softScheck
we identify vulnerabilities others don't

Software-Sicherheit

Kostengünstige Identifizierung von Sicherheitslücken

Threat Modeling, Static Analysis und Dynamic Analysis (Fuzzing)

White Paper

Prof. Dr. Hartmut Pohl, CEO
Hartmut.Pohl@softScheck.com

www.softScheck.com

20. März 2011

Überarbeitete Fassung des 2010 erschienenen Beitrag in der KES - Zeitschrift für Informationssicherheit, Heft 2

Mit dem Einsatz von Tools zum Threat Modeling (Designphase) und Fuzzing (Verifikationsphase) werden auch bisher nicht erkannte Sicherheitslücken identifiziert. Diese Identifizierung erfordert einen weitaus geringeren Aufwand, als wenn die Software schon ausgeliefert ist (Release-Phase) und erst dann korrigiert werden kann - vgl. Abb. 1. Zudem lassen sich diese Verfahren für alle Anwendungsbereiche einsetzen: Individualsoftware, Standardsoftware wie ERP, CRM und auch unternehmensspezifische Ergänzungen, Betriebssysteme, Webbrowser, Webapplikationen, Netzwerk-Protokolle etc.

Der Einsatz herkömmlicher Verfahren zur Behebung von Fehlern und insbesondere nicht veröffentlichten Sicherheitslücken ist sehr kostenaufwändig - viele Sicherheitslücken werden erst nach der Auslieferung der Software an die Kunden - z.T. auch von Dritten - erkannt.

Den exponentiell steigenden Aufwand zur Behebung (Patchen) von Sicherheitslücken zeigt Abb. 2. Tatsächlich steigt der Patchaufwand nach einigen Untersuchung - u.a. des National Institute of Standards and Technology (NIST) - in der Designphase zur Release-Phase auf bis zum 100-fachen.

Aus eigenen Untersuchungen wissen wir, dass der Aufwand zur Identifizierung von Sicherheitslücken mit Threat Modeling und Fuzzing sehr kostengünstig durchgeführt werden kann – auch wenn sich die Software schon auf dem Markt befindet.

Sicherheitslücken werden wie folgt identifiziert:

1. Systematische Suche nach Sicherheitslücken mit den Verfahren Threat Modeling und Fuzzing.
2. Identifizieren der wesentlichen, schwerwiegendsten, (remote) aus dem Internet leicht ausnutzbaren Sicherheitslücken und bewerten der übrigen Sicherheitslücken (Priorisierung).

Fuzzing benötigt zum Erfolg keinen Quellcode (Blackbox-Test): Die Software wird während Ihrer Ausführung untersucht. Hierzu kann die Software in einer Virtuellen Maschine oder auf einem anderen Testsystem ausgeführt werden.

