Rapid in-Depth Analysis für Telematikanwendungen im Gesundheitswesen

Identifizierung nicht erkannter Sicherheitslücken mit Threat Modeling und Fuzzing

Fabian Schwab, B.Sc.; Jörg Lübbert, B.Sc.;
Peter Sakal, B.Sc.; Prof. Dr. Hartmut Pohl

Informationssicherheit, Fachbereich Informatik
Hochschule Bonn-Rhein-Sieg
Grantham-Allee 20
53757 St. Augustin
Fabian.Schwab@h-brs.de
Joerg.Luebbert@h-brs.de
Peter.Sakal@h-brs.de
Hartmut.Pohl@h-brs.de

Abstract: Die Normen DIN EN 61508 und DIN EN 62304 beschreiben Sicherheitsanforderungen für die Entwicklung von Software im medizinischen Umfeld. Diese beinhalten u.a. Vorschriften zur Verifikation und Diagnose (Kapitel C5, DIN EN 61508-7 [Di09]), zur Beurteilung der funktionalen Sicherheit (Kapitel C6, DIN EN 61508-7 [Di09]), zur Implementierung und Verifikation von Software-Einheiten (Kapitel 5.5, DIN EN 62304 [Di07]) und zur Prüfung des Softwaresystems (Kapitel 5.7, DIN EN 62304 [Di07]). Durch die kosteneffektiven Verfahren Threat Modeling und Fuzzing wird diesen Forderungen entsprochen und insbesondere die Identifizierung unveröffentlichter Sicherheitslücken ermöglicht. In einem Forschungsprojekt[1] werden Tools für beide Verfahren analysiert und bewertet. Im Projekt werden mit beiden Verfahren sehr erfolgreich bislang nicht identifizierte (unveröffentlichte) Sicherheitslücken in Individual- und Standardsoftware identifiziert und auch behoben. Im Rahmen der Gesundheitstelematik können durch beide Verfahren die Anforderungen zur Softwareentwicklung und -verifizierung erfüllt und darüber hinaus kann ein weit höheres Sicherheitsniveau erreicht werden.

[1] Förderung des Projektes SoftSCheck durch das Bundesministerium für Bildung und Forschung (Förderkennzeichen 01 IS 09030)

1 Motivation

Software kann nicht fehlerfrei erstellt werden [SP10]. Dies macht das Testen von Software erforderlich, wobei eine enge Bindung zur Qualitätssicherung besteht. Manuelles Testen von Software mit einer großen Menge an Programmcode ist nicht praktikabel [SGA07]. Threat Modeling ermöglicht die Identifizierung von Sicherheitslücken bereits in der Design-Phase. Fuzzing kann sowohl als Black-Box Test (ohne vorliegenden Quellcode) als auch als White-Box Test (mit vorliegendem Quellcode) in der Verification-Phase durchgeführt werden.

Das heuristische Tool-gestützte Verfahren Threat Modeling unterstützt die Identifizierung von Sicherheitslücken. Durch Bewertung und Kategorisierung von Sicherheitslücken können Schutzmechanismen und Gegenmaßnahmen bestimmt werden, um Sicherheitsrisiken zu mindern, zu minimieren, zu beheben oder auch zu akzeptieren [HL06]. Die Erstellung eines Threat Models für einen Systementwurf erfolgt idealerweise bereits in der Design Phase; dadurch ist eine vertrauenswürdige Implementierung von Software bereits lange vor der Veröffentlichung (Release) möglich. Threat Modeling ist ein kosteneffizientes Verfahren zur Identifizierung, Vermeidung und Minderung von Sicherheitslücken und Bedrohungen [My05, MLY05].

Mit Fuzzing steht ein Tool-gestütztes Verfahren zur Identifizierung von Softwarefehlern in der Verification-Phase zur Verfügung. Fuzzing kann dazu beitragen, unveröffentlichte sicherheitsrelevante Fehler zu identifizieren. Dazu werden die Eingabeschnittstellen der zu testenden Software identifiziert, um automatisiert und zielgerichtet Daten an diese zu senden, während die Software durch einen Monitor auf auftretende Fehler überwacht wird. So kann der Identifizierung von Sicherheitslücken durch Dritte und somit der Entwicklung von (Less-Than-)Zero-Day-Exploits (Angriffsprogramme auf unveröffentlichte Sicherheitslücken) entgegengewirkt werden [Po07]; (Less-Than-)Zero-Day-Exploits sind eine der zwanzig häufigsten Angriffsformen [Sa10].

Die Kosten zur Behebung von Sicherheitslücken nehmen im Verlauf des Softwareentwicklungsprozesses exponentiell zu [Ni02]. Wenn Fehler in der Testing- bzw. Verifikationsphase identifiziert werden, steigen die Kosten im Vergleich zur Identifizierung in der Design-Phase um den Faktor 15. Werden Fehler erst in der Release-Phase (oder später) entdeckt, steigen die Kosten um den Faktor 100 (Siehe Abbildung 1: Kosten zur Behebung von Sicherheitslücken im Verlauf der Softwareentwicklung [nach: Jo96]).

Microsoft nutzt die Vorteile von Threat Modeling [HL06] und Fuzzing [GKL08] seit 2003 als festen Bestandteil des eigenen `sicheren` Softwareentwicklungsprozess, dem Security Development Lifecycle (SDL) [HL06].

