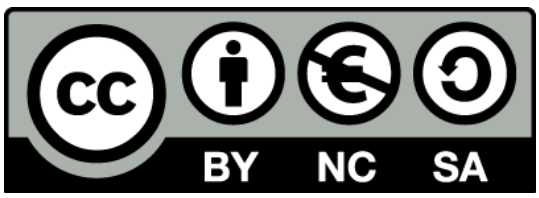

# Beschreibung des Einsatzes von Werkzeugen für Sicherheitstests in Open Source

Mikail Osipov
ossipov@inf.fu-berlin.de
Matrikel-Nr.: 3887133

Betreuer: Prof. Dr. Lutz Prechelt,
Dipl.-Medieninf. Martin Gruhn

Arbeitsgruppe Software Engineering
Institut für Informatik

Berlin, 23. Juni 2008


BY NC SA

# Erklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Berlin, den 23. Juni 2008 .....

(Mikail Osipov)

## Zusammenfassung

Diese Fallstudie untersucht eine Menge von Open Source-Projekten und versucht qualitativ-deskriptiv aufzudecken, wie Werkzeuge für Sicherheitstests in diesen Projekten eingesetzt werden. Im Detail sollen die Stufen des Einsatzes der verschiedenen Werkzeugarten und die Hürden und Probleme beschrieben werden. Es soll verdeutlicht werden, warum sich die Entwickler für diese oder gegen jene Werkzeuge entscheiden und wie sie die Sicherheit im System erhöhen.

# Inhaltsverzeichnis

# 1 Einleitung

Die Einleitung soll einen Einblick über die Ziele der Bachelorarbeit geben und den Zusammenhang zum Thema beschreiben. Zum Anfang soll die Frage beantwortet werden, warum Sicherheit überhaupt ein interessantes Studienobjekt ist und wo der Zusammenhang zu Open Source ist. Um weiter fortzufahren, muss geklärt werden, was Sicherheit allgemein bedeutet und welche Fehler im System eine Bedrohung der Sicherheit darstellen können. Darauf aufbauend wird die Verbindung von Open Source und dem eigentlichen Thema der Arbeit hergestellt.

## 1.1 Warum ist Sicherheit interessant?

Meldungen über geknackte Webseiten, neue Computerwürmer, Verwundbarkeiten und andere sicherheitskritische Ereignisse werden im täglichen Rhythmus auf Sicherheitsportalen und in Sicherheitdatenbanken, wie heise Security, Secunia, SecurityFocus, CVE[1], US-CERT[2] oder anderen Diensten, erfasst. Laut Secunia haben sich nicht nur die Meldungen in den letzten 5 Jahren fast verdoppelt, sondern auch die Anzahl der hochkritischen Verwundbarkeiten, zumal der größte Teil der Angreifer versucht hat, von außen in ein fremdes System einzudringen oder es gänzlich zu deaktivieren (vgl. Secunia [1]). Seit 1991 beschäftigt sich das Bundesamt für Sicherheit in der Informationstechnik mit dem Thema und gibt in regelmäßigen Abständen Leitfäden, Handbücher und Berichte, wie den Lagebericht zur IT-Sicherheit und den Ratgeber über IT-Grundschutz für Unternehmen und zu anderen Themen heraus.

## 1.2 Was hat das alles mit Open Source zu tun?

Open Source-Software taucht auch immer wieder in den Meldungen auf. Da stellt sich die Frage: Was tut die Open Source-Gemeinde um diese Meldungen zu reduzieren bzw. vorzubeugen? Denn Open Source-Software wird nicht nur für nicht-kommerzielle Zwecke genutzt, sondern auch von Unternehmen aktiv eingesetzt, vermarktet oder sogar weiterentwickelt, da jeder Interesse an einem sicheren Produkt hat. Ein weiterer Grund ist, dass sich viele Open Source-Projekte Sicherheit auf die Fahne schreiben und argumentieren, dass Open Source sicherer ist als Closed Source (vgl. auch Wheeler [2]).

## 1.3 Abgrenzung: Safety und Security

Da Werkzeuge für Sicherheitstests das zentrale Thema sind, ist es berechtigt zu fragen, was sie genau testen sollen. Was ist also Sicherheit? Wo ist der Unterschied zwischen Safety und Security, reicht es denn nicht, nur die Funktionen zu testen?

1 Common Vulnerabilities and Exposures
2 Computer Emergency Response Team

Zur Abgrenzung werden absichtlich die englischen Begriffe benutzt, da diese zum einen stärker ausdrücken, was hinter dem Begriff steht, und zum anderen sind diese Begriffe in der Informatik allgemein geläufig.

Safety beschreibt die Eigenschaft des Systems, der Art zuverlässig und robust zu funktionieren, dass zu jedem Zeitpunkt die Systemfunktionen unter allen bekannten Bedingungen gemäß Vorgabe funktionieren und damit das System nicht in einen unbekannten (nicht spezifizierten) Zustand kommen kann (vgl. Viega et al. [3] und Eckert [4]).

Security ist kein klar definierbarer Begriff, es ist davon abhängig in welchem Kontext oder welcher Umgebung sich ein System befindet. Eine Sicherheitsrichtlinie (engl. *security policy*) definiert Ereignisse, die durch böswillige Angreifer nicht eintreten dürfen bzw. Maßnahmen um solche Ereignisse zu vermeiden. Erreicht also ein funktionssicheres System Schutzziele wie Authentizität, Integrität, Vertraulichkeit, Verfügbarkeit, Verbindlichkeit und Privatsphäre um die Sicherheitsrichtlinie zu gewährleisten, so kann ein System als *secure* bezeichnet werden. Security verfolgt zwar ein anderes Ziel als Safety, aber es geht damit stark einher. Ein System kann daher die Schutzziele nicht erreichen, wenn das Fundament der Funktionssicherheit nicht gewährleistet ist (vgl. Viega et al. [3], Eckert [4] und Anderson [5]).

## 1.4 Abgrenzung: Schwachstelle und Verwundbarkeit

Wo ist der Unterschied zwischen Schwachstelle und Verwundbarkeit, ist nicht beides dasselbe? Beinhaltet eine Verwundbarkeit nicht immer eine Schwachstelle bzw. ist eine Schwachstelle nicht immer eine Verwundbarkeit?

Der Unterschied lässt sich anhand eines guten Beispiels von David A. Wheeler erklären. Er beschreibt, dass Borland Anfang der 90er Jahre eine Hintertür in die Datenbank InterBase eingebaut hatte, die vollen Zugriff zur Datenbank ermöglichte. Diese Schwachstelle war jahrelang unbekannt, bis Borland dieses Produkt unter einer Open Source-Lizenz freigab. Die Schwachstelle wurde relativ schnell entdeckt[3] und korrigiert (vgl. D. Wheeler [2]), dabei handelte es sich nicht mal um einen Fehler im Programmcode sondern um eine absichtlich eingebaute Routine.

Eine Schwachstelle bietet also die Möglichkeit das System negativ zu beeinflussen. Sie kann aber erst dann zu einer Verwundbarkeit werden, wenn diese Schwachstelle einer breiten Masse bekannt ist, die diese Schwachstelle auch ausnutzen kann, sodass ein Sicherheitsproblem die nächstmögliche Folge einer Schwachstelle ist. Die Open Source-Gemeinde beugt Verwundbarkeit vor, indem potenzielle Verwundbarkeit einem Security-Team gemeldet werden kann, dieses dann die Meldung analysiert und erst dann die

3 CA-2001-01: http://www.cert.org/advisories/CA-2001-01.html (Stand 2008-06-09)

Schwachstelle mitsamt der Verwundbarkeit öffentlich macht, wenn eine Korrektur zeitgleich dazu erscheint.

## 1.5 Weiterer Verlauf

Diese Bachelorarbeit gliedert sich in mehrere Kapitel, die die Werkzeuge und deren Einsatz von mehreren Blickwinkeln beleuchten soll. Um über Werkzeuge sprechen zu können, muss erstmal klar sein, wie sich diese Werkzeuge einteilen lassen und wie sie sich von einander abgrenzen. Das dritte Kapitel stellt den Kern der Arbeit dar. Mehrere Projekte sollen als Fallstudien verschiedene Blickwinkel auf den Einsatz von Werkzeugen für Sicherheitstests geben. Es sollen die Werkzeuge ermittelt werden, deren Einsatz, Gemeinsamkeiten in den Projekten, sowie die Größe des Faktors der Werkzeuge als Teil der Qualitätssicherung in den Projekten. Interessant hierbei sind natürlich die Best Practices aus den Anwendungen der Werkzeuge und dem Qualitätssicherungsprozess.

# 2 Kategorien

Es existieren verschiedenartige, frei verfügbare Testwerkzeuge im Internet, trotz der funktionalen Beschreibung fällt eine allgemeine Zuordnung zu bestimmten Kategorien auf den ersten Blick schwer. In diesem Zusammenhang ist es wichtig sie in ihrer Funktionalität und Anwendung allgemein zu beschreiben, um klarer zu verstehen, was jedes Werkzeug in seiner Kategorie, unabhängig seiner weiteren Funktionalität, macht.

Diese Beschreibungen sind deshalb wichtig, da im Verlauf der Arbeit immer wieder Werkzeuge aus diesen Kategorien referiert werden, sodass dieses Kapitel eine Zusammenfassung und Übersicht geben soll.

Die Einteilung ist keine neue Erfindung sondern basiert auf der von DeMillo et al [6], die bereits vor mehr als 20 Jahren eine sinnvolle, flache Einteilung mit anderen, aber gleichwertigen Bezeichnungen vorgenommen haben.

Die Lesereihenfolge der Beschreibungen orientiert sich an dem hierarchischen Aufbau der Kategorien, dargestellt durch Abbildung 1, d.h. als erstes liest man die übergeordnete Kategorie (statisch, dynamisch) und im Anschluss die eigentliche Zielkategorie. Der Grund ist, dass die übergeordneten Kategorien allgemeine Informationen enthalten, die auf alle untergeordneten Kategorien zutreffen.

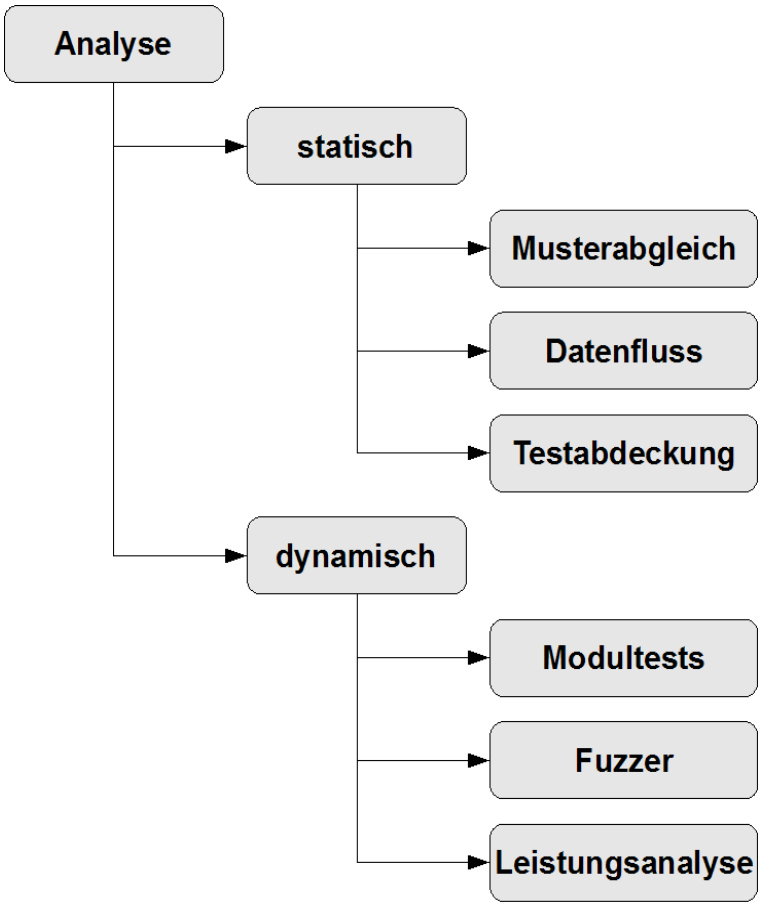



*Abbildung 1: Hierarchischer Aufbau der Kategorien*

## 2.1 Aufbau der Beschreibung

Die Beschreibung unterteilt sich in zwei Punkte. Im ersten Punkt, der eigentlichen Beschreibung, soll die Funktionalität bzw. die Funktionsweise eines Werkzeugs in der jeweiligen Kategorie verdeutlicht werden, dazu gehören auch die allgemeinen Vor- und Nachteile. Der zweite Punkt beschreibt die Anwendung eines solchen Werkzeugs, d.h. was gemacht werden muss, um sie auszuführen um ein Ergebnis zu bekommen.