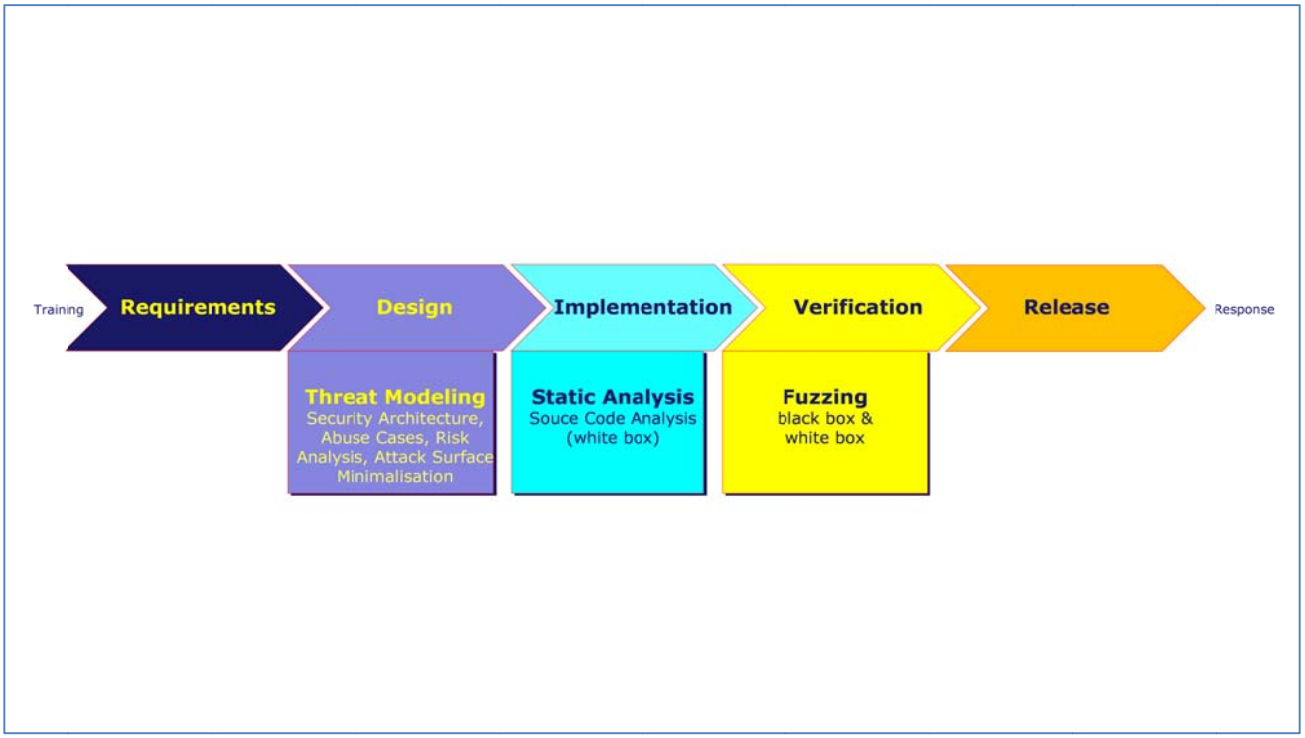

Abb. 1: Lebenszyklus der Entwicklung sicherer Software

1 Keine Software ohne Fehler

Software kann nicht fehlerfrei erstellt werden. Dies macht eine Überprüfung der Software erforderlich. Manuelle Überprüfungen sind angesichts des meist großen Code-Umfangs nicht praktikabel.

In der Welt des traditionellen Software Testing gibt es keine Sicherheitslücken im Design und in der Software Implementierung und (unter Sicherheitsaspekten) nur vollkommene Benutzer. Ziele von Tests sind allein die spezifizierten Funktionalitäten, die getestet werden. Nicht-funktionale Tests werden vernachlässigt.

Mit den Verfahren Threat Modeling und Fuzzing können Sicherheitslücken erkannt werden, die die beispielsweise die folgenden Angriffe ermöglichen:

- Verletzung der Zugriffsregeln
- Formatstring-Angriffe
- SQL-Injections
- Buffer Overflows.

Beim Fuzzing werden hierzu die Eingabeschnittstellen (Attack Surface) gerade mit solchen Eingaben attackiert, die nicht spezifiziert sind, wodurch ein Fehlverhalten der Software provoziert wird. Beim Threat Modeling werden auch Design-Fehler erkannt, indem z.B. Angriffsbäume und Datenflussdiagramme ausgewertet werden.

2 Software-Entwicklungszyklus

Software wird unter Einsatz von Methoden hergestellt, die Sicherheitslücken nicht vollständig vermeiden können – menschliche Fehler können nicht ausgeschlossen werden; selbst wenn Programmierrichtlinien vorhanden sind, werden sie nicht (vollständig) eingehalten und nicht (vollständig) kontrolliert. Dabei existieren wirkungsvolle Tools weit über klassische Verfahren des Testing hinaus, die Sicherheitslücken bereits in der Designphase identifizieren (Threat Modeling) oder spätestens in der Verifikationsphase (Fuzzing) - allzu viele Sicherheitslücken werden aber nach wie vor erst dann identifiziert, wenn Software an den Kunden ausgeliefert ist. Im Folgenden werden die Methoden zur Identifizierung von Sicherheitslücken dargestellt und einige bekannte Tools genannt.

Die Kosten zur Beseitigung von Softwarefehlern, sind abhängig von deren zeitlicher Entdeckung im Software Development Lifecycle (SDL).

Werden Fehler erst nach dem Release – beim Endkunden - entdeckt, steigen die Kosten um den Faktor 100 - vgl. Abb. 2.

Die Qualität von Softwareprodukten ist häufig auf mangelnde Ressourcen in den entwickelnden Unternehmen zurückzuführen. Zudem werden vom Markt sehr kurze Produktlebenszyklen vorgegeben. Dadurch lassen sich Fuzzing und Threat Modeling, als marktgerechte Methode zur Entdeckung von Softwarefehlern nutzen. Hier werden im Vergleich zu traditionellen Methoden nur wenige Ressourcen benötigt – so zeigen Untersuchungen in Projekten von softScheck.

Mittels Fuzzing und Threat Modeling werden Softwarehersteller, Nutzer und solche, die Standardsoftware anpassen (Customizing) in die Lage versetzt, Software-Tests wirkungsvoller und kostengünstiger mit den für ihre Aufgabenstellung geeigneten Tools durchzuführen und dadurch die Software sicherer zu machen: Mit Hilfe dieser Tools lassen sich bisher nicht erkannte Fehler und Sicherheitslücken erkennen.

Im Rahmen des vom Bundesministerium für Bildung und Forschung an der Hochschule Bonn-Rhein-Sieg geförderten Projektes (FKZ 01 IS 09030) softScheck wurden über 300 weltweit verfügbare Tools zum Threat-Modeling und Fuzzing analysiert und bewertet.