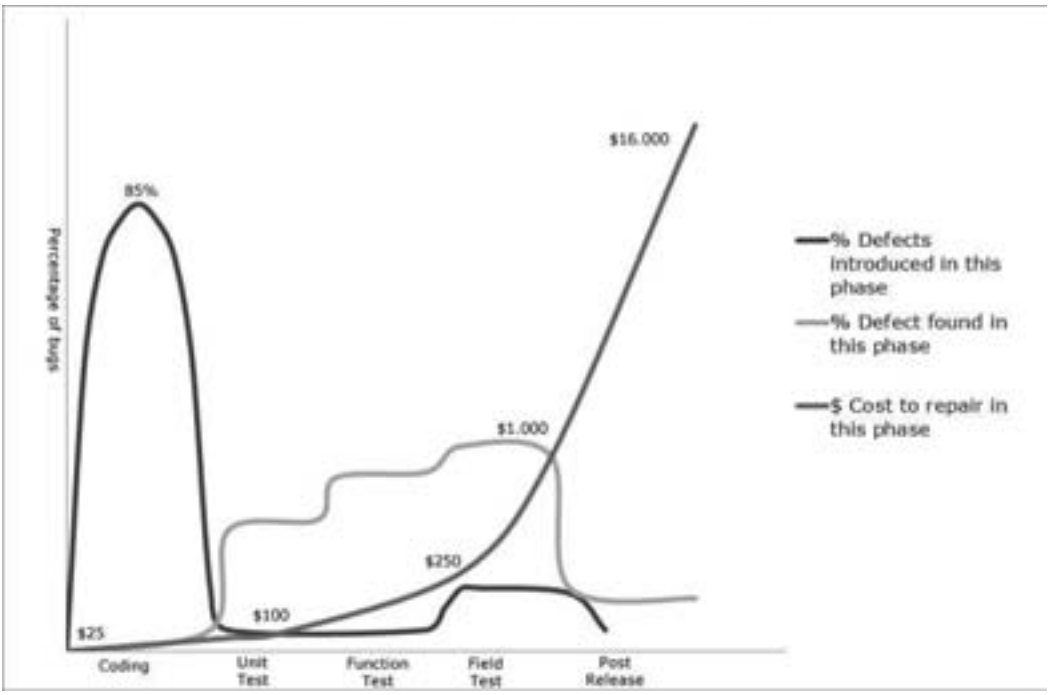

Abbildung 1: Kosten zur Behebung von Sicherheitslücken im Verlauf der Softwareentwicklung [nach: Jo96]

2 Threat Modeling

2.1 Einführung Threat Modeling

Nach vollständiger Identifizierung schützenswerter Komponenten (Assets) sowie zugehöriger Bedrohungen und Sicherheitslücken ist ihre Bewertung erforderlich. Identifizierung und Bewertung der Bedrohungen und Sicherheitslücken kann z.B. durch Attack Trees erfolgen [Sc99]. Auf Grundlage dieser Bewertung kann eine Minderung (Mitigation) der Bedrohungen und eine Behebung der Sicherheitslücken erfolgen. Neben den unterschiedlichen in den Threat Modeling Tools implementierten Maßnahmen (z.B. Redesign, Standard Mitigation, Custom Mitigation oder Accept Risk) ist eine individuelle Behandlung einzelner Bedrohungen und Sicherheitslücken sowie die Kontrolle der implementierten Verfahren erforderlich. Zur Durchführung von Threat Modeling können mehrere Ansätze gewählt werden. Hierbei wird jeweils ein unterschiedlicher Ausgangspunkt gewählt. Neben dem Software-Centric und dem Asset-Centric Ansatz existiert der Attacker-Centric Ansatz. Der systematische Ablauf des Attacker-Centric Ansatzes mit seinen drei Stufen – Sicht eines Angreifers verstehen, Sicherheit charakterisieren und Threats bestimmen [SS04] – ist in Abbildung 2 grafisch dargestellt. In jeder Stufe werden zugehörige Aktionen, mit dem Ziel, das Threat Model genauer zu spezifizieren und weiter auszubauen, durchgeführt. Die Entstehung neuer Bedrohungen für ein bereits durch Threat Modeling spezifiziertes System ist nach [Ow09] zu vernachlässigen.

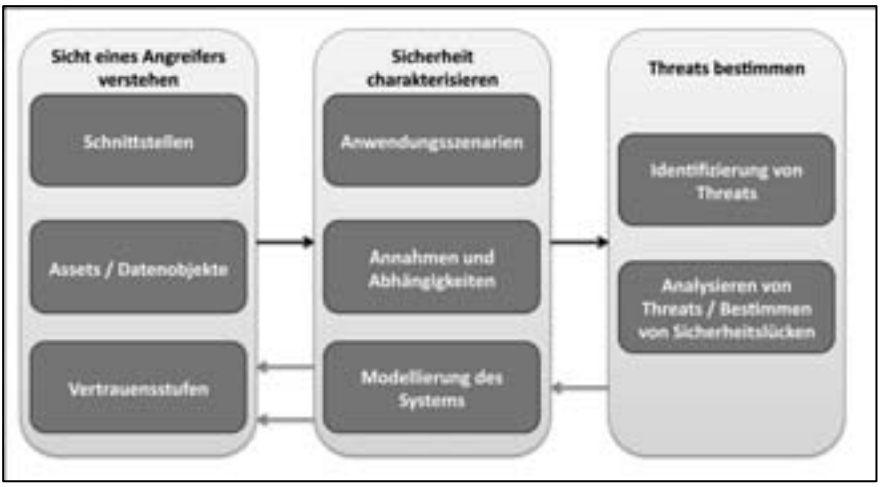

Abbildung 2: Threat Modeling Ablauf des Attacker-Centric Ansatzes [nach: SS04]

Durch Threat Modeling gewonnene Erkenntnisse können ebenso zur Entwicklung strategischer Penetration Tests genutzt werden. Hierbei wird auf Erkenntnisse über Angriffspunkte und Eingabeschnittstellen zurückgegriffen, welche aus der modellierten Systemarchitektur gewonnen werden können.

Neben der Anwendung in der Design Phase der Softwareentwicklung ist es auch möglich ein Threat Model für eine bereits implementierte Systemarchitektur – und auch für bereits ausgelieferte Software – zu erstellen.

2.2 Bewertungsverfahren für Threat Modeling Tools

Um eine einheitliche Klassifizierung der zu untersuchenden Threat Modeling Tools zu gewährleisten, ist die Entwicklung und Begründung differenzierender Bewertungsparameter unabdingbar. Diese müssen unter Berücksichtigung der im Projekt festgelegten Ziele neben einer eindeutigen Produktzuordnung die anfallenden Kosten, den Funktionsumfang, die Entwicklungsmöglichkeiten, den Installationsaufwand, die Benutzerfreundlichkeit und die Qualität der Dokumentation berücksichtigen. Die hier vorgelegten Bewertungsparameter sind in unterschiedliche Kategorien unterteilt. Kategorien beinhalten jeweils Parameter einer Funktionsgruppe. Parameter sind entsprechend ihrer Relevanz gewichtet und fließen in die Berechnung der jeweiligen Kategorie ein. Einzelne Kategorien werden ebenso auf Grundlage Ihrer Relevanz unterschiedlich gewichtet und fließen dementsprechend in das Gesamtergebnis ein.