Es ist zu beachten, dass nicht jedes Werkzeug, häufig aufgrund seiner Funktionalität, sich eindeutig zuordnen lässt.

## 2.2 Statische Analyse

Statische Analyse ist ein Oberbegriff für verschiedene Analysemethoden und Datenerhebungen aus dem Quell- oder Bytecode. Es wird nach Mustern, Verweisen, Verbindungen und Anderem im Code gesucht. Dieses Verfahren arbeitet nach fest definierten Regeln, d.h. es müssen Regeln existieren, die nach Codefragmenten suchen. Die statische Analyse beschränkt sich nicht auf die Suche eines bestimmen Fehlerfalls, sondern auf eine Menge von Fällen, sodass der zu suchende Fehler nicht bekannt sein muss. Zur Ausführung muss nur auf den Quellcodepfad verwiesen werden. Das Ergebnis einer Analyse ist in der Regel eine Reportdatei mit Verweisen auf das potenziell fehlerhafte Codefragment, den Fehlertyp und die möglichen Folgen. Die Vorteile sind: eine sehr einfache Durchführung, eine Untersuchung bereits zu einem frühen Entwicklungsstadium des Quellcodes und eine Abdeckung des gesamten Codebaums. Leider ist die statische Analyse eine ungenaue Abschätzung, d.h. Fehler können nur gefunden werden, wenn es auch entsprechende Regeln gibt, sie können also auch übersehen werden oder als falscher Fehler (engl. *false positive*) gemeldet werden. Zusätzlich lässt sich keine Aussage zum Laufzeitverhalten der zu testenden Anwendung abgeben, da der Code nicht ausgeführt wird (vgl. McGraw [7] und Simon & Simon [8]).

Zu der statischen Analyse gehören Musterabgleich-, Datenfluss- und Testabdeckungsanalyse.

### 2.2.1 Musterabgleich (Pattern)

Musterabgleichswerkzeuge suchen durch vorgegebene Regeln/Muster nach potenziellen Problemen im Programmcode. Im Allgemeinen sind das falsch benutzte/gefährliche Funktionen in Standard-Bibliotheken (z.B. strcpy() in C), ineffiziente Funktionsanwendung (z.B. Stringkonkatenation in Schleifen bei Java), zu komplizierte Ausdrücke oder ungenutzte Variablen. Diese Werkzeuge sind nur im Stande eine oberflächliche Analyse durchzuführen, da sie den Quellcode nur gegen dynamische Muster vergleichen können. In sicherheitskritischen Sprachen (z.B. C), wo es Typunsicherheit, unsichere Methoden

bzw. manuelle Speicherverwaltung gibt, ist es sehr leicht möglich, durch unsachgemäße Programmierung Schwachstellen in den Programmcode einzubauen (vgl. Wagner et al. [9]). Mustervergleich ist in Sprachen wie Java weniger effektiv, da es über unsichere Funktionen oder direkten Speicherzugriff keine Angriffsmöglichkeit gibt.

### 2.2.2 Datenflussanalyse

Die Datenflussanalyse geht einen Schritt weiter als nur Musterabgleich. Durch die Verarbeitung von Quell- bzw. Bytecode wird ein Datenmodell bzw. abstrakter Syntaxbaum (AST) erzeugt, der die oberflächliche Analyse überwindet. Dies bedeutet, dass nun die Benutzung (engl. *variable propagation*) und Ausführungszustände (engl. *execution states*) von Variablen und Funktionen analysiert werden können. Es sind nicht nur die Variablen-, Objekt- und Funktionsnamen dem Werkzeug sichtbar sondern auch deren Typ bzw. Parameter und Rückgabetyp sowie Lese- und Schreibzugriffe. Es können dadurch Null-Pointer-Verweise, Synchronisationsprobleme, falsche Zuweisungen, nicht abgefangene Ausnahmen ermittelt werden. Datenflussanalyse ist heute so allgegenwärtig, dass viele IDEs wie Eclipse einen Teil dieser Fehler direkt finden. Im Gegensatz dazu erfordert die Ermittlung der Weitergabe von Eingabevariablen, die ungefiltert an Funktionen übergeben werden, den Einsatz spezieller Werkzeuge. Eine interessante Anwendung der Datenflussanalyse findet sich in dem Werkzeug LAPSE von Livshits & Lam [10].

### 2.2.3 Testabdeckung (Coverage)

Coverage-Werkzeuge stellen keine eigentlichen Testwerkzeuge dar. Es ist trotzdem aus mehreren Gründen sinnvoll sie zu erwähnen, da sie ein sehr umfangreiches Hilfsmittel darstellen. Diese Werkzeuge analysieren Programmcodes und ermitteln die Zeilen- und Verzweigungsabdeckung, die durch Modultests erreicht werden. Sie können zusätzliche Statistiken wie Code-Komplexität errechnen. Dies bedeutet, je höher die Anzahl an Verzweigungen in einem Codefragment, desto höher die errechnete Zahl, desto höher die Code-Komplexität. Ein solches Werkzeug ist ein guter Indikator um zu ermitteln, welcher Code noch nicht getestet wurde und welcher durch seinen Umfang besonders Beachtung braucht. Es ist aber kein Werkzeug zur Ermittlung der Qualität der Testfälle, d.h. abgedeckter Programmcode heißt nicht zwangsläufig, dass er gut und sinnvoll getestet wurde. Die sinnvolle Anwendung dieser Werkzeuge ist trotzdem eine gute Ergänzung zum Testprozess (vgl. Glover [11]).

## *2.3 Dynamische Analyse*

Dynamische Analyse bedeutet das Ausführen eines vorher entworfenen Testfalls auf ein bestimmtes Element oder Codefragement des Systems unter Beobachtung des Verhaltens. Es ist ein mehrstufiger Prozess aus vorhergehenden Schritten und der eigentlichen An-

wendung. Eine einfache Anwendung wie bei der statischen Analyse ist nicht möglich, zuerst muss man sich mit der Funktionsweise des Werkzeugs beschäftigen, danach das zu testende Problem verstehen und basierend darauf die Testfälle entwickeln. Im Rahmen der Durchführung müssen sowohl der zu testende Teil als auch alle notwendigen Komponenten lauffähig sein (vgl. Simon & Simon [8] und Wikipedia zur Dyn. Analyse [12]).

Trotz dieser Hürden hat die Dynamische Analyse den entscheidenden Vorteil, dass die Integrationsfähigkeit von Modulen, Prozessketten, Laufzeitverhalten und andere Parameter unter realen Bedingungen bzw. realer Simulation getestet werden können.

### 2.3.1 Modultests

Modultests sind klassische Vertreter der Dynamischen Analyse. Ein Modultest führt ein Modul mit vordefinierten Eingabewerten aus und vergleicht das tatsächliche Ergebnis mit einem vorgegebenen Ergebnis. Stimmen diese nicht überein, so wird ein Fehler gemeldet. Als Modul kann eine einzelne Methode, eine Klasse oder sogar eine Suite mit mehreren Klassen bezeichnet werden. Der zu testende Programmcode wird mit dem Testwerkzeug verknüpft und ausgeführt, sodass die Steuerung das Testwerkzeug übernimmt. Diese Werkzeuge lassen sich oft in Build- und Intergrationssysteme einbinden und ermöglichen so ein regelmäßiges und automatisiertes Testen. Ein großer Nachteil ist, dass man für jedes Modul Testfälle erstellen muss, schon bei einem kleinen Programm können es hundert oder mehr Testfälle sein um eine sinnvolle Testabdeckung zu erreichen.

### 2.3.2 Fuzzer

Fuzzing-Werkzeuge führen ein Modul mit einer zufälligen oder parametrisierten Eingabemenge aus um ein Versagen durch ungültige bzw. nicht vollständig verarbeitete Eingaben zu provozieren. Die Vorteile gegenüber reinen Modultests sind, dass sie große Eingabemengen parameterabhängig generieren können, sodass das mühsame Erstellen von Eingabemengen entfallen kann. Sinnvolles Testen mit Fuzzern erfordert (aber) eine geschickte Gestaltung der Eingabemengen im Rahmen der Programm- oder Formatspezifikation und ein gewisses Maß an Verständnis für die Problematik, denn gänzlich willkürliche Eingaben gelangen selten in alle zu testenden Module, und das Analysieren des Problems kann das Werkzeug nicht übernehmen (vgl. Rütten [13] und Sprundel [14]).

### 2.3.3 Leistungsanalyse (Profiler)

Eine Leistungsanalyse ermöglicht das Anhängen eines Werkzeugs an ein laufendes Programm um verschiedene Parameter in Echtzeit zu analysieren. In der Regel betrachtet man den Speicherbedarf, die Ausführungsgeschwindigkeit und nebenläufige Aktionen. Dadurch lassen sich Rückschlüsse über Laufzeit- und Speichereffizienz von Modulen und Algorithmen oder auch Synchronisationsprobleme ziehen.

# 3 Fallstudie

Dieses Kapitel stellt den Kern der Bachelorarbeit dar. Im Verlauf des Kapitels soll deutlich werden, was eine Fallstudie ist, wodurch sie sich definiert und welche Schritte zur Durchführung einer Fallstudie erforderlich sind. Nach der allgemeinen Beschreibung wird in jeweils einem Unterkapitel auf die Schritte einer Fallstudie im Detail eingegangen und beschrieben, wie in dieser Fallstudie vorgegangen wurde. Der Aufbau und die Durchführung orientieren sich an der Arbeit von Robert K. Yin [15] und Susan Soy [16].

Eine Fallstudie ist eine qualitative Forschungsmethode, die eine Situation aus dem wirklichen Leben in ihrem Kontext untersucht. Dabei dient die Situation als Ausgangspunkt für die Anwendung und Evaluation von Ideen und Methoden. Diese nutzen mehrere Quellen zur Beweisführung, die eine beschränkte Menge von wechselseitigen Beziehungen von Ereignissen und Bedingungen der jeweiligen Situation heranzieht.

Der Aufbau einer Fallstudie basiert auf sechs aufeinander folgenden Schritten, sie umfassen folgendes Vorgehen:

1. *Bestimmung und Definition der Forschungsfrage(n)*: Definition einer oder mehrerer Forschungsfrage(n), die sich auf die Situation beziehen und das Forschungsproblem beschreiben, welches der eigentliche Zweck der Fallstudie ist. Durch die Untersuchung von mehreren Fällen aus der Situation, sollen die erworbenen Daten Antwort auf die Forschungsfrage(n) geben.
2. *Auswahl der Fälle und Bestimmung der Datenerhebungs- und Analysemethoden*: Auswahl der Fälle, aus denen die Daten erhoben werden. Jeder Fall ist eine abgeschlossene Fallstudie, deren Ergebnis zum Gesamtergebnis beiträgt. Für die Datenerhebung werden Mittel wie Umfragen, Interviews, Sichtung von Dokumentationen, Beobachtungen und andere Artefakte genutzt. Es ist darauf zu achten, dass jeder Fall mit der gleichen Systematik durchsucht und analysiert wird.
3. *Vorbereitung der Datenerhebung*: Der Forscher soll durch geeignete Methoden die entstehende große Datenmenge sinnvoll systematisch einordnen. Er soll anhand anderer Studien eine Vorstellung bekommen, wie die Durchführung der Datenerhebung und -analyse von statten geht. Vor der eigentlichen Datenerhebung muss ein Pilot mit ein paar Testfällen durchgeführt werden, um zu bestimmten ob die vorgesehen Ziele, Methoden und Ansätze sinnvoll und ergiebig sind, sodass diese in Schritt 2 angepasst werden können.
4. *Sammlung der Daten*: Das Sammeln muss vollständig und identisch aus mehreren Quellen für jeden Fall erfolgen. Wird im Verlauf festgestellt, dass die Suchmethoden, trotz Piloten, nicht sinnvoll sind, so sollte eine Veränderung dokumentiert werden. Der Forscher soll unabhängig von der Datenerhebung Tendenzen und Ideen festhalten.

5. *Evaluation und Analyse die Daten*: Begutachtung der rohen Daten und Herstellen einer Verbindung zwischen Fällen (Daten) und den Forschungsfragen. Eine vielfältige Datenmenge ermöglicht eine Triangulierung, um Ergebnisse und Vermutungen zu stützen. Suche nach Mustern und Gemeinsamkeiten in den Daten. Der Forscher sollte sich nicht von einem bestimmten Einzelfall beeinflussen lassen.
6. *Vorbereitung des Ergebnisses*: Die ermittelten Daten, Schlussfolgerungen und Gemeinsamkeiten müssen in eine lesbare und verständliche Form gebracht werden. Es müssen Grenzen, Probleme, Missstände und Kritik an der Fallstudie angesprochen werden.