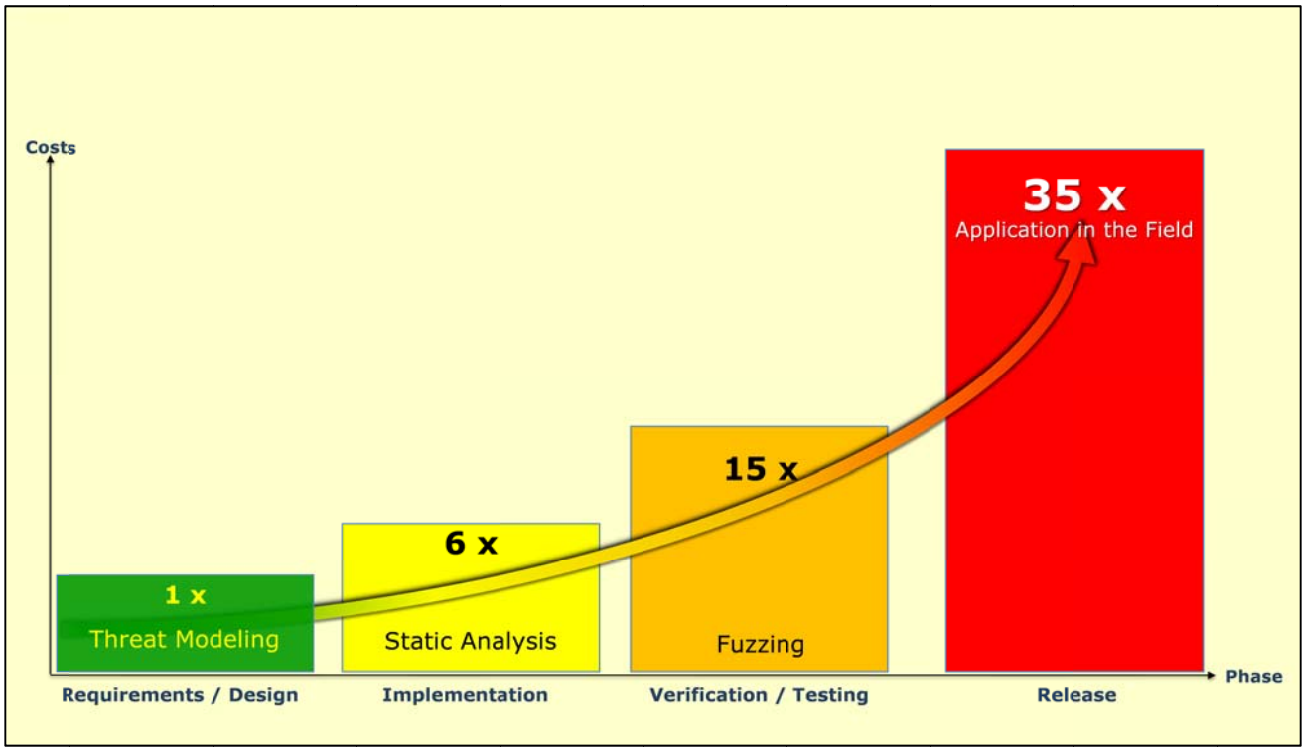

Abb. 2: Kosten der Behebung von Sicherheitslücken in den Software-Entwicklungsphasen

3 Threat Modeling

Dieses heuristische Verfahren unterstützt die methodische Entwicklung eines vertrauenswürdigen Systementwurfs oder einer Architektur in der Design Phase - sodass dies am kosteneffizientesten ist.

Gleichermaßen lassen sich bereits bestehende Systementwürfe und Architekturen verifizieren, mit dem Ziel der Identifizierung, Bewertung und Korrektur von Sicherheitslücken.

