Attacken und Schlussfolgerungen...

Beast (Browser Exploit Against SSL/TLS):

- Chosen-plaintext Attacke unter Ausnutzung von Schwächen des Cipher Block Chaining (CBC) im Zusammenhang des IVs.
- 'Man-in-the-Middle"-Angriff mit dem Ziel das Authentifizierungs-Token zu erhalten.
- Da im CBC-Modus der Ciphertext-Block des vorhergehenden Blocks als IV genutzt wird kann der Angreifer den IV sehen, der für den session-cookie benutzt wird (die Lage des Cookies ist vorhersehbar). Wenn nun der Angreifer selbst einen Klartext wählen kann (versendet im Namen des Opfers), so kann daraus der session-cookie erraten werden (und überprüft werden, ob der cipher übereinstimmt).
- Da nur der ganze Block geraten werden kann, ist dies aber sehr aufwändig. Daher wird versucht, den session-cookie über mehrere Blöcke zu verteilen. Dies wird durch die Kontrolle der Block-Begrenzung (block-boundaries) möglich, nun muss nur noch ein byte pro Block geraten werden. Nach erraten des ersten Bytes wird Block-Begrenzung um ein byte erhöht, usw. Dies wird fortgeführt bis der gesamte Cookie erraten wurde. Wenn der Zufallswert base64 kodiert ist, braucht der Angreifer also nur 32 Runden pro Zeichen.
- **Gegenmaßnahmen**: TLS1.1 hat das Problem entschärft, in dem der impliziter IV durch einen expliziten IV ersetzt wurde. Wegen Client-Kompatibilität sind aber weder TLS1.1 oder 1.2 weit verbreitet. Zumindest als Fallback wird zumeist noch TLS1.0 und SSL3.0 unterstützt. Die Browser haben daher einen workarround implementiert (In dem initial leere Fragmente in die Nachricht eingesetzt werden, wird der IV besser randomisiert, der Suchraum ist dadurch wieder erhöht). Es sollte also zum einen darauf geachtet werden, dass nur gepatchte Browser und möglichst TLS 1.1, besser 1.2 benutzt werden, sowie Cross-Origin-Requests deaktiviert werden (GoogleChrome 16 oder neuer, Microsoft mit MS12-006, Firefox 10 oder neuer, OpenSSL 1.0.0e oder neuer).

Crime (Compression Ratio Info-leak Made Easy):

- Side-Channel Attacke um session-tokens oder andere secrets aus HTTP-Requests zu extrahieren. Möglich wenn SSL/TLS Data-Compressions (Deflate oder gzip) benutzt.
- *Deflate* scant den Input und sucht nach wiederholten Strings, welche dann mit Back-References (Distance, Length) auf das letzte Vorkommen ersetzt werden.
- Beim Angriff wird der HTTP-Request so umgewandelt, dass der Präfix der gewünschten Information (zB secretcookie=) in den Header geschrieben wird. Nach der Compression wird die Länge also kleiner sein, als wenn der eingesetzte String nicht einem anderen String im Request ensprechen würde. Die Idee ist also, den Input zu verändern und die Länge der komprimierten Daten zu messen und zu vergleichen. So kann, ähnlich wie bei BEAST, Zeichen nach Zeichen verglichen werden, ob die Länge des Requests sinkt und damit ein neues Zeichen mit dem gewünschten übereinstimmt. Damit lässt sich nach mehreren Versuchen das komplette Geheimnis herausfinden.
- **Gegenmaßnahmen**: TLS-Compression ausschalten. Folgende Browser haben TLS-Compression per default deaktiviert: alle IE Versionen (da keine IE-Version SSL/TLS compression unterstützt), Chrome 21.0.1180.89 und neuer, FF 51.0.1 und neuer, Opera 21.01 und neuer, Safari 5.1.7 und neuer. Bei Apache 2.2.x mit mod_SSL, muss SSLCompression explizit ausgeschaltet werden (2.2.24 unterstützt SSLCompressionFlag, on

by default, ausstellen!), bei Apache mit mod_gnutls unbedingt mit "!COMP-Deflate" TLS-Compression deaktivieren, MS IIS geht gut (da auch hier kein TLS-Compression unterstützt wird).