Im Verlauf aller sechs Schritte muss die Forschungsfrage stets im Auge behalten werden, um eine sinnvolle Durchführung zu erzielen.

Die aufgeführten Schritte werden in angepasster Form für die Fallstudie in den nachfolgenden Kapiteln diskutiert und dargestellt.

## 3.1 Forschungsfragen

Im Verlauf der Studie sollen zwei aufeinander aufbauende Forschungsfragen beantwortet werden:

1. *Welche Arten von Werkzeugen werden zur Qualitätssicherung bezüglich Sicherheit eingesetzt und welche wurden verworfen?*: Diese Frage soll eingesetzte Werkzeuge über eine Beschreibung qualitativen Charakters zu einer Werkzeugart zuordnen.
2. *Wie werden sie eingesetzt und welcher Mehrwert ergibt sich für die Sicherheit?*: Es soll deskriptiv dargestellt werden, wann im Prozess und wie in den Projekten solche Werkzeuge eingesetzt werden. Zusätzlich ist interessant, was nun diese Werkzeuge tatsächlich in den Projekten leisten und was sie konkret verbessern können.

Eine detaillierte Beantwortung der Fragen stellt eine große Herausforderung dar. D.h. es erfordert tiefergehende Analysen der Projekte, der Testprozesse und umfangreiche Interviews. Dies ist im Rahmen einer Bachelorarbeit ohne umfangreiche Vorarbeit nicht sinnvoll möglich, sodass die Beantwortung der Fragen nur einen beschreibenden Charakter haben wird, um einen guten Überblick über die gegenwärtige Situation in einer Menge von Projekten zu geben.

## 3.2 Wahl der Fälle

Um die Forschungsfrage zu beantworten, muss man sich fragen, was einen Fall überhaupt ausmacht? Welche Fälle könnten die gewünschten Informationen liefern?

Jeder Fall stellt ein explizites Open Source-Projekt dar, von dem ausgegangen wird, dass es Werkzeuge für Tests, vorzugsweise für Sicherheitstests einsetzt. Diese Informationen gilt es zu extrahieren.

Um eine sinnvolle Menge von Projekten zu ermitteln, wurden zwei Möglichkeiten ausgesucht und evaluiert, um brauchbare Projekte zu extrahieren. Zuerst wird versucht, durch Analysen von Ressourcen bei Werkzeugprojekten diejenigen Projekte zu identifizieren, die diese Werkzeuge auch einsetzten. Im Anschluss sind konkrete Projekte mit ihren Ressourcen der Ausgangspunkt für eine Suche nach dem Werkzeugeinsatz.

Die erste Möglichkeit nimmt als Basis ca. 100 Werkzeuge verschiedenster Arten und Größen. Die Auswahl erstreckt sich von recht unbekannten bis hin zu sehr bekannten. Das Ziel ist, die verfügbaren Ressourcen eines Werkzeugs wie Homepage, Mailinglisten, Foren, etc. aufzusuchen und explizit nach Projekten bzw. Projektnamen Ausschau zu halten, die solche Werkzeuge explizit für Sicherheitstests einsetzen. Einige Werkzeuge hatten Listen mit Referenzbenutzern, die aber nicht weiter verfolgt wurden, da sie keine Auskunft zu der Benutzung dieser Werkzeuge geben. Der Fokus lag nur in der oben beschriebenen Suchmethode. Nach mehr als der Hälfte des Suchlaufs wurde diese Suchmethode abgebrochen, da die bis dato erfassten Ergebnisse drei unbeantwortete Forenbeiträge waren, die sich für sinnvolle Einsatzmöglichkeiten dieser Werkzeuge für Sicherheitstests interessierten. Die zweite Möglichkeit sieht den umgekehrten Weg vor. Es soll der explizite Einsatz von Werkzeugen für Sicherheitstests in einer vorgegebenen Menge an Projekten gefunden werden.

Eine Fallstudie hat im Gegensatz zu einer Umfrage zwar keinen repräsentativen, aber dafür einen fundiert qualitativen Charakter, sprich, eine Fallstudie betrachtet immer eine kleine Menge an Fällen um viele Informationen zu extrahieren anstatt nur oberflächlich Daten zu sammeln und statistisch auszuwerten. Die Frage ist dann, wie wählt man eine kleine Menge an Fällen, die hochwertige Daten liefern können? Der Wunsch zum Anfang war, dass alle Projekte im Idealfall in der gestellten Fragestellung aktiv sind. Natürlich kann man so etwas nicht in der Realität erwarten. Beschränkt man sich auf z.B. fünf Fälle, so kann die gewonnene Informationsmenge sehr groß oder auch sehr klein sein. [Um dieser Bedrohung vorzubeugen, wurde absichtlich eine höhere Anzahl an Fällen für die qualitative Analyse gewählt um genau die Fälle auszugleichen, die keine Informationen liefern können oder werden. Es soll also durch einen Vergleich der Projekte miteinander eine fundiertere und breitere Datenbasis geschaffen werden.

Am Anfang der Fallstudie wurden zuerst zwei Projekte (JAMWiki, Apache Tomcat) analysiert. Da das Hauptaugenmerk auf dem Einsatz von Werkzeugen für Sicherheitstests zum Aufdecken von Verwundbarkeiten liegt, wurden die Projektauswahl auf weitere Web-Informationssysteme ausgeweitet, da diese Projekttypen in den letzten Jahren im-

mer wieder unter Beschuss standen und Verwundbarkeiten aufgedeckt wurden. Die weiteren Projekte wurden anhand folgender Kriterien ausgewählt:

- *Popularität*: Das Projekt hat einen hohen Bekanntheitsgrad (allgemein bekannt), ist tausendfach im Einsatz oder ist führend in seiner Funktionsklasse. Dies ermöglicht dem Angreifer durch Ausnutzen einer Schwachstelle eine sehr hohe Anzahl an Systemen gleichzeitig zu beschädigen.
- *Projektgröße*: Die Projekte sind in der Regel CMS[4] oder Web-Server jeglicher Art, sie haben dadurch eine solche Komplexität und einen Codeumfang, dass jede dieser Applikationen Defekte und oftmals übersehene Verwundbarkeiten enthält.
- *Zugänglichkeit*: Die Projektressourcen sollten frei zugänglich sein, um eine Analyse zu ermöglichen.
- *Hohe Downloadzahl*: Auswahl der Projekte anhand hoher Downloadzahlen bei SourceForge.net bzw. beim Betreiber selbst.

Insgesamt werden 15 Web-Informationssysteme betrachtet, darunter zwei Web-Server und 13 Web-Anwendungen für verschiedene Verwendungszwecke.

| Name | Sprache | Funktion | Gründungsjahr |
|---|---|---|---|
| Alfresco | Java | ECM | 2005 |
| Apache HTTPd | C | Web-Server | 1995 |
| Apache OFBiz | Java | eCommerce | 2001 |
| Apache Tomcat | Java | Web-Server | 1999 |
| Bugzilla | Perl | Bug Tracker | 1998 |
| Gallery | PHP | Foto-CMS | 2000 |
| JAMWiki | Java | Wiki-System | 2006 |
| Joomla! | PHP | CMS | 2005 |
| MediaWiki | PHP | Wiki-System | 2002 |
| OpenBravo ERP | Java | eCommerce | 2001 |
| OpenCms | Java | CMS | 1999 |
| phpBB | PHP | Forensystem | 2000 |
| phpMyAdmin | PHP | DB-Frontend | 1998 |
| WebCalendar | PHP | Kalender | 2000 |
| XOOPS | PHP | CMS | 2001 |

*Tabelle 1: Auflistung der untersuchten Projekte (Fälle)*

4 Content Management System

## 3.3 Datenerhebungs und -analysemethoden

Zur Bestimmung der gewünschten Informationen laut Fragestellung werden frei zugängliche Quellen der Projekte miteinbezogen. Dabei werden alle Datenerhebungsmethoden einzeln ausgeführt und zusätzlich mit der vorgesehenen Analysemethode beschrieben:

- *Homepage/Wiki*: Die Suche erfolgte hier auf drei Arten: manuelles Klicken auf Kategorien und Links, Nutzung der eingebauten Suche und Einsatz von Google mit <term> site:<domain>. Gesucht wurde nach interessanten Schlüsselwörtern wie quality assurance, testing, XSS, SQL injection, exploits, fuzzing, automated tests, securty tests, security.
- *Repository/Buildskripte*: Es wurde manuell nach Verweisen bzw. Auffälligkeiten in Make- und Buildfiles (Makefile, build.xml, pom.xml), falls diese vorhanden waren, in Paketen, Klassen und Dateien mit dem Wort *test* im Namen.
- *Bugtracker*: Die Analyse erfolgte sowohl manuell als auch über die eingebaute Suche. Interessant sind die Issues, die in Verbindung mit Sicherheitslücken stehen. Genutzt wurden Suchbegriffe wie injection, exploit, XSS, security.
- *Security-Tracker*: Durchsuchen von SecurityFocus.com und CVE nach bekannten Verletzlichkeiten um herauszufinden, wie die Schwachstellen gefunden wurden (z.B. mit entsprechenden Werkzeugen) und ob diese entstandenen Verwundbarkeiten auch im Bugtracker des Projektes vermerkt sind.
- *Mailinglisten/Foren*: Die Suche erfolgte hier auf zwei Arten: über manuelles Klicken auf Kategorien und Links und die Nutzung der eingebauten Suche. Gesucht wurde nach interessanten Schlüsselwörtern wie quality assurance, testing, XSS, SQL injection, exploits, fuzzing, automated tests, securty tests, security.
- *Umfrage*: Ein Fragenkatalog mit mehreren Multiple-Choice- und Freitextfragen. Es soll eine rein quantitative Erhebung darstellen, die einen Überblick über den Einsatz von Werkzeugen gibt. Die gewonnen Daten sollen als Basis für nächstfolgenden Punkt dienen.
- *Kontakt mit Projekt*: Nach Auswertung aller vorhergehenden Quellen, wurde eine Nachrichtenvorlage entworfen, in der jedes Projekt speziell auf seine Äußerungen, Verweise und Auffälligkeiten aus den vorhergehenden Quellen bezüglich Sicherheit angesprochen wurde, um herauszufinden, was hinter diesen Aussagen steht. Diese Nachrichten wurden an die Entwickler-Mailingliste bzw. an das Entwickler-Forum geschickt. Im Fall einer Antwort wurde auf der Liste bzw. dem Forum weiter diskutiert und nach Möglichkeit mit den Entwicklern direkt Kontakt aufgenommen.

Analog zum Vergleich der Fälle bzw. Projekte findet hier eine Datentriangulierung nach Yin über die Quellen statt. Hier sollen wieder die ergiebigen Quellen die schwachen Quellen ausgleichen.

Man kann sagen, dass der Vergleich auf zwei aufeinander aufbauenden Ebenen stattfindet. Der Vergleich von Projekten untereinander und der Vergleich verschiedener Datenquellen in einem Projekt ermöglichen, die Fragestellung von mehreren Punkten bzw. Blickwinkeln zu beleuchten.

## 3.4 Vorbereitung und Erhebung der Daten

Wie in Kapitel 3.2 bereits bemerkt wurde, war die Suche direkt über die Werkzeuge nicht erfolgreich. Während der Pilotphase und auch des weiteren Verlaufs haben sich mehrere Suchmethoden ebenfalls als nicht ergiebig genug erwiesen und wurden deshalb aus der Erhebung gestrichen:

- *Bugtracker*: Die Fehler, die in Verbindung zu Sicherheit standen, wurden zwar oft in den Bugtrackern erfasst, aber es konnten keine Rückschlüsse aus den Diskussionen gewonnen werden, wie diese Fehler gefunden wurden. Der Informationsgehalt war also so knapp, dass es nicht sinnvoll war diese Quelle weiter zu verfolgen.
- *Security-Tracker*: Hier lag die Hoffnung darin, dass die Berichte mögliche Werkzeugnamen enthielten, mit denen die Schwachstelle für die Verwundbarkeit gefunden wurde, sodass man dies Werkzeuge weiter verfolgen könnte. Zum einem waren nur die Verwundbarkeit und deren Reproduzierbarkeit beschrieben, zum anderen waren die wenigstens Securitymeldungen auch in den Bugtrackern der Projekte vorhanden, obwohl viele Projekte in den Release Notes auf diese Meldungen verwiesen. Hier konnten auch keine interessanten Rückschlüsse gezogen werden.
- *Umfragen*: Die Umfragen hatten einen quantitativen Charakter. Die Auswertung derselben sollte als Basis für eine weitere qualitative Analyse durch direkte Interviews dienen. Aufgrund der teilweise sehr spärlichen Informationen und sehr knappen Rückmeldungen hat sich herausgestellt, dass die quantitative Erhebung durch einen Fragenkatalog in keiner Weise als Ausgangspunkt für eine weitere Analyse genutzt werden kann.

