Der Heartbleed Bug

TEAM:

BERNHARD WAIDHOFER

KEVIN UTERLUTSCH

I. ABSTRACT

In diesem Dokument dreht sich alles um den sogenannten „Heartbleed-Bug“. Mein Team gibt einen Einblick über die Welt des Heartbleeds ebenso wie das Programm „OpenSSL“. Wie ist er entstanden? Wo war er zu finden? Wer war dafür verantwortlich? Was machte diesen Bug so gefährlich?
All diese Fragen werden in den folgenden Zeilen erörtert und genauestens beschrieben.
Ebenfalls ist zu erwähnen, dass unsere Informationen aus verschiedenen seriösen Quellen stammen.

II. GESCHICHTE DES HEARTBLEEDS

Dieser aller Welt bekannte Bug wurde im Jahre 2014 entdeckt. Doch was ist dieser „Heartbleed – Bug“ überhaupt?
Der Heartbleed-Bug war ein schwerwiegender Bug in älteren Versionen der Open-Source-Bibliothek OpenSSL. Dieser wurde 2012 unwissentlich mit der Heartbeat Erweiterung von OpenSSL
hinzugefügt, da den Entwicklern ein großer Implementierungsfehler unterlaufen war, welcher trotz mehreren Tests nicht aufgefallen ist.
Der Fehler betraf mehrere Millionen von Geräten und hatte massive Auswirkungen bzw. Folgen auf die Sicherheit des Internets.

III. DER HEARTBLEED

2012 hat eine kleine Entwicklergruppe von OpenSSL eine neue Version veröffentlicht. Dies war die OpenSSL Version 1.0.1 und fügte hiermit die sogenannte Heartbeat Erweiterung
hinzu. Durch diese Erweiterung war es möglich zu testen, ob der andere Kommunikationsteilnehmer einer Verbindung noch "lebt". Da die Implementation allerdings fehlerhaft war, konnten Angreifer mithilfe eines sogenannten Buffer Overreads sensible Daten wie Benutzernamen, Passwörter oder auch private Schlüssel von Serverzertifikaten zu stehlen.
2014 wurde die Sicherheitslücke durch das Sicherheitsteam von Google entdeckt und Tage später durch einen Patch gefixt. Da der Fehler 2 Jahre lang unbekannt blieb, ist dieser
bis heute (aktueller Stand 2020) immer noch nicht vollständig vom Internet verschwunden.

IV. OPENSSL

In dem vorherigen Kapitel ist das Wort „OpenSSL“ gefallen. Doch was bedeutet OpenSSL eigentlich?
OpenSSL ist ein robustes und kommerzielles Toolkit welches eng zusammen mit der Transport Layer Security (TLS) und Secure Sockets Layer (SSL) zusammenarbeitet. Ebenfalls ist OpenSSL auch als eine universelle Kryptografie Bibliothek bekannt.
OpenSSL wurde verwendet, um Zertifikate zu erzeugen, beantragen und zu verwalten.
Das in C geschriebene Programm bietet ebenfalls noch weitere Funktionen zum Ver- und Entschlüsseln an.

V. GESCHICHTE OPENSSL

Das Programm OpenSSL ist recht interessant. Deshalb hat mein Team beschlossen die Geschichte mit in dieses Dokument zu nehmen.
Wie bereits erwähnt handelt es sich bei diesem Programm um ein C geschriebenes Programm, dass erstmals 1998 veröffentlicht wurde.
Mitte der 1990er Jahre ermöglichte SSLeay auch außerhalb der USA mit starker Verschlüsselung zu kommunizieren.
Damit wurde eine sichere Basis zwischen den USA und anderen Ländern gewährleitet.
Aus was bildete sich der Name „OpenSSL“?
Der Name bildete sich aus den Initialen des Netzwerkprotokolls und des Programmierers.

VI. Schwachstellen der OpenSSL

Nicht nur der Heartbleed Bug zählt zu den bekanntesten Bugs, die das Programm OpenSSL hinnehmen musste. Neben den Heartbleed Bug gab es diverse andere Sicherheitslücken. Ein Beispiel wären die „schwachen Schlüssel“. Dieser schwerwiegende Fehler wurde von einem Team des „Debian-Projekts“ 2008 bekannt gegeben und entdeckt. Das Team erkannte eine Sicherheitslücke in einem OpenSSL-Paket der Distribution. Laut Angaben des Entwickler-Teams existierte diese Sicherheitslücke seit 2006.
Doch was war die Sicherheitslücke?
Durch einen Fehler in einem Patch, war es möglich die Zahlen, die „zufällig“ generiert werden sollten, vorherzusagen.
Wer war davon betroffen?