Die Produkte werden anhand der folgenden Kategorien bewertet:

- Erfassung der Modellierungsmöglichkeiten hinsichtlich Assets, Threats, Threat Mitigation, Vulnerabilities und dem eingesetzten System bzw. der Systemumgebung.
- Folgende Visualisierungsmöglichkeiten werden erfasst:
  - Datenflussdiagramm
  - Bedrohungsbaum

- Anwendungsdiagramm
- Sonstige Möglichkeiten

- Das Berichtswesen (Reporting) erfasst Möglichkeiten des grafischen und textuellen Exports sowie den Reportumfang.
- Folgende Entwicklungsmöglichkeiten werden berücksichtigt:
  - Vorhandener Quellcode
  - Entwicklungsschnittstellen: Möglichkeit zur Anbindung eigener Erweiterungen und/oder zusätzlicher Module)
  - Community-Projekt: Entwicklung und/oder Support durch eine Community
  - Dokumentation der Entwicklungstools
- Beim Installations- und Nutzungsaufwand wird unterschieden nach der Grundinstallation, speziellen Voraussetzungen (z.B. erforderlicher Software), Einarbeitungszeit, Usability und Anwendung (vollständige Modellierung eines Testszenarios).

Die Lizenzkosten der Tools wurden nicht berücksichtigt, da mit Ausnahme eines Tools alle entgeltfrei verfügbar sind. Detailliertere Informationen zum entwickelten Bewertungsverfahren sowie dessen Anwendung wurden bereits gesondert veröffentlicht [Sc10].

2.3 Marktanalyse und Bewertung

Zum Zeitpunkt der Untersuchung waren sieben Threat Modeling Tools (sowohl entgeltfreie, als auch entgeltpflichtige) verfügbar. Um die Interessen von Herstellern zu wahren werden die Ergebnisse im weiteren Verlauf anonymisiert dargestellt. Die Tools können durch folgende Kategorien klassifiziert werden:

- Universal: Das Produkt eignet sich zur Erstellung eines Threat Models ohne spezifische Ausrichtung.
- Netzwerk: Das Produkt eignet sich zur Erstellung eines Threat Models für ein Netzwerkszenario. Es werden insbesondere Elemente eines Netzwerks betrachtet.
- Software Design: Das Produkt eignet sich zur Erstellung eines Threat Models im Rahmen des Software Designs und berücksichtigt spezifische Elemente der Software Entwicklung. Die Integration in den Software-Entwicklungsprozess von Microsoft - dem Security Development Lifecycle (SDL) - wird unterstützt.

Zu den untersuchten Produkten sowie deren Einsatzkategorie siehe Abbildung 3: Verfügbare Threat Modeling Tools.

Im Rahmen der durchgeführten Untersuchungen wurden große Diskrepanzen beim Funktionsumfang - explizit bei Modellierungs- und Visualisierungsmöglichkeiten - festgestellt. Alle untersuchten Tools bieten Funktionen zur Modellierung elementarer Bestandteile [Sc06] eines Threat Models an - Assets, Threats und Vulnerabilities - in unterschiedlichem Umfang. Eines der Tools (Tool C) lässt ausschließlich die Modellierung von Assets zu. Bei dem Tool handelt es sich um eine Sammlung von Symbolen zur Darstellung eines Threat Models. Lediglich drei Tools (Tool

A, Tool E und Tool G) bieten umfangreiche Visualisierungsmöglichkeiten in Form von Datenflussdiagrammen, Bedrohungsbäumen und Anwendungsdiagrammen. Die anderen Tools bieten ausschließlich eigene Visualisierungsformen in unterschiedlichem Umfang.

Name	Hersteller	Kategorie	Version
The CORAS Method	CORAS	Universal	20060714
Microsoft SDL Threat Modeling Tool	Microsoft	Software Design	3.1.3.1
Microsoft Threat Analysis & Modeling	Microsoft	Software Design	3.0
Practical Threat Analysis	PTA Technologies	Universal	1.6 Build 1212
SeaMonster	SeaMonster	Software Design	3.1.3.1
Skybox Secure	Skybox Security	Netzwerk	4.5
Trike	Dymaxion	Universal	1.12a

Abbildung 3: Verfügbare Threat Modeling Tools

Die Bewertung erfolgte auf Grundlage des eigens entwickelten Bewertungsverfahrens, welches die im Projekt festgelegten Ziele und die Anwendung in einem Testszenario abbildet. Eine Bewertung kann ausschließlich subjektiv für einen bedarfsspezifischen Fall erfolgen – es kann daher keine objektive Aussage über eine allgemeingültige Platzierung der einzelnen Tools erfolgen.Die Ergebnisse der einzelnen Parameter, Kategorien und der Gesamtbewertung werden in Prozent der möglichen Punkte dargestellt. Lediglich drei Tools erzielten in der Gesamtbewertung mehr als 70% der möglichen Punkte (Tool A, Tool B und Tool E). Die weiteren Tools weisen große Defizite (weniger als 10% der möglichen Punkte) in mindestens einer Kategorie auf und erzielen in der Gesamtbewertung weniger als 70% der möglichen Punkte. Die Erstellung komplexer und vollständiger Threat Models ist mit drei Produkten möglich. Die von den Tools erzielten Punkte in den einzelnen Kategorien sind in Abbildung 4 grafisch dargestellt.

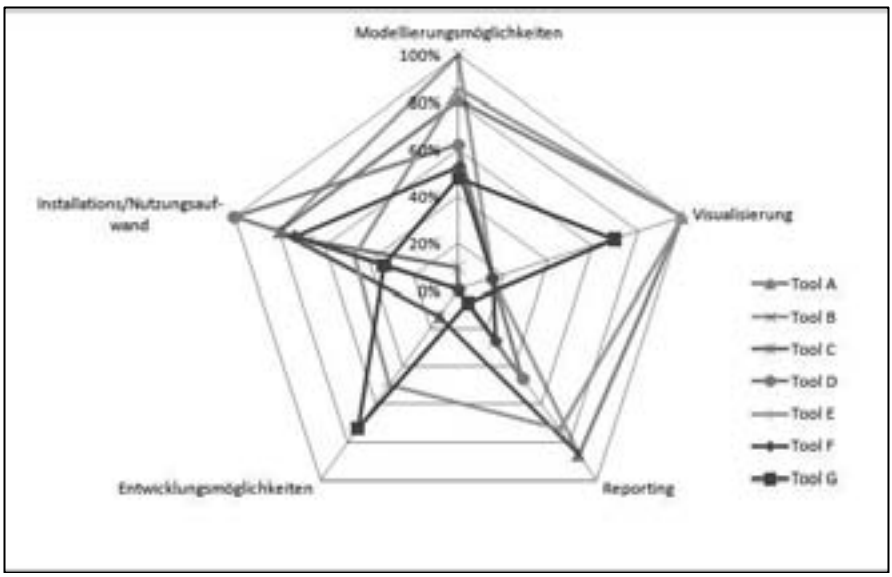

Abbildung 4: Bewertung verfügbarer Threat Modeling Tools

Die Stärken und Schwächen sowie die Gesamtbewertung und der Funktionsumfang der einzelnen Tools werden verdeutlicht:

