Analyse von IT-Sicherheitsproblemen

Torge Hinrichs
torge.hinrichs@haw-hamburg.de

Hamburg University of Applied Sciences,
Dept. Computer Science,
Berliner Tor 7
20099 Hamburg, Germany

Zusammenfassung. Diese Arbeit befasst sich mit der Analyse des „Heartbleed"-Problems. Dabei wird die Problemstellung in die Risikobewertung des US-CERT eingeordnet. Der Fehler wird innerhalb des SSL-Protokolls und innerhalb der openSSL Implementierung im Detail betrachtet. Auf Basis der Klassifizierung als input-validation-Problem werden Ansätze in der aktuellen Literatur betrachtet, die als Grundlage für die Identifizierung verwendet werden.

Schlüsselwörter: Heartbleed, openSSL, input-validation, IT-Security, Testen und Analyse

1 Einleitung

IT-Systeme haben sich als fester Bestandteil einer modernen Gesellschaft etabliert. So haben laut dem Statistischen Bundesamt [Bun15] im Jahr 2015 etwa 92% der in Deutschland ansässigen Unternehmen Computersysteme im Einsatz. Leider wird die Sicherheit dieser Systeme oft vernachlässigt. Der Digitalverband Deutschlands „bitkom" veröffentlichte 2015 eine repräsentative Umfrage in der mehr als 1000 Sicherheitsverantwortliche aus verschiedenen Unternehmen befragt wurden[bit15]. Aus dieser Studie resultiert, dass etwa jedes zweite Unternehmen in Deutschland zum Opfer von Cyberangriffen innerhalb der letzten zwei Jahre wurde. Die dabei entstandenen Schäden für die deutsche Wirtschaft belaufen sich auf schätzungsweise 51 Milliarden Euro pro Jahr. Die Ursachen für diese Probleme werden im „Sophos Security Threat Report 2015"[Lyn15] genauer beleuchtet. Diese sind unter anderem:

Exploits Hierbei werden Schwachstellen in verschiedensten Software Systemen, wie Browsern oder Betriebssystemen dazu genutzt Schadcode auf dem Zielsystem auszuführen.

Internet-of-Things-Angriffe Das Internet der Dinge beschreibt allgegenwärtige oft kleine Computersysteme, Sensoren o.ä., die miteinander vernetzt werden. Angriffe auf diese Systeme sind laut dem Sophos Report[Lyn15] sehr leicht, da diese Geräte sehr klein und günstig sind und damit wenig Möglichkeiten für grundlegende Sicherheitskonzepte bleiben. Des weiteren

werden diese Geräte so schnell zur Marktreife entwickelt, dass keine Zeit für Überprüfungen des Quellcodes bleibt.

Falsch konfigurierte Schlüssel In den vergangenen Jahren zeichnete sich eine positive Entwicklung beim Einsatz von Verschlüsselungen ab. Es werden immer mehr Verschlüsselungen eingesetzt, jedoch werden hierbei oft veraltete Methoden wie der „Data Encryption Standard (DES)“ oder falsch konfigurierte Verfahren verwendet, die nur eine illusorische Sicherheit darstellen.

Schere zwischen Verantwortung und Ausbildung Die Verantwortung für die IT-Sicherheit steigt stetig für die zuständigen Personen, jedoch wird die Aus- und Weiterbildung dieses Personals nur selten weiter voran getrieben, so entsteht ein Defizit zwischen Verantwortung und notwendiger Ausbildung.

Um diese Probleme einordnen und beheben zu können hat das amerikanische „National Institute of Standards and Technology“ (NIST) eine Handlungsrichtlinie [Sca12] veröffentlicht. Mit dieser können die verschiedene Arten der IT-Sicherheitsprobleme erkannt, analysiert und behoben werden. Auf Basis dieser Richtlinien konnte das „United States Computer Emergency Response Team“ (US-CERT) eine Gliederung der Probleme in folgende Kategorien vornehmen [UC14]. Die Tabelle 1 zeigt die Ordnung verschiedener IT-Sicherheitsvorfälle in Kategorien; dabei wird unterschieden zwischen der Kategorie 0, die ausschließlich zu Testzwecken eingesetzt wird, und den übrigen Kategorien. Hierbei nimmt die Schwere des Vorfalls mit steigender Kategorie ab.

