

Chromium, доработка boringssl

У tls соединения есть фингерпринт, почитать что это такое можно вот тут

<https://engineering.salesforce.com/tls-fingerprinting-with-ja3-and-ja3s-247362855967/>

если коротко, то когда происходит инициализация https соединения, отправляется ClientHello пакет, в котором перечислены поддерживаемые клиентом алгоритмы шифрования + экстеншены, это нужно, чтобы договориться о шифровании с сервером. Тк у каждого браузера свой набор алгоритмов и последовательность в пакете, то это позволяет определить браузер или http клиент (его отпечаток/fingerprint). Этот фингепринт используется для защиты от парсинга и ботов. Мы парсим, поэтому есть задача сделать библиотеку, которая сможет эмулировать любой набор фингерпринтов.

Вот тут можно посмотреть свой фингерпринт и почитать инфу дополнительно

<https://tls.peet.ws/>

речь только про ja3 фингерпринт (он самый первый)

TrackMe - fingerprinting API

Your fingerprints

JA3
e88027f8d5003e3bd6bafc34d9f31fda
7714865-4866-4867-49195-49196-49198-49200-52393-52392-49171-49172-156-157-47-53,0-45281-10-16-13-61-35-43-5-45-23-27-11-18-17513-65037-21,29-23-24,0

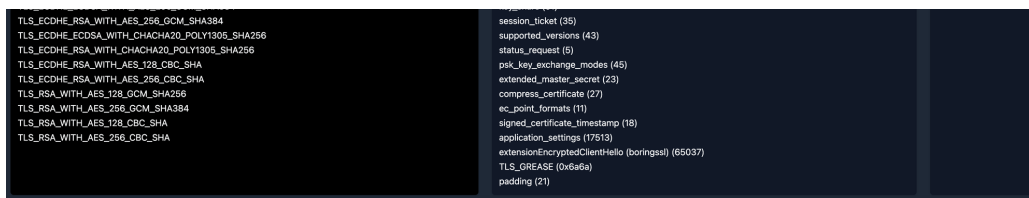
JA4
t13d1517h2_8daaf6152771_39ff608a2b00

Akamai HTTP/2 fingerprint
90224459f8bf70b7d0a8797eb916dbc9
165538,2,0,4,6291456,6,262144/156631080|m.s.p

RealPrint
96b72764e015fb524fd8d005deb7209d
GREASE-772-77102-1,10GREASE-23-23-24/1027-2002-1023-1282-2053-1281-2054-1537/102GREASE-4866-4866-4867-49195-49196-49198-49200-52393-52392-49171-49172-156-157-47-53/0-10-11-13-16-17513-18-21-23-27-35-43-45-5-51-65037-65281-GREASE-GREASE

вот тут список расширений и алгоритмов шифрования для моего браузера, как пример

TLS cipher suites	TLS extensions	TLS curves
TLS_GREASE (0xcaca) TLS_AES_128_GCM_SHA256 TLS_AES_256_GCM_SHA384 TLS_CHACHA20_POLY1305_SHA256 TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	TLS_GREASE (0xcaca) server_name (0) extensionRenegotiationInfo (boringssl) (65281) supported_groups (10) application_layer_protocol_negotiation (16) signature_algorithms (13) key_share (51)	TLS_GREASE (0x2a2a) X25519 (29) P-256 (23) P-384 (24)



те строка

771,4865-4866-4867-49195-49199-49196-49200-52393-52392-49171-49
172-156-157-47-53,51-18-5-23-65037-45-10-27-17513-13-16-35-43-0-11
-65281-41,29-23-24,0

с первого скриншота - это коды алгоритмов шифрования и
расширений на втором скрине

мы уже делали что-то подобное для старой версии хрома (внутри
он использует boringssl либу для ssl соединения), это было давно,
наша доработка позволяла выстраивать последовательность из уже
поддерживаемых алгоритмов шифрования и расширений, весь этот
код есть и будет передан исполнителю

тк хром тяжело и долго собирается, то мы сделали тестовый
проект, который делает запрос по https (с использованием boringssl,
которую использует хром) и возвращает результат в stdout, код
тестового приложения тоже есть

Что нужно сделать:

1. взять наш код для старой версии хрома и доработать тестовый
проект, это позволит менять последовательность и состав пакета
ClientHello, естественно, использоваться будут только
поддерживаемые алгоритмы шифрования + расширения. Смысл
там примерно такой - кладем рядом с проектом файл finger.txt вот
такого содержания (ja3 фингерпринт)

771,4865-4866-4867-49195-49199-49196-49200-52393-52392-4917
1-49172-156-157-47-53,51-18-5-23-65037-45-10-27-17513-13-16-35-
43-0-11-65281-41,29-23-24,0

после чего список алгоритмов и расширений в ClientHello
выстраивается в соответствии с этим fingerprint'ом и

используется для установления ssl соединения

используется при инициализации `tls` соединения

2. написать тест, который будет делать следующее: кладем рядом файл `fingerprint_test.txt`
в нем будет список fingerprints, каждый с новой строки, тест должен проверять - все ли алгоритмы и расширения из fingerprints поддерживает boringssl, если какого-то нет - писать об этом, если все есть - делаем запрос вот сюда <https://tls.peet.ws/api/tls> и проверяем, что текущий fingerprint соответствует полю `ja3`, если не соответствует, значит что-то не так - пишем лог

это все.

Это первый этап, следующий будет заключаться в реализации всех недостающих алгоритмов шифрования и экстеншенов, которые мы найдем в результате тестирования, они все есть в других open source либах, те задача будет интегрировать их в проект. Следующий этап оплачивается отдельно.