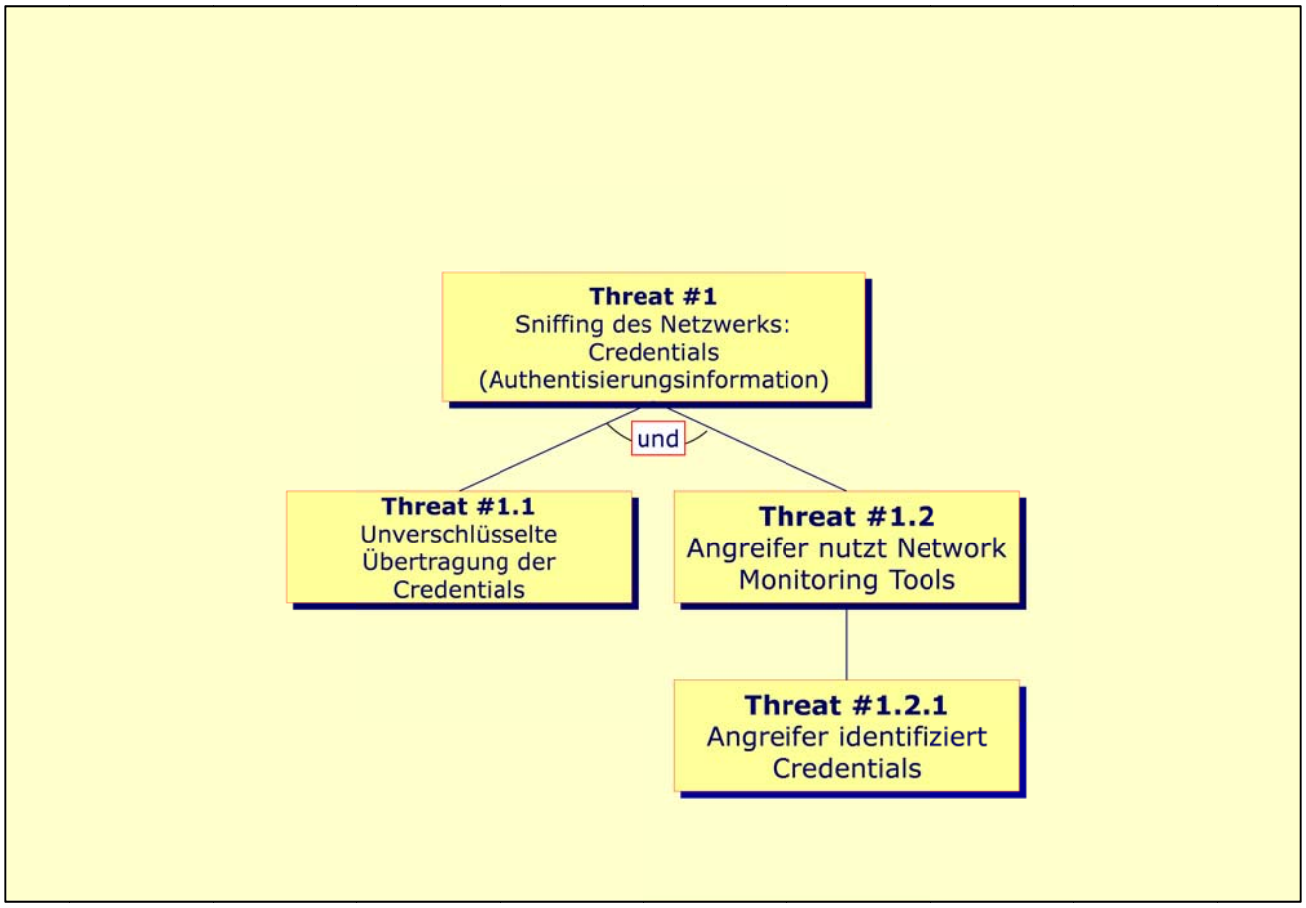

Abb. 3: Fehlerbäume im Threat Modeling

Vorgehen beim Threat Modeling

- Analyse der Dokumentation (falls vorhanden) – insbesondere des Sicherheitsdesigns oder des Quellcodes.
- Untersuchung der Programmablaufpläne
- Beschreibung und Priorisierung erkannter sicherheitsrelevanter Designfehler
- Entwicklung von Fehler Bäumen (attack trees) - vgl. Abb. 3.
- Report Generation: Bericht und Vorgehensempfehlungen für erkannte Sicherheitslücken

Ziel ist weiterhin das Verständnis der Sicherheitsarchitektur, das Erkennen von Designfehlern und die Minimierung möglicher Angriffspunkte (Attack Surface).

Einige Threat Modeling Tools

- Microsoft Threat Analysis & Modeling
- Microsoft SDL Threat Modeling Tool
- Trike

Die Anzahl von mit Threat Modeling gefundenen Sicherheitslücken ist erheblich. So wurden in einem Projekt der Autoren kritische Sicherheitslücken bereits in der Designphase identifiziert, die aus dem Internet ausnutzbar wären - vgl. Abb. 4.

4 Static Analysis

Dieses Verfahren analysiert den Quellcode, ohne ihn auszuführen (im Gegensatz zur Dynamic Analysis, wozu u.a. Fuzzing zählt). Dabei wird in der Implementierungsphase die Konformität mit der Programmiersprache und den Programmierrichtlinien überprüft - wie ein Parser, der eine lexikalische, syntaktische und semantische Analyse des Programmcodes durchführt.