## 3.5 Erhebung der Daten (Fälle)

Nachfolgend wird jedem Projekt ein Kapitel mit einer kurzen Beschreibung der Projektfunktion, einer Zusammenfassung der erwähnenswerten Informationen aus dem Projekt und einem knappen Fazit gewidmet.

Die Zusammenfassung beinhaltet nur die Ergebnisse der Quellen, die durch eine Analyse verwertbare Informationen enthielten, d.h. wird eine Quelle nicht erwähnt, enthielt sie auch keine Informationen bzw. sie wird explizit, aus bestimmten Gründen, als nicht ergiebig erwähnt.

In den Projektabschnitten werden alle benutzen Quellen referiert. Eine Aufgliederung nach Projekt und Quellentyp findet sich im Anhang A, sodass jede benutze Quelle nachvollzogen werden kann.

### 3.5.1 Alfresco

Alfresco ist ein quelloffenes Dokumentenmanagementsystem, das eine kostenlose Alternative zu kommerziellen DMS darstellen soll. Alfresco wurde, unter anderem, von einem der Mitgründer von Documentum gegründet [A-1]. Die Nutzerliste umfasst große Unternehmen und Regierungseinrichtungen [A-2].

Das Wiki enthält ein sehr ausführliches Qualitätssicherungsverfahren, das beschreibt, welche Systemkomponenten wie getestet werden sollen. Der Testprozess soll die Funktionsfähigkeit der Workflowprozesse im System gewährleisten [A-3], [A-4]. Durchgeführt wird der Test mit HP QuickTest Pro, ein kostenpflichtiges Werkzeug zum automatisierten Testen von Web-GUIs durch Record and Replay [A-5]. Dazu existiert ein Subprojekt, das die umfangreichen Testskripte verwaltet [A-6], [A-7]. Darüber hinaus listen die Entwickler eine umfangreiche Liste an Open Source-Werkzeugen in den Bereichen Funktions-, Leistungs-, Modultests, Coverage und Codeanalyse (Pattern, Datenfluss) auf. Die Auflistung beschreibt das Funktionsziel der Werkzeuge und teilweise die Eignung derselben für den Testprozess von Alfresco [A-8] bis [A-12]. Die Beurteilung macht aber nur vage Aussagen zur Benutzung der aufgeführen Werkzeuge. Die Ant-Buildfile enthält Targets für JUnit-Tests, leider finden sich nur Testressourcen im Repository und keine JUnit-Testfälle [A-13]. Das Projekt hat ein eigenes Qualitätssicherungsforum, wo jegliche Themen übers Testen diskutiert werden. Jedoch findet sich dort kein Anhaltspunkt zum Einsatz der gesuchten Werkzeuge [A-14]. Der Kontakt zum Projekt über dieses Forum blieb unbeantwortet [A-16].

Trotz des umfangreichen Testplans und der Evaluation vieler Werkzeuge, gibt es bei diesem Projekt keine Hinweise, dass diese Werkzeuge auch für Sicherheitstests zum Einsatz kommen.

### 3.5.2 Apache HTTPd

Der Apache HTTPd Web-Server ist dank seiner Leistungsfähigkeit, Offenheit und Erweiterbarkeit ein populärer HTTP-Server.

Das Projekt listet in einem Sicherheitsbericht alle bisher gefundenen Verwundbarkeiten auf. Dieser basiert auf den Meldungen des Mitre-CVE. Die Changelog-Dateien jedes Releases listen entsprechende CVE-Meldungen als behoben auf. Apache HTTPd betreibt ein Sicherheitsteam, welches diskret noch nicht bekannte Verwundbarkeiten über eine spezielle eMailadresse akzeptiert, diese behebt und dann die Verwundbarkeit und deren Patch publiziert [A-17]. Zusätzlich geben die Entwickler Tipps für die Anwender zum sicheren Einsatz des Server sowie Schutzmaßnahmen gegen typische Angriffe wie DoS[5] oder CGI[6]-Fallstricke [A-18]. Die Entwickler setzten ein eigenes Test-Framework ein, welches mehrere Werkzeuge beinhaltet. Es wird nicht auf jedes Werkzeug eingegangen, da nur eines einen gewissen Bezug zu Sicherheit hat: Das Perl Framework auf Basis von Apache-Test. Es ist ein dynamisches Testwerkzeug mit einer Fülle von Testfällen im Repository, welche verschiedene Servermodule testen. Interessant ist, dass es einen Ordner mit in Perl geschriebenen Testfällen für Regressionstests für bereits behobene CVE-Meldungen gibt [A-19], [A-20]. Die Entwickler-Mailingliste enthielt nur einen interessanten Beitrag. Der Beitrag war vom Jahr 2003, wo ein Teilnehmer vermutet, dass externe Firmen bereits Sicherheits-Audits des HTTPd-Codes durchgeführt haben und diese Schwachstellen publiziert wurden. Ihm ist leider aber nicht bekannt, ob die Entwickler selbst solche Audits durchführen [A-21]. Auf das Anschreiben mit den Fakten wurde auf der Entwickler-Mailingliste nicht geantwortet [A-22].

Das Security Response-Team und das Perl Framework sind ein Hinweis, dass das Projektteam um Sicherheit jeglicher Art bemüht ist, leider wurden die Fragen diesbezüglich auf der Mailingliste nicht beantwortet.

### 3.5.3 Apache OFBiz

Apache OFBiz ist eine umfangreiche eCommerce-Plattform, die z.B. vielfältige Web-Shops ermöglicht mit dem Vorteil, dass die gesamte Prozesskette vom Verkauf bis hin zu Warenwirtschaft, Buchhaltung und Stammdatenverwaltung integriert ist. Es dient auch als Basis für weitere Plattformen wie z.B. Atlassian JIRA [A-23].

Die ersten beiden Ressourcen bieten sehr wenig Informationen, so sagt der ASF[7] Board Report 2008-03, dass mehr Anstrengung in die Entwicklung von automatisierten Tests investiert werden muss. Schaut man sich jetzt jedoch das Repository an, so ist der Einsatz z.B. von JUnit sehr beschränkt vorzufinden [A-24], [A-25]. Die Entwickler-Mailingliste enthält im Zeitraum von Juni 2006 bis einschließlich Januar 2007 einige erwähnenswerte Nachrichten. Aus zwei Diskussion ist zu entnehmen, dass QAA als Werkzeug für Regressionstests in Use Case-Szenarien genutzt wird [A-37]. Zusätzlich gibt es eine umfangrei-

5 Denial of Service
6 Common Gateway Interface
7 Apache Software Foundation

che Test-Suite im Repository bestehend aus einer Mischung aus JUnit, Jython, BeanShell, JS mit Grinder. Sie besteht aus in XML definierten Testfällen, die unter anderem in Verbindung mit der Web-Oberfläche weitere Use Case-Szenarien testen. Zwei weitere Diskussion beziehen sich auf die Funktionalität und Einsatzfähigkeit von mehreren Web-Testwerkzeugen wie Selenium, Canoo WebTest und Grinder [A-26] bis [A-35]. Der Diskussion ist aber nicht zu entnehmen, ob diese Werkzeuge nun tatsächlich zum Einsatz kommen. Abschließend gibt es einige Meldungen über Verwundbarkeit über XSS[8], die Entwickler sind sich aber nicht sicher, wie man dieses Problem global lösen kann, anstatt es stellenweise zu flicken [A-36]. Auf die Mailanfrage auf der Entwickler-Mailingliste wurde nicht reagiert [A-38].

Hier bietet sich ein ähnliches Bild wie bei Alfresco: Diskussionen über Evaluation und möglichen Einsatz, aber keine konkreten Hinweise für Tests im Sicherheitsbereich. Keiner der gefundenen Ansatzpunkte konnte verifiziert werden, aufgrund der Nichtbeantwortung der Anfragen über Entwickler-Mailingliste.

### 3.5.4 Apache Tomcat

Apache Tomcat ist ein Servlet-Container, der Suns Servlet- und JSP-Spezifikation implementiert. Durch seine Popularität dient er oft als Subsystem für Servlets in Applikationsservern wie Jboss [A-39] oder GlassFish [A-40].

Das Tomcat-Projekt betreibt genau wie Apache HTTPd ein Sicherheitsteam, das sich um Verwundbarkeiten kümmert und einen Sicherheitsbericht pflegt [A-41]. Die veröffentlichten Meldungen spiegeln sich genauso in den Release Notes wieder. Das Repository enthält im Vergleich zu der Projektgröße eine sehr beschränkte JUnit-Testmenge [A-42]. Der Kontakt mit dem Projekt war sehr aufschlussreich und hat Informationen geboten, die so in keiner Weise in den anderen Quellen auftauchen. Es ließen sich mehrere Kernpunkte ausarbeiten. Das Projekt hat keine Werkzeuge für Sicherheitstests evaluiert, trotzdem hat es viele Erfahrungen mit Fuzzern gewonnen. Das Sicherheitsteam bekommt regelmäßig Meldungen über Schwachstellen im Tomcat, die mit solchen Werkzeugen ermittelt wurden. Laut Aussage waren die Werkzeuge weder im Stande, bereits bekannte Schwachstellen aufzufinden, noch haben sie neue Schwachstellen gefunden. Die Ergebnisse enthielten sehr viele Falschmeldungen. Zusätzlich scannt Fortify mit ihrem Produkt Fortify Source Code Analyzer (SCA) viele Open Source-Projekte [A-44], [A-45], unter anderem auch Apache Tomcat. Es ist ein statisches Analysewerkzeug, dass Musterabgleich und auch Datenflussanalyse durchführt. Ein Entwickler bemängelte diesen kostenlosen Service als nutzlos, da Fortify nicht angibt, gegen welche Revision sie scannen und die Reportaufbereilung schwierig zu lesen ist. Wichtig ist aber, dass die Menge an Falschmeldungen viel zu hoch war und dass mit einem solchen Werkzeug nur eine maximale 10 %

8 Cross Site Scripting

Abdeckung bezüglich Sicherheitstests gemacht werden kann. Zusammenfassend bezeichnen die Entwickler die aktuell verfügbaren automatisierten Werkzeuge für Sicherheitstests als sehr minderwertig und nicht verwendbar. Es gibt ihrer Meinung nach keine Werkzeuge, die sinnvolle und brauchbare Ergebnisse liefern. Sie halten aber Werkzeuge wie FingBugs, PMD oder Eclipse (da teilweise statische Analysemethoden vorhanden) für die Suche nach gewöhnlichen Fehlern für sinnvoll. Der Einsatz automatisierter Werkzeuge macht für die Entwickler nur dann Sinn, wenn die Menge an zu testender Software zu umfangreich ist, um sie manuell zu kontrollieren. Die Entwickler halten manuelle Security Code Reviews für ein weitaus effizienteres Werkzeug. Eine Durchsicht, durchgeführt von einem qualifizierten Team bzw. einer spezialisierten Firma, ist im Stande, eine bis zu 90%ige Abdeckung für Sicherheit zu erreichen und die Menge an Falschmeldungen auf ein Minimum zu reduzieren, bei einer gleichzeitig höheren Entdeckungsrate von richtigen Schwachstellen. Diese Vorgehensweise sei, trotz der Kosten auch im Vergleich zu Fortify, vorzuziehen. Das Projekt versucht Codequalität und -sicherheit durch ein dreifaches Review von Commits zu gewährleisten [A-43].

Durch den aufschlussreichen Kontakt mit dem Projekt war es möglich zu ermitteln, dass sich die Entwickler gegen den Einsatz von automatisierten Werkzeugen für Sicherheitstests aussprechen, aber sehr wohl für eine manuelle statische Analyse.

### 3.5.5 Bugzilla

Bugzilla ist ein großes und sehr funktionsreiches Bug-Tracking-System. Es ist das System der Wahl vieler großer Open Source-Projekte wie Mozilla, Eclipse, OpenOffice.org, KDE und andere.

