Der Heartbleed-Bug

Maximilian Bachl

7. August 2014

1 Einleitung

Der „Heartbleed“ genannte Bug, der am 1. April 2014 in der Verschlüsselungssoftware OpenSSL gefunden wurde, ist von renommierten Sicherheitsexperten, wie etwa Bruce Schneier, als *katastrophal* eingestuft worden (Bruce Schneier, 2014). Ein anderer Spezialist schrieb im Forbes-Magazin, dass es sich hierbei um den schwerwiegendsten Fehler seit dem Beginn des kommerziellen Internets handelt (Joseph Steinberg, 2014).

Zu den Seiten, die betroffen waren, zählen viele sehr bekannt Web-Portale wie etwa Google, Amazon, Facebook, Reddit u.a. Auch das Betriebssystem „Android“ hatte eine fehlerhafte OpenSSL-Implementierung. Es wird geschätzt, dass OpenSSL auf circa zwei Drittel aller verschlüsselten Webseiten benutzt wird (Netcraft, 2014).

Bei der Sicherheitslücke in der Implementierung der „Heartbeat“-Erweiterung (Limer, 2014) handelt es sich um einen sogenannten „Buffer-Overread“. Das bedeutet, dass bei einer manipulierten Anfrage mehr Informationen ausgelesen und übertragen werden können, als eigentlich laut Spezifikation gewollt ist. Genauer gesagt ist es möglich bis zu 64kB an zufälligen Informationen aus dem Speicher des angegriffenen Systems zu erhalten.

Die besondere Gefahr dieses Softwarefehlers resultiert daraus, dass er mehrere Jahre existierte, bevor er öffentlich bekannt wurde. Auch als der Bug erkannt und behoben wurde, haben viele Webseiten ihre OpenSSL-Software nicht aktualisiert, wodurch es einige Tage lang möglich war private Daten von betroffenen Websites zu stehlen. Besondere Brisanz erhält der Bug weiterhin durch die Tatsache, dass Quellen behaupten (Riley, 2014), die NSA hätte von dem Fehler schon seit einem längeren Zeitraum gewusst und diesen auch gezielt ausgenutzt. Wenn dies stimmt, würde das die Spionagevorwürfe, die im letzten Jahr wegen des „Prism“-Programms aufkamen, in neues Licht rücken.

Im Folgenden möchte ich die konkreten Auswirkungen von Heartbleed auf das Internet und seine Nutzer thematisieren. Außerdem möchte ich die Fehler analysieren, die das Problem überhaupt erst ermöglicht haben.

2 Die Geschichte des Fehlers

Mitarbeiter von Google und Codenomicon entdeckten den Fehler Ende März/Anfang April 2014 unabhängig voneinander. Um die Nutzer des Internets vor Kriminellen zu schützen, wurde der Fehler erst öffentlich gemacht, nachdem das OpenSSL-Team ihn am 7. April behoben hat (Grubb, 2014) und eine offizielle Nachricht ausgesendet hat (OpenSSL, 2014). Einige große Anbieter wie Facebook oder das Content-Distribution-Network „CloudFlare", wurden schon vorher durch verschlüsselte Nachrichten über den Fehler informiert, und so konnten sie das Problem schon vor dem öffentlichen Bekanntwerden beheben. Es gab aber auch zahlreiche Services wie Twitter und Amazon, die von der Sicherheitslücke erst erfuhren, als sie veröffentlicht wurde. Somit konnten zumindest bei diesen Seiten private Informationen gestohlen werden, bis auch sie OpenSSL upgedatet hatten.

Außerdem fand man im Zuge der darauffolgenden Recherchen heraus, wann der Fehler „committet", also in die zentrale Quellcodeverwaltung des OpenSSL-Projekts eingespeist wurde. Dies geschah zu Silvester im Jahr 2011 (Brodkin, 2014). Der dafür verantwortliche Programmierer ist ein Robin Seggelmann aus Duisburg, der beteuert, den Fehler versehentlich gemacht zu haben (Schulz, 2014).

Rechnet man von der Entstehung von Heartbleed bis zu seiner Entdeckung, so sieht man, dass die schadhafte Version von OpenSSL 27 Monate im Umlauf war und der Fehler so lange ausgenutzt werden konnte.

3 Auswirkungen

Nun stellt sich die Frage, welche Daten Angreifer durch Heartbleed erbeuten konnten. Prinzipiell kann ein Angreifer durch den Fehler den gesamten Speicherbereich von OpenSSL auslesen. Dazu muss er nur genug Angriffe durchführen (Codenomicon, 2014).