Diese Arbeit befasst sich mit Vorfällen der Kategorie 1. Alle hier eingegliederten Ereignisse bezeichnen unerlaubte Zugriffe auf Systeme, Netzwerke, Applikationen, Daten oder andere Ressourcen. Dies hat unmittelbar zur Folge, dass Sicherheitsmaßnahmen umgangen wurden. Daher muss der Umgang mit diesen Vorkommnissen hoch priorisiert werden. Als Demonstrationsbeispiel für diese Art von Vorfällen wird die Schwachstelle in der TLS/SSL Bibliothek„OpenSSL“[Pro14a] betrachtet. Hierbei handelt es sich um den „Heartbleed“-Fehler (CVE-2014-0160) [MIT14]. Dieser ermöglicht Bereiche bis zu 64 Kilobytes aus dem Hauptspeicher zu lesen und somit Klartextpasswörter oder andere vertrauliche Informationen auszuspähen.

1.1 Problemstellung

Ziel dieser Arbeit ist es Ansätze zu identifizieren, mit denen der „Heartbleed“-Fehler im Vorfeld erkannt hätte werden können. Hierbei ist es wichtig, dass dieses nur in einem ersten Ansatz auf „Heartbleed“ angewendet werden soll. In weiteren Arbeiten soll dieser dann generalisiert auf ähnliche Probleme angewendet werden. Der folgende Abschnitt befasst sich mit den Grundprinzipien von OpenSSL sowie einer detaillierte Beschreibung des „Heartbleed“-Fehlers. Außerdem folgt eine ausführliche Analyse des Problems.

Kat.	Name	Beschreibung
0	Übungen/Netzwerk Test	Diese Kategorie dient Testzwecken.
1	Unerlaubter Zugriff	Diese Kategorie greift, wenn auf ein Netzwerk, System, eine Applikation, Daten, oder eine andere Ressource auf logische oder physikalische Weise unerlaubt zugegriffen wird.
2	„Denial of Service“ (DoS)	Bezeichnet Angriffe auf Netzwerke, Systeme oder Applikationen, die die ordnungsgemäße Funktion verhindern oder einschränken.
3	bösartiger Quellcode	Diese Kategorie bezeichnet die erfolgreiche Installation von bösartiger Software auf einem System oder in einer Applikation (Beispiele: Viren, Würmer, Trojanische Pferde o.ä.).
4	falsche Handhabung	In diese Kategorie fallen Handlungen von Personen, die gegen die Verwendungsvorschriften für Computer verstoßen.
5	Scans/ Zugriffsversuche	Diese Kategorie beschreibt jegliche Aktivitäten, die das Ausspähen oder den Zugriffsversuch auf einen Computer, offene Ports, Protokolle, Dienste oder eine Kombination ermöglichen. Der Versuch steht dabei im Mittelpunkt
6	Untersuchungen	Unbestätigte Vorfälle, die möglicherweise schadhaft oder abweichend sein können.

Tabelle 1. Kategorisierung von IT-Sicherheitsproblemen gemäß des US-CERT [UC14]

2 Fallbeispiel: OpenSSL anhand von Heartbleed

OpenSSL (früher SSLeay) ist eine quellenoffene C-Bibliothek, die den „Secure Socket Layer“ (SSL) [Fre11] implementiert. Sie umfasst dabei verschiedene Verschlüsselungsverfahren, sowie die notwendigen Protokolle für die Umsetzung der „Transport Layer Security“(TLS) [Die08].

2.1 Warum OpenSSL?

OpenSSL eignet sich als Beispiel für solche Analysen, da es es sich bei dieser Implementierung um ein komplexes, aber dennoch eigenständiges Projekt mit wenigen Abhängigkeiten handelt. Des Weiteren sind die Quellen frei zugänglich, der Quellcode ist somit einsehbar. Es können also auch Analysen auf dem Quellcode durchgeführt werden und nicht nur auf Assemblerebene. Ein weiterer Punkt für OpenSSL ist die gut dokumentierte Funktionsweise der einzelnen Module; dadurch sind einzelne Pfade durch den Quellcode nachvollziehbar. Ein weiterer Punkt ist, dass eine Analyse[CER15] der CVE's für openSSL in der Zeitspanne 07.2009 bis 03.2015 ergeben hat, dass der Großteil der Probleme, circa 43%, in ein ähnliche Kategorie wie das „Heartbleed“-Problem eingeordnet werden können. Um nachvollziehen zu können wie der „Heartbleed“-Fehler funktioniert, ist es zunächst nötig das Problem in das SSL Protokoll einzuordnen.

