Seguridad del protocolo SSL/TLS. Ataques criptoanalíticos modernos

Autor: Dr. Alfonso Muñoz (@mindcrypt)
Prólogo: D. Juliano Rizzo

Madrid, 27/09/2020

Prólogo – D. Juliano Rizzo (@julianor)

Cuando estés leyendo estas páginas, los protocolos TLS 1.0 y TLS 1.1 ya habrán sido bloqueados por el navegador web más usado del mundo y por lo tanto muchos sitios web forzados a actualizar a versiones más seguras y de esa manera los usuarios estaremos más protegidos. Este progreso es en gran parte un logro de los trabajos de investigación enumerados en este libro. Es un escenario muy distinto al de unos años atrás cuando me interesé en estudiar en profundidad SSL/TLS, por esa época el icono de un candado verde parecía indicador suficiente de la seguridad de la conexión a un sitio web. Ese candadito no indicaba mucho más que el éxito en la negociación entre cliente y servidor de alguna de las versiones de SSL/TLS y algunos de los posibles algoritmos de cifrado implementados por ambas partes. Podíamos ver un candado verde incluso si se había negociado la versión más antigua y vulnerable de SSL con un conjunto de algoritmos débiles y con claves de longitud insuficiente. No dejo de admirar la intensidad y velocidad con la que desde ese momento ha mejorado HTTPS, el protocolo que hace posible que existan los servicios web y de aplicaciones móviles que usamos a diario.

Recorrer las páginas de este libro me remonta al momento que conocí a Thai Duong mientras participamos de una competencia tipo CTF en equipos rivales, y cómo a partir de esa coincidencia empezamos a trabajar juntos investigando y desarrollando ataques prácticos contra protocolos criptográficos y aplicaciones de uso masivo. Publicamos nuestros resultados de variadas formas: en listas de correo, en presentaciones ilustradas con memes y también como prolijos *whitepapers*. Pero en todos los casos nos esforzamos para escribir el código de ataque, probarlo contra objetivos reales y demostrarlo al público. Creíamos que al demostrar un ataque real y práctico motivaría a otros investigadores y al dejar claro el impacto de las vulnerabilidades sería más probable lograr una reacción de quienes podían hacer algo al respecto.

Además de divertirnos y aprender mucho, uno de nuestros aportes fue de alguna manera hacer de puente entre resultados de investigaciones académicas y la industria trabajando en explotar de una forma práctica hallazgos que tal vez ya eran noticia antigua en el ambiente de la investigación pero que no habían sido atendidos por los fabricantes de software.

En algunos casos partimos de una oración en la especificación de un protocolo que dejaba ver un riesgo del diseño, en otros de una breve descripción de un potencial problema en un intercambio de mensajes entre desarrolladores de una biblioteca criptográfica. Para llegar desde esas pequeñas pistas a desarrollar herramientas con la capacidad de comprometer la seguridad de un sistema real, nos ayudó nuestra experiencia en seguridad web e idear técnicas para implementarlas de una manera eficiente. Apuntar a obtener las cookies de autenticación y hacerlo letra por letra o incluso bit por bit es algo que puede parecer obvio en retrospectiva pero que hasta ese momento no formaba parte de los estudios publicados sobre la seguridad de SSL/TLS.

Leer, por ejemplo: "*este protocolo no es seguro contra adversarios que puedan elegir arbitrariamente el contenido de un mensaje antes de ser cifrado*" no tiene el mismo efecto que presenciar cómo la cookie de autenticación de uno de los sitios más usados de Internet se descifra letra por letra en unos minutos, como en la escena menos verosímil de una película de hackers. Pero la próxima vez que leamos en algún documento o identifiquemos en algún sistema una debilidad parecida, tendremos una idea gráfica de cuál puede ser el resultado si se dan las condiciones para el atacante.

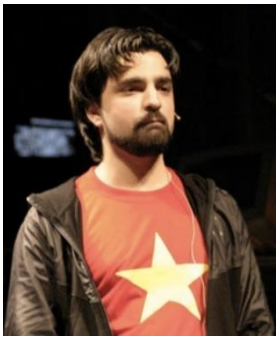
Entender en detalle el origen de las fallas y de las técnicas para abusar de ellas es una parte fundamental del aprendizaje de la seguridad informática. Han pasado años, pero cada tanto recibo un mensaje pidiendo ayuda de algún estudiante al que le han encargado reproducir ataques como BEAST o CRIME. Rescatar software antiguo para experimentar no es de lo más divertido, también se puede experimentar con desafíos de competencias CTF que se han inspirado en vulnerabilidades reales. A

quienes estén leyendo este libro interesados en investigar para mejorar la seguridad de sistemas y protocolos existentes o aprender de los errores del pasado al diseñar nuevos sistemas, les recomiendo elegir alguno de los ataques aquí descritos y reproducirlo experimentando con lo más parecido a un ambiente real que puedan. En mi experiencia, desarrollar un ataque contra sistemas que están en producción es muy motivante y se aprende mucho de los pequeños retos que se van superando y de los intentos fallidos. Es posible que requiera más tiempo superar esos pequeños obstáculos que lograr una demostración de la existencia del problema principal. Pero, esos retos que se encuentran en el camino de la prueba de concepto al ataque real, son muchas veces de los que más se aprende y superarlos nos puede conducir a otros hallazgos importantes.

Los que se acerquen a este tema por primera vez se sorprenderán al ver en el índice nombres como FREAK, ROBOT, y SWEET32; soy orgullosamente culpable de haber iniciado esa tendencia de nombres que empecé con POET (Padding Oracle Exploit Tool) pero claro, el inocente poeta (aunque todavía un ataque poderoso) no llamó la atención en el mundo tanto como la temible bestia. Se suele criticar a quienes publican sus trabajos de investigación poniéndole un nombre, un sitio web y un logotipo, pero es evidente que estos recursos son de ayuda para llamar la atención, hablar del problema, difundirlo y recordarlo.

Además del estilo de nombres, las investigaciones de ataques contra SSL/TLS que la publicación de BEAST empujó, tienen en común el aprovecharse de la posibilidad que tiene el atacante de forzar al navegador web de la víctima a hacer gran número de peticiones de forma automática. Como verán en las siguientes páginas, la posibilidad de iniciar y repetir peticiones ha sido aprovechado para explotar vulnerabilidades criptográficas de todo tipo.

El libro de Alfonso Muñoz invita a más que revisar la historia de un protocolo, sus introducciones a cada ataque y los enlaces a referencias que incluye son buenos puntos de partida para aprender sobre fallas que siguen encontrándose frecuentemente en sistemas criptográficos actuales y que es necesario comprender para poder evitar que afecten a aplicaciones del futuro.



Juliano Rizzo se ha dedicado a la seguridad informática por más de veinte años. Tiene experiencia en la explotación de vulnerabilidades de todo tipo, pruebas de penetración, auditoría de código fuente e ingeniería inversa. Publicó alertas de vulnerabilidades e investigaciones incluyendo ataques prácticos a sistemas criptográficos: ataque padding oracle contra la plataforma Microsoft APS.NET y los famosos BEAST y CRIME contra el protocolo SSL/TLS. Actualmente se dedica a la consultoría a través de su empresa Coinspect, enfocada en la seguridad de criptomonedas y contratos inteligentes.
Twitter: @julianor

Introducción – Dr. Alfonso Muñoz (@mindcrypt)

Comencé en el mundo de la seguridad informática con 14 años, un poco más tarde que muchos de los colegas de profesión que respeto. Me atrajo el mundo más oscuro, al menos como se describía en la década de los 90, del hacking, el cracking, el carding, el phreaking, el viring, etc. Durante ese crecimiento personal y profesional, dicen que la juventud es la única enfermedad que se cura con el tiempo, descubrí la ciencia de la criptología como ese mecanismo ideal para contestar a muchas preguntas relacionadas con el diseño de contramedidas a la cantidad de ataques conocidos.

La criptografía lleva conmigo más de 20 años y para mi es difícil entender un profesional de la seguridad de la información sin amplios conocimientos en esta disciplina. Por suerte, a día de hoy, existen numerosos recursos: libros, cursos online, certificaciones o comunidades para intercambiar conocimientos, pero es cierto que no siempre es sencillo, incluso para profesionales del campo de la seguridad, iniciarse en determinadas prácticas, esquemas, algoritmos o arquitecturas, y sobre todo no es sencillo entender los matices y recomendaciones para utilizar la criptografía en el mundo real.

El texto que tiene entre sus manos surge de la necesidad de explicar de forma "asequible" los ataques criptográficos recientes a las implementaciones de TLS/SSL con varios objetivos: aumentar el conocimiento criptográfico, mayor comprensión para el diseño de mejores protocolos y arquitecturas criptográficas, y habilitar capacidades en analistas que analizan la seguridad de tecnologías criptográficas (hardware o software).

Este escrito no pretende ser excesivamente exhaustivo en los detalles de cada ataque, resumidos en no más de 2 páginas cada uno, pero le facilitará al lector toda la información para entender y profundizar en los mismos al nivel de profundidad que desee.

Siempre he creído que la criptografía ayuda a construir una sociedad más libre y más justa.

Quizás este documento ayude en esa dirección.

Dr. Alfonso Muñoz es experto en seguridad informática, área en la que trabaja profesionalmente desde hace 18 años. Su principal actividad se centra en proyectos técnicos avanzados en seguridad defensiva y ofensiva (Global 500) y su colaboración con organismos públicos. Su actividad ha sido reconocida con diversos reconocimientos académicos e industriales, entre ellos por reportar vulnerabilidades en productos de grandes fabricantes (Google, Microsoft, etc.). Es socio de la empresa CriptoCert que proporciona la primera certificación de la información y criptografía en español a nivel mundial. **Twitter: @mindcrypt**

Actualización del libro

Sólo hay un bien: el conocimiento.
Sólo hay un mal: la ignorancia.

Sócrates *(470 AC-399 AC)*

La primera versión de este libro fue publicada en mayo de 2020. El libro que tiene entre sus manos es una versión actualizada del mismo con nuevos ataques publicados desde la primera edición del mismo. Esta revisión la he aprovechado para actualizar y complementar adicionalmente alguna referencia olvidada. Cualquier recomendación o información de interés que considere de utilidad para este libro puede contactar a través de mi correo personal – **alfonso@criptored.com**

Espero que la lectura sea de su interés.

Índice

Listado de figuras

1. Criptoanálisis de comunicaciones modernas. Desde la cifra clásica a la computación cuántica

Confiar en todo el mundo y no confiar en nadie
son un mismo defecto.
En uno se encuentra más virtud y en el otro,
más seguridad
Séneca

La criptografía es una de las ramas más pesimistas de la ciencia. Asume la existencia de adversarios con capacidad ilimitada de ataque, los cuales pueden leer todos tus mensajes, generar información ilegítima o modificar tus claves aleatorias a su antojo. Curiosamente, también, es una de las ramas más optimistas, mostrando como incluso en el peor escenario inimaginable el poder de las matemáticas y la algoritmia puede sobreponerse a cualquier dificultad[1]. O al menos, así es en teoría.

Desde la criptografía clásica empleada por el emperador Julio César hasta nuestros días, los atacantes han encontrado mecanismos diversos para vulnerar la privacidad de las comunicaciones, analógicas entonces, digitales ahora. Históricamente, la seguridad basada en oscuridad[2] facilitaba esta tarea al atacante, dado que si era capaz de conocer la técnica criptográfica la tarea de descifrado se simplificaba enormemente. Una referencia excelente para comprender el mundo de la criptografía clásica y su criptoanálisis puede verse en el trabajo de William F. Friedman (https://archive.org/details/nsa-friedman).

Siglos después con la aparición de los principios de Kerckhoffs[3] (1883), y la formalización de la teoría de la información por Claude Shannon (1948)[4], definiendo difusión y confusión, la ciencia de la criptografía adquirió tal madurez que, en la práctica, el atacante tenían que recurrir a esquivarla (robar las claves criptográficas), rezar por un fallo en su implementación o diseño[5], forzar los fallos, o realizar ataques de diccionario o fuerza bruta esperando tener suerte, de tal forma que con estas estrategias el atacante pudiera tener alguna oportunidad.

Por ejemplo, si se centra la atención en la implementación o uso de algoritmos o librerías criptográficas se encuentran alguno de los problemas más habituales[6]:

a) **Crear tu propio algoritmo criptográfico o implementar uno existente**. El diseño de un algoritmo criptográfico (incluyendo protocolos y modos de operación) requiere de un conocimiento significativo, matemático y práctico. Incluso, aun cumpliendo esto, la garantía no es absoluta. Un típico ejemplo es el orden de las operaciones para realizar un cálculo determinado, por ejemplo, exponenciación de un número, que puede facilitar la fuga de información a atacantes mediante técnicas de canal lateral[7] (side-channel). Un ejemplo interesante, en este sentido, positivo (hay opiniones a favor y en contra), en cuanto a creación propia de librerías, es la librería de Google BoringSSL (https://github.com/google/boringssl).

b) **Mal uso de librerías o algoritmos**. Incluso cuando se usa librerías "robustas" las garantías no son absolutas, dado que los desarrolladores pueden utilizar las librerías de manera

[1] https://www.premiosfronterasdelconocimiento.es/noticias/fundacion-bbva-premio-fronteras-goldwasser-micali-rivest-shamir-criptografia/
[2] https://es.wikipedia.org/wiki/Seguridad_por_oscuridad
[3] https://es.wikipedia.org/wiki/Principios_de_Kerckhoffs
[4] https://es.wikipedia.org/wiki/Confusi%C3%B3n_y_difusi%C3%B3n
[5] Failures of secret-key cryptography - http://cr.yp.to/talks/2013.03.12/slides.pdf
[6] https://cybersecurity.ieee.org/blog/2015/11/13/use-cryptography-correctly
[7] https://es.wikipedia.org/wiki/Ataque_de_canal_lateral

inadecuada[89], por ejemplo, no eligiendo el algoritmo adecuado o usándolo incorrectamente. Un ejemplo de esto suele ser la incorrecta utilización de vectores de inicialización (IV) o número aleatorios (nonces) en un algoritmo dado. Una excelente recomendación de lectura en este sentido es el proyecto Tink de Google (https://security.googleblog.com/2018/08/introducing-tink-cryptographic-software.html) y el proyecto Wycheproof (https://github.com/google/wycheproof) que permiten entender fallos comunes y recomendaciones para desarrollar software criptográfico con mayores garantías.

c) **Una incorrecta protección de claves criptográficas**. La seguridad de los sistemas criptográficos, habitualmente, recae en las contraseñas. Algunos errores comunes son anotar las claves en el software (código fuente), fallos en la revocación/rotación de claves o directamente el uso claves débiles o predecibles.

d) **Aleatoriedad que no es "aleatoria".** Confusión entre aleatoriedad estadística y aleatoriedad "criptográfica". Las operaciones criptográficas requieren de números aleatorios que tiene ciertas propiedades estrictas de seguridad que no son sencillas de conseguir.

e) **Criptografía no aislada**. El software criptográfico interactúa con otros elementos software o hardware que podrían afectar a su seguridad.

En resumen, la complejidad y evolución de la tecnología, y por qué no destacar también la falta de formación, es tal que los fallos anteriores surgen de manera habitual. Aunque también, históricamente, han existido esfuerzos por introducir fallos intencionadamente. Recientemente los proyectos Bullrun[10] (NSA) y EdgeHill (GCHQ) son un buen ejemplo del interés por debilitar la robustez de algoritmos criptográficos utilizados masivamente en comunicaciones digitales. Edward Snowden[11], en 2013, publicó documentos que afirman la existencia de programas clasificados con el interés de anular las protecciones criptográficas en comunicaciones y datos. Entre las múltiples que se pueden imaginar (robo de contraseñas, infección de redes y terminales[12], criptoanálisis, infección y colaboración de operadoras de telecomunicación, etc.) destaca una bastante peculiar y son las relaciones e influencias en los organismos e industrias principales criptográficas. Su interés es claro, propusieron de manera sutil (viendo que propuestas como el chip Clipper[13] no tuvieron aceptación) algoritmos criptográficos que pudieran ser anulados con mayor facilidad o insertar vulnerabilidades en los sistemas existentes. Por ejemplo, la NSA influyó[14] para promover y adoptar estándares, protocolos y sistemas con debilidades. Así fue, en 2006, cuando la NSA utilizó al organismo de estandarización norteamericano NIST para la estandarización del generador de números aleatorios Dual_EC_DRBG, que más tarde se demostró "intencionadamente" inseguro. Del mismo modo, que se sepa, han trabajado con proveedores de software o hardware criptográfico para que liberen sistemas con debilidades. Un ejemplo significativo, es la relación entre la NSA y la compañía de seguridad RSA. Se ha documentado como se pagó a la empresa RSA, filial de EMC, para que usara la fórmula matemática Dual_EC_DRBG como método generador de números aleatorios por defecto de su software de cifrado Bsafe, aún a sabiendas de la existencia de debilidades. Snowden afirma que

[8] Efail: Breaking S/MIME and OpenPGP Email Encryption using Exfiltration Channels - https://i.blackhat.com/us-18/Thu-August-9/us-18-Mueller-Dresen-EFAIL-Breaking-SMIME-And-OpenPGP-Email-Encryption-Using-Exfiltration-Channels.pdf

[9] Debian OpenSSL Predictable PRNG - https://github.com/g0tmi1k/debian-ssh

[10] https://en.wikipedia.org/wiki/Bullrun

[11] https://en.wikipedia.org/wiki/Global_surveillance_disclosures_(2013%E2%80%93present)

[12] Según los documentos, estas agencias tendrían la capacidad, en determinados entornos, de burlar la protección del protocolo SSL utilizado en comunicaciones mediante VPN (Virtual Private Network) - https://www.theguardian.com/world/2013/sep/05/nsa-gchq-encryption-codes-security

[13] https://es.wikipedia.org/wiki/Chip_Clipper

[14] Estas influencias no siempre tienen el éxito deseado. Un ejemplo reciente es como se descartó los algoritmos SIMON y SPECK de la NSA para su estandarización - https://www.cbronline.com/news/iso-nsa

proyectos como Bullrun estaría sustentados por recursos de unos cuantos cientos de millones de dólares. Al menos, desde 2011, esta cifra superaría los 800 millones de dólares. Sin duda, estas cifras son bastante cuantiosas e incluso algunos autores han llegado a afirmar que esas sumas serían superiores a toda la inversión mundial académica en esta disciplina. En cualquier caso, hasta lo que se conoce, y muchas filtraciones se han conocido en esta última década, parece que no es suficiente[15][16].