Einige Static Analysis Tools

- Pixy
- XDepend

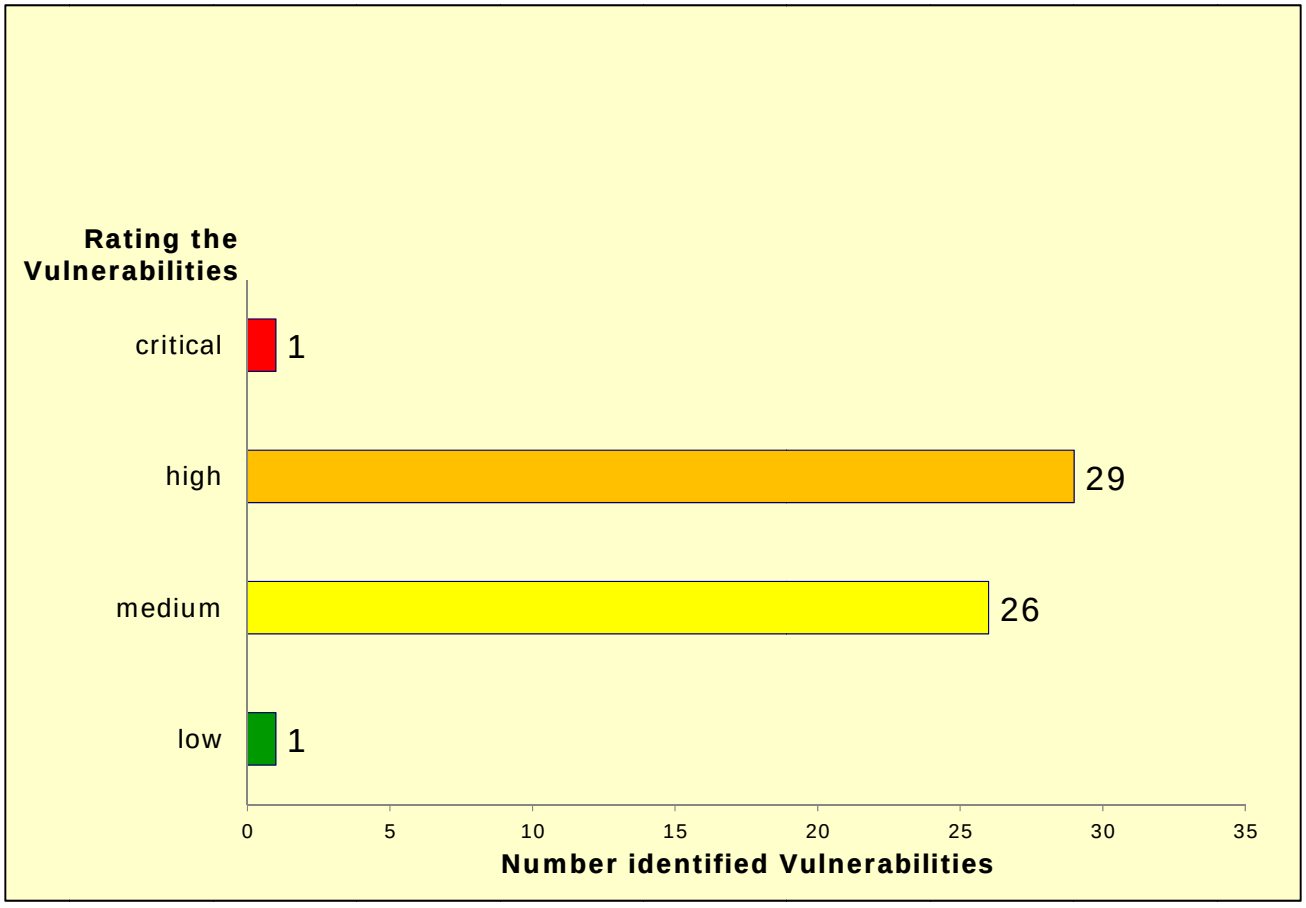

Abb. 4: Mit Threat Modeling erkannte Sicherheitslücken in Internet-Marktplatz

5 Fuzzing

Beim Fuzzing wird die Robustheit der untersuchten Software mit willkürlichen oder auch zielgerichteten Daten überprüft.

So kann sporadische Betriebsausfällen und unbeabsichtigten Datenabflüssen - den häufigsten Folgen sicherheitsrelevanter Softwarefehler - entgegengewirkt werden und so proaktiv hohe Umsatzausfälle, Datenschutzprobleme und Reputationsschäden gemindert werden.

Dazu werden beim Fuzzing Eingabeschnittstellen identifiziert, an die die Daten gesendet werden.

Die Qualität von Fuzzern hängt im Wesentlichen von der Größe des getesteten Eingaberaums (der unendlich ist) und der Qualität der erzeugten Daten ab.

Der Fuzz-Testing Prozess ist in der Verifikationsphase des SDL angesiedelt - vgl. Abb. 1. Hier sollten Fehler spätestens behoben werden, da die Kosten nach dem Release erheblich ansteigen wenn Fehler erst später gefunden werden – siehe Abb. 2.

Wurden die Eingabeschnittstellen der Zielanwendung identifiziert und die vom Fuzzer erzeugten Daten an die Zielanwendung geschickt, überwacht ein Monitoring-Tool die Zielanwendung und meldet dem Tester Anomalien wie z.B. Programmabstürze, hohe

CPU- oder Speicher-Auslastung etc. - vgl. Abb. 5.