2.2 Schematische Funktionsweise SSL

SSL ist ein mehrstufiges Verfahren zur Identifizierung von Verbindungspartnern [Fre11]. Es soll sichergestellt werden, dass es sich bei der verbundenen Gegenstelle (Server) auch wirklich um diese handelt und nicht um einen möglichen Angreifer, der sich als die tatsächliche Gegenstelle ausgibt. Die Grafik 1 zeigt einen schematischen Ablauf des SSL-Protokolls. In einem ersten Schritt muss

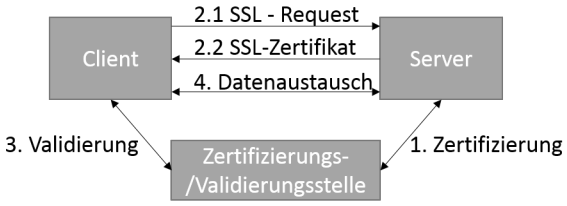

Abb. 1. schematische Funktionsweise des SSL-Protokolls

sich der Server bei einem dritten Teilnehmer zertifizieren. In diesem Prozess wird dem Server ein Zertifikat ausgestellt und ist damit in der Zertifizierungsstelle hinterlegt. Möchte nun ein Client eine Verbindung zum Server aufbauen, so fragt dieser nach dem Zertifikat des Servers (vgl. 2.1 SSL-Request). Der Server antwortet dem Client mit ausgegeben Zertifikat (2.2 SSL-Zertifikat). Der Client kann nun die Authentizität des Zertifikats und damit des Servers überprüfen, indem er eine Anfrage zur Validierung des Zertifikats an die Stelle stellt, die das Zertifikat ausgestellt hat (3. Validierung). Konnte die Identität des Servers bestätigt werden, so kann mit dem Datenaustausch begonnen werden (4. Datenaustausch). Solange Daten versendet oder empfangen werden, kann von der Identität des Servers ausgegangen werden. Findet kein weiterer Datenaustausch oder findet eine Zeitüberschreitung statt, so muss das beschriebene Prozedere wiederholt werden. Dieser Prozess kann verhindert werden, indem permanent Daten ausgetauscht werden, der Strom zwischen Client und Server also nicht abbricht. Der folgende Abschnitt beschreibt den Mechanismus und die Implementierung dieser Eigenschaft in openSSL.

2.3 Vom Heartbeat zu Heartbleed

Um eine konstante Verbindung zwischen Client und Server aufrecht zu erhalten nutzt openSSL so genannte „Heartbeats“. Hierbei handelt es sich um Nachrichten, die prüfen sollen, ob die Gegenstelle noch erreichbar ist. Die folgende Abbildung 2 zeigt den schematischen Ablauf dieser Nachrichten. Hierbei wird eine Anfrage an den Server gestellt, ob dieser noch zu erreichen ist. Kann der Server antworten, so ist die Verbindung noch aktiv und die Identität des Servers muss nicht wiederholt geprüft werden. Wichtig hierbei ist, dass der Client dem Server eine „geheime Antwortphrase“ schickt. Ist der Server in der Lage

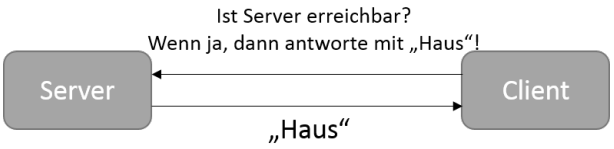

Abb. 2. schematische Funktionsweise von „Heartbeat“-Nachrichten

mit dieser zu antworten ist sichergestellt, dass es sich immer noch um die gleiche Gegenstelle handelt. In der technischen Realisierung wird ein so genanntes SSL3_Record als Teil der Heartbeat-Nachricht versendet. Das Listing 1.1 zeigt einen Auszug aus der Umsetzung des SSL3_Records in openSSL. Wichtig für den weiteren Verlauf ist hierbei, dass diese Struktur einen Zeiger auf die Daten der „Antwortphrase“ und die Länge dieser Antwort beinhaltet. Dies ist notwendig, da Programmiersprachen wie C/C++ nicht über eine integrierte Überwachung der Speichergrenzen verfügen.