Por tanto, si no fuera posible ninguno de los mecanismos anteriores el atacante siempre podría recurrir a los recursos tradicionales: ataques de diccionario o fuerza bruta para revelar las claves criptográficas o mecanismos para su robo, tradicionalmente con malware o ataques de canal lateral.

Por un lado, los ataques de diccionario o fuerza bruta han sido el recurso más utilizado, con arquitecturas de propósito específico o general, para usar la computación masiva como herramienta de criptoanálisis. Ejemplos clásicos pueden verse en propuestas para anular el antiguo estándar de cifrado DES (Descraker[17]) o contra algoritmos de clave pública (Twinkle[18] o Twirl[19]). Si bien es cierto que recursos basados en el uso de GPUs y la herramienta Hashcat[20] es una opción muy socorrida en la actualidad, especialmente con el uso de rainbowtables[21]. Por suerte, a día de hoy, por la madurez de la ciencia de la criptografía y la evolución de la ley de Moore, es posible diseñar un balance adecuado de robustez y renovación de los algoritmos criptográficos, presentando la única duda a esta tendencia el advenimiento de la computación cuántica. El algoritmo de Shor[22] y Grover[23] presentan nuevos retos para fortalecer la criptografía actual y desarrollar nuevos paradigmas y algoritmos, como, por ejemplo, la criptografía postcuántica. Si bien es cierto, que la probabilidad de anulación de la criptografía será mayor por robo de contraseña, especialmente mediante ataques más sofisticados de canal lateral.

Los ataques de canal lateral son muy antiguos conceptualmente y hasta el momento de escribir este libro es uno de los procedimientos de ataque que, probablemente, esté siendo más investigado con el interés de que exista una posible fuga de información que simplifique el criptoanálisis. Algunos de los ataques más documentados[24][25] han sido basados en dispositivos físicos, emanación acústica, electromagnética, magnética, eléctrica, óptica o térmica. Si bien es cierto que en 2018 los ataques basados en la posibilidad de lectura de la cache o instrucciones de la CPU[26] tuvieron bastante impacto desde un punto de vista criptoanalítico. Para estos ataques de caché o CPU, aunque en principio se requiere de software local para progresar con el mismo, no debe descartarse que un atacante pudiera desarrollar capacidades para realizarlos en remoto, como demostró Adi Shamir con otros ataques basados en criptoanálisis acústico (https://www.cs.tau.ac.il/~tromer/acoustic/).

Todos estos ataques permiten observar la necesidad de robustecer al máximo los procedimientos, algoritmos y protocolos criptográficos, así como gestionar debidamente el ciclo de vida de las claves criptográficas, prestando un especial interés en la renovación periódica de las mismas.

---

[15] The NSA's Cryptographic Capabilities. https://www.schneier.com/blog/archives/2013/09/the_nsas_crypto_1.html
[16] https://www.wired.com/2013/09/black-budget-what-exactly-are-the-nsas-cryptanalytic-capabilities/
[17] https://en.wikipedia.org/wiki/EFF_DES_cracker
[18] https://en.wikipedia.org/wiki/TWINKLE
[19] https://en.wikipedia.org/wiki/TWIRL
[20] https://hashcat.net/hashcat/
[21] https://en.wikipedia.org/wiki/Rainbow_table
[22] https://es.wikipedia.org/wiki/Algoritmo_de_Shor
[23] https://en.wikipedia.org/wiki/Grover%27s_algorithm
[24]https://i.blackhat.com/us-18/Wed-August-8/us-18-Camurati-Screaming-Channels-When-Electromagnetic-Side-Channels-Meet-Radio-Tranceivers.pdf
[25] https://i.blackhat.com/us-18/Wed-August-8/us-18-Guri-AirGap.pdf
[26] Meltdown & Spectre - https://i.blackhat.com/us-18/Thu-August-9/us-18-Fogh-Ertl-Wrangling-with-the-Ghost-An-Inside-Story-of-Mitigating-Speculative-Execution-Side-Channel-Vulnerabilities.pdf

No debe olvidarse que cualquier información cifrada podría ser susceptible de ser atacada si es almacenada y en un futuro se descubren los fallos descritos anteriormente. Bien es cierto que, en la práctica, parece razonable, además de almacenar cierto tráfico cifrado, el almacenamiento de claves públicas. Si existe algún error quizás se pueda recuperar la clave privada, descifrar tráfico y acceder a sistemas. Se han documentado diversos proyectos de agencias gubernamentales para el almacenamiento y consulta masiva de información (X-Keyscore NSA, PRISM NSA, Tempora GCHQ, etc.), es algo lógico y un recurso fácil (aunque no barato) si se descubre algún atajo. Y ese es precisamente el problema, que en ocasiones existen atajos, y no siempre es necesario ser una superpotencia para aprovecharlos. Almacenar información cifrada tarde o temprano será de utilidad.

Por ejemplo, en el año 2018, se demostró que mediante el algoritmo GCD (Batch-GCD attack), ataque ya conocido hace años, era posible si existía una enorme cantidad de claves públicas (n,e), deducir factores en común y si estos existían, entonces derivar la clave privada. Para ello, "simplemente" sería necesario un enorme volumen de claves públicas y una enorme capacidad de computación y almacenamiento. Eso sí, no se podría atacar a una clave pública concreta, si no que la clave a romper, no depende de la elección del atacante, depende de las propiedades de todas las claves públicas involucradas en el cómputo. Es un matiz importante, pero en multitud de escenarios con obtener la clave privada de algunas de las claves públicas bajo estudio sería de suficiente interés. Los investigadores Nils Amiet y Yolan Romailler presentaron en agosto de 2018 en la conferencia de seguridad Defcon celebrada en las Vegas su estudio: *Reaping and breaking keys at scale – When crypto meets big data*[27], que demuestra su viabilidad. En su estudio recopilaron más de 240 millones de claves únicas de certificados X.509, 94 millones de claves públicas ssh y más de 10 millones de claves PGP, encontrado que de media 1 de cada 1600 claves era vulnerable. Una cantidad bastante impresionante, de las cuales pudieron obtener la clave privada (RSA y ECC).

Cómo puede verse la ciencia de la criptografía es apasionante y requiere estar al día (recomendaciones, procesos y buenas prácticas) para que sea difícil de batir.

[27] https://research.kudelskisecurity.com/2018/10/16/reaping-and-breaking-keys-at-scale-when-crypto-meets-big-data/

2 Seguridad del protocolo SSL/TLS. Ataques criptoanalíticos modernos

Fatigas, pero no tantas; que a fuerza de muchos
golpes hasta el hierro se quebranta. El que quiera, no lo
diga; haga como que no quiere, y aprenda a pasar fatigas.

Manuel Machado

En el capítulo anterior, se ha descrito una visión global de las variantes de ataques a algoritmos criptográficos, que permite al lector contextualizar el tema cubierto en este apartado, en el que se focaliza el presente libro. Este apartado tiene por objetivo recopilar alguno de los ataques criptográficos más significativos a la seguridad de uno de los protocolos criptográficos más difundidos en la actualidad: TLS/SSL. El apartado pretende reflejar un buen conjunto de variantes de ataques que demuestran lo ya conocido, la implementación y el uso correcto de la criptografía en el mundo real es difícil. Además de conocer de manera sencilla métodos ingeniosos de criptoanálisis, el conocimiento de los mismos facilitará al lector la posibilidad de diseñar mejores sistemas, algoritmos, protocolos criptográficos y auditarlos si fuera necesario.

El objetivo principal del libro es acercar estos ataques de manera sencilla y resumida al lector, proporcionando información adicional muy detallada para su estudio en profundidad.

En este libro no se cubre, aunque los ataques se orientan a este protocolo, la descripción del protocolo SSL/TLS. Aunque no es vital su conocimiento en detalle para comprender los ataques descritos es recomendable la lectura mínima de las siguientes referencias:

- The Illustrated TLS Connection - https://tls.ulfheim.net/
- The Transport Layer Security (TLS) Protocol Version 1.3. RFC8446 - https://tools.ietf.org/html/rfc8446
- https://es.wikipedia.org/wiki/Transport_Layer_Security

En las últimas dos décadas se han publicado numerosos ataques, investigaciones y artículos académicos presentando problemas de seguridad de los protocolos SSL/TLS o de alguno de los algoritmos criptográficos utilizados. En la primera versión de este libro se ha decidido detallar solo alguno de los ataques más significativos (recientes) centrados en anular la criptografía, no obstante, se recomienda al lector la lectura de las siguientes referencias:

- https://www.feistyduck.com/ssl-tls-and-pki-history donde podrá observar algunos de los eventos más significativos en la historia del protocolo SSL/TLS desde el punto de vista de la seguridad y la criptografía.
- Summarizing Known Attacks on Transport Layer Security (TLS)and Datagram TLS (DTLS). RFC7457 - https://tools.ietf.org/html/rfc7457
- Attacks on SSL/TLS Applied Cryptography - https://www.win.tue.nl/applied_crypto/2017/20171130_TLS_attacks.pdf

2.0 BEAST (The Browser Exploit Against SSL/TLS)

Los investigadores Thai Duong y Juliano Rizzo publicaron en septiembre de 2011, en la conferencia de seguridad Ekoparty, su investigación "The *Browser Exploit Against SSL/TLS (BEAST)"* que demostraba que un atacante podría descifrar los datos intercambiados entre dos partes aprovechándose de un fallo de implementación, CVE-2011-3389, cuando se usa el modo de cifrado CBC (Cipher Block Chaining) en SSL o TLS 1.0. El ataque se basa en la posibilidad de construir un ataque adaptativo de texto en claro elegido con vectores de inicialización predecibles (IV). Esta idea ya había sido sugerida años anteriores[28], pero por primera vez se explotó de manera práctica y con impacto.

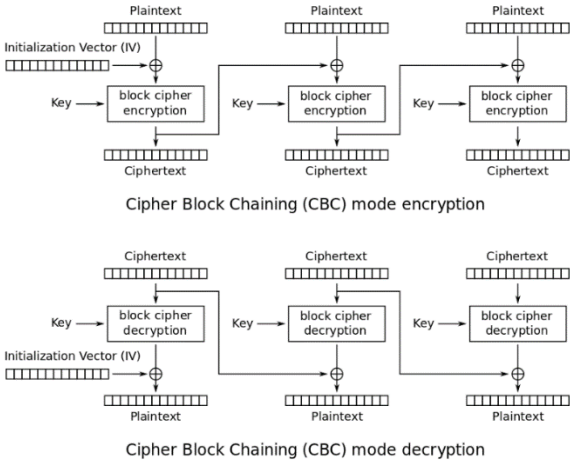


Figura 1. Cifrado de bloque en modo CBC

El ataque se basa en la posibilidad del atacante de sugerir un texto en claro relacionado con un texto cifrado capturado, para analizar esta tarea, saber si es correcto o no, se apoya/necesita de un oráculo criptográfico.

Supongamos que un atacante quiere conoce el significado de un mensaje P_{real} que usa un vector de inicialización conocido IV_1. El atacante sabe que el resultado del texto cifrado (*ciphertext*) es igual a: $E_k(IV_1 \oplus P_{real})$, véase el inicio de la cadena CBC en la imagen anterior. Supongamos ahora que el atacante conoce el vector de inicialización a utilizar en el siguiente bloque, IV_2, en esta situación el atacante puede intentar adivinar P_{guess} inyectando el mensaje en claro $P_{guess} \oplus IV_1 \oplus IV_2$ que generará el siguiente texto cifrado $E_k(P_{guess} \oplus IV_1 \oplus IV_2 \oplus IV_2) = E_k(IV_1 \oplus P_{guess})$. En este punto si los textos cifrados coinciden, el atacante sabrá que $P_{real} = P_{guess}$. Recuérdese que una sesión SSL 3.0 o TLS 1.0 divide la información en una cadena de bloques de información a proteger. Los bloques a cifrar con CBC usarán como IV el cifrado del último bloque.

Intentemos comprender el ataque de una manera más visual.

[28] Vulnerability of SSL to Chosen-Plaintext Attack - http://eprint.iacr.org/2004/111.pdf

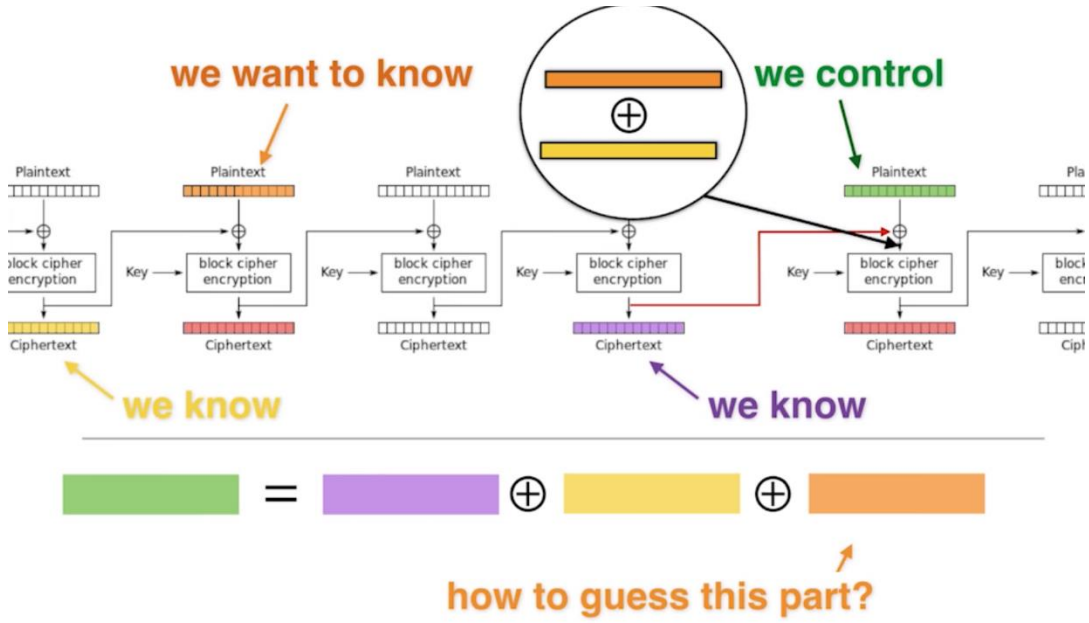


Figura 2. Esquema visual del ataque BEAST

En este ataque un atacante puede ver los mensajes cifrados, los IVs utilizados, y tiene la capacidad de operar sobre cierta información de interés, por ejemplo, es predecible la localización de una cookie aunque vaya cifrada. Si el atacante puede forzar a que se cifren mensajes en claro elegidos por él, típicamente mediante la inyección de un código JavaScript en un navegador web, y enviarlos en nombre de la víctima, entonces el atacante puede realizar comparaciones con los textos cifrados y llegar a predecir y descifrar información.

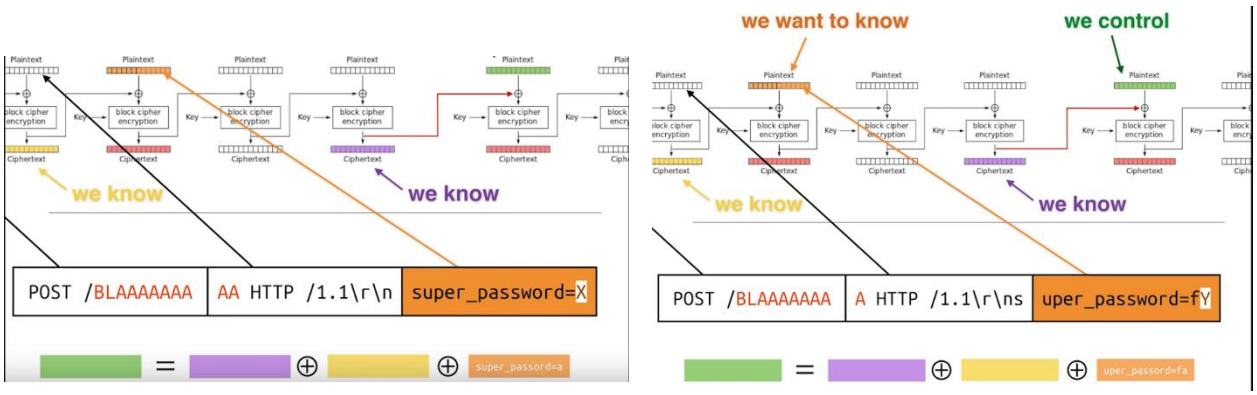


Figura 3. Ejemplo de adivinación de cookies con BEAST

Como el lector puede deducir del tamaño de bloque utilizado típicamente en los algoritmos de cifrado, 128 bits, probar todas las combinaciones posibles hasta obtener un texto cifrado similar al deseado para recuperar la información en claro no es posible. Por este motivo, el ataque se centra en recuperar de manera inteligente una información pequeña, pero de gran impacto en seguridad, como puede ser una cookie de sesión. Adivinar, por ejemplo, 16 octetos de una clave de sesión aleatoria no es factible directamente, pero analizando el formato de las cabeceras, las cookies y jugando con ciertos umbrales es posible recuperar información poco a poco mediante prueba y error. Por ejemplo, supongamos una cookie cifrada cuyo texto en claro es *Session_Token:Edx38NesewqeNI872Def32sfe*, desplazando los bloques es posible ajustarlos de forma que el bloque en claro a sugerir sea de la forma ``_Session_Token:?" donde ? es el primer octeto a adivinar (256 posibilidades). Una vez los textos cifrados coincidan el atacante conocerá el octeto en claro verdadero, en este caso la E, y jugará de nuevo con los límites de los bloques para intentar adivinar el segundo carácter ``Session_Token:E?", y así sucesivamente.

La prevención contra este ataque reside en la actualización de la versión de TLS y la recomendación de no utilizar modos de cifrado CBC, y sí otros, por ejemplo, RC4. Cómo se verá en ataques documentados posteriormente, esta recomendación, a su vez, introdujo nuevos vectores de ataque.