- Tool A zeichnet sich durch gute Modellierungsmöglichkeiten sowie durch eine sehr gute Visualisierung aus; das Reporting ist gut ausgebildet; der Installations- und Nutzungsaufwand ist gut und Entwicklungsmöglichkeiten dieses Community-Projekts sind befriedigend.
- Tool B zeichnet sich durch gute Modellierungsmöglichkeiten und ein gutes Reporting aus; die Entwicklungsmöglichkeiten sind in geringen Umfang vorhanden und gut dokumentiert; die Visualisierung ist ausreichend (nur eigene Visualisierungsformen) und der Installations- und Nutzungsaufwand ist gut.
- Tool C bietet unzureichende Modellierungsmöglichkeiten und Berichtsfunktionen; Funktionen zur Visualisierung und Entwicklungsmöglichkeiten sind nicht vorhanden; der Installations- und Nutzungsaufwand ist befriedigend; es handelt sich um eine Sammlung von Symbolen zur Darstellung eines Threat Models.
- Tool D bietet befriedigende Modellierungsmöglichkeiten und ein ausreichendes Reporting; die Visualisierung ist mangelhaft und Entwicklungsmöglichkeiten sind nicht vorhanden; der Installations- und Nutzungsaufwand ist sehr gering und daher als sehr gut zu bewerten.
- Tool E zeichnet sich durch gute Modellierungsmöglichkeiten und eine sehr gute Visualisierung aus; das Reporting ist gut ausgebildet; die Entwicklungsmöglichkeiten dieses Community-Projekts sind gut und es sind Entwicklungsschnittstellen vorhanden; der Installations- und Nutzungsaufwand ist hoch - jedoch noch mit ausreichend zu bewerten.

- Tool F bietet befriedigende Modellierungsmöglichkeiten und einen guten Installations- und Nutzungsaufwand; die Visualisierung und das Reporting sind mangelhaft; Entwicklungsschnittstellen sind nicht vorhanden.
- Tool G bietet gute Entwicklungsschnittstellen und eine gute Visualisierung; die Modellierungsmöglichkeiten sind befriedigend; der Installations- und Nutzungsaufwand ist mit ausreichend zu bewerten und das Reporting ist ungenügend.

Zusammenfassend ergibt sich eine Spitzengruppe gebildet aus den Tools A, B und E. Die Tools C, D, F und G zeigen - bei Anwendung der hier zugrunde gelegten Bewertungsparameter - nur mittlere oder sogar unzureichende Leistungen. Abbildung 5 stellt die bewerteten Tools hinsichtlich Ihrer Effektivität bei der Erstellung eines Threat Models sowie hinsichtlich Ihres Funktionsumfangs dar.

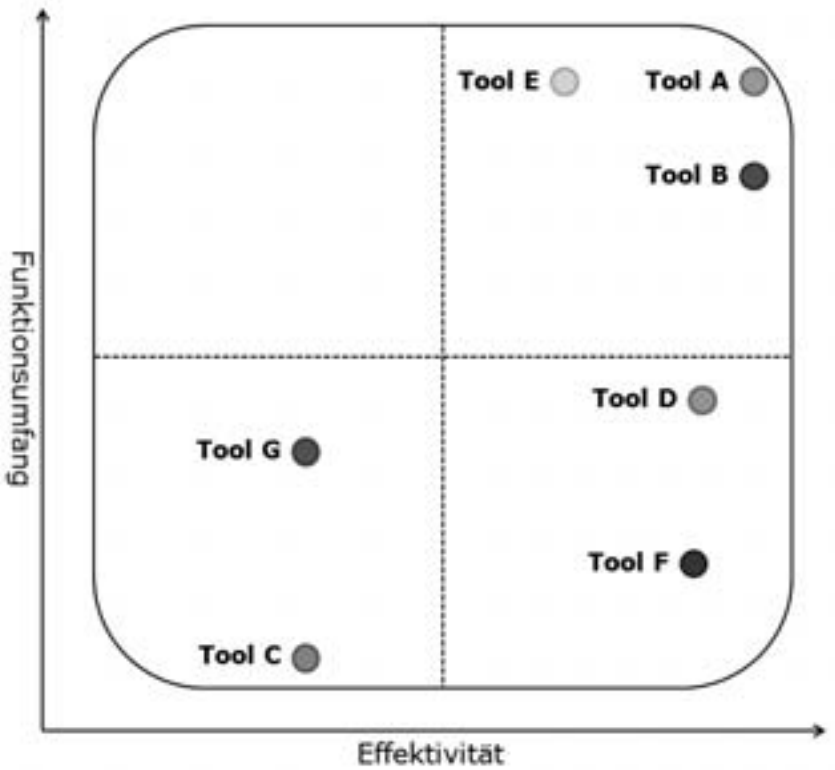

Abbildung 5: Vergleich der untersuchten Threat Modeling Tools

2.4 Fallstudie: Erfolgreiches Threat Modeling

Bei der exemplarischen Anwendung von Threat Modeling auf eine bereits implementierte und an Kunden ausgelieferte Individualsoftware konnten zahlreiche, mit klassischen Verfahren nicht erkannte, Sicherheitslücken identifiziert und behoben werden. Bei der untersuchten Software handelt es sich um eine komplexe E-Commerce-Plattform. Vor Release der Software wurde diese u.a. durch einen Web-Application-Scanner überprüft. Mit diesem wurden 30 schwerwiegende Sicherheitslücken gefunden, welche im Anschluss behoben wurden. Durch eine erneute Untersuchung wurde dies verifiziert. Durch die Anwendung von Threat Modeling auf die bereits implementierte, veröffentlichte und mit etablierten Verfahren untersuchte Software konnten acht weitere Sicherheitslücken, davon 5 schwerwiegende, identifiziert werden. Die Klassifizierung erfolgte anhand der SDL Security Bug Bar [HL06]. Bei den Sicherheitslücken handelt es sich sowohl um Implementierungsfehler als auch um Designfehler (z.B. der unberechtigte Zugriff auf eine Datenbank), welche durch die frühzeitige Anwendung von Threat Modeling hätten vermieden werden können. Für eine detaillierte Aufstellung der identifizierten Sicherheitslücken siehe Abbildung 6: Durch Threat Modeling identifizierte Sicherheitslücken in veröffentlichter Individualsoftware [nach: Ju10].