Die betroffenen Daten, die sich im Speicherbereich von OpenSSL befinden, werden von Codenomicon (2014) in vier Kategorien eingeteilt:

1. Die **privaten Schlüssel** des SSL-Zertifikats des Servers. Bei dem Zertifikat handelt es sich um ein sogenanntes X.509-Zertifikat, das benutzt wird, damit

der Client die Identität des Servers bestätigen kann. Hat man den privaten Schlüssel erbeutet, kann man den gesamten verschlüsselten Datenverkehr des Servers entschlüsseln und mitlesen. Dies gilt ebenso für alle verschlüsselten Daten, die in der Vergangenheit gesendet wurden.

Des Weiteren kann sich ein Angreifer, wenn er den Schlüssel hat, als die Webseite ausgeben, von der er den Private-Key erbeutet hat. Dies ermöglicht etwa Phishing-Angriffe.

Um diese Angriffsmöglichkeit zu unterbinden, muss der betroffene Web-Service seine Zertifikate als ungültig markieren und neue erstellen. Erst wenn dies erfolgt ist, können keine neuen verschlüsselten Daten mehr von Angreifern mitgelesen werden.

Die besondere Problematik hierbei ist, dass es einige Web-Browser gibt, die die Sperrlisten für Zertifikate nicht überprüfen oder nicht zuverlässig überprüfen (Seltzer, 2014). Somit kann dieses Problem erst entschärft werden, wenn die betroffenen Zertifikate abgelaufen sind (Zertifikate verlieren nach einer gewissen Zeitdauer ihre Gültigkeit). Dies ist bei den meisten Zertifikaten aber erst nach zwei bis fünf Jahren der Fall (Huston, 2009).

2. Die **Session-Keys, Usernamen und Passwörter** der Benutzer der betroffenen Web-Services.

3. **Geschützter User-Inhalt**, wie etwa vertrauliche Nachrichten (etwa auf Facebook). Ebenso zählen hierzu Kreditkartendaten und andere Zahlungsinformationen, die in den falschen Händen großen finanziellen Schaden verursachen können.

4. **Technische Detailinformationen**, wie etwa die benutzten Speicheradressen und genaue Informationen zur benutzten OpenSSL-Implementierung. Diese Daten sind die am wenigsten hilfreichen, da sie nach einem Upgrade der Sicherheitssoftware nicht mehr zu gebrauchen sind. Auch ein simpler Neustart der Sicherheitssoftware lässt die Speicheradressen unbrauchbar werden, da bei jeder Neuinitialisierung des Programms der Speicher neu zugewiesen wird.

4 Technische Grundlagen

Wie schon erwähnt handelt es sich bei dem Fehler um einen „Buffer-Overread“ in der „Heart-Beat“-Erweiterung von OpenSSL (Cassidy, 2014). Dieses neue Feature ermöglicht es, verschlüsselte Verbindungen auch nach der Übertragung der Daten

aufrecht zu erhalten. Somit muss die Verbindung später nicht nochmals aufgebaut werden, wenn wieder Daten gesendet werden, was Zeit spart.

Die Funktionsweise des Heart-Beats wird im Folgenden erklärt. Die beiden Beteiligten sind hier Person A und Person B.

1. Person A will sicherstellen, ob die Verbindung zu B noch existiert und fragt daher B, ob er noch da ist.
2. Dazu sendet A der Person B ein Paket, etwa mit dem Inhalt „Blubb“. Er sagt dazu, dass „Blubb“ 5 Zeichen lang ist.
3. B erhält die Nachricht und antwortet mit der Nachricht, die A ihm geschickt hat. Er sendet also wieder „Blubb“ zurück.
4. Nun weiß A, dass die Verbindung zu B noch besteht.

Der Heartbleed-Bug resultiert daraus, dass B nicht überprüft, ob die Nachricht auch tatsächlich so lange ist, wie A ihm sagt (Limer, 2014). Angenommen A ist ein Angreifer, so würde A die Nachricht „Blubb“ an B senden, aber sagen, dass „Blubb“ 500 Zeichen lang ist. B würde das nicht überprüfen und die Antwort würde folgendermaßen aussehen: „Blubb<*495 beliebige andere Zeichen aus dem Speicher von OpenSSL*>“.