Listing 1.1. Auszug aus dem SSL3_Record-Struct [ope16]

```
1 struct ssl3_record_st{
2         int type;                 /* type of record */
3         unsigned int length;      /* How many bytes available */
4         [...]
5         unsigned char *data;      /* pointer to the record data */
6         [...]
7 } SSL3_RECORD;
```

Die „Heartbeat“-Prozedur(dtls1_process_heartbeat) verarbeitet die an den Server gesendeten Nachrichten. Hierbei wird das SSL3_Record gelesen und ausgewertet. Das folgende Listing 1.2 zeigt einen Auszug aus der Prozedur dtls1_process_heartbeat.

Listing 1.2. Auszug aus der Prozedur dtls1_process_heartbeat [ope14]

```
1  int dtls1_process_heartbeat(SSL *s){
2          unsigned char *p = &s->s3->rrec.data[0], *pl;
3          unsigned short hbtype;
4          unsigned int payload;
5          unsigned int padding = 16; /* Use minimum padding */
6          [...]
7          /* Read type and payload length first */
8          hbtype = *p++;
9          n2s(p, payload);
10         pl = p;
11         [...]
```

Zunächst wird ein Pointer auf das SSL3_Records angelegt und der Typ ausgelesen. Der Pointer wird erhöht, um nun auf das Längenfeld des Structs (vgl. Listing 1.1) zu zu greifen. Diese Daten geben die vom Client definierte Länge

der „Antwortphrase“ an. Die Länge wird benötigt, um ausreichend viel Speicherplatz für die eigentliche Nachricht im Hauptspeicher zu reservieren. Die Längeninformationen werden als zwei separate Bytes übertragen und müssen konvertiert werden. Das Makro n2s() übernimmt diese Aufgabe und schreibt die transformierte Information über die Länge des Structs in die Variable payload. Aus dieser Umwandlung ergibt sich, dass der maximale Wert dieses Feldes $2^{16} = 65535$ betragen kann. Außerdem wird durch das Ablegen des Pointers p in pl ein Pointer auf die vom Client gesendeten Daten angelegt.
Die Daten müssen nun im Hauptspeicher abgelegt werden. Dazu muss Speicher reserviert werden. Das Listing 1.3 zeigt das Allozieren des Hauptspeichers für die vom Client gesendeten Daten. Wichtig hierbei ist, dass keine Plausibilitätsüberprüfung zwischen den tatsächlichen Daten und der vom Client übertragenen Länge vorgenommen wird. Diese nicht vorhandene Prüfung macht den „Heartbleed“-Fehler möglich. Die Prozedur OPENSSL_malloc() reserviert Speicher für ein komplettes SSL3_Record.

Listing 1.3. Auszug aus der Prozedur dtls1_process_heartbeat. Allozieren des Hauptspeichers für Client Daten. [ope14]

```
1 unsigned char *buffer, *bp;
2 int r;
3 /* Allocate memory for the response, size is 1 byte
4  * message type, plus 2 bytes payload length, plus
5  * payload, plus padding
6  */
7 buffer = OPENSSL_malloc(1 + 2 + payload + padding);
8 bp = buffer;
```

Bis zu diesem Zeitpunkt ist noch kein Fehler ersichtlich. Die einzige Möglichkeit bisher ist, möglicherweise deutlich mehr Speicher im Hauptspeicher zu belegen als notwendig ist, jedoch maximal 65 Kilobyte. Das Sicherheitsproblem entsteht erst, wenn diese Daten gelesen werden. Das Listing 1.4 zeigt die Stelle in der dtls1_process_heartbeat Prozedur, die die Daten aus dem Speicher liest und an den Client zurück schickt.

Listing 1.4. Auszug aus der Prozedur dtls1_process_heartbeat. Lesen der Daten für den Versand zum Client [ope14]

```
1 /* Enter response type, length and copy payload */
2 *bp++ = TLS1_HB_RESPONSE;
3 s2n(payload, bp);
4 memcpy(bp, pl, payload);
```

An dieser Stelle des Quellcodes wird der reservierte Block aus dem Hauptspeicher genommen und direkt mit der memcpy() Funktion in das Antwort-SSL3_Record geschrieben. Gemäß dem Fall es wurde nun zu viel Speicher reserviert als für die eigentlichen Daten notwendig war, so werden die Bytes an den Client übertragen, die hinter den eigentlichen Daten im Hauptspeicher liegen. Der Hauptspeicher wird von allen Prozessen des Servers gleichermaßen genutzt.