Das Projektteam verwaltet genau wie Apache HTTPd einen Sicherheitsbericht, der die Verwundbarkeiten aufzählt und Referenzen zu den Versionen/Releases setzt [A-46]. Die Homepage enthält einige Informationen über den Sicherheitsprozess bei Bugzilla. Es wird behauptet, dass bei der Entwicklung der Software alle Module exzellent gesichert und von Grund auf sicher entworfen worden sind [A-47]. Zusätzlich gibt es Anweisungen, die neuen Entwicklern Best Practices nahe bringen sollen [A-48]. Es wird ein selbst geschriebenes Perl-Werkzeug eingesetzt. Es ist eine Test-Suite, die ähnlich wie Lint funktioniert und jeden Commit auf schlechten Code untersucht [A-49]. Wenn ein Durchlauf fehlschlägt, so wird der Commit zurückgewiesen [A-50]. Das Werkzeug leistet aber viel mehr als einen Musterabgleich wie Lint. Der Kontakt über die Entwickler-Mailingliste war sehr positiv und verriet, dass dieses Werkzeug eine Datenflussanalyse durchführt. Es untersucht, ob Eingabevariablen direkt und gefiltert referiert werden. Ein ungefilterter Zugriff auf eine Variable kann immer die Ursache einer Schwachstelle sein, so dass ein Commit in diesem Fall auch zurückgewiesen wird [A-51].

Dies ist das erste Projekt, das direkt selbstentwickelte Werkzeuge einsetzt um Schutzmaßnahmen im System durchzusetzen.

### 3.5.6 Gallery

Gallery ist ein anpassbares, großes CMS, das speziell fürs Foto-Hosting entwickelt wurde. Es ist in PHP geschrieben und lässt sich in andere PHP-basierende CMS integrieren.

Im Wiki findet sich, dass Schwachstellen durch das Framework von Gallery vorgebeugt wird [A-52]. Die Entwicklung der Software ist stark testgetrieben. Eine ausführliche Dokumentation beschreibt den Entwicklungsprozess, der auch eine Testumgebung von mehr als 3000 PHPUnit-Klassen für mehrere Plattformen beinhaltet [A-53] bis [A-55]. Interessant ist, dass diese Testmenge für jede bekannte Verwundbarkeit einen Regressionstest vorsieht, um diese bzw. ähnliche Schwachstellen nicht aufzureißen [A-57]. Der Kontakt mit dem Projekt brachte eine große Fülle an Informationen zu Tage, die man in mehrere Punkte gliedern kann. Programmatisch-präventive Maßnahmen, wie z.B. Prepared Statements, indirekter Zugriff auf Eingabenvariablen, Filterung von sämtlichen Eingaben, strikte Typisierung um PHP-bedingte Probleme auszuschließen, durchgeführt, um Schwachstellen zu vermeiden. Regelmäßiges Testen der Sicherheit wird mit speziellen Unit-Tests, der Burp Suite, Wireshark und Firebug durchgeführt. Fuzzer werden aber nicht eingesetzt, da die Qualität der Ergebnisse aufgrund einer hohen Anzahl an Falschmeldungen als sehr minderwertig zu betrachten ist. Das Gallery-Team verlässt sich auf regelmäßige Security Code Reviews, die auf Commits und der aktuellen Codebasis durchgeführt werden, dabei werden die Durchsichten projektintern durchgeführt und auch an eine spezialisierte, externe Firma (Gotham Digital Science[9]) vergeben, die noch zusätzlich zu den Code Reviews Fuzzer und automatisierte Penetrationswerkzeuge einsetzt. Die externe Code Review wurde bereits mehrfach durchführt und soll vor jedem Hauptrelease erfolgen [A-58].

Das Gallery-Projekt stellt dar, dass Sicherheit für sie sowohl prozess- als auch testgetrieben gewährleistet werden kann.

### 3.5.7 JAMWiki

JAMWiki ist ein kompaktes Wikisystem, dass auf in Java geschrieben ist und MediaWiki nachempfunden ist.

Aufgrund der kleinen Projektgröße, ist keine stark ausgeprägte Aktivität in Richtung Sicherheitstests vorhanden. Jedoch gibt es Aktivitäten in mehrere Richtungen. Das Projekt versucht programmatisch Schwachstellen vorzubeugen (z.B. Prepared Statements, JS abgeschaltet in allen Eingabefeldern) [A-61] und setzt JUnit auch für Sicherheitstests ein

9 http://www.gdssecurity.com/ (2008-06-15)

[A-60]. Für seine kleine Projektgröße, ist die Anzahl an JUnit-Testcases sehr hoch. Des weiteren existieren hier projektübergreifende Tests für den Wikisyntax-Parser. Ein Fuzzer, der bei MediaWiki eingesetzt wird, soll hier auch zum Einsatz kommen [A-59].

Das JAMWiki-Projekt versucht ein Mindestmaß an Sicherheit mit programmatisch-präventiven Maßnahmen und Modultests zu gewährleisten.

### 3.5.8 Joomla!

Joomla! ist ein weiteres Content Management System, das mit Mambo verwandt ist und eine große Popularität genießt.

Die Homepage bzw. Wiki beherbergen recht wenig Informationen für Entwickler. Ein paar Ratgeber erklären, wie man XSS und Code Injection (Best Practices) vermeiden kann [A-62], [A-64]. Unter Sicherheit versteht das Projekt, dass es *nicht den* Weg der Sicherheit gibt und dass Sicherheit *immer* Erfahrungssache ist [A-70]. Im Gegensatz dazu legt das Team Wert auf größtmögliche Sicherheit bei den Anwendern. Unzählige Ratgeber weisen den Administrator darauf hin, worauf es bei einem sicheren Betrieb ankommt [A-63], [A-67] bis [A-69]. Aus dem Forum lässt sich entnehmen, dass es eine Joomla! Test Suite gibt, aber der Einsatz derselben ließ sich aus den Beiträgen und der Homepage nicht ermitteln [A-71], [A-65], [A-66]. Ein relativ knapper Kontakt ergab doch eine wichtige Äußerung zum Thema Werkzeuge. Das Team äußert sich negativ gegenüber Fuzzern, sie haben SQL Injection-Fuzzer auf Joomla! angewendet und die Werkzeuge waren nicht im Stande bis zu den entscheidenden Stellen im Code vorzudringen, sprich, sie erreichten nicht die Datenbank. Darüber hinaus war die Rate der Falschmeldungen sehr hoch. Das Projektteam hat diese Werkzeugart verworfen und verfolgt sie nicht weiter [A-72].

Leider war in diesem Projekt nicht viel zu ermitteln, es wurde jedoch die Meinung zur Einsatzfähigkeit von Fuzzern für Sicherheitstests geäußert, aber nicht, welche weitere Vorgehen für die Gewährleistung von Sicherheit existieren (vgl. Gallery).

### 3.5.9 MediaWiki

MediaWiki ist eine CMS der besonderen Art. Hervorgegangen aus Wikipedia, die diese Software als erste eingesetzt haben, wurde diese Software für alle frei gegeben. Es ist deshalb besonders, weil jeder die Möglichkeit, hat den Inhalt zu verändern oder etwas hinzuzufügen, bei einem klassischen CMS dürfen dies nur besondere Nutzer. Mittlerweile werden Wikis auf vielen Webseiten und Open Source-Projekten zur gemeinsamen Zusammenarbeit eingesetzt.

Wie viele andere Projekte, betreibt auch MediaWiki ein Sicherheitsteam, dass sich diskret um Schwachstellen kümmert und diese in den nächsten Releases behebt [A-73]. Im Repository findet sich im Gegensatz zu vielen anderen PHP-basierenden Projekten nur eine

Hand voll an PHPUnit-Testcases [A-75]. Der Kontakt offenbart, dass MediaWiki eins der wenigen Projekte ist, welches Fuzzer mit Erfolg benutzt. Ein auf PHP portiertes Fuzzing-werkzeug (fuzz-tester.php auf Basis von mangleme) testet den Wikisyntax-Parser mit Erfolg. Es wurden bereits einige Schwachstellen entdeckt und geschlossen [A-74], [A-76]. Der Schwerpunkt liegt auf Security Code Audits auf bestehendem Code und auf allen ankommenden Commits und Patches. Programmatisch-präventive Maßnahmen, wie keine direkte Nutzung von Eingabevariablen ohne vorherige Filterung, sind auch ein zentraler Punkt [A-77].

MediaWiki geht wie viele andere Projekte mit programmatisch-präventiven Maßnahmen und manuellen Werkzeugen vor, setzt aber erfolgreich einen speziellen Fuzzer ein, wo andere solche Werkzeuge verworfen haben.

### 3.5.10 Openbravo ERP

Openbravo ist ein vollintergriertes, webbasiertes Enterprise Resource Planning-System, angepasst für kleine und mittelständische Unternehmen.

Die Homepage beschreibt lediglich funktionales Testen anhand von Anwendungsfällen [A-78]. Auf das Anschreiben erfolgte keine Antwort [A-79].

### 3.5.11 OpenCms

OpenCms ist das einzige Java-basierende CMS in der Fallstudie. Entwickelt von einer Kölner Firma scheint das CMS in der Stille eine sehr starke Verbreitung beim Einsatz von Firmenwebseiten zu haben [A-80].

Praktisch keine Quelle war ergiebig. Der Kontakt wurde auch nicht beantwortet [A-82]. Lediglich das Repository weist eine hohe Anzahl an JUnit-Testfällen auf [A-81].

### 3.5.12 phpBB

phpBB ist eine großes Forensystem mit mehr 300 000 Zeilen [A-89] an Code. Es wird von vielen Communities und Open Source-Projekten eingesetzt.

Die Homepage verrät sehr wenig über den Qualitätssicherungsprozess. Im Forum wurde über den Einsatz von Selenium diskutiert [A-85]. Ein Beitrag vom November 2007 sagt, dass kein einheitliches Test-Framework existiert, jeder Entwickler testet nach seinem Geschmack lokal, wobei mit Version 3.2 endlich eine umfangreiche Menge an PHPUnit-Testfällen eingeführt werden soll [A-84]. Das Projekt bezeichnet sich selbst als das sicherste PHP-Forensystem [A-86]. Der Kontakt mit phpBB war der aufschlussreichste von allen, es wurde ein persönliches Interview mit einem der Hauptentwickler geführt. Das Interview lässt sich im Anhang A in Stichpunkten nachlesen. Hier werden die Kernaussagen mit Beispielen zusammengefasst: phpBB nutzt kein explizites Sicherheitstestwerk-

zeug und äußert sich recht kritisch solchen gegenüber. Das Projekt hat automatisierte Werkzeuge wie Fuzzer mehrfach evaluiert und festgestellt, dass sie weder im Stande waren Schwachstellen aufzuspüren, noch konnten sie bekannte Verwundbarkeiten reproduzieren. Die Werkzeuge haben die zu testenden Stellen nie erreicht und wiesen darüber hinaus sehr viele Falschmeldungen auf. Sogar kommerzielle Werkzeuge wie Acunetix [A-90] lieferten keine brauchbaren Ergebnisse. Am Beispiel von Acunetix gab es 160 Meldungen 100 % davon waren Falschmeldungen. Die Entwickler schlagen einen anderen Weg der Gewährleistung der Sicherheit ein. Vielen Schwachstellen wird durch programmatisch-präventive Maßnahmen (z.B. Typisierung, Prepared Statements, Parameterfilterung) vorgebeugt, gerade das bereitet den automatisierten Werkzeugen Probleme. Bei manuellen Tests führt das Team Security Code Reviews durch. In regelmäßigen Abständen wird mithilfe von manuellen Penetrationstests der Programmcode Zeile für Zeile analysiert und verbessert. Zusätzlich verlässt man sich hier auch auf externe Code Audits von spezialisierten Firmen. In dem Fall von der Firma SektorEins[10] unter der Leitung von Stefan Esser, der in der Szene als „PHP-Gott“ angesehen wird. Jedoch werden andere Werkzeuge für Sicherheitstests eingesetzt. Ein selbstentwickeltes Werkzeug auf Basis von grep führt eine statische Analyse durch (vgl. Bugzilla). Zusätzlich sind die Entwickler der Meinung, dass PHP selbst das Problem darstellt, durch die schwache Typisierung entstehen viele Schwachstellen. Gute Sicherheit in PHP zu gewährleisten, bedeutet vor allem strikte Typisierung, Schöpfen aus Erfahrung und manuelle Code Reviews [A-87] bis [A-89].

phpBB lieferte sehr ausführliche Informationen zu Werkzeugen und anderen Prozessen. Hier ließ sich wieder erkennen, dass automatisierte Werkzeuge als mangelhaft angesehen werden und manuelle Code Audits als effizienter betrachtet werden.