Durch den Angriff kann A also Daten aus dem Speicherbereich von OpenSSL bekommen. Die Höchstgrenze sind $2^{2 \cdot 8} = 2^{16} = 65536$ Bytes, da das Feld für die Länge in der Heart-Beat-Nachricht zwei Bytes lang ist.

5 Einsatz von Heartbleed

Da ein Angreifer, der Heartbleed ausnutzt kaum Spuren hinterlässt, ist es schwierig zu sagen, wo und wann es eingesetzt wurde (**?**).

Es gibt aber einige Fälle von denen man weiß, allerdings keine besonders schwerwiegenden Fälle. So wurden etwa kanadische Sozialversicherungsdaten gestohlen sowie auch Daten des britischen Erziehungsratgebers „Mumsnet“.

Allerdings wurde in diesen beiden Fällen der Fehler erst nach der Veröffentlichung ausgenutzt, bevor er von den betroffenen Seiten behoben wurde.

5.1 Nutzung durch die National Security Agency (NSA)

Das Nachrichtenportal „Bloomberg“ schrieb, es habe zwei Insider aus der NSA interviewt, die behaupten, die NSA habe vom Heartbleed-Bug schon länger gewusst

und diesen auch aktiv genutzt. Laut dieser Kontaktpersonen habe der Inlandsgeheimdienst den Fehler auch bewusst nicht veröffentlicht, um ihn länger für seine eigenen Zwecke nutzen zu können. Angeblich wurden so Passwörter und alle anderen Arten privater Nutzerdaten erbeutet. (Riley, 2014)

Die NSA wies die Vorwürfe zurück (**?**) und sagte, auch sie hätten von dem Fehler nur aus den Medien erfahren.

Laut Riley (2014) ist sonst niemand bekannt, der den Fehler vor der Veröffentlichung genutzt haben könnte.

6 Mitwirkende Faktoren und Ursachen

Hier soll auf Faktoren eingegangen werden, die zum Heartbleed-Bug beigetragen haben und ihn ermöglicht haben.

6.1 Linus' Law

Lange Zeit dachte man, dass Open-Source-Software (OSS) prinzipiell sicherer als Closed-Source-Software ist, weil bei OSS der Quellcode öffentlich ist und ihn somit auch mehr Leute auf seine korrekte Funktionsweise überprüfen können.

Diese Mentalität wird in Eric Raymonds „Linus' Law" zum Ausdruck gebracht: *„Given enough eyeballs, all bugs are shallow"*. Durch die große Anzahl an Entwicklern, die sich den Code ansehen, werden also Fehler tendenziell unwahrscheinlich.

Bei OpenSSl hat dieses Gesetz allerdings nicht gegolten. Die Gründe hierfür werden im nächsten Absatz erläutert.

6.1.1 Das OpenSSL-Team

Zum Fehler beigetragen hat der Aufbau des OpenSSL-Teams an sich. OpenSSL hat nur einen einzigen Vollzeitentwickler, nämlich den britischen Mathematiker Stephen Henson. Dies ist nicht ausreichend angesichts der Tatsache, dass das Projekt mittlerweile aus über 400000 Zeilen Programmcode besteht. Um so eine große Menge an Code zu warten reicht eine Person nicht aus. (Stokel-Walker, 2014; Vaughan-Nichols, 2014)

Als Heartbleed öffentlich wurde haben viele große Institutionen erkannt, dass derart essentielle Projekte wie OpenSSL mehr finanzielle Ressourcen brauchen und darum wurde die „Core Infrastructure Initiative" gegründet. Das Ziel ist, durch mehr Entwickler solche Fehler in Zukunft vermeiden zu können.

Man kann also sagen, dass Linus' Law zwar prinzipiell gilt: Wenn mehr Leute den Code betrachten, werden Fehler eher gefunden. Bei vielen Projekten, wie bei OpenSSL, gibt es aber schlicht nicht genug Leute, die den Code auf Fehler überprüfen.

6.2 Verwendung der Programmiersprache *C*

Wie schon erwähnt ist die Ursache für Heartbleed ein „Buffer-Overread". Diese Art von Fehlern kann nur in Programmiersprache entstehen, die die Größenbeschränkungen von Zeichenketten nicht überprüfen. C ist eine der wenigen populären Sprachen, die den Zugriff auf sogenannte „Strings", also Zeichenketten, nicht überprüfen (Wheeler, 2014). In anderen Programmiersprachen wie etwa Java oder Python, könnte Heartbeat nicht funktionieren, da ein Fehler ausgegeben werden würde, wenn z.B. beim String „Blubb" auf das vierhundertste Element zugegriffen werden soll.

