OCSP, и идти туда – сервер отправит и свой сертификат, и закэшированные ответы про “этот сертификат валиден”. Это экономит время.

Данный механизм реализован в серверах Microsoft начиная с IIS 7.5 и, по сути, не управляем – т.е. он просто есть и работает. Повлиять на время кэширования или частоту опроса OCSP-сервера не получается.

Однако есть возможность повлиять на схему проверки сертификата в сценарии “к нашему HTTPS-сайту подключается клиент, у которого есть x.509-сертификат”. Ведь в этом сценарии тоже надо проверять подлинность сертификата – только не “клиент проверяет сервер”, а наоборот – сервер проверяет подключающегося клиента.

Управление этой функцией доступно через контекст **http** в ATcmd; команда **quickaia** позволяет выбрать между вариантом “быстрой проверки” – когда проверяется только клиентский сертификат, и “полной” – когда проверяется вся цепочка до корневого. По умолчанию проверка “быстрая” – а вот так включается полная:

```
C:\Users\Ruslan\Desktop\atcmd\x64\Release64\atcmd.exe

ATcmd 2.1, build 311
(c) 2011-2017 by Ruslan V. Karmanov @ Advanced Training (rk@atraining.net)
Check for new versions @ http://www.atraining.ru/soft/

Press ? to get quick help

Autoupdate disabled

ATRaining-WIN10(config)#http
ATRaining-WIN10(config-http)#?

  duo                Client's HTTP Upgrade header will be processed
  http2tcp           Enable HTTP/2 over TCP support
  http2tls           Enable HTTP/2 over TLS support
  mfl                Set upper limit for each HTTP-header
  mrb                Max size of request (+ headers)
  quickaia           Make short check of client certificate (only certificate, no AIA)

  end                Exit to global configuration mode
  exit               Return to previous configuration mode
  no                 Disable something or return value to default
  quit               Quit

ATRaining-WIN10(config-http)#no quickaia

Full client certificate mode check mode enabled

ATRaining-WIN10(config-http)#
```

[Включаем полноценную проверку клиентского HTTPS-сертификата](#)

[\(кликните для увеличения до 979 px на 512 px\)](#)

Более подробная информация про OCSP есть в статье про [настройку и оптимизацию OCSP, OCSP Stapling, OCSP Must-Staple и OCSP Expect-Staple](#).

Пока всё. Теперь – краткие советы по оптимизации быстродействия.

Ускоряем работу SSL/TLS

Ускорение работы – как самой работы, так и установки сессии – очень важно по множеству причин. Что можно сделать в плане ускорения с “абстрактной” TLS/SSL-сессией, не привязываясь к какому-то конкретному сценарию?

Уменьшайте число согласовываемых протоколов

Чем меньше вариантов для согласования – тем лучше. Если у вас, допустим, доступный только для сотрудников с новыми системами сайт – оставьте только TLS 1.2. Если речь о доступном снаружи или для пользователей со старыми системами – TLS 1.0. Можно целиком убрать согласование TLS 1.1, потому что сейчас по сути нет систем, которые поддерживают его, но не 1.2. Поэтому целесообразно оставить только 2 варианта – TLS 1.2 (обычный) и TLS 1.0 (в целях совместимости).

Учитывайте аппаратные тонкости

В современных процессорах алгоритм шифрования AES реализован “на чипе” – поэтому для них вопрос “как именно шифровать данные сессии” не стоит.

Но вот у мобильных клиентов, на процессорах медиатек или снапдрагон, а то и более экзотичных, этой функции нет. Поэтому “шифровать всё AES-256” для них – не особо удачный вариант; на многомегабитных скоростях 4G/WiFi им придётся

честно и программно обрабатывать массивы информации. Задачу можно облегчить, если сделать первыми в списке предлагаемых сервером cipher suites комплекты с CHACHA20, например так:

- ECDHE - ECDSA - CHACHA20 - POLY1305
- ECDHE - RSA - CHACHA20 - POLY1305
- DHE - RSA - CHACHA20 - POLY1305

Это ощутимо снизит затраты вычислительной мощности у мобильных устройств – процентов на 20-60. А следовательно ускорится работа, уменьшится расход батареи и нагрев.

Выключите пересогласование

Пересогласование ключей – что безопасное, что нет – занимает время. Если ваши клиенты подключаются короткими сессиями вида “проверить почту и отвалиться”, то смысла в стартовом обсуждении “а что мы будем делать через час и как пересогласуемся” нет – оно не случится.

Используйте SHA-2/512

Использовать старые алгоритмы хэширования – плохо, поэтому имеет смысл использовать только хэши семейства SHA-2. Но, что удивительно, для семейства SHA-2 логика “раз на выходе меньше бит, то быстрее” – [не работает](#). Выбирайте SHA-512, потому что он реально быстрее считается на 64х битовых системах (а современная серверная система уже с гарантией 64х битная).

Используйте только ECDH, а не DH

ECDH согласовывается быстрее классического DH, а уж если сравнивать ECDH с DH с сильной группой – например, 4096 бит – ощутимо быстрее.

Вкратце всё.

Заключение

В итоге всё может выглядеть примерно так:

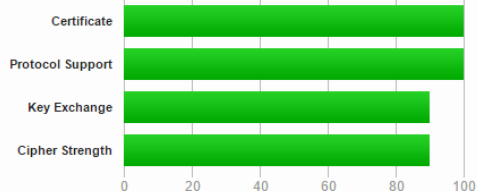
SSL Report: host1.advancedtraining.ru (178.159.49.235)

Assessed on: Mon, 16 Jan 2017 13:25:46 UTC | [HIDDEN](#) | [Clear cache](#)

[Scan Another »](#)

Summary

Overall Rating



Visit our [documentation page](#) for more information, configuration guides, and books. Known issues are documented [here](#).

This site works only in browsers with SNI support.

HTTP Public Key Pinning (HPKP) deployed on this server. Yay!

HTTP Strict Transport Security (HSTS) with long duration deployed on this server. [MORE INFO »](#)

DNS Certification Authority Authorization (CAA) Policy found for this domain. [MORE INFO »](#)

[Результат хорошей настройки TLS в Windows Server](#)
([кликните для увеличения до 1123 px на 680 px](#))

securityheaders.io

Sponsored by APPLAUSE

[Home](#) [About](#)

Scan your site now

Scan

☒ Hide results ☒ Follow redirects

Security Report Summary



Site: <https://host1.advancedtraining.ru/>

IP Address: 178.159.49.235

Report Time: 16 Jan 2017 13:55:15 UTC

Report Short URL: Hidden scans do not get a short URL.

Headers:

☒ Strict-Transport-Security ☒ X-Frame-Options ☒ Public-Key-Pins ☒ X-Content-Type-Options

☒ X-XSS-Protection ☒ Content-Security-Policy

[Другой результат хорошей настройки TLS в Windows Server](#)
([кликните для увеличения до 1280 px на 521 px](#))

(может даже лучше, но на картинке пример рабочего сайта, где не закручивались гайки по криптографии – просто используется и корректно настроен TLS 1.2).

Настройка, защита и оптимизация TLS – это важный момент для обеспечения безопасности практически любой современной IT-системы, т.к. TLS присутствует и используется практически во всех серверных продуктах. Как видно, помимо “просто включить и не париться” есть много других моментов, нужных для эффективной работы. Надеюсь, что эта статья чем-то помогла разобраться в данной задаче.

Удачного применения знаний!