### 3.5.13 phpMyAdmin

PMA ist webbasierendes Verwaltungswerkzeug für MySQL-Datenbanken.

Hier schweigt sich das Projekt wie andere auch aus. Außer ein Sicherheitsteam und Anstrengungen XSS zu vermeiden [A-91], [A-92], existieren im Repository ca. 10 PHPUnit-Dateien [A-93]. Auf den Kontakt hat das Projektteam nicht geantwortet [A-94].

### 3.5.14 WebCalendar

WebCalendar ist ein PHP-basierendes, multifunktionales Kalendersystem.

Außer einem PHPUnit-Testcase [A-95], waren keine weiteren Informationen zu extrahieren. Der Kontakt wies zwar programmatisch-präventive Maßnahmen wie Bereinigung

10 http://www.sektioneins.de/ (2008-06-15)

und Validierung von Eingaben auf, aber Fragen zu Testprozessen allgemein wurden nicht beantwortet [A-96].

### 3.5.15 XOOPS

XOOPS ist ein weiteres CMS, welches auf SourceForge.net gehostet wird.

Es gibt ein Qualitätssicherungsverfahren, dass recht unverständlich ist und mit Testen nichts zu tun hat [A-97]. Der Kontakt ergab, dass das Projekt selbst weder testet noch präventiv schützt. Es nutzt ein externes Modul namens „Protector“, das gegen alle möglichen Angriffe (XSS, Injection u.a.) schützen soll. Informationen zu dem Werkzeug gibt es wenig im Netz und die Webseite des Entwicklers ist auf Japanisch. Auf weitere Anfragen, ob das Modul hält, was es verspricht, wurde nicht geantwortet [A-98] bis [A-100].

## *3.6 Ergebnisse*

In diesem Kapitel werden die erhobenen Daten gesammelt, bewertet und analysiert. Es soll deutlich werden, wie und welche Daten verwendet werden, welche Kernpunkte und Schlussfolgerungen aus den Daten gezogen werden können und in welchen Punkten die Studie Schwächen hat, welche Ansatzpunkte sie für eine weitere Studie liefern könnte oder wo noch weitere Untersuchungen notwendig sind.

### 3.6.1 Evaluation der Daten

Nach einer Evaluation der gesammelten Datenmenge, trifft genau die Befürchtung ein, die in Kapitel 3.2 (Wahl der Fälle) geäußert wurde. Nicht jeder Fall lieferte die gewünschten Daten, obwohl sechs von acht ausgeschlossenen Projekten auch wirklich Tests durchführen. Der Ausschluss erfolgt aber nicht unbegründet. Hält man sich nochmal die erste Forschungsfrage vor Augen, so gibt es interessante Beobachtungen wie die Regressionstests bei Apache HTTPd oder die umfangreiche Werkzeugaufzählung bei Alfresco. Die Komplexität der Systeme und der nicht nachvollziehbare Nutzen machen eine eindeutige Zuordnung schwierig. Zumal der Kontakt mit Fragen Hinweise zum möglichen Einsatz von Regressionstests oder andere Maßnahmen hätte liefern können. Antworten auf diese Fragen blieben auf den Entwickler-Mailinglisten aber aus.

Ausgeschlossen werden folgende Projekte: Alfresco, Apache HTTPd, Apache OFBiz, Openbravo ERP, OpenCms, phpMyAdmin, WebCalendar und XOOPS. Man muss aber auch hier zwischen den Fällen unterscheiden, die unzureichende Daten haben und denen, die überhaupt keine Daten geliefert haben. Betrachtet man die vorhandenen Daten nun genauer, so kann man hier gewisse Gruppierungen entdecken. Die erste Gruppe besteht aus den Fällen WebCalendar und XOOPS. Zwar haben Entwickler von XOOPS im Mailkontakt auf ein Schutzmodul verwiesen, aber es konnte kein Testverfahren aufgefunden

werden. Es waren aber die einzigen zwei von den acht, die auf den Kontakt überhaupt geantwortet haben. Zwei weitere Projekte (Apache HTTPd, phpMyAdmin) betreiben Sicherheitsteams, die sich um Schwachstellen kümmern. Werkzeuggestütztes Testen in den Projekten ist auf verschiedene Arten ausgeprägt. Apache HTTPd, phpMyAdmin, Apache OFBiz und OpenCms verfügen über Modultests. Apache HTTPd hat sogar eine Menge von Regressionstests für CVE-Meldungen. Zu den Modultests führen Alfresco, OFBiz und Openbravo ERP Webtests durch. D.h., sie testen Anwendungsfälle und Prozesse auf der Web-Oberfläche.

Eine weitergehende, differenzierte Analyse wird daher auf den verbleibenden Projekten (Apache Tomcat, Bugzilla, Gallery, JAMWiki, Joomla!, MediaWiki, phpBB) durchgeführt.

### 3.6.2 Auffälligkeiten

Im Verlauf der Evaluation und Analyse sind ein paar Auffälligkeiten aufgetreten, die einer Erwähnung bedürfen. Es wird aber bewusst auf eine weitere Erklärung der Ursachen bzw. Interpretation dieser Auffälligkeiten verzichtet, da der Kern der Arbeit eine reine Beschreibung der Situation vorsieht.

- *Vergleich PHP gegen Java*: Von den 15 untersuchten Projekten sind sechs Projekte in Java geschrieben und sieben Projekte in PHP. Die Verteilung der Ergiebigkeit zeigt aber keine Ausgeglichenheit. Nur zwei Java Projekte lieferten interessante Informationen und haben tatsächlich auf die Anschreiben geantwortet. Im Gegensatz dazu haben vier PHP-Projekte sinnvolle Informationen geliefert. Vergleicht man nur die Anzahl der Antworten, so haben sogar sechs PHP-Projekte geantwortet.
- *Initiatoren hinter Java-Projekten*: Schaut mal sich die sechs Java-Projekte an, so kann man feststellen, dass drei Projekte (Alfresco, Openbravo ERP, OpenCms) von kommerziellen Unternehmen unter einer Open Source-Lizenz entwickelt werden. D.h., die Software wird jedem frei zur Verfügung gestellt und der Umsatz wird in der Regel mit Zusatzprodukten oder Support gemacht. Auffällig ist nun, dass kein einziger Entwickler dieser kommerziell getriebenen Projekte auf das Anschreiben geantwortet hat.
- *Informationsgehalt der Quellen*: Schaut man sich im Laufe der Datenerhebung und -evaluation den Informationsgehalt der Quellen an, stellt man fest, dass fast nur der direkte Kontakt mit den Projekten ergiebige Informationen zu Tage brachte. Es muss aber beachtet werden, dass die Kontakte auf den vorhergehenden Quellen aufbauen, um ein sinnvolles Anschreiben zu entwickeln.

### 3.6.3 Analyse der Daten

Aus den Quellen der verbleibenden Projekte lassen sich fünf verschiedene Maßnahmen heraus destillieren. Diese Maßnahmen werden durch den Einsatz verschiedener Werkzeuge durchgeführt. Dabei testen die Entwickler mit diesen Werkzeugen das System und die expliziten Schutzmaßnahmen. Diese Hauptpunkte ermöglichen bereits die Beantwortung der 1. Forschungsfrage im Allgemeinen, die nachfolgende Tabelle 3 im Speziellen.

Hauptpunkte der Aussagen:

1. *Spezialisierte Werkzeuge für Sicherheitstests*: Die Projekte äußerten ihre Meinung zur der Evaluation, Einsatzfähigkeit bzw. Nichteinsatzfähigkeit von spezialisierten Werkzeugen für Sicherheitstests. In der Regel waren Fuzzer Gegenstand der Diskussion.
2. *Interne Code Reviews*: Die Entwickler beschreiben, zu welchen Zeitpunkten Reviews durchgeführt werden und wo deren Nutzen/Vorteile liegen.
3. *Externe Code Reviews*: Zusätzlich zu internen Reviews werden auch Reviews von spezialisierten externen Unternehmen durchführt. Die Entwickler beschreiben den Mehrwert dieser kostenpflichtigen Analyse.
4. *Eigene Analysewerkzeuge*: Mehrere Projekte haben für ihren Bedarf eigene Analysewerkzeuge für Sicherheit entwickelt. Es wird auf die Funktionsweise und das Funktionsziel eingegangen.
5. *Modultests*: Die Entwickler setzen Modultests (PHPUnit, JUnit) ein, um ihre Software auf bestimmte Sicherheitsaspekte zu testen.

*Hinweis*: Sowohl Punkt 1 und Punkt 4 verweisen auf die Evaluation und den Einsatz von Werkzeugen, man hätte sie als einen Punkt aufführen können. Darauf wurde absichtlich verzichtet, da fertige und selbstentwickelte Werkzeuge bei den Projekten in einem völlig anderem Licht stehen, sodass hier eine Unterscheidung einer besseren Abgrenzung dient.

| Projekt | Spezialisierte Werkzeuge | Interne Reviews | Externe Reviews | Eigene Werkzeuge | Modultests | Schutzmaß nahmen |
|---|---|---|---|---|---|---|
| Joomla! | x | | | | | |
| JAMWiki | | | | | x | x |
| Bugzilla | | | | x | | x |
| MediaWiki | | x | | x | | x |
| A. Tomcat | x | x | x | | | |
| Gallery | x | x | x | | x | x |
| phpBB | x | x | x | x | x | x |

*Tabelle 2: Evaluation und Anwendung verschiedener Testwerkzeuge*

Tabelle 2 stellt dar, welches Projekt welchen Punkt evaluiert und durchführt hat. Zur Übersichtlichkeit sind die Projekte nach Häufigkeit der Aussagen gruppiert. Die expliziten Schutzmaßnahmen sind zur besseren Abgrenzung durch eine Doppellinie abgetrennt.

Die konkreten eingesetzten Werkzeuge lassen sich tabellarisch so darstellen:

| Projekt | Werkzeug | Kategorie |
|---|---|---|
| Bugzilla | runtests.pl | Musterabgleich, Datenflussanalyse (selbst entwickelt) |
| Gallery | PHPUnit | Modultests |
| Gallery | Burp Suite | Modultests, Fuzzer, Leistungsanalyse |
| JAMWiki | JUnit | Modultests |
| MediaWiki | fuzz-tester.php | Fuzzer (selbst entwickelt) |
| phpBB | grep-Erweiterung | Musterabgleich (selbst entwickelt) |

*Tabelle 3: Eingesetzte Werkzeuge*

Tabelle 3 stellt nur die Werkzeuge dar, die tatsächlich im Testeinsatz sind.

Die dargestellten Informationen in Tabelle 2 lassen sich ein Diagramm überführen:

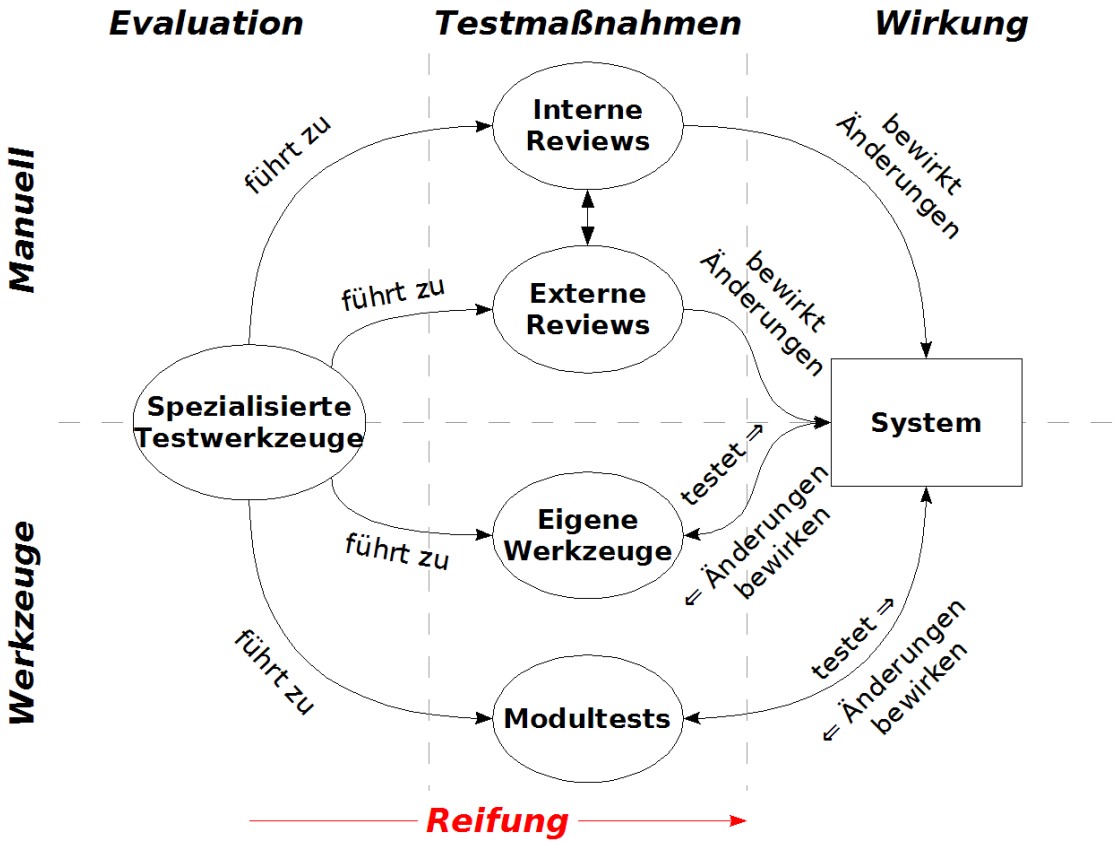



*Abbildung 2: Stufen des Testens*

Abbildung 2 Zeigt, wie die Verwendung von Werkzeugen stufenweise erfolgt. Aus den Erfahrungen mit spezialisierten Testwerkzeugen gelangt man zur nächsten Stufe. Sie

stellt dar, dass andere Werkzeuge beim Testen eines System eingesetzt werden und verdeutlicht die Auswirkung des Testens und den Rückfluss aus dem System an die Testmaßnahmen.