Más información en:

eko7 - 2011 - BEAST: Surprising crypto attack against HTTPS - Thai Duong & Juliano Rizzo
https://m.youtube.com/watch?v=-BjpkHCeqU0

BEAST: An Explanation of the CBC Attack on TLS. David Wong
https://www.cryptologie.net/article/413/beast-an-explanation-of-the-cbc-attack-on-tls/
http://antoanthongtin.vn/Portals/0/TempUpload/pProceedings/2014/9/26/tetcon2012_juliano_beast.pdf

A diversion: BEAST Attack on TLS/SSL Encryption
https://blog.cryptographyengineering.com/2011/09/21/brief-diversion-beast-attack-on-tlsssl/
file:///C:/Users/anonymous/Downloads/ssl_attacks_survey%20(3).pdf

2.1 CRIME, TIME, BREACH y HEIST. Comprensión de datos y fuga de datos basado en tamaño.

En los últimos años se han publicado muy buenos ejemplos de cómo poder esquivar la criptografía, obtener la información en claro, sin necesidad de conocer la clave criptográfica utilizada o de anular los algoritmos criptográficos. Además, son un excelente ejemplo de porqué utilizar e implementar criptografía en el mundo real es difícil especialmente, en estos ejemplos concretos, por su interrelación con otros protocolos o algoritmos ajenos a la criptografía. Los ataques cubiertos en este apartado iniciaron un buen conjunto de ejemplos para demostrar que la afirmación anterior era cierta.

Compression Ratio Info-leak Made Easy (CRIME), CVE-2012-4929, es un ataque sobre el protocolo SSL/TLS que fue desarrollado por los investigadores Juliano Rizzo y Thai Duong. CRIME conceptualmente es un ataque de canal lateral que permite descubrir cookies/tokens de sesión u cualquier otra información secreta (típicamente pequeña para que el ataque sea realista) consultando el tamaño comprimido de la información enviada en las peticiones HTTP. Este ataque tiene especial relevancia en sesiones web protegidas por SSL/TLS al utilizar dos esquemas de compresión específicos (DEFLATE[29] y gzip) de utilidad para reducir la congestión en la red y acelerar el tiempo de carga de las páginas web. Rizzo y Doung publicaron los detalles en la conferencia de seguridad Ekoparty en septiembre 2012, impactando en la mayoría de software comercial, incluido Mozilla y Google.

De manera simplificada CRIME aprovecha vulnerabilidades que utilizan ataques basados en texto en claro elegido y una ingeniosa fuga de información basada en los algoritmos de compresión para obtener información en claro (teóricamente cifrada). CRIME saca ventaja de la compresión SSL/TLS y SPDY[30] aunque muchos otros protocolos cifrados y comprimidos podrían adolecer del mismo problema. Esta idea no es nueva pero su aplicación práctica fue significativa. De hecho, estos ataques están basados en principios ya descubiertos años atrás por, al menos, el criptógrafo John Kelsey en su trabajo *Compression and Information Leakage of Plaintext*. (https://www.iacr.org/cryptodb/archive/2002/FSE/3091/3091.pdf)

En cualquier caso, el ataque es aparentemente sencillo, el atacante observa el tamaño de la información cifrada enviada por el navegador web mientras al mismo tiempo fuerza al navegador web a realizar peticiones específicas al mismo servidor destino. El atacante fuerza a enviar una información junto con la cookie de sesión, por ejemplo. Si el tamaño de la información cifrada por el canal aumenta con una alta probabilidad la información insertada será diferente a la presente en la cookie de sesión, por el contrario, si el tamaño de la información cifrada se reduce (a mayor coincidencia la ratio de compresión es mayor) entonces es probable que la información inyectada esté presente en la cookie de sesión. Con esta idea y a modo de oráculo, un atacante puede ir haciendo múltiples sugerencias hasta conocer el texto en claro de la cookie de sesión y, por tanto, recuperar la información aparentemente protegida[31] [32] [33] [34]. Para que este ataque sea posible se tienen que dar una serie de condiciones: la compresión SSL/TLS se utilice en ambos extremos[35], el atacante pueda ejecutar código JavaScript en la víctima y poder tener acceso en modo captura de paquetes (sniffer)

---

[29] Transport Layer Security Protocol Compression Methods - https://tools.ietf.org/html/rfc3749

[30] SPDY es un protocolo de transporte desarrollado por Google similar a HTTP, pero optimizado para reducir la latencia en el consumo de contenido web, así como la seguridad de los mismos. Ejemplos del ataque CRIME en SPDY puede verse en: https://www.imperialviolet.org/2012/09/21/crime.html.

[31] https://www.ekoparty.org/archive/2012/CRIME_ekoparty2012.pdf

[32] https://www.nccgroup.trust/us/about-us/newsroom-and-events/blog/2012/september/details-on-the-crime-attack/

[33] https://en.wikipedia.org/wiki/CRIME

[34] https://www.nccgroup.trust/globalassets/our-research/us/whitepapers/ssl_attacks_survey.pdf

[35] En el mensaje *ClientHello*, el cliente establece la lista de algoritmos de compresión que conoce, y el servidor responde, en el *ServerHello*, con el algoritmo de compresión que se utilizará. Es posible definir como método compresión nula.

de la sesión cifrada entre cliente-servidor (lo cual puede ser común mediante un ataque de MiTM).

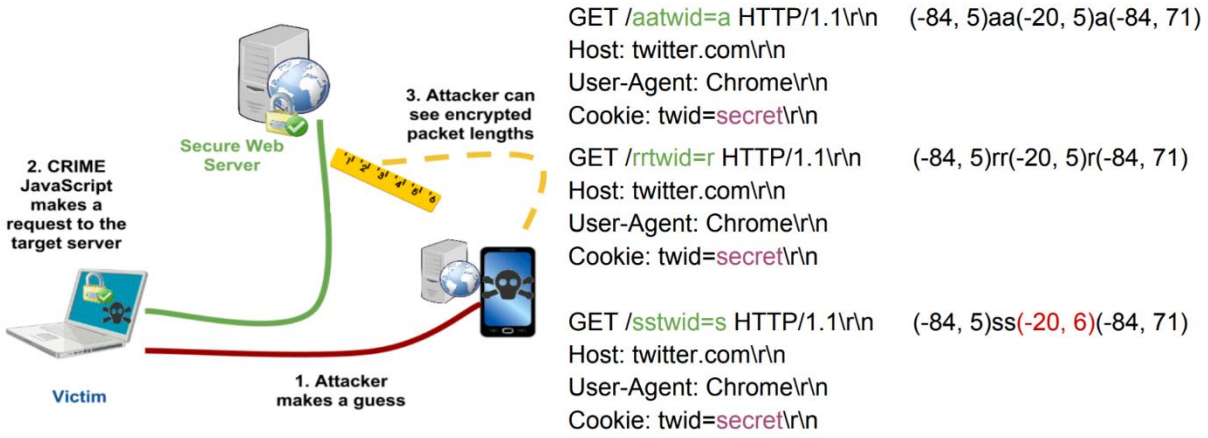


Figura 4. Ejemplo de ataque CRIME

Es significativo destacar como el uso del algoritmo RC4[36] en lugar de cifrados de bloques como CBC ayuda a progresar este ataque. Curiosamente soluciones temporales recomendadas frente a ataques previos como BEAST. La solución más sencilla a este problema consiste en deshabilitar la compresión TLS y SDPY, si aplica.

Una vez solucionado este ataque surgieron variantes[37] [38] del mismo buscando el mismo objetivo pero solucionando algunas de las limitaciones del ataque CRIME. Un ejemplo destacable es el ataque **TIME (Timing Info-leak Made Easy)** desarrollado por Tal Be'ery y Amichai Shulman, en un ataque de texto en claro elegido sobre las respuestas HTTP (CRIME lo hace sobre las peticiones) analizando diferencias temporales para deducir el tamaño[39]. Pero sin duda el ataque que más repercusión mediática tuvo fue **BREACH (Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext)**, con un modo de funcionamiento muy similar a CRIME pero centrado en las respuestas HTTP y el efecto de la compresión del protocolo HTTP (en lugar del de SSL/TLS ya solventado para mitigar CRIME) en el cuerpo de la respuesta[40] [41]. La ejecución de este ataque tiene una serie de restricciones: el servidor a atacar debe usar compresión HTTP e información relacionada con las entradas de usuario, así el secreto a desvelar, por ejemplo, relativo a un CSRF token, se debe mostrar en la respuesta HTTP. Es importante destacar que BREACH es independiente de la versión de TLS/SSL y funciona independientemente del algoritmo criptográfico utilizado. Aunque es cierto que deshabilitar la compresión HTTP impide el ataque tiene un coste en perdida de rendimiento que tiene que ser contemplado. Otras contramedidas se han propuesto, pero sin ser definitivas, entre ellas aleatorizar el tamaño de la respuesta[37].

[36] https://www.imperva.com/docs/HII_Attacking_SSL_when_using_RC4.pdf
[37] HEIST - https://www.blackhat.com/docs/us-16/materials/us-16-VanGoethem-HEIST-HTTP-Encrypted-Information-Can-Be-Stolen-Through-TCP-Windows-wp.pdf
[38] eko13 - 2017 - Jose Selvi - FIESTA: an HTTPS side-channel party https://www.youtube.com/watch?v= guZ30nXHkUg
[39] https://media.blackhat.com/eu-13/briefings/Beery/bh-eu-13-a-perfect-crime-beery-wp.pdf
[40] https://media.blackhat.com/us-13/US-13-Prado-SSL-Gone-in-30-seconds-A-BREACH-beyond-CRIME-Slides.pdf
[41] http://breachattack.com/

2.2 Lucky13

Los investigadores Nadhem J. AlFardan y Kenneth G. Paterson publicaron en 2013 la investigación "Lucky Thirteen: Breaking the TLS and DTLS Record Protocols"[42] demostrando como era posible, al menos teóricamente, recuperar texto en claro de diferentes versiones de los protocolos TLS y DTLS (Datagram TLS) mediante ataques temporales de canal lateral al proceso de descifrado de la información, apoyándose en la modificación a medida del padding generado, usando para ello los principios clásicos de ataque basado en padding oracle[43].

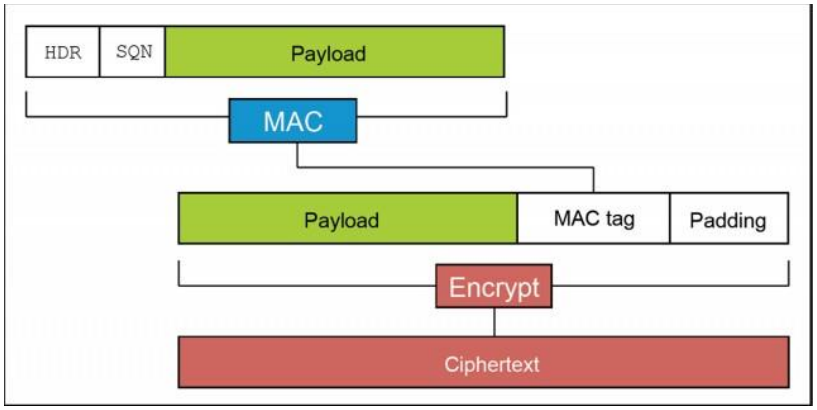


Figura 5. Proceso de cifrado D(TLS)

Estos ataques aplican a las implementaciones que utilizan como modo de cifrado CBC, al menos TLS 1.1 y 1.2 y DTLS 1.0 y 1.1, sin necesitar la capacidad de inyectar texto en claro elegido en la víctima, a diferencia del ataque BEAST, con lo que un atacante podrá progresar el ataque simplemente si tiene la capacidad de interceptar y manipular la comunicación cifrada, mediante un ataque de hombre en el medio (MiTM). El atacante solo necesita poder capturar los bloques de información cifrada e inyectar a su antojo, cambiando el orden, bloques de información cifrada. Observé el lector que los bloques que contienen la información de padding no está protegida por el algoritmo de MAC.

El atacante, por tanto, puede reemplazar los bloques con información de relleno con otros bloques cifrado elegidos por él, de la comunicación, y observar la cantidad de tiempo que necesita el servidor para responder. Su investigación demuestra que para un mensaje TLS que contiene un padding incorrecto el servidor tardará menos tiempo en responder, no se calcula el MAC, que uno que es correcto. Este hecho permite, al menos teóricamente, crear un oráculo al que se le pueda sacar ventaja criptoanalítica, obteniendo información en claro mediante el envío de múltiples mensajes en múltiples sesiones, en el caso de TLS. Los investigadores documentaron, en un peor caso, la necesidad de 2^{23} sesiones TLS para conseguir extraer el contenido en claro de una cookie de sesión (autenticación).

El profesor Matthew Green hace una descripción muy acertada de las limitaciones, importantes, de este ataque, especialmente en TLS, y las contramedidas posibles a implementar, muchas de ellas incorporadas en las nuevas versiones de las librerías TLS más difundidas. Se recomienda al lector la lectura de: https://blog.cryptographyengineering.com/2013/02/04/attack-of-week-tls-timing-oracles y evoluciones de este ataque[44].

[42] El número 13 del título del ataque viene inspirado en los 13 bytes de la cabecera de datos que son incorporados en el cálculo del algoritmo MAC.
[43] Padding_oracle_attack - https://en.wikipedia.org/wiki/Padding_oracle_attack
[44] Lucky 13 Strikes Back - http://users.wpi.edu/~teisenbarth/pdf/Lucky13%20AsiaCCS2015.pdf

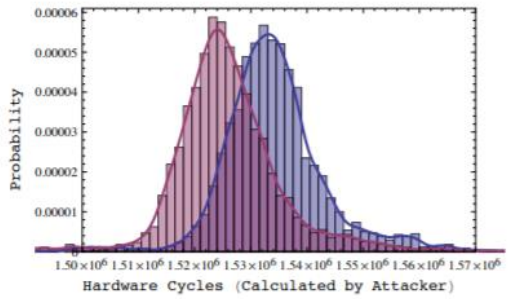


Figure 2: Distribution of timing values (outliers removed) for distinguishing attack on OpenSSL TLS, showing faster processing time in the case of M_0 (in red) compared to M_1 (in blue).

L	Success Probability
1	0.756
2	0.769
4	0.858
8	0.914
16	0.951
32	0.983
64	0.992
128	1

Table 1: OpenSSL TLS distinguishing attack success probabilities.

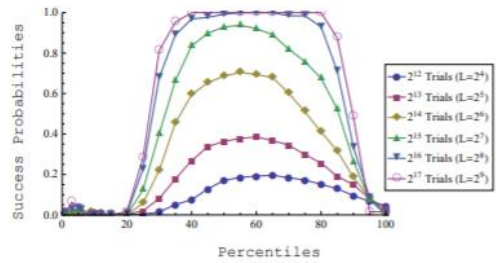


Figure 5: OpenSSL TLS partial plaintext recovery: percentile-based success probabilities for recovering P_{15}^* assuming P_{14}^* known.

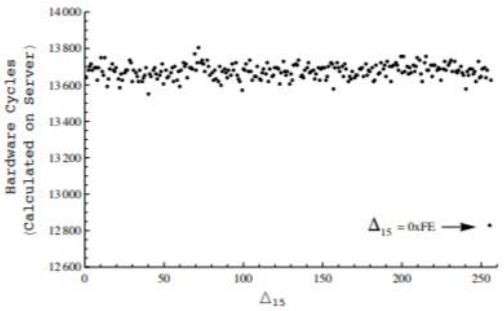


Figure 3: OpenSSL TLS median server timings (in hardware cycles) when $P_{14}^* = 0\text{x}01$ and $P_{15}^* = 0\text{xFF}$. As expected, $\Delta_{15} = 0\text{xFE}$ leads to faster processing time.

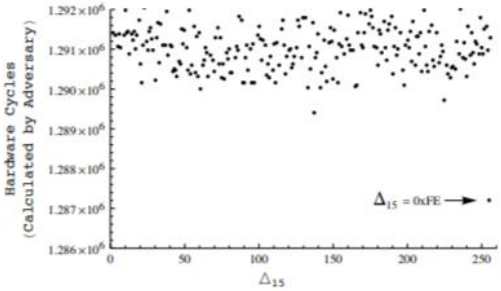


Figure 4: OpenSSL TLS median network timings in terms of hardware cycles when $P_{14}^* = 0\text{x}01$ and $P_{15}^* = 0\text{xFF}$. As expected $\Delta_{15} = 0\text{xFE}$ leads to faster processing time.

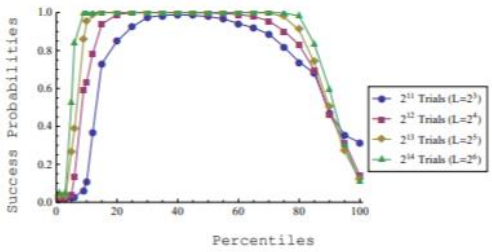


Figure 6: OpenSSL DTLS partial plaintext recovery: percentile-based success probabilities for recovering P_{15}^* with P_{14}^* known, $n = 10$.

Figura 6. Ataque Lucky13 a TLS/DTLS

Más información:

http://www.isg.rhul.ac.uk/tls/TLStiming.pdf
http://www.isg.rhul.ac.uk/tls/Lucky13.html

2.3 Seguridad de RC4 en SSL/TLS. RC4 biases

En 2013, los investigadores *Nadhem AlFardan* et al. publicaron su investigación "On the Security of RC4 in TLS" cubriendo la problemática de utilizar RC4 en conexiones SSL/TLS y las opciones que tendría un atacante para recuperar información en claro, típicamente una cookie de sesión.

Para entender estos ataques sobre RC4 es importante recordar que este algoritmo es un cifrador de flujo inventado en 1987 por Ron Rivest entre cuyas ventajas destaca:

1. No requiere relleno o padding ni IVs, lo cual significa que es inmune a ataques anteriores a TLS via BEAST o Lucky13. El uso de este algoritmo fue una recomendación como solución a estas amenazas.

2. RC4 es rápido. Esto significa menor computación y requisitos hardware, lo que le hizo interesante para grandes proveedores tecnológicos como Google.