TIME (*Timing Info-leak Made Easy)*

- CRIME hatte zwei Nachteile: Erstens benutzte es HTTP-Requests. Die TLS-Compression wurde daraufhin von den großen Browsern wie der Server-Software deaktiviert. Zweitens basiert CRIME auf einer Man-in-the-Mddle-Attacke.
- TIME basiert daher auf HTTP-*Responses* mittels Chosen-plaintext-Attacken. Es werden Timing-Informationen benutzt, um auf die Größe des komprimierten Payloads zu schließen.
- Die HTTP-Requests werden mit JavaScript erzeugt, welches eine Schwachstelle von SOP ausnutzt. Da Multimedia tags von SOP ausgenommen sind (also Images von anderen Domanins zugelassen sind) kann automatisch und interaktiv die URL gesetzt werden und dabei die Load-Time gemessen werden. Damit wird SOP gebrochen (ein Datenleak zwischen einer zur anderen Domain zugelassen).
- **Gegenmaßnahmen**: Daher ist es nötig die „X-Frame-Options“ in die Seitenkonfiguration einzufügen. (Apache: Header always append X-Frame-Options SAMEORIGIN). Dieser Header sollte von allen Browsern unterstützt werden, wird aber in alten Browsern nicht unterstützt. X-Frame-Options sind ab folgenden Browsern unterstützt: Basic-support: Chrome 4.1.249.1042, Firefox 3.6.9, IE 8.0, Opera 10.5, Safari 4.0; ALLOW-FROM support: No Support in Chrome, FF 18.0, IE 8.0, No support in Safari, Opera ??

BREACH (*Browser Reconnaissance & Exfiltration via Adaptive Compression of Hypertext*)

- Nach CRIME fokussiert sich BREACH ähnlich wie TIME auf die Größe der Komprimierten HTTP-Responses und extrahiert Geheimnise aus dem Response-Payload. Es werden also wieder Information aus der Kombination von Kompression und Verschlüsselung benutzt.
- BREACH funktioniert ähnlich wie CRIME, diesmal nicht auf TLS- sondern auf HTTP-Kompromission (Nur der Body der Pakete wird dabei komprimiert). Es werden wieder geratene Zeichen in den HTTP-Request eingefügt und die Größe der Komprimierten und Verschlüsselten Responses gemessen. Eine kürzere Antwortzeit weißt auf ein richtig erratenes Zeichen im geheimen Wert hin. Dies wird dann Zeichen für Zeichen wiederholt.
- Es wird als Wahrscheinlich angenommen, dass BREACH vor allem benutzt werden könnte um CSRF-Token zu bekommen und damit CSRF zu knacken. Diese Token sollen zwar eigentlich bei jedem Request geändert werden, doch das ist nicht immer der Fall.
- **Gegenmaßnahmen**: Es gibt derzeit noch keine perfekten Gegenmaßnahmen. Die HTTP-Kompromission abzustellen würde das Problem beheben, aber für Server und WebClients einen erheblichen Overhead bedeuten. Bisher wurde bisher empfohlen, temporär, Django's Gzip Kompression in der Web-Server-Config abzuschalten um das Problem bezüglich der CSRF-Token zu beheben (Apache disable mod_deflate).

Lucky13

- Eine neue Timing-Attacke auf TLS/DTLS die einem man-in-the-middel-attacker erlaubt, den Klartext aus einer TLS-Verbindung zu erlangen. Die Attacke basiert auf einem timing-bug der TLS-Data-Decryption wenn der CBC-Modus benutzt wird (was Standard ist).
- Der MAC wird bei TLS nicht auf den verschlüsselten Text angewandt, sondern zuvor auf

den Klartext.. Nachgestellt wird ein Padding. Auf den ganzen Block wird dann die CBC-Verschlüsselung angewandt. Beim entschlüsseln wird dann der Block entschlüsselt, und mit dem vorherigen Verschlüsselten Block XORed. Danach wird das Padding überprüft und, nur wenn das Padding valide ist, die Daten mit dem MAC überprüft. Das Padding ist also nicht durch den MAC geschützt. Wenn hier ein Fehler auftritt, gibt der Server einen Fehler aus, in dem steht ob der MAC oder das Padding fehlerhaft sind. Seit der Padding-Orcle-Attacke werden keine expliziten Fehlermeldungen mehr zurückgegeben. Dies verhindert aber keine Timing-Attacke, da eine Zeitdifferenz zwischen einem fehlerhaften Padding und einem fehlerhaften MAC entsteht. Seit TLS1.1 wird die Session geschlossen, falls eine Entschlüsselung fehlschlägt. Es ist damit also nicht mehr möglich immer weiter Requests zu senden. Allerdings funktioniert die Attacke auch über mehrere verschiedene Sessions hinweg. Daher wird in TLS1.2 auch dann der MAC überprüft, wenn das Padding fehlerhaft ist. Wenn das Padding fehlerhaft ist, kann aber die Größe der Nachricht nicht mehr festgestellt werden und damit kann auch der MAC nicht wirklich berechnet werden. Daher wird der ganze Block benutzt um die MAC zu berechnen, was dann etwas länger dauert. Genau dieser Timing-Bug wird von Lucky13 ausgenutzt.