Wo die Daten dieser Prozesse abgelegt werden kann nicht vorhergesagt werden. Es ist daher möglich, dass Daten wie Passwörter oder sensible Dokumente in den ausspähbaren Speicherbereich abgelegt werden. Konnten keine sensiblen Daten durch einen solchen Angriff erbeutet werden, so kann dieser beliebig oft wiederholt werden.

Der folgende Abschnitt befasst sich mit einer genaueren Analyse des Problems, sowie der Behebung und Lösungsansätzen, wie dieser Fehler zukünftig vorgebeugt werden kann.

3 Testverfahren und Ansätze

Technisch gesehen ist der „Heartbleed"-Fehler ein so generates input-validation-Problem; hierbei stimmen die vom Client angegebenen Informationen nicht mit den tatsächlichen Informationen über die Daten überein. Ein Beispiel: Der Nutzer schickt wie in Abbildung 2 die Schlüsselphrase „Haus"(4 Bytes), als Länge werden aber 65 Kilobytes angegeben. OpenSSL sah zum Zeitpunkt des „Heartbleed"-Fehlers keine Möglichkeit zur Validierung dieser Eingaben vor. Somit konnten 64.9 Kilobyte Daten aus dem Hauptspeicher unberechtigter Weise aus dem Speicher gelesen werden. Das Update „1.0.1g "[Pro14b] konnte das Problem, wie Listing 1.5 zeigt, beheben (vgl. Listing1.2).

Listing 1.5. Fix der Funktion dtls1_process_heartbeat [ope14]

```
1 /* Read type and payload length first */
2 if (1 + 2 + 16 > s->s3->rrec.length)
3         return 0; /* silently discard */
4 hbtype = *p++;
5 n2s(p, payload);
6 if (1 + 2 + payload + 16 > s->s3->rrec.length)
7         return 0; /* silently discard per RFC 6520 sec. 4 */
8 pl = p;
```

Der Fix sieht vor, dass zunächst geprüft wird, ob die Größe der gesendeten Daten größer ist als 0 (Siehe Listing 1.5, Zeile 2), indem die Größe des SSL3_Records größer sein muss als die Summe der Bytes aller Felder im Struct ($1 + 2 + 16 = 19$ *Bytes*). Des Weiteren wird auf eine ähnliche Weise geprüft, ob der die Größe des Structs plus die vom Client angegebene Länge der Daten mit der tatsächlichen Größe des gesamten Structs übereinstimmt. Sollte einer dieser Fälle nicht korrekt sein wird die weitere Verarbeitung abgebrochen. An dieser Stelle ist zu kritisieren, dass keinerlei Mitteilung oder Logging durchgeführt wird. Ein Reporting hätte den Vorteil, dass der Angriff protokolliert wird, das System erhält somit Informationen darüber, dass eine mögliche Bedrohung besteht, es kann untersucht werden, ob noch weitere Angriffe stattfinden und es kann darauf reagiert werden. Ohne eine solche Information bleibt ein solcher Angriffsversuch verborgen.

Wie können solche Probleme zukünftig verhindert werden?
Hierbei soll konkret ein Verfahren gesucht werden, das den Fokus auf die Erkennung des Fehlers auf den unteren Ebenen (Quellcode, Struktur, etc) legt. Der von Hao et al.[HJC+14] beschriebe Ansatz ist für diese Fragestellung nicht brauchbar, da hierbei „Security Management“ in den Schwerpunkt gerückt. Dabei soll „Security Management“ ein Basis-Bestandteil der Entwicklung auch in den auf dem Produkt aufbauenden Applikationen sein. Es handelt sich somit um einen Ansatz auf der Projekt- bzw. Architektur-Ebene.

Am Beispiel openSSL: Die Architekten von SSL sollen in das „Security Management“ in allen Phasen der Entwicklung von z.B. openSSL oder darauf basierenden Technologien eingebunden werden, um so mögliche Fehler frühzeitig zu erkennen und zu beheben. Für die Praxis gestaltet sich dieser Ansatz als schwer umsetzbar. Die Umsetzung einer solchen Methode ist ressourcen- und zeitintensiv und wenn überhaupt nur in großen Unternehmen denkbar. Außerdem ist openSSL ein Opensource Projekt. Die Entwickler arbeiten über die Welt verteilt, was einen solchen Prozess weiter erschwert.