1. SSH
2. OpenVPN
3. DNSSEC
4. Schlüssel in X.509 Zertifikaten
5. Sitzungsschlüssel, die in SSL/TLS Verbindungen genutzt wurden

Schlüssel, die mit GnuPG oder GnuTLS erzeugt wurden, waren nicht betroffen.

VII. SSL und TLS

Mit OpenSSL war es möglich sich sicher im World Wide Web zu bewegen. Dies war möglich durch die SSL und TLS Verschlüsselung, um Daten sicher von A nach B zu bringen.
Besonders bei Transaktionen war eine SSL und TLS Verschlüsslung überaus wichtig.
Dazu ein Beispiel:
Ein Verkäufer (Peter) möchte einen Online Shop erstellen, um seine Waren außerhalb des Geschäftes (online) zu verkaufen. Es gibt jetzt natürlich 2 Varianten wie man einen Online Shop „aufsetzen“ kann.
Variante 1 wäre unverschlüsselt (Ohne SSL und/oder TLS). Dies hat zufolge, dass bei einer Transaktion zwischen Käufer und Händler sich Dritte Zugriff auf Kundendaten verschaffen können. Was zu Folge hat, dass sich Angreifer (Dritte) sich mit den Daten der Kunden weitere Transaktionen möglich sind, ohne dass der Eigentümer (ursprünglicher Kunde) davon etwas merkt.
Diese Methode ist selbstverständlich inakzeptabel und wird so in keinen seriösen Unternehmen gehandhabt.
Es gibt natürlich noch die 2. Variante.
Hier wird die Transaktion von SSL und/oder TLS Verschlüsselung geschützt.
Dazu muss sich der Verkäufer (Peter) eine Zertifizierungsanforderung (CSR) an eine Zertifizierungsstelle (CA) schicken. Die CA prüft natürlich die Identität des Servers sowie den Inhaber der Website. Ist alles in Ordnung, so wird eine Zertifizierung (SSL) an den Verkäufer vergeben. Diese SSL ist unique und ist ausschließlich für den Verkäufer bereitgestellt.

VIII. OpenSSL und Verschlüsselung

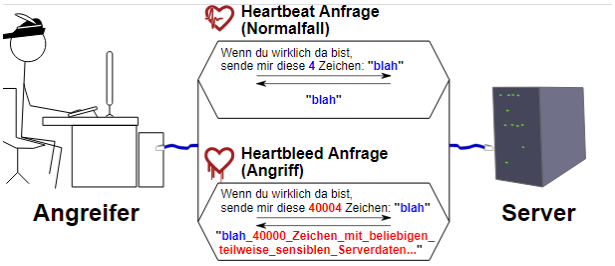
Was hat OPENSSL jetzt mit SSL und TLS zu tun?
Im Grunde genommen ist das Programm OPENSSL eine „Erleichterung“. Ohne diesem Programm war es nicht einfach ein Zertifikat zu erstellen. Man müsste den Quellencode mühsam per Hand schreiben. Was sehr viel Zeit in Anspruch genommen hat.

IX. Die Heartbeat-Erweiterung

Die Heartbeat-Erweiterung der Bibliothek OpenSSL wurde 2012 eingeführt. Mit dieser Funktion war es möglich Verbindungen den Kommunikationsteilnehmer die Anteilnahme des Partners zu überprüfen.
Hierbei schickt einer der Teilnehmer dem Anderen eine gewisse Menge an Daten (bis zu 16 KByte), um zu sehen ob die Verbindung überhaupt noch besteht.
Konkret sieht der Ablauf folgendermaßen aus:
1. A sendet B eine Heartbeat Anfrage. Diese beinhaltet die payload (ein beliebiger String) und die payload_length (die Länge von payload).
2. B antwortet A mit einer Heartbeat Antwort. Diese beinhaltet die gleiche payload wie bei der Anfrage.
3. A erhält die Antwort von B und prüft, ob die payload aus der Anfrage gleich dem String aus der Antwort ist.