El problema con RC4 es que los bytes que salen del proceso de cifrado no son "suficiente" aleatorios, tienen pequeños sesgos/imperfecciones (bias). Algunos de estos sesgos/imperfecciones se conocían desde hace años, pero no se consideraron útiles hasta fechas cercanas a esta publicación. Pensemos un momento en el funcionamiento de este algoritmo. Si se cifra el mismo mensaje (texto en claro) con muchas claves RC4 diferentes, debería obtener un montón de textos cifrados de apariencia totalmente aleatoria. Pero estos pequeños sesgos significan que no serán aleatorios: un análisis estadístico[45] de diferentes posiciones del texto cifrado mostrará que algunos valores ocurren con más frecuencia. Al obtener muchos cifrados diferentes del mismo mensaje, bajo claves diferentes, se puede calcular estas pequeñas desviaciones y, por lo tanto, averiguar qué se cifró.

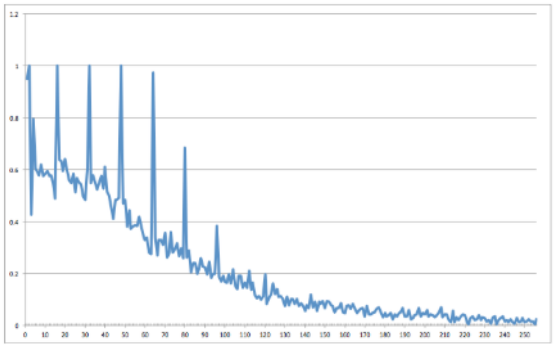
El funcionamiento natural de los protocolos de comunicación, por ejemplo, HTTPS, hace viable que se pueda cifrar un mismo mensaje muchas veces. Por ejemplo, la cookie de sesión que se envía a un proveedor, GMAIL, por ejemplo, para acceder. En este ejemplo, si la conexión está cifrada con RC4 cada vez que se genera una nueva conexión se está enviando una nueva copia cifrada de la misma cookie, dado que está usando claves diferentes para el establecimiento de la conexión cifrada. Esta particularidad permite al atacante construir la lista de textos cifrados que necesita. Habitualmente para esta tarea el atacante inyectará código JavaScript en el navegador de la víctima que hará peticiones concurrentes lo que garantiza al atacante construir "rápidamente" miles o millones de peticiones a analizar.

Aunque este ataque es conceptualmente interesante y útil para futuras propuestas es cierto que necesita generar un número elevado de conexiones[41] para recuperar una cantidad limitada de texto en claro. Esta característica minimizó en parte la importancia de este ataque en usuarios finales que utilizaban TLS.

Algunas de las contramedidas propuestas para solventar el fallo fueron: conmutar/volver a modos cifrado con CBC, utilizar modos de cifrado autenticado (AEAD) como AES-GCM, corregir el comportamiento de RC4 o modificar el comportamiento de los navegadores para dificultar la tarea de recopilación de sesiones al atacante.

[45] Existe muy buenos trabajos para comprender como aprovecharse de esas anomalías estadísticas para progresar estos ataques. Una buena referencia es el trabajo de los mismos autores titulado Biases in the RC4 keystream (128 bit uniform keys) - http://www.isg.rhul.ac.uk/tls/biases.pdf. Los ataques documentados permitirán descifrar más o menos información utilizando un número grande de conexiones/sesiones TLS. Por ejemplo, para recuperar 220 bytes en claro de una comunicación cifrada se necesitaría del orden de 2^{30} conexiones/sesiones TLS.

Tiempo después el ataque Bar Mitzvah demostraría la importancia, de nuevo, de prescindir de RC4[46] [47].



Probability of recovering a plaintext byte (y axis) at a particular byte position in the RC4-encrypted stream (x axis), assuming 2^24 ciphertexts. Increasing the number of ciphertexts to 2^32 gives almost guaranteed plaintext recovery at all positions. (source)

Figura 7. Probabilidad de recuperación de un byte en claro de una determinada posición en el flujo cifrado RC4

Más información en:

http://www.isg.rhul.ac.uk/tls/biases.pdf
https://blog.cryptographyengineering.com/2013/03/12/attack-of-week-rc4-is-kind-of-broken-in/
http://www.isg.rhul.ac.uk/tls/
https://www.usenix.org/system/files/conference/usenixsecurity13/sec13-paper_alfardan.pdf

[46] SSL/TLS suffers Bar Mitzvah Attack - https://www.darkreading.com/attacks-breaches/ssl-tls-suffers-bar-mitzvah-attack-/d/d-id/1319633? - https://en.wikipedia.org/wiki/Bar_mitzvah_attack
[47] Attacking SSL when using RC4 - http://www.imperva.com/docs/HII_Attacking_SSL_when_using_RC4.pdf

2.4 Triple Handshake Attack. 3SHAKE

Los investigadores Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cedric Fournet, Alfredo Pironti y Pierre-Yves Strub publicaron en 2014 su investigación "*Triple Handshakes and Cookie Cutters: Breaking and Fixing Authentication over TLS*" donde demuestran la posibilidad de nuevos ataques de suplantación a través de renegociaciones[48] de sesión en el protocolo TLS.

La idea de este ataque es conceptualmente sencilla. Un cliente TLS intenta conectarse a un servidor legítimo, pero se conecta a un servidor malicioso intermedio que mediante un ataque de hombre en el medio es capaz de procesar tres intentos de establecimiento de conexión (handshakes sucesivos) entre el cliente y el servidor legítimo consiguiendo suplantar al cliente en el tercer intento de conexión (handshake). Un ataque interesante, aunque sin mucho impacto en la práctica.

En la documentación adjunta tiene descrito los ataques en detalle. A modo de ejemplo, se ejemplificará un ataque en el cual un cliente C desea conectarse a un servidor S que utiliza intercambio de claves con el algoritmo RSA y permite renegociación y reanudación de sesión. En primer lugar, véase el diagrama de flujo adjunto, el cliente C (TLS) se conecta a un servidor A malicioso pensando que se conecta al servidor legítimo. El servidor malicioso (A) se conecta al servidor S usando el mismo identificador cr que el cliente y devuelve al cliente C legítimo los valores sr y SID, adicionalmente A fuerza a C y S usar el intercambio de claves con RSA permitiendo solo la negociación de los algoritmos criptográficos correspondientes. A continuación, A coge el valor PMS cifrado (Pre-Master Secret) y lo vuelve a cifrar para S. Con este escenario, completa dos handshakes para obtener una nueva sesión en cada conexión. Estas dos sesiones comparten la misma clave y parámetros (sid, ms, cr, sr) pero tienen diferentes certificados de servidor y mensajes de finalización (*verify_data*).

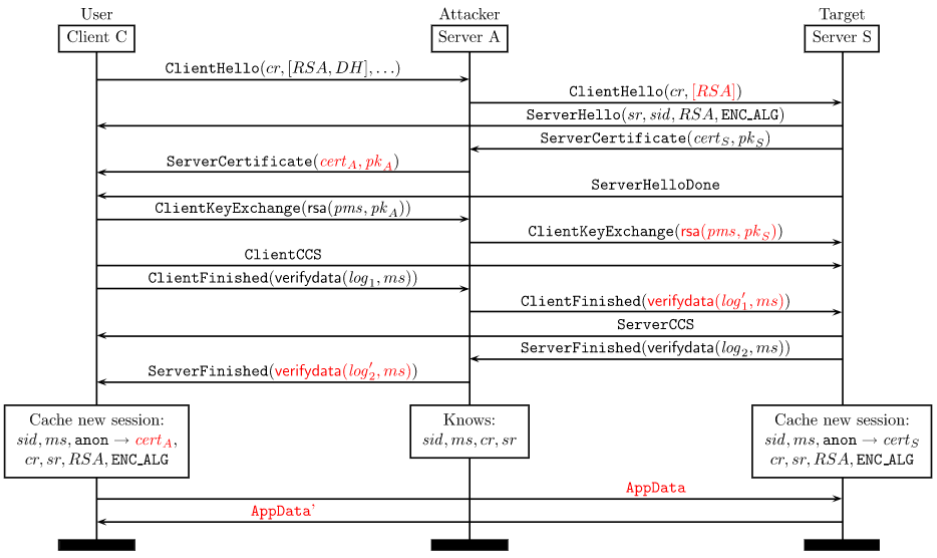


Figura 8. Ejemplo de ataque 3SHAKE - Parte I

En segundo lugar, C reconecta con A y le pide que reanude su sesión anterior. A, a su vez, vuelve a conectarse a S y reanuda también su sesión anterior. Dado que todos los parámetros relevantes en las dos sesiones son los mismos, A de hecho puede simplemente reenviar los mensajes de handshake sin cambios entre C y S. Al completar este proceso, las dos conexiones nuevamente tienen las mismas

[48] Este ataque puede recordar al menos conceptualmente, aunque con mejoras, a propuestas previas como las recogidas en http://www.educatedguesswork.org/2009/11/understanding_the_tls_renegoti.html o https://mailarchive.ietf.org/arch/msg/tls/Y1O3HUcq9T94rMLCGPTTozURtSI pero evadiendo las protecciones introducidas en la RFC5746 (Transport Layer Security –TLS- Renegotiation Indication Extension)

claves, pero ahora también tienen los mismos mensajes de finalización (verify_data). A conoce las nuevas claves de conexión, por lo que puede continuar enviando datos en cualquier conexión hacia C o S. En este punto, C todavía cree que tiene una conexión con A, y S todavía cree que tiene una conexión con algún cliente anónimo, por lo que ninguno de ellos ha sido suplantado todavía. Sin embargo, dado que el cliente y el servidor, verify_data, ahora son los mismos en ambas conexiones, estas conexiones tendrán los mismos valores.

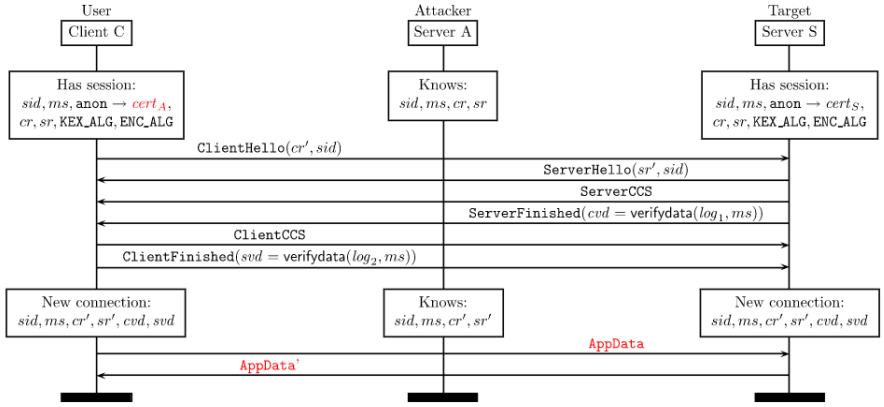


Figura 9. Ejemplo de ataque 3SHAKE - Parte II

En tercer lugar, S exige la renegociación con la autenticación del cliente en su conexión con A, posiblemente en respuesta a la solicitud de A de algún recurso restringido. Luego, A reenvía la solicitud de renegociación a C, y C acuerda autenticarse con su certificado de cliente en A. Al final de la renegociación, A ya no conoce las claves de conexión ni el secreto maestro; solo son conocidos por C y S. Por consiguiente, A ya no puede leer ni enviar mensajes en estas conexiones.

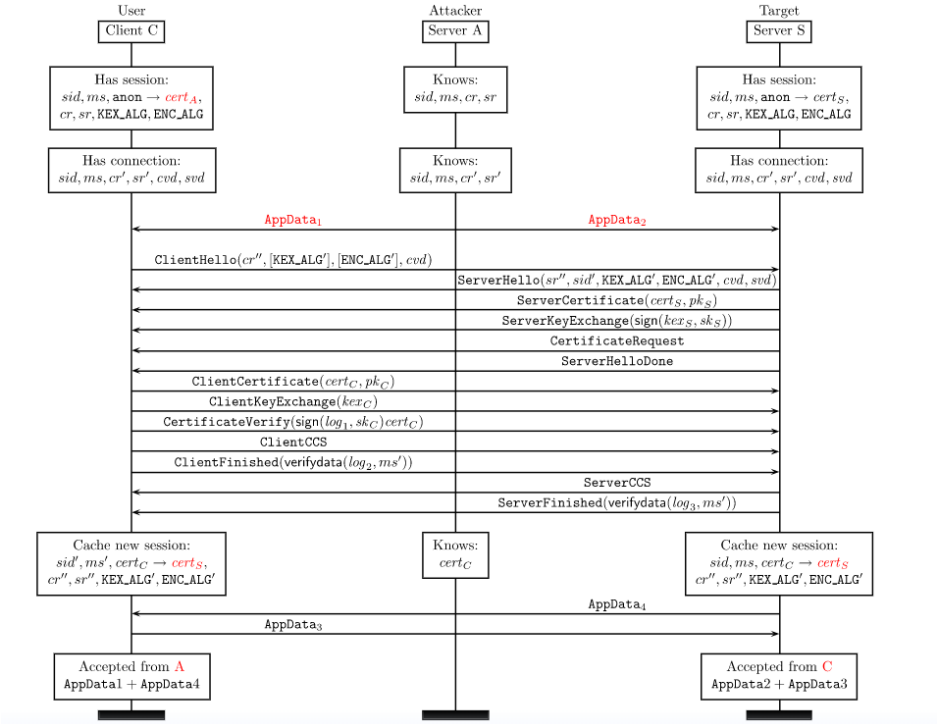


Figura 10. Ejemplo de ataque 3SHAKE - Parte III

Más información puede encontrarse en:

https://mitls.org/downloads/tlsauth.pdf - https://mitls.org/pages/attacks/3SHAKE
https://blog.cryptographyengineering.com/2014/04/24/attack-of-week-triple-handshakes-3shake/

2.5 Heartbleed

Heartbleed hace referencia a un fallo de seguridad (CVE-2014-0160), descubierto en 2014, introducido en la librería criptográfica de OpenSSL (1.0.1 a 1.0.1f). OpenSSL es la librería más popular de opensource criptográfica y es la utilizada por SSL/TLS para cifrar los datos en las comunicaciones en Internet, proteger servidores de correo, servidores de chat (protocolo XMPP), VPNs, etc. Por tanto, no es un problema de la especificación del protocolo SSL/TLS como tal, pero sí un problema de implementación de una librería que le da soporte en aplicaciones, sistemas y servidores.

El fallo viene debido a una incorrecta validación de los parámetros de entrada en la implementación de la extensión Hearbeat (RFC 6520) del protocolo TLS. La idea del ataque es sencilla. Un atacante construye un mensaje *Hearbeat request* con una pequeña información (payload) pero introduce en el campo reservado para indicar cuál es el tamaño de la información enviada un valor muy grande. El receptor del mensaje, en las versiones de OpenSSL afectadas, no verifica correctamente que el tamaño indicado corresponda realmente con el tamaño de la información enviada (bounds checking) y en el mensaje de vuelta, como consecuencia de esa incorrecta validación, se produce un desbordamiento de la memoria reservada y se consigue enviar información adicional. Con este procedimiento el atacante puede recuperar hasta 64Kilobytes de la memoria de la víctima, repitiendo este proceso de forma reiterada. En función de lo que se encuentre en ella el fallo permitiría obtener de los sistemas mensajes en claro que se consideran confidenciales (cookies de sesión, passwords, etc.), claves privadas (lo que facilitaría un potencial descifrado de información), etc.

La contramedida eficaz a este ataque consiste en la utilización de versiones más modernas de la librería que las afectadas y la actualización/renovación/revocación de información privada que pudiera haber sido accedida. Por ejemplo, revocación de certificados digitales (su clave privada), etc.

Más información en:

http://heartbleed.com
https://en.wikipedia.org/wiki/Heartbleed

2.6 New Bleichenbacher side channels and attacks

Los investigadores Christopher Meyer et al. publicaron en 2014 un interesante ataque titulado: *Revisiting SSL/TLS Implementations: News Bleichenbacher Side Channels and Attacks (https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/meyer)*, en el cual aplican el ataque de Bleichenbacher[49] (*adaptive chosen-ciphertext attack*[50]) documentado en 1998 a recientes implementaciones de TLS que utilizan RSA PKCS#1 v1.5, en lugar de RSA-OAEP. El objetivo del ataque descifrar una información cifrada sin conocimiento de la clave (en general no debería poderse obtener la clave privada).

Este ataque es similar al original de Bleichenbacher, se parte de un texto cifrado y desde ese se van generando variaciones. El atacante envía esos nuevos textos cifrados (ciphertext) al receptor, típicamente el servidor, que actúa como un oráculo y observa su respuesta sobre la validez o no del mensaje que ha descifrado. Con cada respuesta el atacante, mediante fuga de información, podría llegar a deducir alguna parte de la información en claro. Si esto tiene éxito, repetirá los pasos hasta conseguir descifrar el mensaje cifrado original de partida. Por tanto, la clave aquí es si el receptor, el servidor, actúa o no como oráculo desde el punto de vista del atacante.

En los últimos años diferentes contramedidas se han propuesto frente a los ataques de tipo Bleichenbacher (RFC 2246 o RFC5246) y coinciden en una cosa: tratar los mensajes formateados de forma incorrecta en una manera indistinguible a los bloques RSA formateados correctamente.

El ataque de Christopher Meyer et al. se apoya en el ataque Bleichenbacher original, en principio no se podría aplicar con las contramedidas implantadas, pero demuestran que en ciertas implementaciones de TLS actuales es posible convertir al receptor, el servidor, en un oráculo válido y por tanto aplicar el ataque de hace más de 20 años. Su prueba de concepto se basa en ataques temporales de canal lateral (*timing side channels*). El servidor tarda tiempo diferente en procesar mensajes formateados de diferente forma, y esto permite la fuga de información convirtiendo al servidor en un oráculo. Entre sus hallazgos se encuentran fallos de seguridad en la librería JSSE (Java Secure Socket Extension) o contra dispositivos basados en seguridad hardware que utilizan Cavium NITROX SSL accelerator chip. Por ejemplo, en el caso de la librería JSSE después de enviar diferentes mensajes formateados en PKCS#1 JSSE responde con diferentes errores (INTERNAL_ERROR O HANDSHAKE_FAILURE).

Este ataque recuerda una recomendación muy importante. Las implementaciones criptográficas deben tener mucho cuidado para usar funciones constantes en el tiempo. De no ser así, pequeñas fugas por canal lateral podrían producir resultados catastróficos.