- Bei der Attacke werden wieder Byte für Byte eines Geheimnisses gesucht, in dem verschieden Werte durchprobiert werden. Ist der MAC falsch, so dauert die Überprüfung serverseitig ein kleines bisschen länger. Daa allerdings für die Lucky13 Attacke sehr geringe Netzwerk-Latenz notwendig ist und eine sehr große Anzahl an TLS-Handshakes notwendig werden, da für jedes Byte mehrere zehntausend Verbindungen aufgebaut werden müssen, scheint die Attacke nicht sehr wahrscheinlich.
- **Gegenmaßnahmen**: Die Benutzung von TLS1.2 wäre Ideal, da hier AES-GCM und AES-CCM implementiert sind und sie somit statt CBC als Authentifizierter Verschlüsselungs-Algorithmus benutzt werden könnten. TLS1.2 wird allerdings noch fast gar nicht eingesetzt. Randomisierte Timing-Verzögerungen zum Entschlüsselungsvorgang hinzuzufügen würden die Attacke nicht verhindern aber noch unwahrscheinlicher machen. Neue Versionen von openSSL (1.0.1d, 1.0.0k oder 0.9.8y) sowie GnuTLS (2.12.23, 3.0.28 und 3.1.7) haben Gegenmaßnahmen implementiert und sind daher zu empfehlen.

RC4 Biases in TLS

- RC4 ist eine sehr oft genutzte Stream-Cipher, unter anderem implementiert in SSL/TLS, vor allem aus Performance gründen. Neuerdings sind zwei Attacken (full plantext recovery attacks) auf RC4 veröffentlicht worden.
- RC4 benutzt einen key scheduling Algrotihmus (KSA) der ein initiales Array mit werten zwischen 0 und 255 befüllt und diesen für jeden Index des Arrays mit dem Schlüssel mischt. Weiterhin benutzt RC4 ein pseudo-random generation Algorithmus (PRGA) welcher das Array des KSAs für jedes einzelne Klartextzeichen durchmischt und den Output einzeln mit dem Klartextzeichen XORed um den verschlüsselten Text zu erhalten.
- Die neusten Attacken auf RC4 haben starke Vorhersehbarkeit (Bias) der ersten 257 Bytes der Ciphers gefunden, welche auf etwa den ersten 200 Bytes des Klartextes beruhen. (Beispielsweise ist das zweite verschlüsselte Byte mit der Wahrscheinlichkeit von 1/128 statt 1/256 eine 0.) Die einzelnen Klartextzeichen können nun mit den meist wahrscheinlichen biases berechnet werden, welche für diese stelle bekannt sind.
- **Gegenmaßnahmen**: Da die Attacke bisher nicht optimiert für den Gebrauch ist, ist es unwahrscheinlich, dass sie für den Großteil der Implementationen eine Gefahr ist. Es ist trotzdem davon auszugehen, dass RC4 keine Zukunft mehr hat. Und zu AES-GCM (Galois/Counter Mode) zu wechseln.

Forward Secrecy

- Wenn ein Schlüssel aufgedeckt wird kann auch aufgezeichnete, ältere Kommunikation entschlüsselt werden. Um das zu verhindern können Sitzungsschlüssel eingesetzt werden. Ein Angreifer, dem ein derartiger Sitzungsschlüssel bekannt wird, kann deshalb nur einen Teil der Daten entschlüsseln. Allerdings sind sämtliche Sitzungsschlüssel der Gefahr ausgesetzt, dass derjenige Langzeitschlüssel kompromittiert wird, der dafür verwendet wird die Sitzungsschlüssel selbst gesichert zu übertragen. Durch die Kenntnis dieses Langzeitschlüssel könnte ein möglicher Angreifer sämtlichen Datenverkehr entschlüsseln, insbesondere also auch die Übertragung der Sitzungsschlüssel und somit Zugriff auf den gesamten Datenverkehr erhalten.
- Dies wird durch *Perfect Forward Secrecy* unterbunden. Ein möglicher Angreifer kann trotz Kenntnis des Langzeitschlüssels keinerlei Rückschlüsse auf die ausgehandelten Sitzungsschlüssel ziehen. Die Sitzungsschlüssel sind nur den beiden involvierten Parteien bekannt. Technisch werden die Sitzungsschlüssel nach Verfahren ausgehandelt, welche auf dem Diffie-Hellman-Schlüsselaustausch basieren oder mit diesem verwandt sind.
- Einem c't-Artikel zufolge setzen nur wenige Web- und E-Mailserver das Verfahren ein, obwohl dadurch die Internetsicherheit verbessert werden könnte. Von den großen internationalen IT-Unternehmen war Google das einzige Unternehmen, das den Standard verwendete. Facebook hat angekündigt, ab Herbst 2013 PFS zu unterstützen.

Key-Pinning

- Typischerweise werden Zertifikate zur Überprüfung durch die Signatur Hierarchie validiert
- Pinning ist der Prozess des Zuordnens eines Hosts mit dem erwarteten X509-Zertifikat oder dem öffentlichen Schlüssel. Sobald ein Zertifikat oder der öffentliche Schlüssel bekannt ist, wird das Zertifikat oder den öffentlichen Schlüssel zugeordnet und an den Host 'gepinned'.
- Der Public-Key oder die Zertifikat-Details werden auf der Client-Seite gespeichert und mit jenen verglichen, die über das Netzwerk gesendet werden.
- **Tack** implementiert das Public-Key Pinning (da es aber noch nicht ausgereift ist, wird es bisher kaum eingesetzt?)