Einen Ansatz auf den oben genannten unteren Ebenen bieten Cao et al. [CGLX15] in einer Arbeit über input-validation auf Androidgeräten. Hierbei konnte gezeigt werden, wie durch Ansteuern von Android Modulen und Services mit Fehlinformationen input-validation-Fehler in insgesamt 16 von 90 getesteten Services aufgedeckt wurden. Umgesetzt wurde dieser Ansatz mit einem sogenannten „Binding Fuzzer“. Hierbei handelt es sich um ein semi-automatisches Tool, dass fehlerhafte Anfragen an die zu testenden Android System Services sendet und deren Antworten protokolliert und auswertet. Bei genauerer Betrachtung fällt auf, dass es sich hierbei um klassisches „Blackbox Testing“[Bat15] handelt. Da mit diesem Ansatz bereits erfolgreich gezeigt wurde, dass Android-spezifische input-validation-Probleme aufgedeckt werden können ist eine Übertragung auf die „Heartbleed“-Problematik sinnvoll. Die Ergebnisse dieses Vorgehens werden in einer weiteren Arbeit diskutiert [Hin16b].

Ein weiterer Ansatz, der für die Ermittlung des „Heartbleed“-Fehlers in Betracht gezogen werden kann, ist eine Arbeit von Lui und Tan [Tan09]. Hierbei ist es gelungen automatisiert akzeptierte und verweigerte Inputs anhand des Quellcodes zu bestimmen. Des Weiteren können mit diesem Ansatz automatisiert Black-Box Testfälle generiert werden. Diese können genutzt werden, um mögliche Inkonsistenzen in der Spezifikation aufzudecken. Auf diese Weise können weitere Black- und White-box Testverfahren unterstützt werden. Außerdem wurde in der Arbeit von Lui und Tan ein Test Kriterium vorgestellt, welches die Eignung der Testfälle für die input-validation messbar macht. Der Fokus dieses Ansatzes und des erstellten Tools liegt auf webbasierten Applikationen, die in Java geschrieben wurden. Hierbei wird ein Kontrollflussgraph und ein Datenflussgraph anhand des Java Bytecodes erstellt. Anhand dieser Informationen kann dann die Verarbeitung der Nutzereingaben nachvollzogen werden und geeignete Testfälle erzeugt werden. Für das Testkriterium werden anhand des Datenflussgraphen

Pfade ermittelt, die mögliche input-validation-Probleme beinhalten. Nun kann mit den erstellten Tests geprüft werden zu wie viel Prozent der Tests diese Pfade überdecken. Diese Zahl kann als Indikator für die Qualität der Tests im Bezug auf input-validation-Probleme heran gezogen werden. Für die in dieser Arbeit vorgestellten Problemstellung eignet sich auch dieser Ansatz als Ausgangspunkt für nähere Betrachtungen. Besonders die Erstellung von Kontroll- und Datenflussgraphen ist genauer zu untersuchen. Hierbei ist es wichtig zu prüfen, inwieweit sich die geschilderten Verfahren auf Sprachen wie C/C++ übertragen lassen.
Um solche Schritte durchzuführen sind weitere Arbeiten und Projekte notwendig. Insbesondere sollen dabei die Ansätze von Cao und Tan auf den „Heartbleed"-Fehler übertragen werden. Der folgende Abschnitt befasst sich mit den bereits durchgeführten praktischen Versuchen, sowie notwendigen Schritten für zukünftige Vorgehen und die damit verbundenen Chancen und Risiken.