Das Diagramm trifft nicht vollständig auf jedes Projekt zu, es spiegelt den Konsens aus den Projekten wieder. Man kann durch Weglassen einzelner Elemente unvollständige Projekte darstellen. Die folgenden Abschnitte versuchen zu erklären, was die einzelnen Elemente bedeuten und wie die Verbindungen zwischen ihnen entstanden sind. Es soll dargestellt werden welche Werkzeuge die Projekte einsetzen (Forschungsfrage 1), wie der Einsatz jeweiliger Werkzeuge erfolgt und welche Auswirkungen sie haben (Forschungsfrage 2).

Der erste Abschnitt der Abbildung 2 behandelt die Evaluation spezialisierter Werkzeuge für Sicherheitstests und die Schlussfolgerung daraus hin zu anderen Testmaßnahmen. Viele Projekte haben sich meist, aber nicht explizit, zu Fuzzern geäußert. Ein Projekt (Apache Tomcat) hat die Ergebnisse solcher Werkzeuge evaluiert, sprich, externe Entwickler haben solche Werkzeuge eingesetzt und dem Projekt die Ergebnisse zur Verfügung gestellt. Die anderen drei Projekte haben solche Werkzeuge selbst evaluiert. Überraschenderweise deckten sich die Meinungen der Entwickler. Die Werkzeuge waren weder im Stande, bereits bekannte Schwachstellen zu reproduzieren, noch konnten sie neue finden. Die Anzahl an Falschmeldungen war sehr hoch. So sagten phpBB und Joomla!, dass Fuzzer gar nicht an die Stellen im Code bzw. System durchdringen, wo sie eigentlich nach einer Schwachstelle suchen wollen bzw. sollen. Die Entwickler vom Apache Tomcat und phpBB gingen weiter und sagten, dass sogar kostenpflichtige Werkzeuge nicht helfen und die beworbenen Eigenschaften nicht aufweisen. Ein Selbstversuch von phpBB mit dem Werkzeug Acunetix lieferte 160 Meldungen, davon waren 100 % Falschmeldungen. Es hat nicht die gewünschten zu testenden Stellen im System erreicht. Der Apache Tomcat-Code wird von Fortify Software kostenlos gescannt. Jedes Werkzeug erreicht einen bestimmten Prozentsatz an untersuchtem Code (vgl. Testabdeckung). Laut Entwickler erreichen Werkzeuge wie Fortify Software eine maximale Abdeckung bzgl. Sicherheit bei 10 %. Zu der geringen Abdeckung lieferte es zu viele Falschmeldungen. Es gibt jedoch eine positive Beobachtung. Das Gallery-Projekt nutzt die Burp Suite um HTTP-Requests zu manipulieren und um sie automatisch zu erzeugen. Es sollen dadurch Abstürze in der Software provoziert werden. Trotz des einen positiven Einsatzes, spricht sich jedes Projekt, das seine Meinung zu spezialisierten Werkzeugen geäußert hat, kritisch und negativ solchen Werkzeugen gegenüber aus. Durch den Mangel an Erfolgen, werden diese Werkzeuge als nicht sicherheitsfördernd erachtet, die Entwickler raten daher vom Einsatz ab.

Vergleicht man nun das Ergebnis der Evaluation mit den Daten in Tabelle 2, so stellt man fest, dass gerade die Projekte, die von spezialisierten Werkzeugen enttäuscht wurden, manuelle Testmaßnahmen bevorzugen im Gegensatz zu denen, die sich nicht über solche

Werkzeuge geäußert haben. Der vertikale Abschnitt *Evaluation* spiegelt die negativen Erfahrungen mit spezialisierten Testwerkzeugen wider, von dem vier Pfeile den Übergang hin zu dem Einsatz anderer Testmaßnahmen darstellen.

Ausgehend von der Evaluation wird zuerst auf die manuellen Testmaßnahmen eingegangen und im Anschluss auf die werkzeuggestützten Maßnahmen. Sie spiegeln den zweiten vertikalen Abschnitt wider.

Code Reviews scheinen mehreren Projekten ein effizientes Mittel zur Sicherheitssteigerung und Entwicklung neuer Schutzmaßnahmen zu sein. Apache Tomcat hat einen strikten Begutachtungsprozess. Jeder Commit/Patch wird von drei Entwicklern analysiert, getestet und bewertet. Gallery und Media Wiki führen den Begutachtungsprozess als Peer Review durch. Diese Maßnahme soll fehlerhaften Code erst gar nicht ermöglichen Teil des Systems zu werden. Wie man am Diagramm erkennen kann, erfordert jede Änderung am System auch eine Reihe von Tests, d.h. eine solche vorbeugende Maßnahme kann den Testaufwand nach Fehlern reduzieren. Zusätzlich legen MediaWiki und besonders phpBB Wert auf regelmäßige interne Code Reviews. Gallery führt diese hingegen nur sporadisch durch. Die Entwickler betonten, dass gute Sicherheit auf Erfahrung und Know-how basiert, sodass sie in externen Code Reviews weitere bedeutende Verbesserungen sehen. So berichtet ein Entwickler von Apache Tomcat, der selbst Leiter eines Sicherheitsunternehmens ist, dass ein solches Unternehmen bzw. eine trainierte Gruppe eine bis zu 90 %ige Sicherheitsabdeckung (vgl. Coverage) erreichen könnte, bei einer drastischen Minimierung der Falschmeldungen. Die Entwickler von phpBB und Gallery gehen noch weiter, sie lassen ihre Software von solchen Unternehmen tatsächlich regelmäßig testen und stellen fest, dass Schwachstellen vorhanden sind, die sie selbst nie bemerkt hätten.

Code Reviews geben den Projekten eine Möglichkeit zur deutlichen Steigerung der code und Systemqualität, aber mit der Kehrseite des notwendigen Wissens. Gallery und phpBB verlassen sich deshalb auf Sicherheitsunternehmen, um dieses zu erreichen und um zusätzlich selbst aus den Erkenntnissen neue Einblicke und Sichtweisen zu gewinnen.

Wirft man einen weiteren Blick auf die Spalten *Eigene Werkzeuge* bis *Schutzmaßnahmen* in Tabelle 2, so stellt man fest, dass nur die Projekte explizite Schutzmaßnahmen treffen, die weitere Testwerkzeuge einsetzen. Betrachtet man diese Erkenntnis in Abbildung 2, sieht man, dass es einen geschlossenen Kreislauf zwischen Werkzeugen und System gibt. Ein erfolgreicher Test führt zu einer Änderung im System. Eine neue Änderung im System erfordert die Anpassung oder Erstellung von weiteren Testfällen.

Drei Projekte haben auf ihren Bedarf hin Werkzeuge entwickelt, die Schwachstellen auffinden sollen. Ein Entwickler von MediaWiki hat in den letzten Jahren einen Fuzzer entwickelt, der den Wikisyntax/HTML-Parser testet. Durch den Einsatz dieses Werkzeugs war der Entwickler im Stande mehrere Schwachstellen aufzudecken und zu beheben.

Bugzilla und phpBB führen statische Analysen durch. In beiden Fällen sollen Codefragmente gefunden werden, die nicht die empfohlenen Schutzmaßnahmen verwenden oder eine andere Bedrohung darstellen könnten. Bugzillas Werkzeug führt eine Perl-basierte Datenflussanalyse durch, die auf jedem Commit angewendet wird. Jeder Commiter muss sicherstellen, dass der Test keine Fehler meldet, da der Commit sonst nicht ins Repository gelangt. Das Sicherheitsteam von phpBB geht einen anderen Weg. Sie suchen mithilfe eines Werkzeug auf grep-Basis[11] mit einen Musterabgleich nach potenziell gefährlichen Codestellen.

Modultests sind für rein funktionale Tests ausgelegt. Die Projekte stellen trotzdem dar, wie man sie sinnvoll einsetzten kann. JAMWIki nutzt JUnit für Regressiontests gegen XXS beim Wikisystax-Parser. Durch die Verwendung von Maven werden alle Tests automatisch im Build- und Releaseprozess ausgeführt. Diese Tests sind auf Anregung jenes Entwicklers entstanden, der auch den MediaWiki-Fuzzer entwickelt hat. Gallerys große Testmenge enthält speziell präparierte Testfälle, um die Schutzmaßnahmen gegen XSS oder SQL-Injection zu testen bzw. diese zu brechen. Die Tests werden von verschiedenen Entwicklern auf verschiedenen Plattformen durchgeführt. Die Häufigkeit ist nicht bekannt. In der gegenwärtigen Version (3.0) von phpBB gibt es noch keine Modultests. Im Interview versicherte der Entwickler, dass mit Version 3.2 massiv auf den Einsatz von Modultests gesetzt werden soll. Trotz der Nachteile derselben, ist der er davon überzeugt, dass man durch gut konstruierte Testfälle die Schutzmaßnahmen und das System so testen kann, dass ein höheres Sicherheitsniveau erreicht werden kann.

### 3.6.4 Kritik

Es gibt mehrere Kritikpunkte, die man an der Fallstudie äußern muss. Durch die Beschränkung auf Web-Informationssysteme bekommt man eine einseitige Sicht auf den Einsatz der Werkzeuge. Andere Softwarearten wie Betriebssysteme (z.B. OpenBSD) oder Verschlüsselungssysteme (z.B. GPG, OpenSSH, OpenSSL) sind prädestinierte Fälle für eine Studie, da diese Projekte Sicherheit als einen zentralen Bestandteil ihrer Entwicklung sehen. Zum anderen kann man der Fallstudie eine gewisse Oberflächlichkeit bei den Analysemethoden vorwerfen. Vorhandene Quellen wurden nur bis zu einem bestimmten Grad untersucht und dann als ergiebig oder nicht ergiebig eingeteilt. Es kann also durchaus sein, dass verworfene Projekte doch Werkzeuge für Sicherheitstests einsetzten, aber diese nicht als solche anerkannt wurden.

### 3.6.5 Zusammenfassung

Unter Berücksichtigung der genannten Einschränkungen bzw. Kritik lässt sich ein vielseitiges Ergebnis präsentieren. Der Einsatz automatisierter Werkzeuge ist ein schwieriges

11 Unix-Kommando zur Suche von Strings in einer Datei

Feld. Viele Projekte resignieren bei speziellen Testwerkzeugen aufgrund ihrer Schwächen bei der Ausführung. Kritikpunkte sind die vielen Falschmeldungen und das Testen an den falschen Stellen. Jedoch gibt es wenige, die gelernt haben, fertige Testwerkzeuge sinnvoll einzusetzen. Angefangen von der spezialisierten Burp Suite, nutzen Entwickler klassische Modultests um automatisiert die Schutzmaßnahmen zu testen. Spezialisierte Werkzeuge stellen kein Allheilmittel dar, auch wenn deren Featurelisten tolle Funktionen versprechen. Es ist ein Zusammenspiel von vielfältigen Maßnahmen, um das System von verschiedenen Blickwinkeln und Standpunkten zu beleuchten und es zu verbessern. Weg von den fertigen Werkzeugen, hin zu hausgemachten Lösungen, ist das Rezept von vielen Projekten. Sie entwickeln und setzen erfolgreich selbstgeschriebene Werkzeuge ein und unterstützten eine gute Programmierung mit regelmäßigen Security Code Reviews.