Más información en:

https://web-in-security.blogspot.com/2014/08/old-attacks-on-new-tls-implementations.html
https://www.usenix.org/system/files/conference/usenixsecurity14/sec14-paper-meyer.pdf

[49] http://archiv.infsec.ethz.ch/education/fs08/secsem/Bleichenbacher98.pdf
[50] https://en.wikipedia.org/wiki/Adaptive_chosen-ciphertext_attack

2.7 POODLE. Padding Oracle On Downgraded Legacy Encryption

POODLE (Padding Oracle On Downgraded Legacy Encryption), CVE-2014-3566, es un ataque, descubierto en octubre de 2014 por Bodo Möller, Thai Duong y Krzysztof Kotowicz (Google Security Team), que se aprovecha de la compatibilidad que los clientes TLS tienen con versiones previas del protocolo, en este caso mediante un ataque downgrade a SSL 3.0 (*SSL 3.0 fallback*). El atacante necesita para progresar con el ataque que el atacado ejecute SSL 3.0 (o se pueda forzar a que lo utilice), que el cliente ejecute un pequeño código (por ejemplo, un código JavaScript en una página web que se ejecuta en su navegador) y tener la capacidad de interceptar el tráfico cifrado (ataque de MiTM). Con estas circunstancias, el atacante podría en media revelar un octeto de un bloque de información cifrado realizando 256 peticiones (SSL 3.0), a continuación, se entenderá por qué esta cifra. En la práctica, este ataque permitirá robar cookies de sesión o tokens sin necesidad de conocer la clave criptográfica.

El problema real de este ataque reside en el uso que hace el protocolo SSL3.0 del modo de cifrado CBC[51] y el error al no proteger adecuadamente los bloques de relleno (padding) mediante el código MAC (Message Authentication Code), lo que permite a un atacante sacar ventaja de ello[52].

Si en una comunicación existe un bloque de relleno, entero o no, el último octeto indicará cuantos octetos se han utilizado de relleno (esto podría cambiar en función del estándar de relleno utilizado). Por ejemplo, en un bloque de 8 octetos si ese bloque fuera de relleno el último octeto tendría un valor 7, indicando que los 7 octetos anteriores son de relleno. Entendiendo esto y cómo funciona el cifrado en bloque CBC y el MAC, un atacante puede progresar un ataque basado en oráculo. Veamos a continuación el funcionamiento.

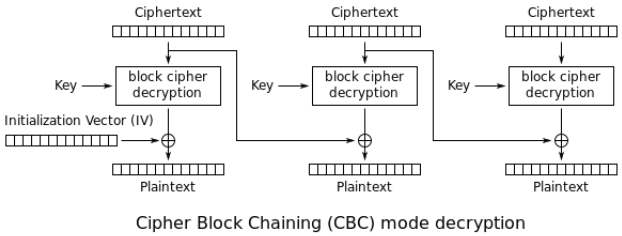


Figura 11. Modo de descifrado CBC (Cipher Block Chaining)

En una información cifrada el último bloque está destinado a relleno, entero o no. El atacante puede introducir bloques de tal forma que si el formato es adecuado el receptor lo aceptará (validará el MAC de la información en claro) y si no lo descartará, esto facilitará un ataque de oráculo (saber si lo que inyecto es correcto o no). En un bloque de relleno completo, por ejemplo, sabemos que el último octeto debería ser un valor 15 (en la posición 16), supongamos un criptograma de 16 octetos, lo típico para un cifrado simétrico de bloque 128 bits, indicando que el resto de octetos del bloque son de relleno. Conocido el valor 15, el criptograma anterior y el flujo en un modo de cifrado CBC, puede conocerse el octeto de la posición 15 de la información en claro del último criptograma. Aparentemente esto no sería de utilidad dado que el último criptograma hace referencia a información de relleno, pero es precisamente en este punto donde el atacante obtiene ventaja. Dado que el atacante puede realizar un ataque de hombre en el medio puede insertar como último bloque algún criptograma previo. Si el criptograma elegido da la casualidad que verifica el formato de relleno, recuerde el último

[51] El mecanismo de relleno (padding) intenta completar la información a cifrar para que sea de un tamaño múltiplo al tamaño de bloque de cifrado.

[52] En 2013, Lucky13 es otro buen ejemplo de ataque al modo de relleno en un modo de operación CBC, en este caso facilitándolo con ataques temporales. Más información en: Lucky Thirteen: Breaking the TLS and DTLS Record Protocols - http://www.isg.rhul.ac.uk/tls/TLStiming.pdf

octeto es el que indica el tamaño, entonces en esa situación (se da en media 1 de cada 256 veces) el atacante puede recuperar un octeto de la información en claro.

Este ataque lo repetirá sucesivamente para recuperar más octetos hasta intentar conseguir la información en claro deseada. Para impedir este ataque se puede proceder a deshabilitar soporte a SSL 3.0 o los cifradores en modo CBC, aunque en general, es recomendable la configuración de TLS_FALLBACK_SCSV para impedir ataques basados en downgrade. Recientemente han surgido variantes de este ataque conocidas como Zombie Poodle y Golden Poodle[53] resaltando de nuevo los riesgos del uso de CBC.

Más información en:

https://en.wikipedia.org/wiki/POODLE
https://security.googleblog.com/2014/10/this-poodle-bites-exploiting-ssl-30.html
https://www.openssl.org/~bodo/ssl-poodle.pdf
https://github.com/mpgn/poodle-PoC
https://blog.cryptographyengineering.com/2014/10/15/attack-of-week-poodle/

[53] Zombie POODLE, GOLDENDOODLE, & How TLSv1.3 Can Save Us All - https://i.blackhat.com/asia-19/Fri-March-29/bh-asia-Young-Zombie-Poodle-Goldendoodle-and-How-TLSv13-Can-Save-Us-All.pdf

2.8 SMACK y FREAK. State machine attacks on TLS

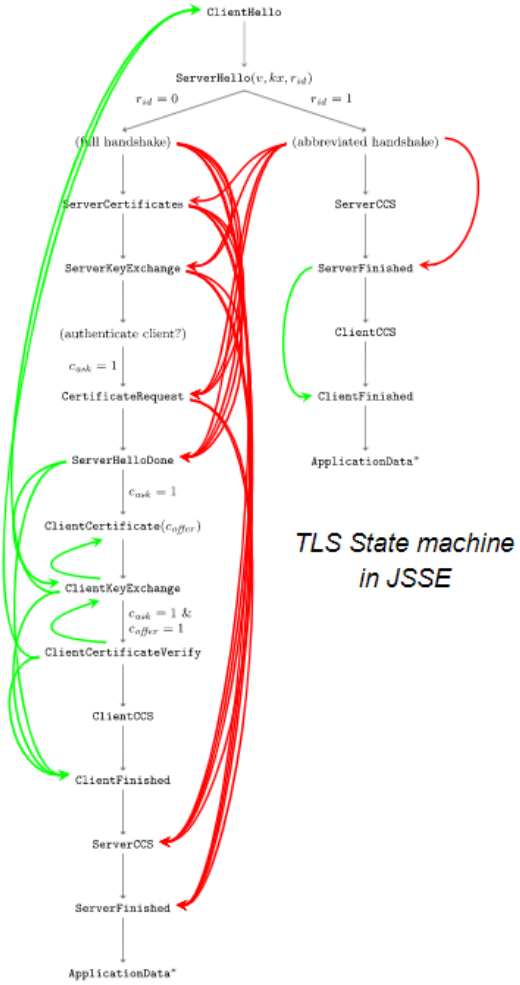
Las implementaciones del protocolo TLS manejan múltiples versiones del protocolo, extensiones, modelos de autenticación, métodos de intercambio de claves, etc. En cada combinación existen diferentes secuencias de mensajes intercambiados entre cliente y servidor. Los ataques que se cubren en este apartado profundizan en el problema (la dificultad) de diseñar complejas máquinas de estados para trabajar con los diferentes flujos de mensajes.

En 2015, los investigadores Benjamin Beurdouche et al. publicaron su trabajo *A Messy State of the Union: Taming the Composite State Machines of TLS* donde recopilan una colección de vulnerabilidades detectadas al analizar implementaciones TLS muy conocidas (opensource). Esta colección de ataques se denomina genéricamente como SMACK: State Machine AttaCKs on TLS. Para progresar los diferentes ataques se asume la capacidad del atacante para realizar ataques de MiTM que le permita manipular los mensajes TLS handshake, típicamente manipulando el sistema DNS (Domain Name System)[54]

Algunos ataques documentados son:

a) SKIP-TLS: Message Skipping Attacks on TLS

Diferentes conjuntos de algoritmos criptográficos utilizan diferentes secuencias de mensajes. En los ataques descubiertos, las implementaciones de TLS permiten incorrectamente suprimir algunos mensajes incluso aunque sean requeridos para seleccionar un conjunto de algoritmos dado. La explicación es sencilla: las librerías intentan reutilizar tanto código como sea posible entre los diferentes conjuntos de algoritmos. Sin embargo, estas asunciones son peligrosas. Por ejemplo, los clientes JSSE permiten a sus extremos suprimir todos los mensajes relacionados con intercambio de clave o autenticación. En concreto, un atacante en la red puede enviar el certificado de cualquier servidor web, y suprimir el resto de los mensajes del protocolo. Un cliente JSSE vulnerable aceptaría el certificado y comenzaría a descifrar los datos de aplicación. En otras palabras, este ataque demostraba que las implementaciones JSSE de TLS no proporcionaban ningún tipo de seguridad en los últimos años. Algún ejemplo del impacto se reflejó en **JSSE** (CVE-2014-6593) o **OpenSSL** (CVE-2015-0205).



[54] Dns rebinding (https://en.wikipedia.org/wiki/DNS_rebinding) o Domain name seizure (https://en.wikipedia.org/wiki/Domain_name#Seizures)

b) FREAK: Factoring RSA Export Keys

El ataque FREAK permite forzar el uso de algoritmos criptográficos débiles (*export cipher suites*) que están presentes en determinadas implementaciones y en muchos casos aparentemente deshabilitados por defecto utilizando el flujo de mensajes negociados, para lo que un atacante necesita poder realizar un ataque de hombre en el medio para forzar la selección de esos algoritmos criptográficos. Por ejemplo, se podría forzar al uso de un algoritmo RSA de 512 bits (fácilmente factorizable). Librerías como OpenSSL (CVE-2015-0204) o BoringSSL se vieron afectadas, entre otras.

Más información en:

https://mitls.org/pages/attacks/SMACK
http://www.ieee-security.org/TC/SP2015/papers-archived/6949a535.pdf
https://en.wikipedia.org/wiki/FREAK
https://blog.cryptographyengineering.com/2015/03/03/attack-of-week-freak-or-factoring-nsa/

2.9 Logjam. Degradando el protocolo TLS a algoritmo criptográfico débil.

Logjam (CVE-2015-4000) es un ataque que permite a un atacante con capacidad de interceptar/manipular el tráfico de red (MiTM) degradar el protocolo TLS para utilizar un intercambio de clave débil basado en Diffie-Hellman. El intercambio de claves con Diffie-Hellman permite a protocolos en Internet negociar una clave compartida para facilitar una conexión segura. Este esquema es fundamental en HTTPS, SSH, IPsec, SMTPS, y protocolos basados en TLS.

La degradación del protocolo TLS permite usar parámetros de 512 bits que facilita descifrar comunicaciones en tiempo real. Este ataque recuerda al anteriormente analizado, FREAK, aunque Logjam se basa en fallos en el protocolo TLS en lugar de vulnerabilidades en la implementación y ataca el intercambio de claves usando Diffie-Hellman, en lugar de con el algoritmo RSA. El ataque afecta a cualquier servidor que soporte *DHE_EXPORT ciphers* y afectó a todos los navegadores web modernos y en el 8,4% del top de 1 millón de dominios más relevantes.

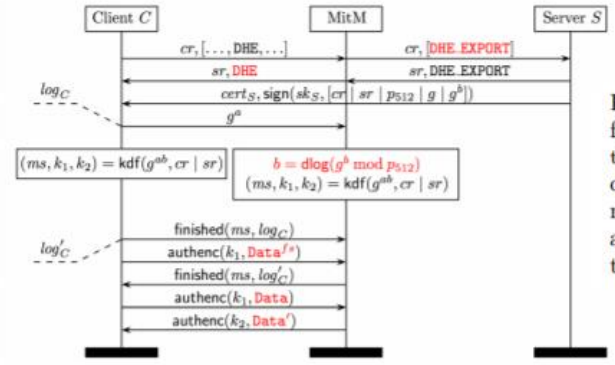


Figure 2: **The Logjam attack.** A man-in-the-middle can force TLS clients to use export-strength DH with any server that allows DHE_EXPORT. Then, by finding the 512-bit discrete log, the attacker can learn the session key and arbitrarily read or modify the contents. Data^{fs} refers to False Start [30] application data that some TLS clients send before receiving the server's Finished message.

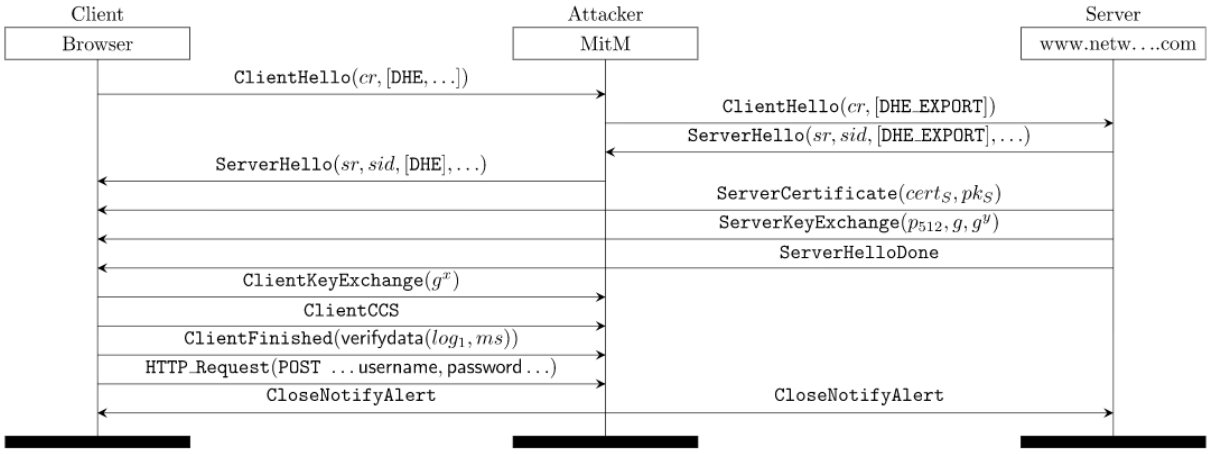


Figura 12. Descripción del ataque Logjam

El ataque reflejado en la imagen anterior (*DHE_EXPORT Downgrade and offline decryption of TLS False Start Attack*) solo requiere que un servidor soporte parámetros de 512-bits (si no está habilitado non-export DHE ciphers) o soporte de *DHE_EXPORT cipher suites*. El cliente debe usar la extensión "TLS_False_Start", es decir, el cliente envía datos de aplicación antes de recibir el mensaje de finalizado por el servidor en el establecimiento de la sesión (handshake TLS). Por último, el atacante necesita realizar un ataque de MiTM para ser capaz de interceptar, alterar y reenviar el tráfico.

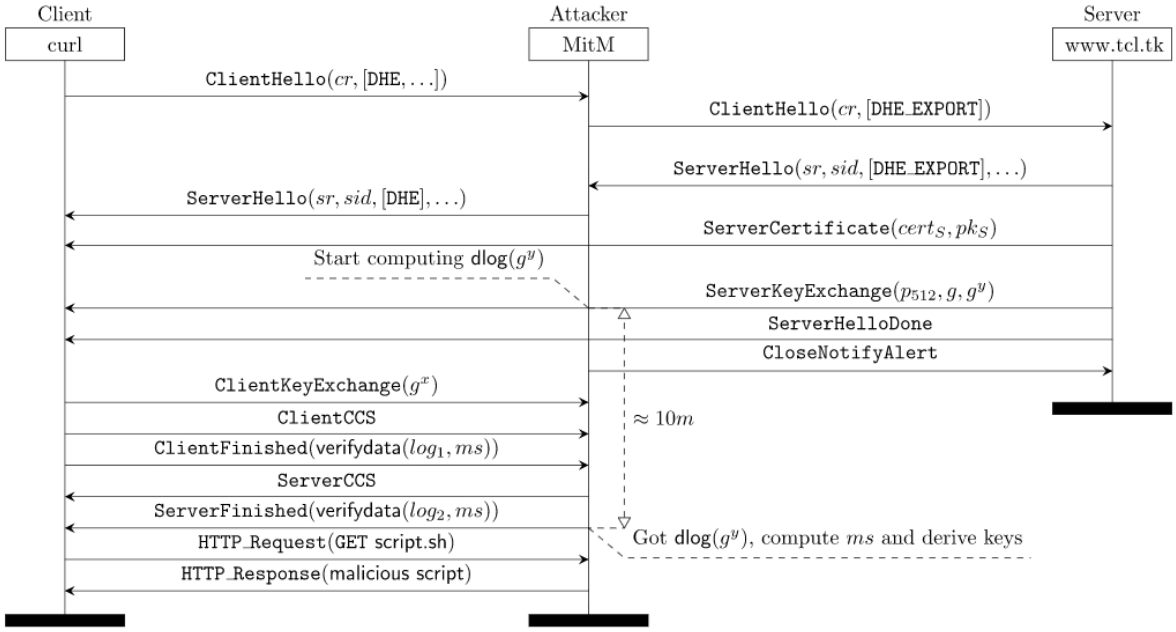


Figura 13. Suplantación de servidor mediante Logjam

En el ataque reflejado en la imagen anterior (*DHE_EXPORT downgrade and man-in-the-middle server impersonation*), el cliente tiene que ser capaz de esperar una cantidad de tiempo importante a que el handshake se complete. Esto se debe a que el atacante debe computar la clave de conexión durante el proceso de handshake (coste de minutos). Algunos clientes lanzarán timeouts que cerrarán la conexión antes de computar la clave. Si no se da este escenario el atacante sería capaz de suplantar al servidor.

En la web *Logjam Attack Proof of Concept Demostrations* (https://weakdh.org/logjam.html) los autores desarrollaron los ejemplos anteriores proporcionando más detalles prácticos de criptoanálisis mediante pruebas de concepto funcionales.