4 Weitere Schritte

Für die Erkennung des „Heartbleed"-Fehlers konnte bisher eine abstrahierte Demonstrationsanwendung erstellt werden. Diese wird im Projektbericht [Hin16a] näher beschrieben. In dieser Projektausarbeitung wird beschrieben, welche Kernkomponenten notwendig sind, um den „Heartbleed"-Fehler zu abstrahieren und in ein Testsystem einzubetten. Die Abbildung 3 zeigt einen Überblick über die Abstraktion. Hierbei wurden die Komponenten aus openSSL auf ein Minimum reduziert, sodass die grundlegende Funktionalität erhalten bleibt, aber das Nachvollziehen von Pfaden die Komponenten der Applikation erleichtert wird. Dieses

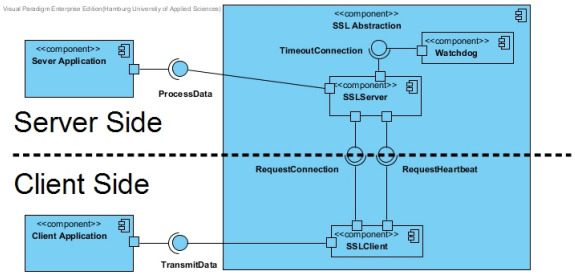

Abb. 3. Abstraktion des „Heartbleed"-Problems

System kann dazu genutzt werden in weiteren Arbeiten die in Abschnitt 3 vorgestellten Verfahren zu validieren und zunächst auf die Abstraktion angewendet werden. Hierbei ist es wichtig die gewonnenen Erkenntnisse in einen größeren Zusammenhang zu setzten. Dies führt dann zu „Thesis Outline".

4.1 Thesis Outline

In einen größeren Kontext gesetzt, ist die Frage zu stellen, ob die bei der Analyse von „Heartbleed“ gewonnenen Erkenntnisse auch auf andere Software übertragbar ist. Wichtig hierbei ist, dass eventuelle Risiken evaluiert werden müssen. So muss ermittelt werden, ob „Heartbleed“ in allen Ausprägungen zuverlässig erkannt werden kann und inwieweit sichergestellt oder sogar garantiert werden kann, dass Verallgemeinerungen des Fehlers in anderer Software identifizierbar sind. Daraus ergibt sich eine Kernfragestellung für eine folgende Projektstudie [Hin16b]: „Kann die Erkennung von Verwundbarkeiten in openSSL auf andere Software übertragen werden?“ In dieser Studie müssen auch Alternativen für den Analyseansatz identifiziert werden. Denkbar sind hierbei Ansätze aus dem modell-basierten Testen. Für diesen Fall ändert sich die Fragestellung: „Kann ein Modell gefunden werden, welches Probleme wie „Heartbleed“mit den dazu gehörigen Charakteristika im Detail beschreibt?“. Es muss dabei ermittelt werden, ob die sehr Quellcode-spezifischen Eigenschaften in einem Modell abgebildet werden können, oder ob durch das Abstrahieren in ein Modell notwendige Informationen zur Identifikation verloren gehen. Eine weitere Alternative sind Formale Methoden. Hierbei können mathematische und logisch fundierte Verfahren, wie Modelchecking, genutzt werden, um das Verhalten einer Applikation zu belegen. Formale Analysen gewinnen immer größere Bedeutung in der Analyse von IT-Sicherheitsproblemen. So veröffentlichte das Bundesamt für Sicherheit in der Informationstechnik (BSI) ein Handbuch für die Anwendung von Formalen Methoden in sicherheitsrelevanten Systemen [GG13]. Mögliche Risiken sind dabei ähnlich wie bei den modell-basierten Tests. Auch hier müssen Modelle erstellt werden, die das Verhalten des „Heartbleed“-Problems nachbilden: Allgemein bleibt das Risiko, dass keiner dieser Ansätze zielführend ist. Wichtig ist hierbei frühzeitig eine Erfolgsprognose zu erstellen, um nicht zu viel Zeit zu verlieren. Sollte sich keiner der beschriebenen Alternativen als erfolgversprechend zeigen, kann noch eine Kombination verschiedener Verfahren und Methoden evaluiert werden.

5 Fazit

Zusammenfassend konnte in dieser Arbeit die Notwendigkeit ausreichend motiviert werden sich mit der Analyse des „Heartbleed“-Problems auseinander zu setzen. Außerdem wurde die Problemstellung in die Risikobewertung des US-CERT eingeordnet. Für diese Arbeit wurde der Ansatz gewählt, die Untersuchung zunächst am speziellen Beispiel analysieren und dann zu prüfen in wie weit die Erkenntnisse auf weitere, ähnliche Probleme übertragbar sind. Der Fehler konnte innerhalb des SSL-Protokolls und innerhalb der openSSL Implementierung im Detail betrachtet werden. Ebenso konnte gezeigt werden, wie dieser Fehler korrigiert wurde und anhand dieser Informationen wurde der Fehler als input-validation-Problem klassifiziert. Auf Basis dieser Klassifizierung konnten Ansätze in der aktuellen Literatur gefunden werden, die als Grundlage für die Identifizierung verwendet werden können. Für weitere Untersuchungen in

einer Projektstudie wurden zunächst zwei dieser Ansätze ausgewählt. Sollten diese Ansätze nicht den gewünschten Erfolg bringen, konnten bereits alternative Ansätze identifiziert werden, die auch in der genannten Studie untersucht werden sollen.