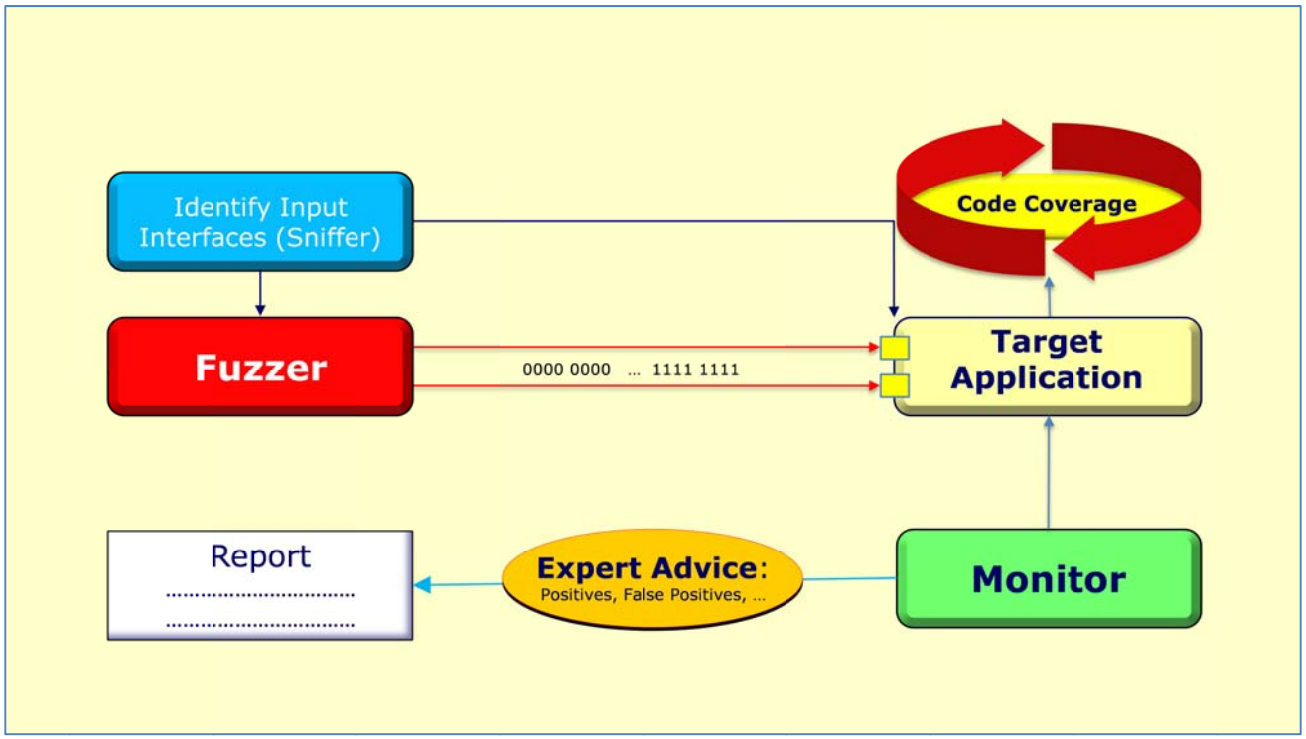

Abbildung 5: Fuzzing-Prozess

Im Anschluss erfolgt durch den Security Analyst eine Analyse, in der u.a. die Reproduzierbarkeit, Ausnutzbarkeit aus dem Internet und die Schwere der Sicherheitslücke bestimmt werden.

Für das Fuzzing wird der Quellcode der Zielanwendung nicht benötigt, was die Einbeziehung externer Security Analysten vereinfachen kann.

Je nach Typ lassen sich Fuzzer lokal und remote einsetzen. Zu den lokalen Fuzzern werden z.B. die Kommandozeilen-Fuzzer gezählt.

Netzwerkbasierte Anwendungen werden remote gefuzzed. Dazu kommen Fuzzer, die Web-Applikationen und Browser fuzzen.

Weiterhin können sog. dumb und smart Fuzzer unterschieden werden.

Smart-Fuzzer testen programmgesteuert selbständig ein Zielprogramm - oftmals ohne weitere Vorbereitung oder Begleitung durch den Anwender; sie sind häufig lizenzpflichtig. Ein wichtiges Bewertungskriterium für Fuzzer ist die Testabdeckung (code coverage); häufig ermöglichen nur smart Fuzzer das Eindringen in das Zielprogramm und Austesten des Programmcodes.

Dumb-Fuzzer können den Aufbau des Zielprogramms nicht erkennen und sie generieren nur ungesteuerte Eingabedaten. Wegen dieser fehlenden Programmsteuerung setzen sie eine erhebliche Erfahrung beim Anwender voraus; dafür sind sie häufig entgeltfrei aus dem Internet herunterladbar - vgl. Abb. 6.

Einige Fuzzing Tools

- AxMan
- beSTORM
- Defensics
- FileFuzz
- FTPStress Fuzzer
- Fuzz
- Peach
- SPIKE
- SPIKEfile

Fuzzing Frameworks bieten die Möglichkeit speziell für die eigene Zielanwendung angepasste Fuzzer zu entwickeln – ähnlich einem Baukasten. Der Entwicklungsaufwand ist jedoch verhältnismäßig hoch – zumal für alle gängigen Anwendungen bereits fertig nutzbare Fuzzer existieren. Fuzzing Frameworks eignen sich besonders bei neuen proprietären Zielanwendungen wie z.B. neue Netzwerkprotokolle. Kommerzielle Tools zeichnen sich häufig durch die Abkehr von der Kommandozeilensteuerung zu einer graphischen Benutzeroberfläche aus.