Der Grund, warum C trotz dieser Einschränkungen benutzt wird, ist, dass C sehr alt ist und darum viel Software vorhanden ist, die in dieser Programmiersprache geschrieben sind. Außerdem ist C relativ einfach und hat nur geringe Komplexität (Cheng, 2011).

6.3 Fokus auf Performance zulasten der Sicherheit

Ein weiterer beitragender Faktor ist, dass OpenSSL eine eigene Implementierung der „malloc"-Funktion benutzt. Mit „malloc" fordern Programme Speicherplatz an und reservieren neuen Speicher. Man hat diese Funktion in OpenSSL neu implementiert, da in den 1990er Jahren, also also das Projekt ins Leben gerufen wurde, gängige Implementierungen von „malloc" keine gute Performance lieferten.

Die Neuimplementierung verwaltet den Speicher allerdings durch eine Liste, die alle freien Speicherbereiche auflistet. Diese Liste funktioniert nach dem Last-In-First-Out-Prinzip. Wenn OpenSSL neuen Speicher verlangt, so bekommt es immer den Speicherabschnitt, der zuletzt freigegeben wurde. (Unangst, 2014)

Das Problem hierbei ist, dass dadurch immer ein neuer Speicherbereich zurückgegeben wird, der noch sehr aktuelle Daten enthält. Darum erhalten Angreifer bei Heartbleed immer aktuelle Daten. (Higgins, 2014)

Ein anderes Problem ist, dass man bei OpenSSL neuen Speicher nicht mit Nullen überschreibt (López, 2014). Das heißt, dass neu reservierte Speicherbereiche immer noch alte Informationen enthalten. Auch dies hat man wegen der Performance gemacht, da das Initialisieren mit Nullen eine gewisse Zeit dauert.

7 Schlussfolgerung

Eine große Rolle bei Heartbleed spielte menschliches Versagen, da der fehlerhafte Code von vor der Veröffentlichung jemand anderem (dem OpenSSL-Hauptentwickler) reviewt wurde und der Bug auch von ihm nicht entdeckt wurde (Timson, 2014). Es gab also zwei Menschen, den Programmierer, Robin Seggelmann, und den Überprüfer, Stephen Henson, die den Fehler nicht erkannt haben.

Es ist aber meiner Meinung nach nicht ausschließlich die Schuld dieser beiden Entwickler, da es für so wenige Personen beinahe unmöglich ist, den Überblick über ein Projekt mit mehrerer 100000 Zeilen Code zu behalten. Das „Linus' Law" ist zwar prinzipiell gültig, aber Open-Source-Software ist nur dann sicherer als Closed-Source, wenn der Code auch tatsächlich gelesen und überprüft wird. Dies war bei OpenSSL nicht in ausreichendem Maß der Fall.

Allerdings wurde dieses Problem nach Heartbleed von vielen Konzernen erkannt und mit der „Core Infrastructure Initiative" wird in Zukunft durch mehr finanzielle Mittel entgegengesteuert.

Ein anderer wichtiger Aspekt ist, dass selbst nachdem der Fehler bereits implementiert war, es normalerweise einige Sicherheitsmechanismen geben müsste, die die Auswirkungen von Heartbleed minimiert hätten.

Diese Mechanismen wurden jedoch außer Kraft gesetzt, etwa durch die Verwendung der verhältnismäßig unsicheren Programmiersprache C, sowie das Neuimplementieren der Speicherverwaltung als auch der Verzicht auf das Initialisieren des Speichers mit Nullen. Dies sind alles Folgen eines zu großen Fokus auf Geschwindigkeit im OpenSSL-Projekt. Hierbei wurde aber durch den Schwerpunkt auf Performance gleichzeitig die Sicherheit nicht genügend beachtet.