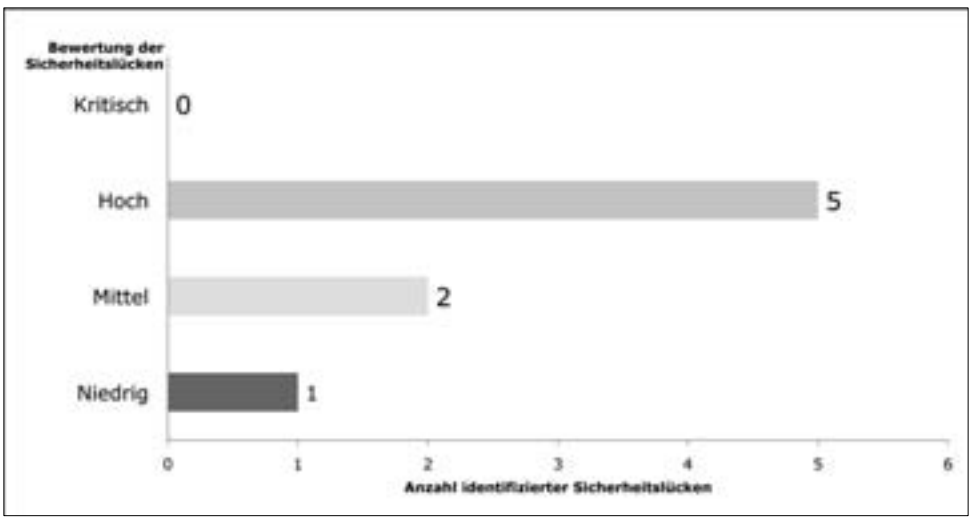

Abbildung 6: Durch Threat Modeling identifizierte Sicherheitslücken in veröffentlichter Individualsoftware [nach: Ju10]

Im Rahmen eines weiteren Forschungsprojektes[2] wurde ein zu entwickelnder Prototyp in der Design-Phase durch Threat Modeling untersucht. Auf Grundlage der Ergebnisse konnten im Entwicklungszyklus zahlreiche potentielle Bedrohungen identifiziert und vermieden werden (z.B. Denial of Service oder Information Disclosure)

[2] KMU-innovativ - Verbundprojekt EOMS: RFID in der Logistik - Offene Systeme auf der Basis von EPC und ONS (Förderkennzeichen 01IS08043B)

3 Fuzzing

3.1 Einführung Fuzzing

Fuzz-Testing (Fuzzing) ist eine Software-Überprüfungsmethode, die im Rahmen des Security Development Lifecycle (SDL) idealerweise in der Verifikationsphase eingesetzt wird [LH05], aber auch später nach Auslieferung der Software erfolgreich ist. Die Verifikationsphase befindet sich zwischen der Implementierungs- und der Release-Phase des SDL (Abbildung 1).

Der Fuzzing-Lifecycle beschreibt die Durchführung von Fuzzing [TDM08]:

1. Identifikation von Eingabeschnittstellen,
2. Generierung von Eingaben,
3. Versand von Eingaben,
4. Überwachung der Zielsoftware (Monitoring),
5. Durchführung einer Exception-Analyse und
6. Erstellung von Berichten (Reporting).

Nach erfolgreicher Identifizierung von Schnittstellen können Eingabedaten durch einen Fuzzer generiert und an die zu testende Software gesendet werden. Durch einen Monitor wird die Software während des Fuzzing auf auftretende Fehler überwacht, die durch möglichst gezielt gewählte Eingabedaten provoziert werden.

3.2 Bewertungsverfahren

Für die Bewertung von Fuzzern wird ein mehrstufiges Schema aus Bewertungsparametern, Kategorien und Gewichtungen herangezogen. Folgende Kategorien von Bewertungsparametern werden in diesem Schema verwendet und sind unterschiedlich gewichtet:

- Fuzzing-Methoden: Bewertet wird die Möglichkeit, einen Fuzzer für unterschiedliche Aufgaben einzusetzen. Darunter fallen z.B. die Unabhängigkeit des Produkts, Schnittstellen zu interpretieren, das Erlernen von Protokollspezifikationen oder die Möglichkeiten, die Zielsoftware zu interpretieren.
- Sonstiger Umfang: Bewertung sonstiger von Fuzzern bereitgestellter Methoden und Funktionen, um die Qualität des Fuzzing zu erhöhen. Hier werden insbesondere Möglichkeiten bemessen, aufgetretene Fehler zu identifizieren, einzugrenzen, auszuwerten und zu präsentieren.
- Software-Ergonomie: Bewertung insbesondere der Effektivität, Effizienz und Zufriedenstellung bei der Durchführung von Fuzzing im Umgang mit dem Produkt. Weiter werden Funktionale-, Dialog- sowie Ein- und Ausgabekriterien bewertet.

- Dokumentation: Bewertung des Umfangs und der Qualität der bereitgestellten Dokumentationsressourcen, wie Benutzerhandbuch, technische Dokumentation, integrierte Hilfesysteme etc.
- Entwicklungsmöglichkeiten: Bewertung der durch das Produkt bereitgestellten Funktionen zur Ergänzung eigener Funktionen oder zur Erweiterung bestehender.
- Kosten: Bewertung der anfallenden Kosten, insbesondere Anschaffungskosten, Wartungskosten und Personalkosten werden exemplarisch anhand von Fallstudien quantifiziert.

3.3 Marktanalyse

Es existieren über 100 Fuzzer. 25% aller Fuzzer können Web-Applikationen und 34% sonstige Netzwerkprotokolle testen. Das Testen von Dateiformaten wird von 15% aller Fuzzer unterstützt. Web-Browser können durch 10% und APIs durch 7% aller Fuzzer untersucht werden. Vgl. Abbildung 7: Marktübersicht von Fuzzern.