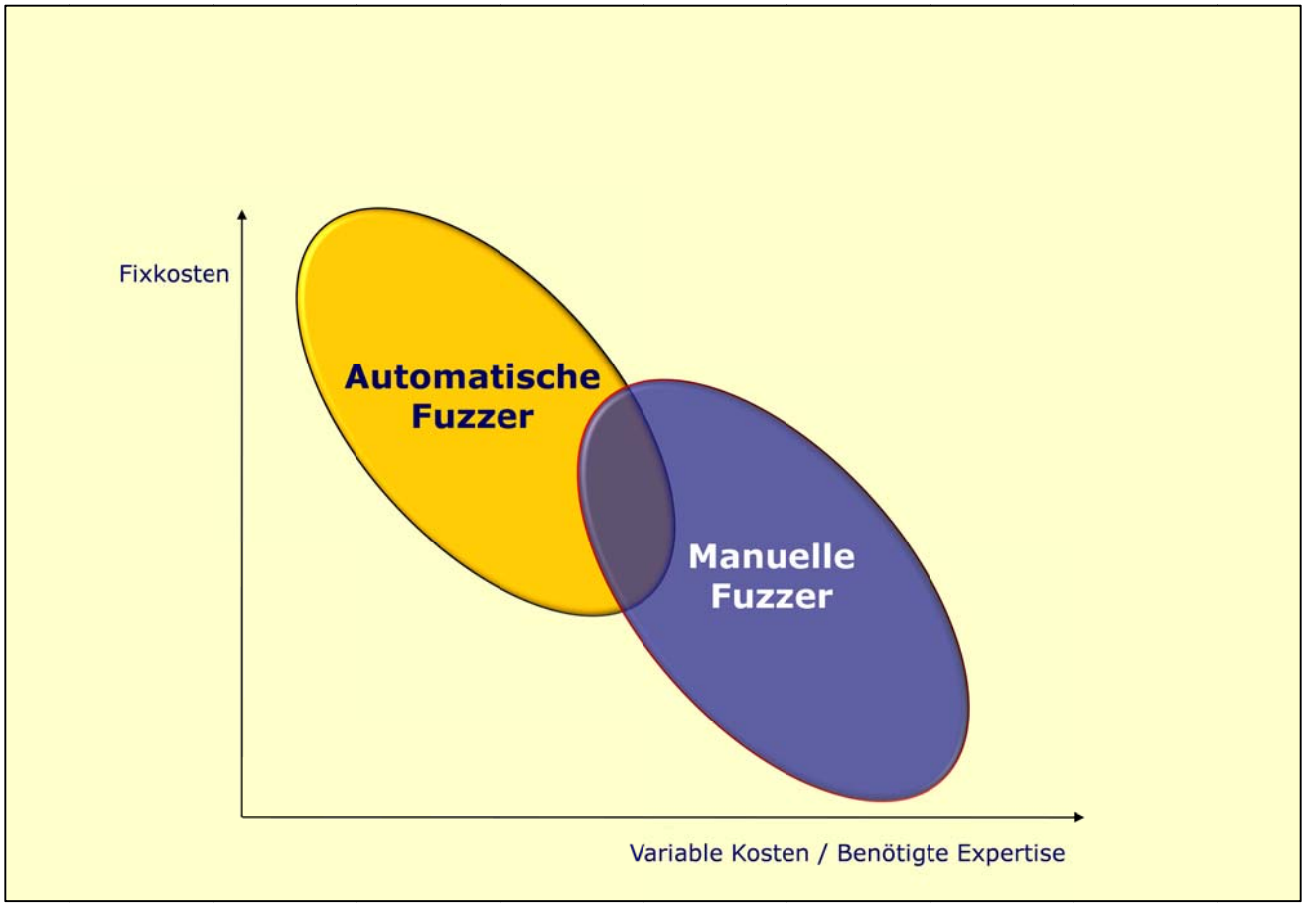

Abb. 6: Divergenz Produktkosten - Personalaufwand

6 Ergebnisse

Mit Fuzzing und Threat Modeling stehen Tool-gestützte Verfahren zur Identifizierung insbesondere sicherheitsrelevanter Softwarefehler zur Verfügung. So kann Zero-Day-Angriffen, die eine der zwanzig am häufigsten auftretenden Angriffsformen sind, entgegenwirkt und die Anwendungssicherheit effizient und marktgerecht gesteigert werden. Dabei lässt sich der Fuzzing Ansatz durch die zur Verfügung stehenden Methoden adäquat an die kürzer werdenden Software-Entwicklungszyklen anpassen.

Bei Softwareherstellern kann durch die Kombination der erläuterten Verfahren und die Integration in den SDLC ein höheres Return on (Security) Investment erreicht werden. Zudem kann die Qualität der Software gesteigert, können Markteinführungszeiten reduziert und damit insgesamt Kosten eingespart werden. Diese Aspekte sind auch schon von KMU erkannt worden.

Neben der Kostenersparnis kann die Reputation des Softwareherstellers – durch sicherere Software – gesteigert werden.

Threat Modeling und Fuzzing kann für alle Anwendungsarten eingesetzt werden - von Protokollen bis hin zur Individualsoftware und Webapplikationen. Zur Unterstützung einer bedarfsgerechteren Auswahl von geeigneten Tools hat das Projekt softScheck eine Taxonomie entwickelt, die gesondert veröffentlicht wird.

Ergebnisse des Verfahrenseinsatzes

- Systematische Suche und auch erfolgreiche (!) Identifizierung der wesentlichen Sicherheitslücken.
- Und zwar in **jeder** Software: Individualsoftware, Standardsoftware wie ERP, CRM und auch unternehmensspezifische Ergänzungen, Betriebssysteme etc.
- Kein Quellcode wird zum Threat Modeling und Fuzzing benötigt - ausführbare Dateien reichen völlig aus: Black-Box.

Die Anzahl der gefundenen Sicherheitslücken durch Fuzzing (meist Black Box: Ohne Vorliegen des Quellcode) ist enorm. Dies ist der Grund für die zunehmende Verbreitung und Beliebtheit des Verfahrens.