Con este conocimiento, la contramedida esencial para protegerse frente a todos estos ataques se fundamenta en deshabilitar el soporte para *export cipher suites* y garantizar un uso adecuado de algoritmos criptográficos, como Diffie-Hellman de 2048 bits o el uso de curvas elípticas.

Más información en:

https://weakdh.org/
https://weakdh.org/imperfect-forward-secrecy-ccs15.pdf
https://weakdh.org/logjam.html
https://en.wikipedia.org/wiki/Logjam_(computer_security)

2.10 SLOTH. Security Losses from Obsolete and Truncated Transcript Hashes

Karthikeyan Bhargavan et al. publicaron en 2015 la investigación *Transcript Collision Attacks: Breaking Authentication in TLS, IKE, and SSH*, o coloquialmente Sloth.

Sloth, CVE-2015-7575, es un ataque que afecta a clientes y servidores que utilicen TLS, especialmente TLS 1.2, y den soporte a firmas basadas en MD5 (RSA-MD5 o ECDSA-MD5). Es importante resaltar que hace referencia a los algoritmos de firmado y no de cifrado, los cuales podrían estar bien elegidos. Sloth, al igual que Poodle, Freak o Logjam, hace uso de esta compatibilidad con algoritmos criptográficos inseguros. En esta circunstancia y dada una serie de condiciones, un atacante, con cierta capacidad de computación para progresar ataques de colisión y recursos de almacenamiento, podrá anular la seguridad criptográfica.

La siguiente imagen resume las pérdidas de seguridad (SLOTH) debido a ataques de colisión en protocolos criptográficos conocidos. En cada protocolo, se identifica un mecanismo de protocolo y una estimación aproximada de la seguridad esperada. Por ejemplo, si un servidor TLS usa una clave RSA de 3072 bits para la firma y admite SHA-256 como algoritmo hash, la seguridad esperada de las firmas del servidor es de 128 bits. Incluso si el servidor admite firmas MD5, dado que la complejidad del segundo ataque de preimagen de las firmas MD5 sigue siendo 2^{128}, se puede esperar una seguridad de 128 bits para las firmas del servidor. Sin embargo, en esta investigación, se describe un ataque de colisión en las firmas del servidor TLS que toma 2^{64} conexiones, 2^{64} de almacenamiento y 2^{64} hashes por conexión. Por lo tanto, la seguridad efectiva se reduce a la mitad a alrededor de 64 bits. Las pérdidas de seguridad de otros mecanismos, como la autenticación de clientes TLS, son aún más dramáticas, lo que lleva a ataques prácticos a clientes y servidores en el mundo real.

En la siguiente imagen puede verse el coste real de algunos de los ataques documentados:

Protocol	Property	Mechanism	Attack	Collision Type	Precomputation	Work/connection	Wall-clock Time	Preimage Cost	Security Loss
TLS 1.2	Client Auth	RSA-MD5	Impersonation	Chosen Prefix		2^{39}	1 hour (48 cores)	2^{128}	89 bits
TLS 1.2	Channel Binding	Truncated HMAC (96 bits)	Credential Forwarding	Generic		2^{48}	20 days (4 GPUs)	2^{96}	48 bits
TLS 1.2	Client Auth	RSA-SHA1 or ECDSA-SHA1	Impersonation	Chosen Prefix		2^{77}		2^{160}	83 bits
TLS 1.2	Server Auth	RSA-MD5	Impersonation	Generic	2^{X} connections +storage	2^{128-X}		2^{128}	X bits
TLS 1.3	Server Auth	RSA-SHA1 or ECDSA-SHA1	Impersonation	Chosen Prefix		2^{77}		2^{160}	83 bits
TLS 1.1	Handshake Integrity	MD5 | SHA1	Downgrade	Chosen Prefix		2^{77}		2^{160}	83 bits
IKEv1	Initiator Auth	HMAC-MD5	Impersonation	Generic		2^{64}		2^{128}	64 bits
IKEv2	Initiator Auth	RSA-SHA1 or DSA-SHA1	Impersonation	Chosen Prefix	2^{77}			2^{160}	93 bits
SSH 2	Key Exchange Integrity	SHA1	Downgrade	Chosen Prefix		2^{77}		2^{160}	83 bits

Figura 14. Impacto del ataque Sloth en cliente y servidor TLS

En TLS, el cliente se autentica presentando un certificado X.509 y operando con diferentes algoritmos de hash y firma. En TLS 1.2 los clientes y servidores pueden negociar los algoritmos de firma y hash que se admiten. Esto permitió el uso de algoritmos de hash más modernos y robustos, como SHA-256 y SHA-512, pero desafortunadamente también permitió el uso de algoritmos hash más débiles, como MD5. En realidad, se demostró que las librerías TLS en Java (SunJSSE) y en los servidores Akamai admitían firmas RSA-MD5 tanto para la autenticación del cliente como del servidor. Incluso implementaciones que no anuncian soporte para RSA-MD5, como NSS (antes de la versión 3.21), BouncyCastle (Java antes de la versión 1.54, C # antes de la versión 1.8.1), PolarSSL / mbedTLS (antes de la 2.2.1), GnuTLS (antes de la versión 3.3.15), y OpenSSL (antes de la versión 1.0.1f) acepta sorprendentemente las firmas RSA-MD5.

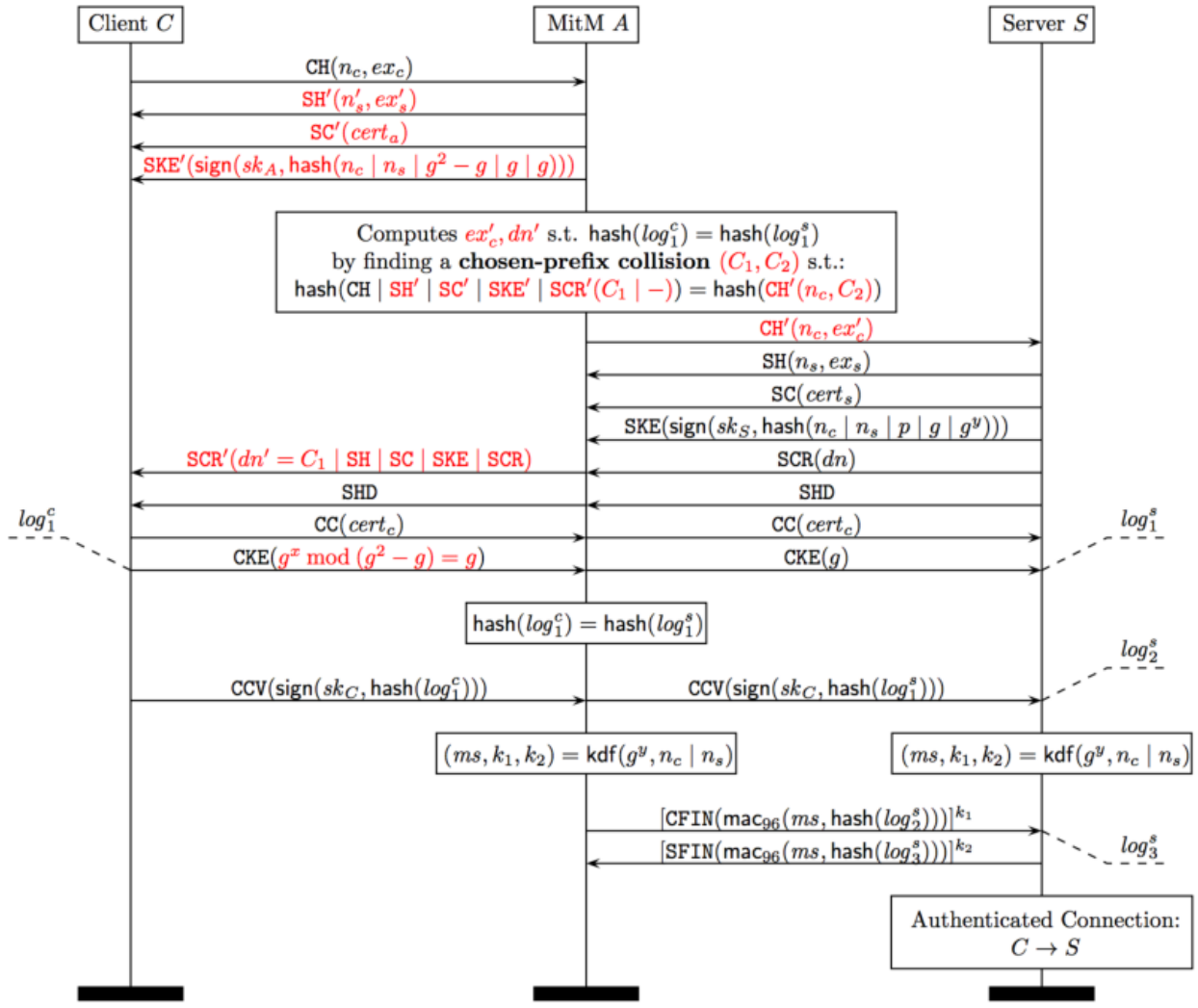


Figura 15. Ejemplo de ataque Sloth

En la figura anterior puede verse un ejemplo de suplantación de cliente mediante un ataque de hombre en el medio. Supongamos que un cliente TLS C y el servidor S son compatibles con las firmas RSA-MD5 para la autenticación del cliente, y supongamos que el cliente está dispuesto a usar el mismo certificado para autenticarse en algún servidor A malicioso. Se puede ver como A puede montar un ataque de colisión de transcripción de hombre en el medio que le permite suplantar a C en S. Para lograr este ataque, el atacante A debe calcular una colisión con el prefijo MD5 elegido, una entre C y A, y la otra entre A y S. La complejidad del ataque depende de la dificultad de encontrar dichas colisiones. Para MD5, se sabe que tales colisiones requieren la computación de 2^{39} hashes, lo que puede lograrse en varias horas en las instancias de Amazon EC2.

Más información para comprender su impacto en librerías como OpenSSL, GnuTLS, BouncyCastle, etc., puede verse en:

https://www.mitls.org/pages/attacks/SLOTH
https://www.mitls.org/downloads/transcript-collisions.pdf

2.11 DROWN. Decrypting RSA with Obsolete and Weakened eNcryption

Los investigadores *Nimrod Aviram et al*. publicaron en 2016 en su investigación "DROWN: Breaking TLS using SSLv2" el ataque Drown (**D**ecrypting **R**SA with **O**bsolete and **W**eakened e**N**cryption)[55] un ataque sobre TLS que se aprovecha del soporte, y de diversos fallos, del protocolo SSLv2 en un servidor (aunque no se utilice) como un oráculo para descifrar conexiones cifradas con TLS. En la práctica, implementa una variante del ataque de Bleichenbacher RSA padding-oracle[56].

DROWN es un buen ejemplo de ataque de protocolo cruzado, en inglés, *cross protocol attack*. Este tipo de ataque hace uso de errores en la implementación de un protocolo (SSLv2) para atacar la seguridad de las conexiones realizadas bajo un protocolo diferente, en nuestro caso TLS. En este caso concreto, el punto de unión de ambos protocolos es el soporte del cifrado RSA (RSA-PKCS#1v1.5). TLS presenta contramedidas frente a los ataques más típicos a RSA mientras que SSLv2 no.

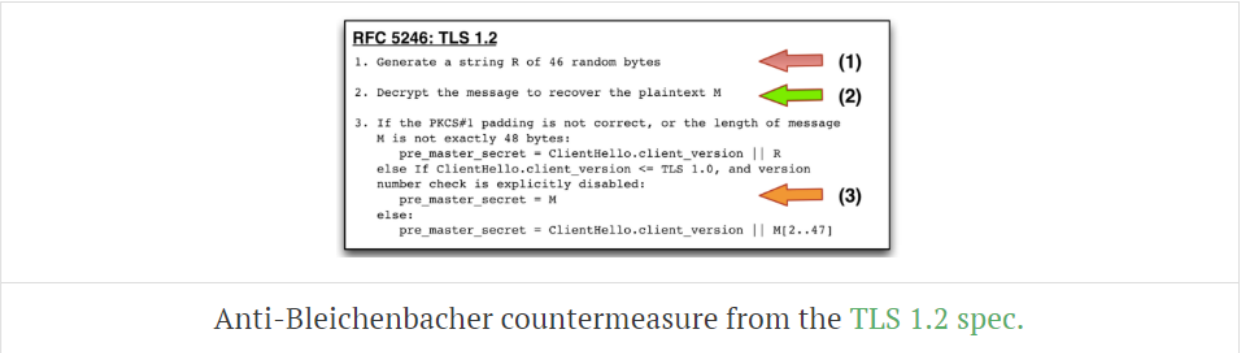


Figura 16. Contramedidas al ataque Bleichenbacher en TLS 1.2 (RFC5246)

El ataque de Bleichenbacher proporcionar un ataque devastador contra servidores SSL, dado que el estándar SSL para el establecimiento de la conexión segura con RSA implica que el cliente cifre un secreto, llamado PMS (Pre-Master Secret), utilizando la clave pública RSA del servidor y lo envíe por la conexión (cable o aire). Un atacante que pueda interceptar el PMS cifrado puede ejecutar el ataque Bleichenbacher contra el servidor, enviándole miles de valores relacionados (bajo la apariencia de nuevas conexiones SSL), y usar las respuestas del servidor (por ejemplo, error o no) para gradualmente descifrar el valor cifrado de PMS. Con este valor en posesión del atacante puede computar las claves de la sesión SSL y descifrar la comunicación, típicamente almacenada.

Para evitar este ataque se han publicado multitud de contramedidas, una se basa en que el servidor cuando detecta que un cifrado con RSA se descifra incorrectamente (por ejemplo, por problemas de formato o padding) miente. En vez de retornar un error (que podría utilizar el atacante para ir descifrando poco a poco la información), el servidor genera un PMS aleatorio y continua con el resto del protocolo como si este valor fuera el válido para descifrar la comunicación. Este valor hará que más adelante la conexión no termine correctamente, pero es suficiente para evitar el ataque.

Si el atacante envía un texto cifrado RSA válido para ser descifrado, el servidor lo descifra y obtiene algún valor PMS. Si el atacante envía el mismo texto cifrado válido por segunda vez, el servidor descifrará y obtendrá nuevamente el mismo valor de PMS. De hecho, el servidor siempre obtendrá el mismo PMS, incluso si el atacante envía el mismo texto cifrado válido cien veces seguidas. Por otro lado, el atacante envía repetidamente el mismo texto cifrado no válido, el servidor elegirá un PMS diferente cada vez. Esta observación es crucial.

[55] CVE-2016-0800 - CVE-2015-3197 - CVE-2016-0703
[56] http://archiv.infsec.ethz.ch/education/fs08/secsem/Bleichenbacher98.pdf

En teoría, si el atacante tiene un texto cifrado que podría ser válido o no válido (y al atacante le gustaría saber cuál es el verdadero), puede enviar el mismo texto cifrado para que se descifre repetidamente. Esto llevará a dos condiciones posibles. En la condición (1) donde el texto cifrado es válido, el descifrado producirá el "mismo PMS cada vez". La condición (2) para un texto cifrado no válido producirá un "PMS diferente cada vez". Si el atacante pudiera decidir de alguna manera la diferencia entre la condición (1) y la condición (2), podría determinar si el texto cifrado era válido. Sólo esa determinación sería suficiente para resucitar el ataque de Bleichenbacher. Afortunadamente en TLS, el PMS nunca se usa directamente; primero se pasa a través de una función hash y se combina con un conjunto de números aleatorios para obtener un master secret. Este resultado luego se usa en cifrados y funciones hash robustas. Gracias a la robustez de la función hash y los cifrados, las claves resultantes están tan ofuscadas que el atacante literalmente no puede saber si está observando la condición (1) o (2). Por desgracia, el mismo comportamiento no sucede en SSLv2.

En SSLv2 no hay PMS, el valor cifrado es usado como Master Secret y empleado directamente para derivar la clave de sesión. Además, en los modos de cifrado permitidos para exportación, la Master Secret podría ser tan pequeña como de 40 bits, y usada con el correspondiente algoritmo criptográfico débil (la suite de algoritmos criptográficos “débiles” permitidos para exportación). Esto significa que un atacante puede enviar múltiples textos cifrados y calcular por fuerza bruta el resultado para las claves de menor tamaño. Una vez recuperadas las claves para un conjunto pequeño de sesiones, el atacante será capaz de determinar si están en la condición (1) o (2). Esto revive el ataque de Bleichenbacher. La mayoría de los servidores configurados con SSLv2 y TLS soportan usar la misma clave privada RSA para descifrar sesiones de ambos protocolos. Un ataque Bleichenbacher en la implementación SSLv2, como se ha descrito, puede ser usado para descifrar el contenido de una sesión TLS basada en RSA.

Finalmente, un atacante para descifrar una sesión de TLS necesita capturar 1000 sesiones TLS usando el intercambio de clave con RSA, realizar 40.000 conexiones SSLv2 al servidor víctima y realizar 2^{50} operaciones criptográficas simétricas. Con el uso de GPUs, en una configuración sencilla, es posible descifrar un criptograma protegido por RSA de 2048 bits en menos de 18 horas. Este ataque se demostró de utilidad para vulnerar más de 11.5 millones de servidores HTTPS, muchos de los cuales no soportaban SSLv2, pero si compartían claves RSA con otros servidores que si lo hicieron. Esto facilitó progresar el ataque.