Literatur

Bat15. McKay Bath. *Test Analyst und Technical Test Analyst*. Rocky Nook, 2015.

bit15. deutscher Digitalverband bitkom. Digitale Angriffe auf jedes zweite Unternehmen, 2015. (Stand 03.02.2016) `https://www.bitkom.org/Presse/Presseinformation/Digitale-Angriffe-auf-jedes-zweite-Unternehmen.html`.

Bun15. Statistisches Bundesamt. Nutzung von Informations- und Kommunikationstechnologien in Unternehmen, 2015. (Stand 03.02.2016) `https://www.destatis.de/DE/Publikationen/Thematisch/UnternehmenHandwerk/Unternehmen/InformationstechnologieUnternehmen5529102157004.pdf?__blob=publicationFile`.

CER15. DFN CERT. Auszug aus CVE-Datenbank 07.2009 - 03.2015, 2015. vertrauchlich zur Verfügung gestellt.

CGLX15. Chen Cao, Neng Gao, Peng Liu, and Ji Xiang. Towards Analyzing the Input Validation Vulnerabilities Associated with Android System Services. In *Proceedings of the 31st Annual Computer Security Applications Conference*, ACSAC 2015, pages 361–370, New York, NY, USA, 2015. ACM.

Die08. T. Dierks. The Transport Layer Security (TLS) Protocol Version 1.2. Technical report, IETF, 2008.

Fre11. A. Freier. The Secure Sockets Layer (SSL) Protocol Version 3.0. Technical report, IETF, 2011.

GG13. Dr. Hubert Garavel and Dr. Susanne Graf. *Formal Methods for Safe and Secure Computer Systems*, 2013.

Hin16a. Torge Hinrichs. Projektbericht: Analyse von IT-Sicherheitsproblemen. Technical report, HAW-Hamburg, 2016.

Hin16b. Torge Hinrichs. Projektbericht: Identifizierung von input-validation-Problemen. Technical report, HAW-Hamburg, 2016. Working Title.

HJC+14. Yongle Hao, Yizhen JiaL, Baojiang Cui, Wei Xin, and Dehu Meng. OpenSSL HeartBleed: Security Management of Implements of Basic Protocols. In *P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2014 Ninth International Conference on*, 2014.

Lyn15. James Lyne. Sophos Security Threat Report, 2015. (Stand 03.02.2016) `https://www.sophos.com/threat-center/medialibrary/PDFs/other/sophos-trends-and-predictions-2015.pdf`.

MIT14. MITRE. CVE-2014-0160, 2014. (Stand 03.02.2016) `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0160`.

ope14. openSSL.org. Commit Heartbleedfix dtls1_process_heartbeat, 2014. (Stand 03.02.2016) `http://git.openssl.org/gitweb/?p=openssl.git;a=commitdiff;h=96db902`.

ope16. openSSL.org. Quellcode ssl3_record_st, 2016. (Stand 03.02.2016) `https://github.com/openssl/openssl/blob/master/ssl/record/record.h`.

Pro14a. The OpenSSL Project. https://www.openssl.org, 2014. (Stand 03.02.2016) `https://www.openssl.org/`.

Pro14b. The OpenSSL Project. Patchnotes 2014, 2014. (Stand 03.02.2016) https://www.openssl.org/news/vulnerabilities.html#y2014.

Sca12. Paul Cichonski, Tom Millar, Tim Grance, Karen Scarfone. Computer Security Incident Handling Guide. Technical report, National Institute of Standards and Technology, 2012.

Tan09. Hui Liu; Hee Beng Kuan Tan. Covering code behavior on input validation in functional testing. *Information and Software Technology*, 2009.

UC14. US-CERT. Federal Agency Incident Categories, 2014. (Stand 03.02.2016) https://www.us-cert.gov/government-users/reporting-requirements.