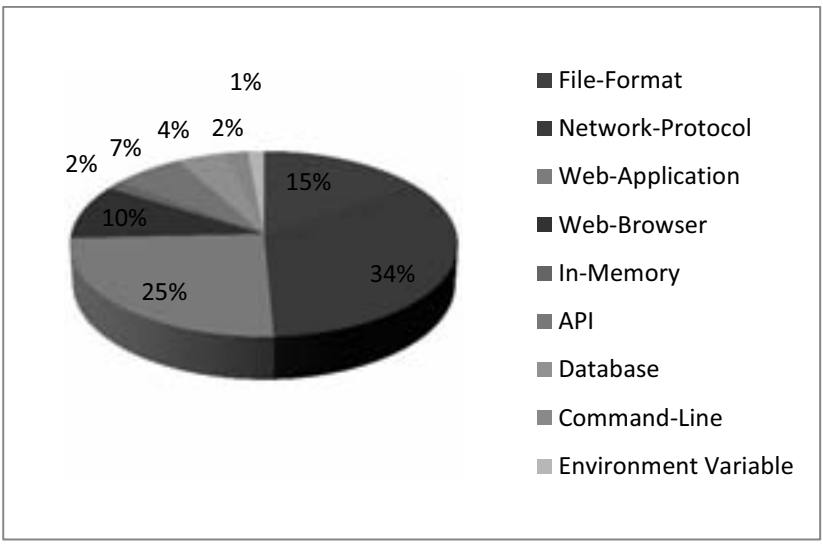

Abbildung 7: Marktübersicht von Fuzzern

Im Folgenden werden zwei Fuzzer von je zwei Fuzzer-Arten aufgrund ihrer Verbreitung exemplarisch untersucht, die das Testen mehrerer Schnittstellen unterstützten. Die untersuchten und bewerteten Fuzzer können der Abbildung 8: Einige untersuchte Fuzzer – entnommen werden.[3].

Fuzzer der Kategorie „Multi-Protocol Fuzzer“ unterstützen im Lieferumfang bereits mehrere Protokolle und können somit mehrere Schnittstellen untersuchen. Fuzzer der Kategorie „Fuzzing Framework“ müssen vor Verwendung auf die jeweiligen Protokolle angepasst werden.

[3] Die Ergebnisse der Untersuchung aller verfügbaren Fuzzer ist Teil des Forschungsprojekts und wird über diese Veröffentlichung hinaus nochmals veröffentlicht

Name	Kategorie	Version
beSTORM	Multi-Protocol Fuzzer	3.7.5 (4480)
Peach	Fuzzing Framework	2.3.4
Defensics	Multi-Protocol Fuzzer	3.8.3
Sulley	Fuzzing Framework	r156

Abbildung 8: Einige untersuchte Fuzzer

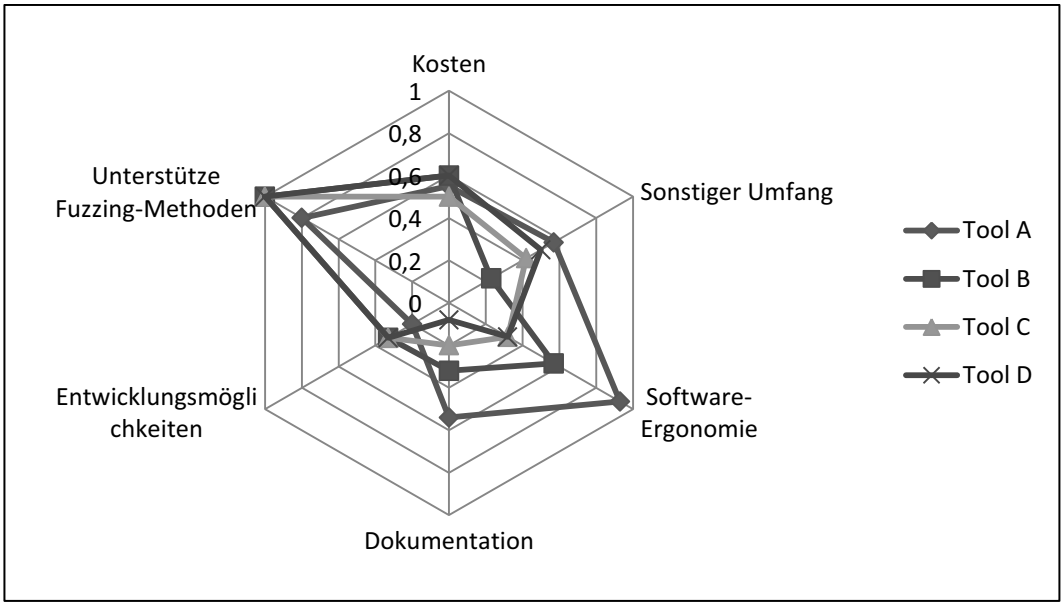

Abbildung 9: Bewertung der Fuzzer

Tool A zeichnet sich insbesondere durch sehr hohe Software-Ergonomie aus. Die Dialoge sind sehr benutzerfreundlich und selbsterklärend gestaltet; die Dokumentation ist vollständig und ausführlich, der Einsatz weiterer Medien ist jedoch wünschenswert. Die Weiterentwicklungsmöglichkeiten sind hier allerdings gering. Im Vergleich zu den anderen Tools ist der Umfang unterstützer Fuzzing-Methoden geringer.

Tool B bietet befriedigende Software-Ergonomie und Dokumentation. Wenig ausgeprägt ist der sonstige Umfang. Dafür werden sehr viele Fuzzing-Methoden unterstützt, die eine große Bandbreite von Anwendungsgebieten ermöglicht.

Tool C und D erlangen sehr geringe Wertungen in der Benutzerfreundlichkeit. Dafür sind die Einsatzmöglichkeiten sowie der Umfang bei beiden Produkten hoch. Sie unterscheiden sich nur geringfügig voneinander, wobei sich Tool C leicht von Tool D abhebt.

Bei den Kosten der Produkte gibt es geringe Unterschiede. Tool A und B weisen höhere Anschaffungskosten, Tool C und D höhere Personalkosten auf.

Insgesamt hebt sich Tool A am meisten ab, unterstützt jedoch nicht alle Fuzzing-Methoden. Keines der Produkte erhält mehr als 60% der möglichen Punkte in der Kategorie Umfang. Daher empfiehlt sich grundsätzlich die Nutzung von mindestens zwei Fuzzern.