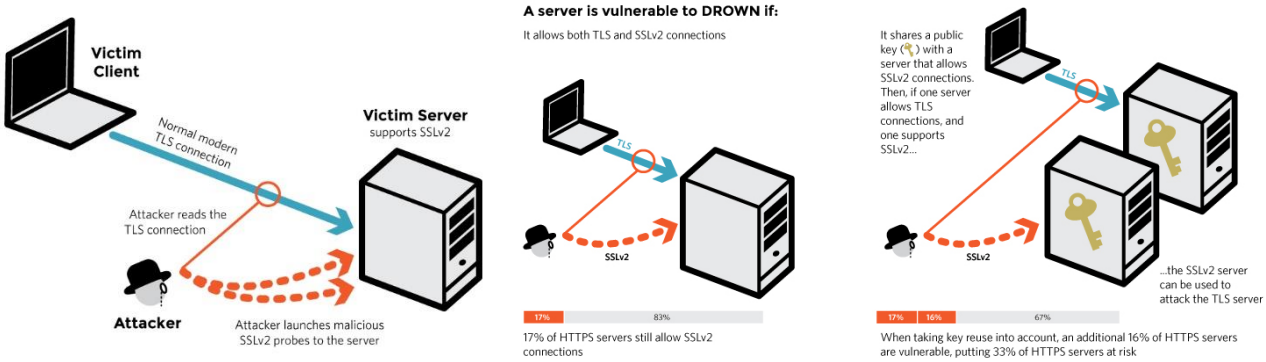


Figura 17. Descripción del ataque DROWN

Más información en:

https://drownattack.com/drown-attack-paper.pdf
https://drownattack.com/
https://blog.cryptographyengineering.com/2016/03/01/attack-of-week-drown/

2.12 SWEET32. Birthday attacks on 64-bits block ciphers

Los protocolos criptográficos como TLS, SSH, IPsec o OpenVPN utilizan algoritmos criptográficos simétricos operando con bloques de información en las comunicaciones entre cliente y servidor. A menudo, para estos algoritmos se centra la fortaleza de su seguridad en el tamaño de su clave, aunque el tamaño del bloque de información a procesar, como es bien conocido en la comunidad criptográfica, es un parámetro que también se debe cuidar.

Un tamaño de bloque pequeño facilita progresar ataques de cumpleaños (*birthday attack*[57]) y hacer que la comunicación cifrada sea vulnerable. Es precisamente en esta característica en la que se centra el ataque SWEET32[58], especialmente para algoritmos con tamaño de bloque más pequeño que lo recomendable de 128 bits. Es precisamente lo que demuestran los investigadores de forma práctica para los algoritmos Triple-DES o Blowfish de tamaño de bloque 64 bits, que siguen utilizándose en protocolos de amplio uso como TLS (web) o OpenVPN. Por ejemplo, los atacantes demuestran que monitorizando una conexión de larga duración que utilice Triple-DES en HTTPS entre un navegador web (cliente) y un servidor web es posible recuperar una cookie de sesión procesando y capturando 785 GB ($2^{36.6}$ bloques). Este volumen de información puede generarse en menos de 2 días mediante un código JavaScript malicioso introducido en una página web.

```
<html>
  <body>
    <script>
    var W = new Array;
    for (var i=0; i<8; i++) {
        var x = new Worker("worker.js");
        W.push(x);
    }
    </script>
  </body>
</html>
```

attack.html

```
var url = "https://10.0.0.1/index.html";
var xhr = new XMLHttpRequest;

// Expand URL to ~4kB using a query string
// Alternatively, force a large cookie
url += "?";
var x = 10000000;
for (var i=0; i<=500; i++) {
    url += x++;
}


while(true) {
    xhr.open("HEAD", url, false);
    xhr.withCredentials = true;
    xhr.send();
    xhr.abort();
}
```

worker.js

Figura 18. Ejemplo de inyección de código JavaScript en el ataque Sweet32

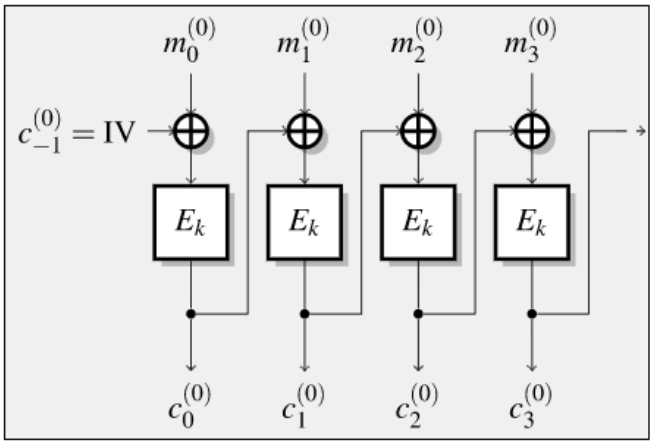
El problema de Sweet32 es que se apoya en un problema bien conocido en criptografía, en cifradores de bloques es necesario cambiar la clave antes de cifrar $2^{n/2}$ bloques, siendo n el tamaño de bloque en bits, y esto es un problema para la mayoría de modos de operación de bloques (CBC, CTR, GCM, OCB, etc). En un cifrador moderno de bloques 128 bits como AES se necesitaría una cantidad de tráfico del orden de 256EB, sin embargo, cifradores de bloque de 64 bits incumpliría esa recomendación a los 32GB, algo asumible en una conexión de alta velocidad en un espacio corto de tiempo.

En este escenario el atacante puede ser capaz de identificar información, cómo por ejemplo credenciales o tokens de autenticación. Recuérdese que entre los mecanismos de autenticación básicos se encuentran aquellos basados en cookies de sesión (RFC6265) o mecanismos básicos de autenticación HTTP (RFC7617), por ejemplo, *Authorization: Basic dGVzdDoxMjPCow=*. Estas credenciales son enviadas por el navegador en cada solicitud y si se fuerza el número adecuado de

[57] https://en.wikipedia.org/wiki/Birthday_attack
[58] CVE-2016-2183, CVE-2016-6329

peticiones se conseguirá colisiones al cifrador de bloque que permitirá recuperar los secretos. Por ejemplo, supongamos un cifrado con tamaño de bloque 64 bits en modo CBC, típicamente una conexión HTTPS donde cliente y servidor han negociado Triple-DES. La víctima ya se ha logeado en una página web y tiene una cookie de sesión. El ataque ejecutará un código JavaScript en el navegador de la víctima, enviará peticiones HTTP a la url objetivo (servidor), cada petición contendrá, por ejemplo, la cookie de sesión. Si el número de peticiones es cercano a 2^{32} solicitudes, se espera que una colisión suceda entre un bloque cifrado correspondiente a la cookie (c_i) y un bloque conocido (c_j) que contiene una parte conocida de la petición. El ataque de colisión contra CBC revelará la cookie de sesión:

$$p_i = p_j \oplus c_i\text{-}1 \oplus c_j\text{-}1$$



The CBC mode

Un matiz importante en este ataque es como de práctico puede llegar a ser. En teoría, las recomendaciones indican que la clave criptográfica (la sesión) debe cambiarse antes de $2^{n/2}$ bloques procesados, aunque muchas implementaciones, como demuestra Sweet32, no fuerza a la limitación en el uso de una clave criptográfica. Es decir, podría suceder que existieran conexiones de muy larga duración que utilizaran la misma clave criptográfica, aunque es cierto por las pruebas de los investigadores que este número es muy reducido (0,6% del top 10.000 de Alexa).

Por tanto, para que el ataque sea viable se necesita poder tener conexiones de larga duración, el atacante tiene que tener acceso a la red del objetivo para capturar tráfico y tiene que tener la capacidad de que el cliente cifre texto en claro conocido (JavaScript malicioso) elegido por el atacante. Lo que está claro, no obstante, el ataque hace recordar la necesidad de utilizar siempre tamaño de bloque iguales o superiores a 128 bits.

Más información en:

https://sweet32.info/
https://sweet32.info/SWEET32_CCS16.pdf

2.13 ROCA – The Return of Coppersmith's attack

La vulnerabilidad ROCA (Return Of Coppersmith's Attack), CVE-2017-15361, es una debilidad criptográfica que permite recuperar la clave privada de la clave pública en dispositivos afectados por esta vulnerabilidad. En general, suele hacer referencia a los problemas introducidos por la librería RSALib (Infineon Technologies) en la generación de claves RSA, en concreto en la generación de números primos[59], que permite mediante la aplicación de una variante del método de Coppersmith[60] anular la protección de claves de 512, 1024 y 2048 bits, lo que afectó a numerosas tarjetas inteligentes (dnis electrónicos incluidos), TPM (Trusted Platform Module), etc.

Key size	M	Size of M	Size of M'	Naïve BF # attempts ($ord_M(65537)/2$)	Our BF # attempts ($ord_{M'}(65537)/2$)	Time per attempt	Worst case
512 b	$P_{39}\# = 167\#$	219.19 b	140.77 b	$2^{61.09}$	$2^{19.20}$	11.6 ms	1.93 CPU hours
1024 b	$P_{71}\# = 353\#$	474.92 b	285.19 b	$2^{133.73}$	$2^{29.04}$	15.2 ms	97.1 CPU days
2048 b	$P_{126}\# = 701\#$	970.96 b	552.50 b	$2^{254.78}$	$2^{34.29}$	212 ms	140.8 CPU years
3072 b	$P_{126}\# = 701\#$	970.96 b	783.62 b	$2^{254.78}$	$2^{99.29}$	1159 sec	$2.84 * 10^{25}$ years
4096 b	$P_{225}\# = 1427\#$	1962.19 b	1098.42 b	$2^{433.69}$	$2^{55.05}$	1086 ms	$1.28 * 10^{9}$ years

Figura 19. Descripción general de los parámetros utilizados (M original y M optimizado) y el rendimiento de algoritmos de factorización para las longitudes de clave utilizadas comúnmente. Las mediciones de tiempo para múltiples intentos se tomaron en un núcleo de una CPU Intel Xeon E5-2650 v3 con velocidad de reloj de 3.00 GHz, y las estimaciones de tiempo en el peor de los casos se extrapolan de las solicitudes y los tiempos promedio requeridos por intento. El tiempo de factorización esperado es la mitad del tiempo de peor caso.

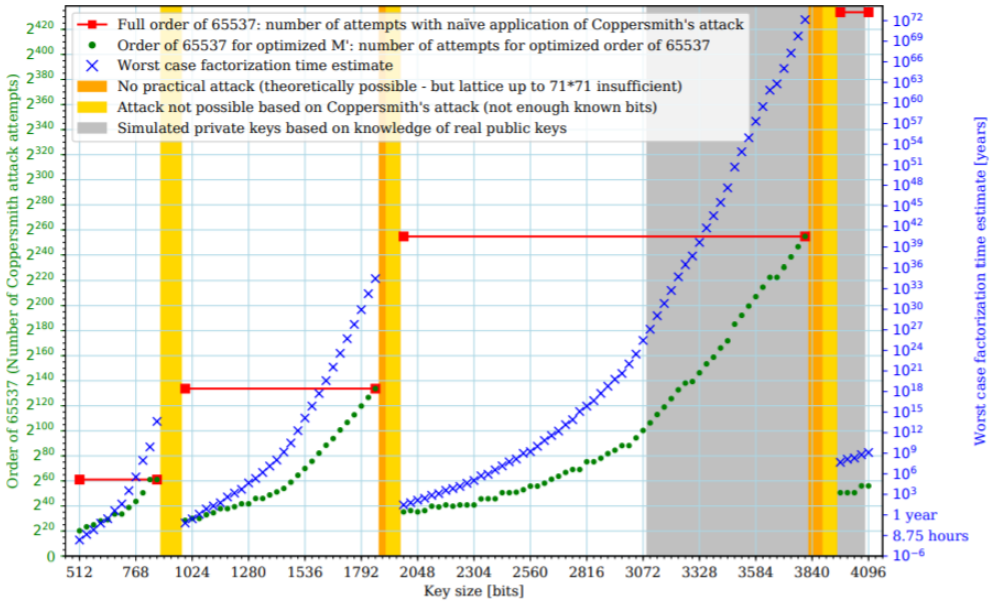


Figura 20. Complejidad de factorización de claves generadas por RSALib con diferentes tamaños de claves desde 512 a 4096 bits en escalones de 32 bits.

Más información en:

https://crocs.fi.muni.cz/_media/public/papers/nemec_roca_ccs17_preprint.pdf
https://en.wikipedia.org/wiki/ROCA_vulnerability
https://github.com/crocs-muni/roca

[59] La primalidad en RSALib se testea de la forma $k*M + (65537^a \bmod M)$. Esta decisión y el formato de los primos creados facilita el cálculo de logaritmos discretos mediante el algoritmo de Pohlig-Hellman (https://en.wikipedia.org/wiki/Pohlig%E2%80%93Hellman_algorithm).
[60] https://en.wikipedia.org/wiki/Coppersmith%27s_attack

2.14 The ROBOT. Return Of Bleichenbacher's Oracle Threat

Hanno Böck, Juraj Somorovsky y Craig Young en agosto de 2018 demostraron que el ataque descubierto en 1998 por Daniel Bleichenbacher[61] [62] seguía siendo posible de reproducir. El lector observará que muchos de los ataques recientes se han basado en este paradigma.

Daniel Bleichenbacher descubrió que los mensajes de error devueltos por servidores SSL al procesar errores en el padding en mensajes PKCS#1 v1.5 permitían realizar ataques de oráculo (adaptive-chosen ciphertext). Procesando un número importante de estos mensajes, que provocan errores, era capaz de deducir el mensaje en claro al que correspondía un mensaje cifrado enviado. En los últimos años se publicaron diversas contramedidas para evitar este ataque. Por ejemplo, la RFC3218 (Preventing the Million Message Attack on Cryptographic Message Syntax) que se puede consultar en: https://tools.ietf.org/html/rfc3218.

Hanno Böck et al. descubrieron 2018 en su investigación "Return Of Bleichenbacher's Oracle Threat (ROBOT)" descubrieron nuevos mecanismos para poder replicar este tipo de ataque basándose en otro tipo de oráculo (no en formato de mensajes PKCS y padding), en este caso, utilizando TCP resets, TCP timeouts o mensajes de alerta duplicados.

Implementation	Server response		TLS flow	Oracle	Reference / ID
	Valid message	Invalid message			
Facebook					
1st vulnerability	20	47	full	strong	-
2nd vulnerability	20	TCP FIN	shortened	strong	-
F5					
Variant 1	TCP timeout	40	shortened	strong	CVE-2017-6168
Variant 2	One alert (40)	Two alerts (40)	full	strong	CVE-2017-6168
Variant 3	TCP timeout	40	shortened	weak	CVE-2017-6168
Variant 4	One alert (40)	Two alerts (40)	full	weak	CVE-2017-6168
Variant 5	20	80	full	strong	CVE-2017-6168
Citrix Netscaler					
with CBC cipher suites	Connection reset	TCP timeout	full	strong	CVE-2017-17382
with GCM cipher suites	51	TCP timeout	full	strong	CVE-2017-17382
Radware					
Radware Alteon	51	TCP reset	full	strong	CVE-2017-17427
Cisco					
Cisco ACE	20	47	full	strong	CVE-2017-17428
Cisco ASA	TCP timeout	TCP reset	full	weak	CVE-2017-12373
Erlang					
Erlang version 19 and 20	10	51	full	strong	CVE-2017-1000385
Erlang version 18	20	51	full	strong	CVE-2017-1000385
Palo Alto Networks					
PAN-OS	One alert (40)	Two Alerts (40)	full	weak	CVE-2017-17841
IBM					
IBM Domino	20	47	full	weak	(unfixed)
IBM WebSphere MQ	?	?	?	?	CVE-2018-1388
WolfSSL					
WolfSSL prior to 3.12.2	TCP timeout	Alert 0	shortened	weak	CVE-2017-13099
Bouncy Castle					
Bouncy Castle 1.58	`ChangeCipherSpec`	80	shortened	weak	CVE-2017-13098

Table 1: Overview of vulnerable implementations and affected servers found in our research. TLS alerts are referenced by their numbers: 10 (`unexpected_message`) 20 (`bad_record_mac`), 40 (`handshake_failure`), 47 (`illegal_parameter`), 51 (`decrypt_error`), and 80 (`internal_error`).

El ataque ROBOT solo afecta a modos de cifrado en TLS que utilicen cifrado RSA. En este caso, el atacante podría almacenar y descifrar la información a su antojo, en caso de progresar el ataque. Numerosos productos y software open-source se vieron comprometidos: F5, Citrix, Radware, Palo Alto Networks, IBM, Cisco, etc.

Más información en:

https://robotattack.org/
https://www.usenix.org/system/files/conference/usenixsecurity18/sec18-bock.pdf

[61] https://en.wikipedia.org/wiki/Daniel_Bleichenbacher
[62] Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1 http://archiv.infsec.ethz.ch/education/fs08/secsem/bleichenbacher98.pdf

2.15 The 9 Lives of Bleichenbacher's CAT: new cache attacks on TLS implementations (1.3)

Eyal Ronen et al. publicaron, a finales de 2018, la investigación titulada "The 9 Lives of Bleichenbacher's CAT: New Cache ATtacks on TLS Implementations". En esta investigación se demostró que incluso implementaciones modernas[63], TLS (incluso la 1.3[646566]) y QUIC, con contramedidas a variantes del ataque de Bleichenbacher, analizadas anteriormente en el libro, eran vulnerables a ataques de canal lateral mediante los tiempos de acceso a la cache[67] (Cache ATacks, CAT). Esta aproximación permitió romper el intercambio de claves con RSA[68] en 7 de las implementaciones TLS más famosas (OpenSSL, Amazon s2n, MbedTLS, Apple CoreTLS, Mozilla NSS, WolfSSL, y GnuTLS) forzando un ataque de downgrade a TLS 1.2, cubriendo aproximadamente el 6% de servidores de Internet.

El ataque puede resumirse fácilmente.

1. SSL/TLS, excepto TLS 1.3, puede usar el algoritmo RSA para el intercambio de un secreto compartido, usado típicamente para proteger los datos de la sesión (comunicación).

2. Si un atacante consigue ese secreto compartido puede descifrar la sesión, o conseguir una cookie de sesión (un ejemplo clásico en la literatura), o suplantar a uno de los extremos de la comunicación.

3. Para forzar el ataque, descrito por los investigadores, se utiliza una filosofía tipo ataque BEAST, es decir, es necesario realizar un ataque de hombre en el medio, MiTM, e inyectar algún código en uno de los extremos de la comunicación, típicamente un código JavaScript en un navegador web, que facilite la creación de nuevas conexiones TLS a un objetivo concreto.

4. Para atacar TLS 1.3, la versión más moderna en el momento de escribir este libro, se necesita que, típicamente el servidor en una comunicación web, soporte una versión más antigua de este protocolo. Los escenarios más típicos son:

    a. El servidor usa una clave pública RSA que utiliza para firmar las claves de sesión y una versión más antigua del protocolo TLS utiliza las mismas claves RSA. Esto facilita ataques de protocolo cruzado como se pudo ver en el ataque DROWN.

    b. Ambos extremos de la comunicación soportan una versión más antigua del protocolo TLS que permite el intercambio de claves de sesión utilizando RSA.