8 Literatur

[Brodkin 2014] Brodkin, Jon: *Heartbleed developer explains OpenSSL mistake that put Web at risk*. http://arstechnica.com/information-technology/2014/04/heartbleed-developer-explains-openssl-mistake-that-put-web-at-risk/. Version: April 2014

[Bruce Schneier 2014] Bruce Schneier: *Heartbleed*. https://www.schneier.com/blog/archives/2014/04/heartbleed.html. Version: 2014

[Cassidy 2014] Cassidy, Sean: *Diagnosis of the OpenSSL Heartbleed Bug*. http://blog.existentialize.com/diagnosis-of-the-openssl-heartbleed-bug.html. Version: 2014

[Cheng 2011] Cheng, Harry H.: *Ten Reasons to Teach and Learn Computer Programming in C?* http://iel.ucdavis.edu/publication/WhyC.html. Version: 2011

[Codenomicon 2014] Codenomicon: *Heartbleed Bug.* http://heartbleed.com. Version: April 2014

[Grubb 2014] Grubb, Ben: *Heartbleed disclosure timeline: who knew what and when.* http://www.smh.com.au/it-pro/security-it/heartbleed-disclosure-timeline-who-knew-what-and-when-20140415-zqurk.html. Version: 2014

[Higgins 2014] Higgins, Kelly J.: *Did A Faulty Memory Feature Lead To Heartbleed?* http://www.darkreading.com/vulnerabilities---threats/did-a-faulty-memory-feature-lead-to-heartbleed/d/d-id/1204505. Version: 2014

[Huston 2009] Huston, Brent: *Best Practices for Certificate Expiration | MSI :: State of SecurityMSI :: State of Security.* http://stateofsecurity.com/?p=548. Version: 2009

[Joseph Steinberg 2014] Joseph Steinberg: *Massive Internet Security Vulnerability – Here's What You Need To Do.* http://www.forbes.com/sites/josephsteinberg/2014/04/10/massive-internet-security-vulnerability-you-are-at-risk-what-you-need-to-do/. Version: 2014

[Limer 2014] Limer, Eric: *How Heartbleed Works: The Code Behind the Internet's Security Nightmare.* http://gizmodo.com/how-heartbleed-works-the-code-behind-the-internets-se-1561341209. Version: 2014

[López 2014] López, Javier S.: *How Java™ Could Have Prevented Heartbleed.* http://www.forumsys.com/api-security/java-prevented-heartbleed/. Version: 2014

[Netcraft 2014] Netcraft: *April 2014 Web Server Survey.* http://news.netcraft.com/archives/2014/04/02/april-2014-web-server-survey.html. Version: 2014

[OpenSSL 2014] OpenSSL: *OpenSSL Security Advisory.* https://www.openssl.org/news/secadv_20140407.txt. Version: 2014

[Riley 2014] Riley, Michael: *NSA Said to Exploit Heartbleed Bug for Intelligence for Years.* http://www.bloomberg.com/news/2014-04-11/

nsa-said-to-have-used-heartbleed-bug-exposing-consumers.html. Version: April 2014

[Schulz 2014] Schulz, Stefan: Programmierer über seinen „Heartbleed"-Bug „Ein Fehler, keine Beteiligung, kein Zusammenhang". In: *Frankfurter Allgemeine Zeitung* (2014), April. http://www.faz.net/aktuell/feuilleton/debatten/ueberwachung/programmierer-nennt-heartbleed-bug-einen-fehler-12891134.html. – ISSN 0174–4909

[Seltzer 2014] Seltzer, Larry: *Chrome does certificate revocation better.* http://www.zdnet.com/chrome-does-certificate-revocation-better-7000028589/. Version: 2014

[Stokel-Walker 2014] Stokel-Walker, Chris: *The Internet Is Being Protected By Two Guys Named Steve.* http://www.buzzfeed.com/chrisstokelwalker/the-internet-is-being-protected-by-two-guys-named-st. Version: 2014

[Timson 2014] Timson, Lia: *Who is Robin Seggelmann and did his Heartbleed break the internet?* http://www.smh.com.au/it-pro/security-it/who-is-robin-seggelmann-and-did-his-heartbleed-break-the-internet-20140411-zqtjj.html. Version: 2014

[Unangst 2014] Unangst, Ted: *heartbleed vs malloc.conf*. http://www.tedunangst.com/flak/post/heartbleed-vs-mallocconf. Version: 2014

[Vaughan-Nichols 2014] Vaughan-Nichols, Steven J.: *Cash, the Core Infrastructure Initiative, and open source projects.* http://www.zdnet.com/cash-the-core-infrastructure-initiative-and-open-source-projects-7000028842/. Version: 2014

[Wheeler 2014] Wheeler, David A.: *How to Prevent the next Heartbleed.* http://www.dwheeler.com/essays/heartbleed.html. Version: 2014