X. HEARTBLEED FUNKTIONALITÄT

Wie kommt der Bug jetzt zustande bzw. wie kann dieser ausgenutzt werden?
Aufgrund der fehlerhaften Implementierung des 2. Schrittes der Heartbeat-Erweiterung, kann ein Angreifer aus der Gegenstelle unzählige Daten erhalten. Das ist deswegen der
Fall, da bei jeder Speicherzuweisung eines payload Buffers die payload_length hergenommen wird. Ohne Prüfung der payload_length mit der wirklichen Länge von payload kann der Angreifer einen Buffer Overread hervorrufen, mit dessen es möglich ist, den restlichen Buffer mit Daten benachbarter Speicherbereiche aufzufüllen, die Benutzernamen und
Passwörter sein könnten.
Ein beispielhafter Ablauf des Leaks sieht folgendermaßen aus:
1. Der Angreifer sendet eine Heartbeat Anfrage. Diese beinhaltet eine 5 Byte payload und eine 2000 Byte payload_size.
2. Die Gegenstelle antwortet mit einer Heartbeat Antwort. Diese beinhaltet die eigentliche 5 Byte payload, aufgefüllt auf 2000 Byte mit 1995 Byte Daten benachbarter Speicherbereiche.



Unterschied Heartbeat/Heartbleed (https://cutt.ly/HeartBeat)

XI. BUFFER OVERREAD

In dem vorherigen Kapitel wurde der sogenannte „Buffer Overread“ genannt.
Doch was bedeutet er?
Beziehungsweise, was hatte er mit den „Heartbleed“ zu tun. In diesem Kapitel werden all diese Fragen beantwortet.
Ein Buffer Overread ist in der Programmierung eine Anomalie, bei der ein Programm während des Lesens eines Buffers über dessen Grenzen hinaus geht und benachbarte Speicherbereiche liest. Dieser funktioniert beispielsweise folgendermaßen:
Das Programm versucht nun alles was innerhalb eines Buffers liegt zu lesen.
Nehmen wir an, dass der Buffer selbst eine Grenze von 2000 Bytes hat. Allerdings möchte er Daten von 2500 Bytes lesen.
Das Programm musste irgendwo die restlichen 500 Bytes herbekommen, doch woher?
Der benachbarte Speicher liefert die Antwort.
Da die Daten innerhalb eines Buffers komplett ausgelesen sind, werden die „restlichen Daten“ einfach aus dem Arbeitsspeicher gelesen.
Was hat das nun zufolge?
Es ist ein massives Problem, denn wenn ein Angreifer eine Buffer Anfrage sendet, welche vom Empfänger nicht rein von den Daten außerhalb des Arbeitsspeichers gefüllt werden kann, ist der Empfänger „gezwungen“ interne Daten weiter zu geben.
Das macht einen Buffer Overread, vor allem in Programmiersprachen wie C und C++, so gefährlich, da diese keinen Schutz gegen unerlaubte Pointerzugriffe gewähren können. Das ist beispielsweise der Fall bei der Verwendung der Methode memcpy. Zum Schutz gegen solcher Buffer Overreads, müssen die jeweiligen Grenzen immer vor der Verwendung solcher gefährlichen Methoden geprüft werden.
Zur Veranschaulichung eines Buffer Overreads dient folgende Abbildung:

```
2580  buffer = OPENSSL_malloc(1 + 2 + payload +
          padding);
2581  bp = buffer;
2582
2583  /*Enter response type, length and copy payload*/
2584  *bp++ = TLS1_HB_RESPONSE;
2585  s2n(payload, bp);
2586  memcpy(bp, pl, payload);
2587  bp += payload;
2588  /* Random padding */
2589  RAND_pseudo_bytes(bp, padding);
2590
2591  r = ssl3_write_bytes(s, TLS1_RT_HEARTBEAT,
          buffer, 3 + payload + padding);
```

Code Ausschnitt aus t1_lib.c in openssl-1.0.1f

XII. SCHLUSSFOLGERUNG

Das Programm „OpenSSL“ galt als eines der berühmtesten Softwares seiner Zeit. Trotz seiner vielen Fehler wurde das Programm doch weitestgehend/weltweit genutzt.
Ebenso gilt der Heartbleed-Bug als einer der bedeutendsten Bugs in der Geschichte des Internets.
Sicherheitsmaßnahmen wurden verstärkt, Initiativen wie die Core Infrastructure Initiative, die kleinere IT-Projektgruppen wie das damalige OpenSSL Team unterstützen, wurden eingeleitet, sowie Teams wie beispielsweise das Project Zero von Google wurden gegründet, die sich mit der Entdeckung von unbekannten Bugs beschäftigen.