5. Proceder a una degradación del protocolo a TLS 1.2. Para ello se modifican las peticiones-respuestas en el proceso de negociación entre cliente y servidor para la negociación de los algoritmos criptográficos a utilizar y versión del protocolo.

---

[63] https://www.nccgroup.com/us/about-us/newsroom-and-events/blog/2019/february/downgrade-attack-on-tls-1.3-and-vulnerabilities-in-major-tls-libraries/

[64] Nuevos ataques están surgiendo a la seguridad proporcionada por TLS 1.3 https://www.wolfssl.com/wolfssl-and-the-selfie-attack/

[65] Whats new with TLS 1.3 - https://medium.com/@vanrijn/what-is-new-with-tls-1-3-e991df2caaac

[66] Nuevos ataques a implementaciones, en la relación entre TLS 1.3 y TLS 1.2, están en aumento: "CVE-2020-13777: TLS 1.3 session resumption works without master key, allowing MITM" - https://gitlab.com/gnutls/gnutls/-/issues/1011

[67] Utilizando para ello técnicas de ataque a caché famosas en la literatura: Flush+Reload, Prime+Probe o Branch-Predition..

[68] Los investigadores llegaron a conseguir descifrar claves de 2048 bits en menos de 30 segundos.

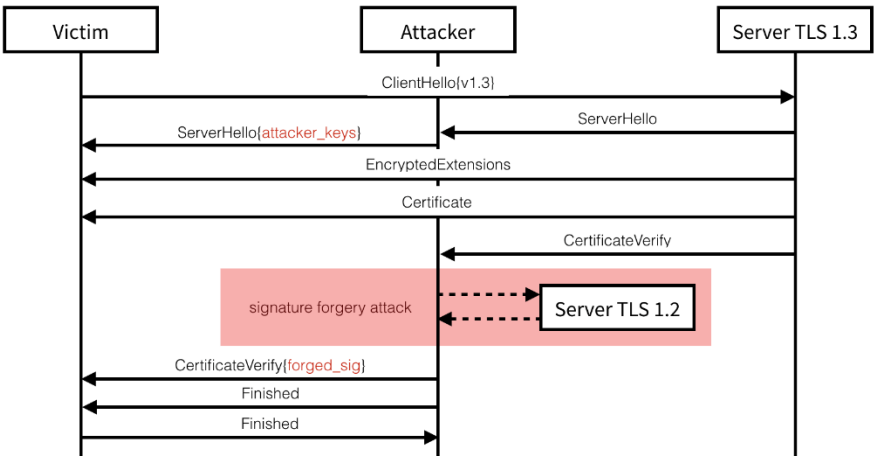


Figura 21. Esquema general de ataque 9 lives of bleichenbachers cat

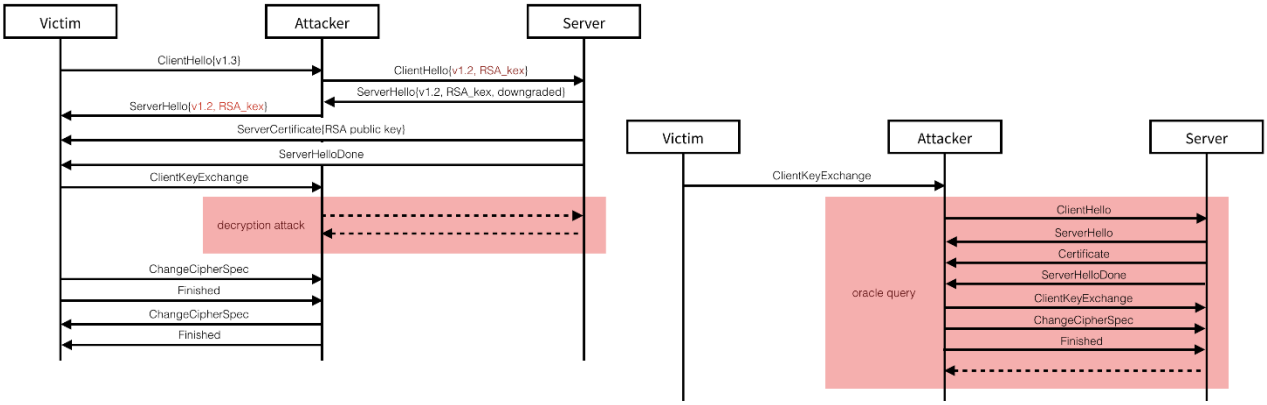


Figura 22. Proceso completo de ataque a TLS 1.3 con 9 lives

6. Si el ataque progresa el cliente utilizará la clave pública RSA del certificado para cifrar el secreto compartido (*TLS premaster secret*) enviando un mensaje *ClientKeyExchange* y finalizando el proceso inicial de negociación (*handshake TLS*) con los mensajes *ChangeCipherSpec* y *Finished*. Los mensajes de finalización requieren una etiqueta de autenticación de todo el proceso con lo que es necesario realizar el proceso de descifrado en un tiempo relativamente corto dado que los navegadores modernos tienen timeouts que cierran las conexiones incompletas (existen técnicas para retrasar esos timeouts, para más detalle ver la documentación recomendada). Uno de los principales hitos de los investigadores recae en acelerar el proceso de descifrado para hacer el ataque práctico.

7. El ataque, adicionalmente a la posibilidad de realizar un downgrade del protocolo (necesidad de posición de MiTM) y de las características adecuadas en uso de claves y algoritmos de RSA, requiere que el atacante ejecute código en la máquina "servidora" (en un sistema que comparta hardware con él). Esta ejecución de código es la que le permitirá realizar el ataque de canal lateral mediante acceso a la memoria y la que finalmente ayudará, a modo de oráculo (como se analizó en caso similares en los ataques descritos anteriormente) el poder detectar diferencias temporales a la hora de procesar los mensajes cifrados que envía el atacante, a descifrar el secreto compartido. Un escenario típico recae en sistemas que tenga acceso a hardware físico compartido, por ejemplo, un servidor web en cloud que comparte el mismo hardware físico con otra instancia donde tenemos un atacante no privilegiado.

Notation. We extend the notation of Bardou et al. [7] to refer to various oracles. Specifically, our notation is:

- FFFF denotes an oracle that gets as input a ciphertext and returns true only if the corresponding plaintext passes all four checks. This is the same as the Bad-Version Oracle (BVO) of Klíma et al. [42].
- FFFT denotes an oracle that returns true for ciphertexts corresponding to plaintexts that pass the first three checks, ignoring the fourth check.
- FFTT is an oracle that only verifies first two checks. This is the Bleichenbacher oracle described in Section II-C
- FTTT denotes an oracle that returns true if the decrypted plaintext passes the first check and disregards the last three checks.
- TTTT is an oracle that disregards the four checks, returning true for ciphertexts whose corresponding plaintexts start with 0x0002.
- M denotes a Manger oracle (Section II-D).
- I denotes an Interval oracle (Section II-E).

TABLE I
SUMMARY OF IDENTIFIED PADDING ORACLES.

	Data Conv.	PKCS #1 v1.5 Verification	TLS Mitigation
OpenSSL	M	M	
OpenSSL API	M	FFTT	
Amazon s2n		FFFT	
MbedTLS	I	FFTT, FFFT*	
Apple CoreTLS			FFTT, FFFT, FFFF
Mozilla NSS	M	M, TTTT, FTTT*	FFFF
WolfSSL	M	M, FFTT	FFTT, FFFF
GnuTLS	M	M, TTTT, FFTT	FFTT, FFFT
BoringSSL		*Not Vulnerable*	
BearSSL		*Not Vulnerable*	

Figura 23. Resumen de ataques de padding a implementación TLS/SSL modernas

Más allá del impacto final de este ataque, y de las restricciones necesarias para hacerlo práctico, es un excelente ejemplo del ingenio utilizado para romper en la práctica la protección de las comunicaciones cifradas. Adicionalmente, permite extraer una buena serie de sugerencias y contramedidas para evitar ataques similares:

1. Prohibir el uso del intercambio de claves con RSA. Utilizar el su lugar otras alternativas como por ejemplo intercambio de claves con ECDH (Elliptic Curve Diffie-Hellman).

2. Evitar la reutilización de certificados digitales.

3. Programación constante en tiempo para dificultar a un atacante la creación de oráculos criptográficos.

4. Timeouts bajos y claves largas dificultan en gran medida los ataques prácticos, especialmente en navegadores web.

5. Hardware dedicado para operaciones criptográficas, al menos, las más sensibles.

Más información en:

The 9 Lives of Bleichenbacher's CAT: New Cache ATtacks on TLS Implementations
https://eyalro.net/project/cat/
https://eprint.iacr.org/2018/1173
https://github.com/mimoo/RSA_PKCS1v1_5_attacks
https://www.cryptologie.net/article/461/the-9-lives-of-bleichenbachers-cat-new-cache-attacks-on-tls-implementations/

The 9 Lives of Bleichenbachers CAT: New Cache ATtacks on TLS Implementations
https://www.youtube.com/watch?v=tv8PNkTIpUU y
https://www.youtube.com/watch?v=nEIgFQTnlsw

2.16 Ataque Raccoon - 09/2020

Raccoon attack es un ataque de canal lateral temporal que afecta al protocolo TLS 1.2 y versiones anteriores, afectando a servidores web y redes privadas virtuales (OpenVPN, SSL VPN, …), entre otros. Este ataque es muy difícil de replicar (ataque del lado del servidor[69]), tanto por la necesidad de configuraciones específicas por parte de uno de los extremos de la comunicación (típicamente el servidor) y la necesidad de un oráculo preciso mediante el uso de un canal lateral temporal (en la práctica estar *"muy"* cerca del servidor). El ataque se sustenta, por tanto, en dos condiciones:

- El servidor reutiliza claves públicas DH (Diffie-Hellman) en los mensajes enviados en el TLS handshake. Esta circunstancia se dará si el servidor tiene configurado de forma estática el uso de TLS-DH como suite de cifrado o si el servidor usa la version efímera de Diffie-Hellman (TLS-DHE) y reutiliza dichas claves efímeras para multiples conexiones.
- Capacidad del atacante para ejecutar con precision un canal lateral temporal (medir diferencias temporales) que le permitiría descubrir si el primer byte del secreto compartido por Diffie-Hellman comienza con 0 o no[70].

¿Cómo funciona el ataque de manera sencilla?

El intercambio de claves Diffie-Hellman (DH) es un método bien establecido para intercambiar claves en conexiones TLS. Cuando se usa Diffie-Hellman, ambos pares TLS generan claves privadas al azar (a y b) y calculan sus claves públicas: g^a mod p y g^b mod p. Estas claves públicas se envían en los mensajes TLS KeyExchange. Una vez que se reciben ambas claves, tanto el cliente como el servidor pueden calcular una clave compartida g^{ab} mod p, llamada *premaster secret*, que se utiliza para derivar todas las claves de sesión TLS con una función de derivación de clave específica.

El ataque Raccoon explota un canal lateral de especificación TLS. En TLS 1.2 (y todas las versiones anteriores) se prescribe que todos los bytes cero iniciales en el secreto del premaster se eliminen antes de usarse en cálculos posteriores. Dado que el secreto de premaster resultante se utiliza como entrada en la función de derivación de claves, que se basa en funciones hash con diferentes perfiles de tiempo, las mediciones de tiempo precisas pueden permitir a un atacante construir un oráculo a partir de un servidor TLS. Este oráculo le dice al atacante si un secreto de premaster calculado comienza con cero o no. Por ejemplo, el atacante podría escuchar a escondidas el mensaje enviado por el cliente, reenviarlo al servidor y determinar si el secreto de premaster resultante comienza con cero o no.

Aprender un byte del premaster secret no debería ayudar mucho al atacante. Sin embargo, aquí el ataque se vuelve interesante. Imagine que el atacante interceptó un mensaje *ClientKeyExchange* que contenía el valor g^a. El atacante ahora puede construir valores relacionados con g^a y enviarlos al servidor en distintos TLS handshakes. Más concretamente, el atacante construye valores $g^r_i * g^a$, que conducen a los secretos del premaster g^{ri*b*}gab. Según el comportamiento del tiempo del servidor, el atacante puede encontrar valores que conduzcan a secretos de premaster que comiencen con cero.

---

[69] Los navegadores dependen de la suite de cifrado negociada para interactuar, aunque es cierto que podría bloquear cualquier comunicación para un conjunto de algoritmos criptográficos determinado. Las versiones modernas de los navegadores web deberían bloquear DHE.

[70] Este ataque no afecta ni a TLS 1.3 ni a ECDH. Los primeros bytes a cero se conservan y las claves no se reutilizan. Variantes de TLS 1.3 podrían estar afectadas, como ETS o eTLS debido a la reutilización de claves. En principio, DTL sí está afectado.

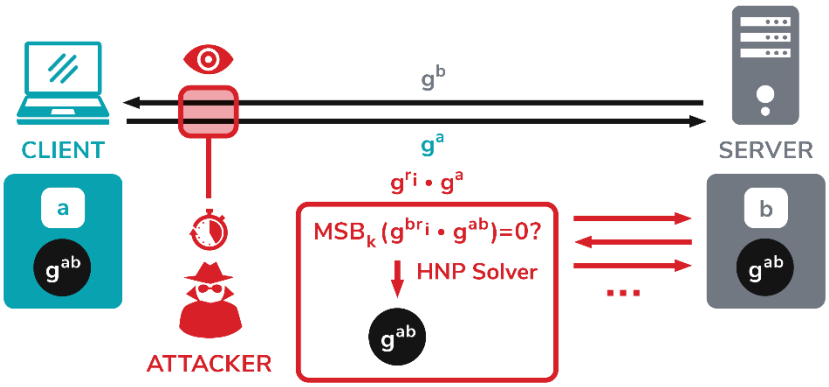


Figura 24. Esquema de ataque criptográfico de Raccoon attack

Al final, esto ayuda al atacante a construir un conjunto de ecuaciones y usar un *solucionador* para el problema de números ocultos (HNP – Hidden Number Problem) para calcular el secreto de premaster original establecido entre el cliente y el servidor.

10 Conclusions

Beyond the specifics of the attack, we argue that its existence can also teach us broader lessons for cryptographic protocols.

Forgoing forward secrecy is dangerous. Forward secrecy is a well-known security goal for cryptographic protocols and was intensively analyzed in the context of TLS in [61]. Our attack exploits the fact that servers may reuse the secret DH exponent for many sessions, thus forgoing forward secrecy. In this context, Raccoon teaches a lesson for protocol security: For protocols where some cryptographic secrets can be continuously queried by one of the parties, the attack surface is made broader. The Raccoon attack showed that we should be careful when giving attackers access to such queries.

Secrets should be constant-size. The dangers of non-constant-time implementations are well-known. For example, they have been repeatedly demonstrated to break ECDSA as used in TLS. One of the reasons for these breaks is that the processing of variable-length secret values within the implementation usually results in non-constant execution time. We argue that future protocol designs should make sure that all their secrets (including intermediate values and their internal number representation) are of fixed size.

Countermeasures. The most straightforward mitigation for the Raccoon attack against TLS is to remove support for TLS-DH(E) entirely. Besides that, server operators can disable DHE key reuse. Then the attacker cannot compute the shared secret even with perfect measurements, as one equation does not give enough information to recover the shared secret. To prevent timing-based side channels in legacy applications with length-varying secrets, the implementation must ensure that the execution of the function is still constant time. This can be done as in the Lucky13 mitigation or by computing the values for different fake parameters and discarding the fake ones afterwards. However, one has to be very careful when implementing such mitigations; previous research has shown that this kind of mitigations adds code complexity and may still leave the side channel open [5] or introduce even more severe vulnerabilities [60].

Figura 25. Recomendaciones criptográficas para mitigar Raccoon attack - *https://raccoon-attack.com/RacoonAttack.pdf*

Más información en:

https://raccoon-attack.com/
https://raccoon-attack.com/RacoonAttack.pdf

3 Conclusiones y recomendaciones

El éxito no es el final,
el fracaso no es fatal:
es el coraje de continuar
lo que cuenta.
Winston Churchill

La lectura de este libro permite al lector reflexionar sobre las precauciones necesarias a la hora de seleccionar algoritmos criptográficos (una buena referencia puede verse en el repositorio de Aaron Toponce[71]), reutilizar certificados y claves criptográficas, la interacción de la criptografía, y cómo se puede debilitar, con diversos modos y protocolos (por ejemplo, combinado con algoritmos de compresión).

Reflexionar sobre el peligro en el uso de hardware compartido y ataques laterales de naturaleza diversa, pero en especial derivados de malas implementaciones, que realizan cálculos diversos de manera no constante en el tiempo, la necesidad de evitar ataques de inyección de código en los extremos de una comunicación que dificulte el uso oráculos criptográficos, la necesidad criptográfica de evitar las compatibilidades hacia atrás de algoritmos y protocolos, etc[72].

En octubre de 2019, impartí una ponencia en la conferencia de seguridad Navaja Negra resumiendo algunos de los aspectos cubiertos en este libro. Quizás el vídeo y el material generado pueda ayudarle en la comprensión del material presente en este documento.

- Navaja Negra 9 - Reversing Cryptographic attacks over SSL/TLS - Alfonso Muñoz. https://www.youtube.com/watch?v=m1Gwi6jKPCE

- https://github.com/mindcrypt/talks/blob/master/Reversing%20cryptographic%20attacks%20over%20SSL-TLS%20-%20Alfonso%20Mu%C3%B1oz%20NavajaNegra%202019.pdf

Deseo que pueda aprender y generalizar las recomendaciones y aprendizaje que cada ataque y su amplia bibliografía refleja este escrito. La criptografía nos permite construir una sociedad mejor. Espero que este libro ayude en esa dirección.

[71] Cryptographic best practices - https://gist.github.com/atoponce/07d8d4c833873be2f68c34f9afc5a78a
[72] Adicionalmente, siempre es recomendable, por lo menos para la auditoría criptográfica de servidores web, el uso del servicio gratuito SSL Labs de la empresa Qualys para el análisis sencillo de la presencia de muchos de los ataques descritos en este libro - https://www.ssllabs.com/ssltest/

Seguridad del protocolo SSL/TLS. Ataques criptoanalíticos modernos

Autor: Dr. Alfonso Muñoz (@mindcrypt)
Prólogo: D. Juliano Rizzo