Abbildung 9 – Bewertung der Fuzzer – stellt die Ergebnisse grafisch dar.

Sollen Fuzzer für mehrere Schnittstellen eingesetzt werden, sollte der Grad unterstützter Schnittstellen berücksichtigt werden. Die benötigte Expertise zum Einsatz von Fuzzern ist ein weiteres Maß, anhand dessen Fuzzer betrachtet werden können. Die Produkte D und C unterstützen mehr Schnittstellen als Produkte A und B, setzen jedoch eine höhere Expertise voraus. Vgl. Abbildung 10: Vergleich von Fuzzern anhand von Schnittstellen und benötigter Expertise.

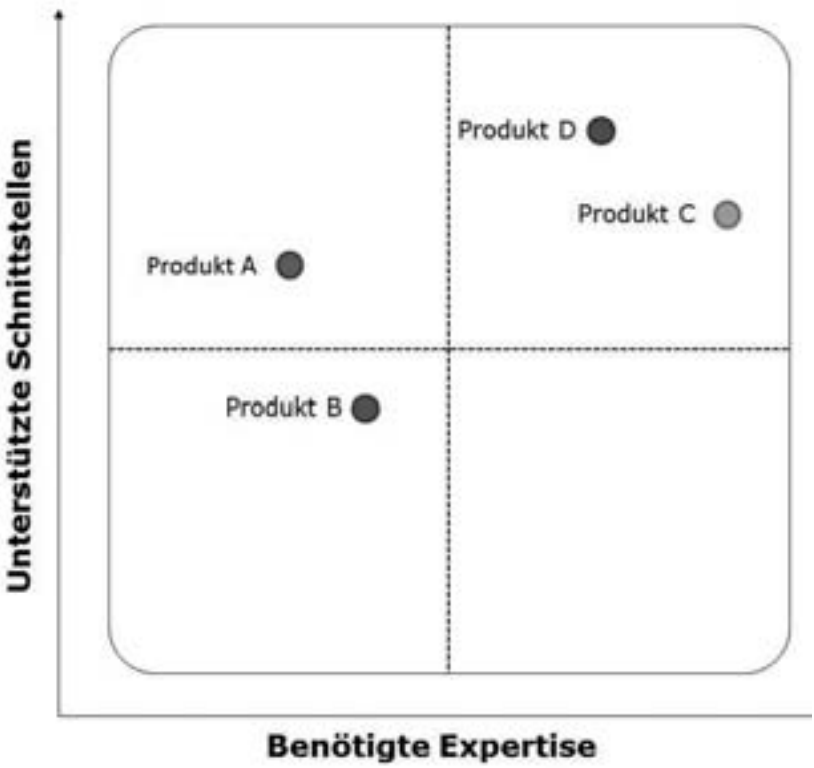

Abbildung 10: Vergleich von Fuzzern anhand von Schnittstellen und benötigter Expertise

3.4 Fallstudie: Erfolgreiches Fuzzing

In einer Fallstudie des Projekts SoftSCheck der Hochschulle Bonn-Rhein-Sieg wurde unter Einsatz von Fuzzing-Tools in einer kommerziellen (daher namentlich nicht genannten) Web-Applikation eine Vielzahl von Sicherheitslücken identifiziert. Darunter fallen u.A. Cross Site Scripting-, Cross Site Request Forgery-, Session Fixation- und Authentifizierungssicherheitslücken [Ow10]. Die Schweregrade der Sicherheitslücken wurde nach dem Common Vulnerability Scoring System [MSR07] errechnet und auf die Kritikalitäten „Kritisch“, „Hoch“, „Mittel“, „Niedrig“ abgebildet (Abbildung 11: Kritikalität identifizierter Sicherheitslücken eines Fallbeispiels). Je kritischer eine Sicherheitslücke ist, desto größer sind der potenzielle Schaden und desto geringer der Aufwand zur Ausnutzung der Sicherheitslücke.

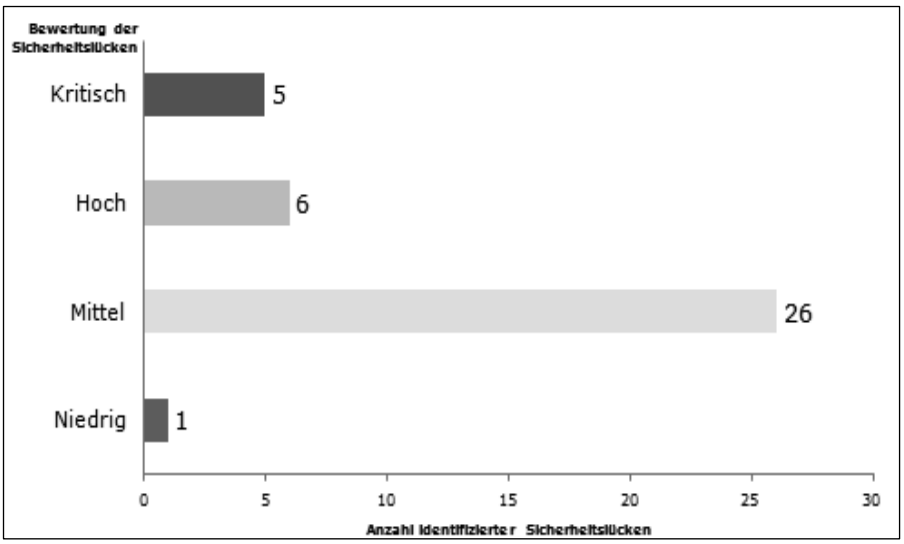

Abbildung 11: Kritikalität mit Fuzzing identifizierter Sicherheitslücken eines Fallbeispiels