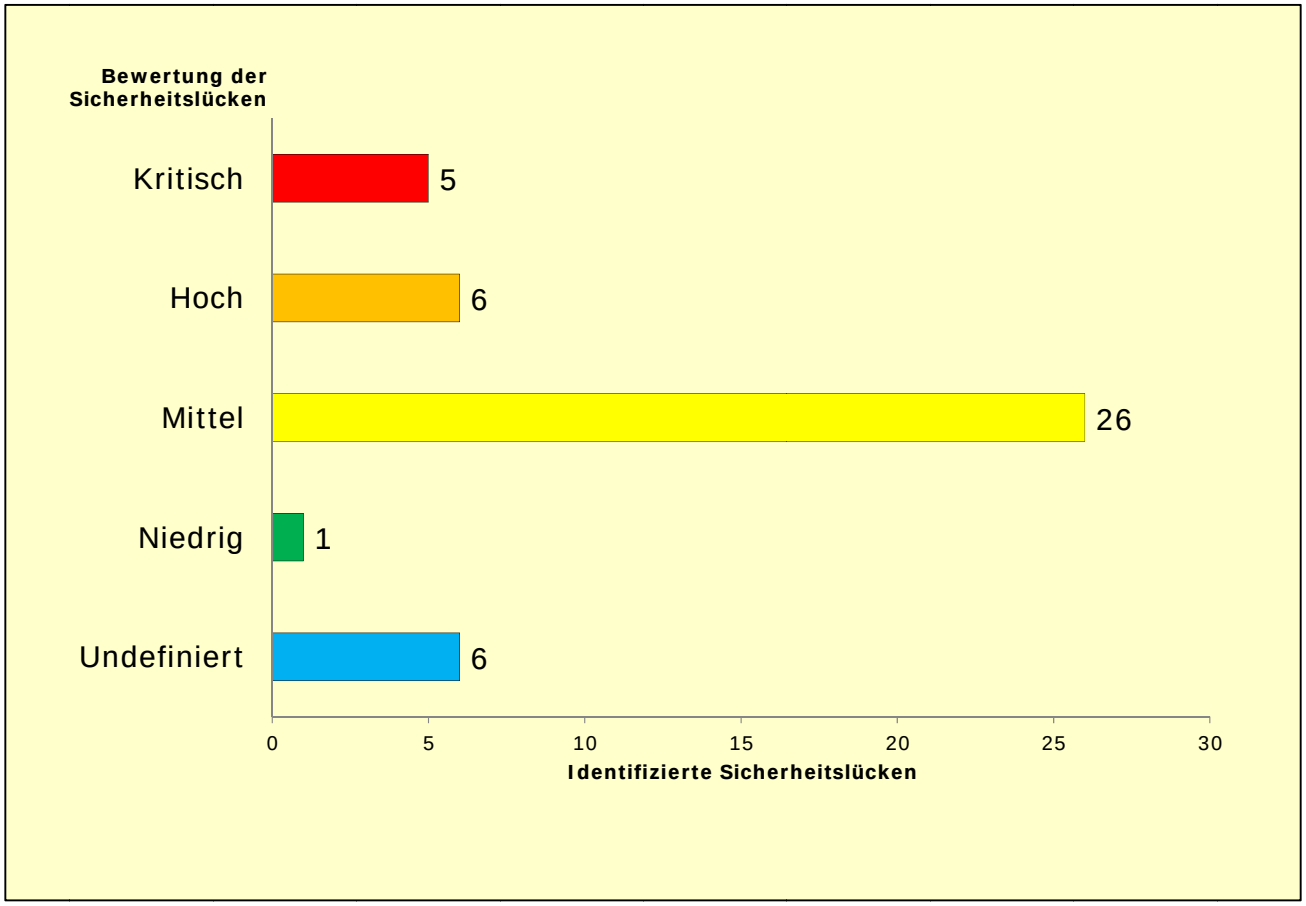

Abb. 7: Mit Fuzzing erkannte Sicherheitslücken in Standardsoftware

So wurden in einem Projekt der Autoren in bereits an Kunden ausgelieferter Standardsoftware fünf kritische (bisher nicht unveröffentlichte) Sicherheitslücken identifiziert, die aus dem Internet ausnutzbar waren - vgl. Abb. 7. Dies obwohl beim Hersteller ein umfangreiches Richtlinienwerk auch hinsichtlich der Programmierung 'sicherer' Software existiert.

Die erfolgreichen und kostengünstigsten Verfahren sind:

- **Threat Modeling**: Systematische und Tool-gestützte Verfahren. Sicherheitslücken werden bereits in der Designphase identifiziert. Sicherheitslücken können priorisiert werden: Aus dem Internet ausnutzbar und/oder geringer Aufwand für den Angreifer.
- **Fuzzing**: Black-Box - kein Quellcode erforderlich - ausführbare Dateien reichen völlig aus! Gängige Verfahren - seit mehr als 5 Jahren bei den weltweit größten Softwarehäusern im Einsatz - auch in KMU.
- **Static Analysis**: Tool-gestütztes Code Reading ergänzt die beiden Verfahren.

7 Weiterführende Literatur

Beyond Security (Ed.): Black Box Testing. McLean 2008. http://www.beyondsecurity.com/black-box-testing.html

Beyond Security (Ed.): Beyond Security introduces 80/20 rule for 'smart' blackbox testing in new version of beSTORM. McLean 2006. http://www.beyondsecurity.com/press/2006/press12090601.html

Codenomicon (Ed.): Buzz on Fuzzing. Cupertino 2007. http://www.codenomicon.com/products/buzz-on-fuzzing.shtml

Fox, D.: Fuzzing. DuD 30, 2006, 12, 798. 2006.

Peter, M.; Karen, S.; Romanosky, S.: Complete Guide to Complete Guide to the Common Vulnerability Scoring System Version 2.0 Gaithersburg 2007 http://www.first.org/cvss/cvss-guide.pdf

Pohl, H.: Zur Technik der heimlichen Online-Durchsuchung. DuD 31, 9, 2007. http://www.dud.de/binary/DuD_Pohl_907.pdf

Rathaus, N.; Evron, G.: Open Source Fuzzing Tools. Amsterdam 2007.

Doyle, F.; Fly, R.; Jenik, R.; Manor, D. Miller, C.; Naveh, Y.: Open Source Fuzzing Tools. Amsterdam 2007

Sutton, M,; Greene, A.; Amini, P.: Fuzzing - Brute Force Vulnerability Discovery. New York 2007.

Takanen, A.; Demott, J.D.; Miller, C.: Fuzzing For Software Security Testing And Quality Assurance. Norwood 2008