4 Anwendung auf Telematikanwendungen im Gesundheitswesen

Die Normen DIN EN 61508 und DIN EN 62304 beschreiben Sicherheitsanforderungen für die Entwicklung von Software im medizinischen Umfeld. Diese beinhalten u.a. Vorschriften zur Verifikation und Diagnose (Kapitel C5, DIN EN 61508-7 [Di09]), Beurteilung der funktionalen Sicherheit (Kapitel C6, DIN EN 61508-7 [Di09]), Implementierung und Verifikation von Software-Einheiten (Kapitel 5.5, DIN EN 62304 [Di07]) und Prüfung des Softwaresystems (Kapitel 5.7, DIN EN 62304 [Di07]). Diese Forderungen und auch höherwertige Anforderungen an das Sicherheitsniveau können durch die Anwendung von Threat Modeling und Fuzzing in vollem Umfang erfüllt werden. Durch Anwendung von Threat Modeling auf die Gesamtstruktur der Gesundheitskarte, sowie auf einzelne Bestandteile dieser, können Bedrohungen und Sicherheitslücken frühzeitig erkannt und mit minimalen Kosten behoben werden. Durch Anwendung von Fuzzing können - durch klassische Verfahren nicht erkannte - Sicherheitslücken identifiziert werden. Die Anwendung der Verfahren Threat Modeling und Fuzzing ist kosteneffizient und

ergänzend zu der Anwendung formalisierter Sicherheitsüberprüfungen (z.B. Common Criteria) durchzuführen. Derzeit wird eine Integration von Threat Modeling in die Common Criteria geprüft mit dem Ziel den Zertifizierungsprozess sicherer Software zu verkürzen. Durch den Einsatz von Threat Modeling und Fuzzing kann ein höheres Sicherheitsniveau der Telematikanwendungen im Gesundheitswesen erreicht werden.

5 Fazit / Ausblick

Die bisherigen Erfolge haben gezeigt, dass mit den Verfahren Threat Modeling und Fuzzing selbst bei Standardsoftware sehr viele aus dem Internet ausnutzbare kritische Sicherheitslücken gefunden werden konnten - trotz eines hohen Sicherheitsstandards in den Programmierrichtlinien [SP10]. Es ist zu erwarten, dass auch bei Software für die Gesundheitstelematik ähnlich hervorragende Ergebnisse erreicht werden.

Literaturverzeichnis

[Di07] DIN Deutsches Institut für Normung e.V. (Hrsg.): Medizingeräte-Software - Software-Lebenszyklus-Prozesse (IEC 62304:2006). Berlin 2007.

[Di09] DIN Deutsches Institut für Normung e.V. (Hrsg.): Funktionale Sicherheit sicherheitsbezogener elektrischer/elektronischer/programmierbarer elektronischer Systeme - Teil 7: Anwendungshinweise über Verfahren und Maßnahmen (IEC 65A/528/CDV:2008). Berlin 2009.

[GKL08] Godefroid, P.; Kiezun, A.; Levin, M.Y.: Grammar-based Whitebox Fuzzing. o.O. 2008. http://research.microsoft.com/en-us/projects/atg/pldi2800.pdf

[HL06] Howard, M.; Lipner, Steve: The Security Development Lifecycle. SDL: A Process for Developing Demonstrably More Secure Software. Redmond 2006.

[Jo96] Jones, C.: Applied Software Measeurement. New York 1996.

[Ju10] Juhasz, S.: Dokumentation und Überprüfung des Sicherheitsniveaus von Individualsoftware: Einsatz der Bedrohungsmodellierung (Threat Modeling). Sankt Augustin 2010.

[LH05] Lipner, S.; Howard, M.: The Trustworthy Computing Security Development Lifecycle. o.O. 2005. http://msdn.microsoft.com/en-us/library/ms995349.

[Lü10] Lübbert, J.: Bewertung von Netzwerkprotokoll-Fuzzern. Entwicklung einer Taxonomie zur Klassifizierung und Aufbau von Parametern zur Evaluation von Fuzzern. Sankt Augustin 2010.

[MLY05] Myagmar, S.; Lee, Adam J.; Yurcik, W.: Threat Modeling as a Basis for Security Requirements. Symposium on Requirements Engineering for Information Security. Paris 2005. http://www.suvda.com/papers/threat_sreis05.pdf.

[MSR07] Mell, P; Scarfone, K; Romanosky, S: A Complete Guide to the Common Vulnerability Scoring System Version 2.0. o.O. 2007. http://www.first.org/cvss/cvss-guide.pdf

[My05] Myagmar S.: Threat Modeling networked and data-centric systems, Urbana 2005, http://www.projects.ncassr.org/threatmodeling/myagmar-msthesis.pdf

[Ni02] National Institute of Standards and Technology (NIST) (Ed.): The Economic Impacts of Inadequate Infrastructure for Software Testing. Gaithersburg 2002. http://www.nist.gov/director/prog-ofc/report02-3.pdf

[Ow09] OWASP Foundation (Ed.): Threat Risk Modeling. Columbia 2009. http://www.owasp.org/index.php/Threat_Risk_Modeling

[Ow10] OWASP Foundation (Ed.): Category:Vulnerability – OWASP. Columbia 2010. http://www.owasp.org/index.php/Category:Vulnerability

[Po07] Pohl, H.: Zur Technik der heimlichen Online Durchsuchung. DuD, Ausg. 31. 2007, 684 - 688.

[Sa09] Sakal, P.: Identifikation und Evaluation von Fuzzing Tools, zur Nutzenmaximierung bei der Identifikation und Lokalisierung sicherheitsrelevanter Softwarefehler. Sankt Augustin 2009.

[Sa10] SANS (Ed.): The Top Cyber Security Risks. o.O. 2010 http://www.sans.org/top-cyber-security-risks/?ref=top20

[Sc99] Schneier, B.: Attack Trees: Modeling Security Threats. In: Dr. Dobb's Journal, v. 24, n.12. San Francisco 1999. http://www.schneier.com/paper-attacktrees-ddj-ft.html

[Sc06] Schumacher, M.; Fernandez-Buglioni, E.; Hybertson, D.; Buschmann, F.; Sommerlad, P.: Security Patterns. Integrating Security and Systems Engineering. Hoboken 2006.

[Sc10] Schwab, F.; Findeisen, A.; Sakal, P.; Pohl, H.: Bedrohungsmodellierung (Threat Modeling) in der Softwareentwicklung.GI-Edition: Lecture Notes in Informatics. Berlin 2010.

[SGA07] Sutton, M.; Greene, A.; Amini, P.: Fuzzing – Brute Force Vulnerability New York 2007.

[SP10] Sakal, P.; Pohl, H.: Entwicklungshelfer und Stresstester - Tool-gestützte Identifizierung von Sicherheitslücken in verschiedenen Stadien des Softwarelebenszyklus. In: <kes> - Die Fachzeitschrift für Informations-Sicherheit, 2, 2010

[SS04] Swiderski, F.; Snyder, W.: Threat Modeling. Redmond 2004.

[TDM08] Takanen, A.; Demott, J.D.; Miller, C.: Fuzzing for Software Security Testing and Quality Assurance. Norwood 2008.