# 4 Ausblick

Diese Fallstudie wirft mit ihren Ergebnissen mehrere Fragen auf und zeigt Möglichkeiten der weiteren Untersuchung. Wie wird der Trend in den nächsten Jahren bei spezialisierten Werkzeugen wie Fuzzern sein? Im Moment haben solche Werkzeuge einen großen Schwachpunkt. Sie versuchen einen breiten bzw. allgemeinen Raum für ihren Spezialfall (z.B. XSS oder SQL-Injection) abzudecken. Diese Allgemeinheit ermöglicht kein Verständnis der speziellen Anpassungen und Schutzfunktionen in einem System. Die Wirkung des Werkzeugs verpufft, weil eben genau dieses Verständnis fehlt und nur durch eigene Ansätze gelöst werden kann. Ermöglichen die gewonnen Erkenntnisse eine Verbesserung der gegenwärtigen Werkzeuge? Betrachtet man die Kritik noch einmal, wäre eine vergleichbare Fallstudie in anderen Softwarearten ein interessanter Gegenpol zu dieser Fallstudie. Es gibt aber auch weiterreichende und tiefergehende Möglichkeiten. Die Projekte Gallery, phpBB oder MediaWiki waren schöne Beispiele, die gezeigt haben wie sie auf verschiedenste Weise versuchen Herr der Lage zu werden. Basierend auf dieser Fallstudie wäre es möglich, einen fähigen PHP-Entwickler in diese Projekte zu injizieren, mitzuentwickeln, zu diskutieren und um die Testprozesse bei einem dieser beiden Projekte wirklich in der Tiefe zu untersuchen bzw. aufgrund der Ergebnisse sogar Werkzeuge entwickeln, die andere PHP-Projekte sinnvoll einsetzen könnten. Es bleibt weiter spannend, wie die Community versucht, aus ihrem eigenen Antrieb heraus selbst nach Lösungen zu suchen und auch, wie die Industrie versucht, ihre kritisierten Werkzeuge zu verbessern.

# Literaturverzeichnis


[1] Secunia, *Advisory Statistics*, Secunia, 2008, URL: http://secunia.com/advisory_statistics/, Stand: 2008-04-07.

[2] David A. Wheeler, *Why Open Source Software? Look at the Numbers!*, David A. Wheeler, 2007, URL: http://www.dwheeler.com/oss_fs_why.html, Stand: 2008-05-26.

[3] John Viega, Gary McGraw, *Building Secure Software: How to Avoid Security Problems the Right Way*, 1. Auflage, Indianapolis, IN: Addison Wesley Professional, 2001.

[4] Claudia Eckert, *IT-Sicherheit: Konzepte - Verfahren - Protokolle*, 3. Auflage, München: Oldenbourg Wissenschaftsverlag, 2004.

[5] Ross J. Anderson, *Security Engineering: A Guide to Building Dependable Distributed Systems*, 1. Ausgabe, Hoboken, NJ: Wiley Computer Publishing, 2001.

[6] Richard A. DeMillo, W. Michael McCracken, R.J. Martin, John F. Passafiume, *Software Testing and Evaluation*, 1. Ausgabe, Menlo Park, CA: The Benjamin/Cummings Publishing Company, Inc., 1987.

[7] Gary McGraw, B. Chess, "Static Analysis for Security", *IEEE Security & Privacy*, vol. 2, no. 6, Washington, DC: IEEE Computer Society, 2004, S. 76-79.

[8] Daniel Simon, Frank Simon, "Statische Code-Analyse als effiziente Qualitätssicherungsmaßnahme im Umfeld eingebetteter Systeme", *9. Workshop Software-Reengineering*, Mainz: Johannes Gutenberg-Universität Mainz - Institut für Informatik, 2007, S. 49-52.

[9] D. Wagner, J. S. Foster, E. Brewer, and A. Aiken, "A First Step Towards Automated Detection of Buffer Overrun Vulnerabilities", *Proc. Network and Distributed System Security Symp.*, Reston, VA: Internet Society, 2000, S. 15 ff.

[10] Benjamin Livshits, Monica S. Lam, "Finding Security Vulnerabilities in Java Applications with Static Analysis", *Proceedings of the 14th Usenix Security Symposium*, Baltimore, MD: USENIX, 2005, S. 271-286.

[11] Andrew Glover, *In pursuit of code quality: Don't be fooled by the coverage report*, IBM developerWorks, 2006, URL: http://www-128.ibm.com/developerworks/java/library/j-cq01316/index.html?ca=drs, Stand: 2008-05-24.

[12] Wikipedia, *Dynamische Analyse*, Wikipedia, Die freie Enzyklopädie, 2007, URL: http://de.wikipedia.org/w/index.php?title=Dynamische_Analyse&oldid=29933042, Stand: 2008-05-24.

[13] Christiane Rütten, "Schwachstellensuche mit Fuzzing", *c't*, vol. -, no. 18, Hannover: Heise Zeitschriften Verlag, 2006, S. 210 ff.

[14] Ilja van Sprundel, "Fuzzing: Breaking software in an automated fashion", *22nd Chaos Communication Congress*, Hamburg: CCC, 2005, 5 Seiten.

[15] Robert K. Yin, *Case Study Research: Design and Methods*, 1. Ausgabe, Charlotte, NC: Baker & Taylor, Inc., 1984.

[16] Susan K. Soy, *The Case Study as a Research Method*, Austin, TX: University of Texas at Austin, 1997.

# Anhang A

In diesem Anhang werden alle benutzen Quellen für die Erhebung der Daten erfasst. Die Aufteilung erfolgt pro Projekt.

## Alfresco

Homepage: http://www.alfresco.com/

Aus Homepage/Wiki:


[A-1] About Alfresco - The Open Source Alternative for Enterprise Content Management: http://www.alfresco.com/about/ (Stand 2008-04-20)

[A-2] Alfresco Customer Overview: http://www.alfresco.com/customers/ (Stand 2008-04-20)

[A-3] Qa test plan – AlfrescoWiki: http://wiki.alfresco.com/w/index.php?title=Qa_test_plan&oldid=17284 (Stand 2008-04-20)

[A-4] Test Plan – AlfrescoWiki: http://wiki.alfresco.com/w/index.php?title=Test_Plan&oldid=17282 (Stand 2008-04-20)

[A-5] Com test tools – AlfrescoWiki: http://wiki.alfresco.com/w/index.php?title=Com_test_tools&oldid=17680 (Stand 2008-04-20)

[A-6] AlfrescoForge: Alfresco Test Automation: Projektinfo: http://forge.alfresco.com/projects/qtp/ (Stand 2008-04-20)

[A-7] AlfrescoForge: Alfresco WCM Website Framework: Projektinfo: http://forge.alfresco.com/projects/wsf/ (Stand 2008-04-20)

[A-8] OSS Code Analysers – AlfrescoWiki: http://wiki.alfresco.com/w/index.php?title=OSS_Code_Analysers&oldid=17287 (Stand 2008-04-20)

[A-9] OSS Code Coverage – AlfrescoWiki: http://wiki.alfresco.com/w/index.php?title=OSS_Code_Coverage&oldid=17286 (Stand 2008-04-20)

[A-10] OSS Func TT – AlfrescoWiki: http://wiki.alfresco.com/w/index.php?title=OSS_Func_TT&oldid=17285 (Stand 2008-04-20)

[A-11] OSS Perf TT – AlfrescoWiki: http://wiki.alfresco.com/w/index.php?title=OSS_Perf_TT&oldid=17289 (Stand 2008-04-20)

[A-12] OSS Unit TT – AlfrescoWiki: http://wiki.alfresco.com/w/index.php?title=OSS_Unit_TT&oldid=17288 (Stand 2008-04-20)


Aus Repository/Buildskripte:


[A-13] build.xml: http://svn.alfresco.com/repos/alfresco-open-mirror/alfresco/HEAD/root/build.xml Revision 1784 (Stand 2008-04-20)


Aus Mailinglisten/Foren:

[A-14] Alfresco • View forum - Quality Assurance: http://forums.alfresco.com/viewforum.php?f=31 (Stand 2008-04-20)

[A-15] Alfresco • View topic - Alfresco and cross-site attack: http://forums.alfresco.com/viewtopic.php?f=14&t=6032&p=19490&hilit=xss#p19490 (Stand 2008-04-20)

Aus Kontakt:

[A-16] Alfresco • View topic - Assuring Security by testing: http://forums.alfresco.com/viewtopic.php?f=31&t=12233&p=40473#p40473 (Stand 2008-05-10)

## Apache HTTPd

Homepage: http://httpd.apache.org/

Aus Homepage/Wiki:

[A-17] Reporting Security Problems with Apache - The Apache HTTP Server Project: http://httpd.apache.org/security_report.html (Stand 2008-04-20)

[A-18] Security Tips - Apache HTTP Server: http://httpd.apache.org/docs/trunk/misc/security_tips.html (Stand 2008-04-20)

Aus Repository/Buildskripte:

[A-19] Regressionstests: http://svn.apache.org/viewvc/httpd/test/framework/trunk/t/security/?pathrev=647766 (Stand 2008-04-20)

[A-20] Perl-Testframework: http://svn.apache.org/viewvc/httpd/test/framework/trunk/?pathrev=647766 (Stand 2008-04-20)

Aus Mailinglisten/Foren:

[A-21] Re: Security Audit: http://www.mail-archive.com/dev@httpd.apache.org/msg15681.html (Stand 2008-04-20)

Aus Kontakt:

[A-22] Assuring Security by testing: http://www.mail-archive.com/dev@httpd.apache.org/msg40593.html (Stand 2008-05-10)

## Apache OFBiz

Homepage: http://ofbiz.apache.org/

Aus Homepage/Wiki:

[A-23] Apache OFBiz User List - The Open For Business Project Wiki - Apache OFBiz Documentation Site - Powered by Confluence - Hosted by Hotwax Media and Contegix: http://docs.ofbiz.org/pages/viewpage.action?pageId=4311 (Stand 2008-04-21)

[A-24] ASF Board Report 2008-03 - OFBiz Project Administration Workspace - Apache

OFBiz Documentation Site - Powered by Confluence - Hosted by Hotwax Media and Contegix: http://docs.ofbiz.org/pages/viewpage.action?pageId=3532 (Stand 2008-04-21)

Aus Repository/Buildskripte:

[A-25] Repository: http://svn.apache.org/viewvc/ofbiz/trunk/?pathrev=650113 (Stand 2008-04-21)

[A-26] Test-Suite: http://svn.apache.org/viewvc/ofbiz/trunk/framework/testtools/?pathrev=650113 (Stand 2008-04-21)

Aus Mailinglisten/Foren:

[A-27] Nabble - OFBiz - Dev - Dev - Automated regression testing tool for Java API: http://www.nabble.com/Dev---Automated-regression-testing-tool-for-Java-API-td3075269.html#a3075269 (Stand 2008-04-21)

[A-28] Nabble - OFBiz - Dev - Dev - Automated Testing in OFBiz: http://www.nabble.com/Dev---Automated-Testing-in-OFBiz-td4849934.html#a4849934 (Stand 2008-04-21)

[A-29] Nabble - OFBiz - Dev - More on automating testing: http://www.nabble.com/More-on-automating-testing-td6038820.html#a6100352 (Stand 2008-04-21)

[A-30] Nabble - OFBiz - Dev - More on testing: http://www.nabble.com/More-on-testing-td7653601.html#a7655166 (Stand 2008-04-21)

[A-31] Nabble - OFBiz - Dev - OFBiz Security Doc: http://www.nabble.com/OFBiz-Security-Doc-td14554061.html#a14554061 (Stand 2008-04-21)

[A-32] Nabble - OFBiz - Dev - Ofbiz Test Automation Services Offered: http://www.nabble.com/Ofbiz-Test-Automation-Services-Offered-td8638186.html#a8638186 (Stand 2008-04-21)

[A-33] Nabble - OFBiz - Dev - OFBiz Testing Initiative: http://www.nabble.com/OFBiz-Testing-Initiative-td7119966.html#a7146151 (Stand 2008-04-21)

[A-34] Nabble - OFBiz - Dev - Preparing Test Data for OFBiz JUnit Tests: http://www.nabble.com/Preparing-Test-Data-for-OFBiz-JUnit-Tests-td16021567.html#a16063190 (Stand 2008-04-21)

[A-35] Nabble - OFBiz - Dev – Selenium: http://www.nabble.com/Selenium-td8635367.html#a8687470 (Stand 2008-04-21)

[A-36] Nabble - OFBiz - User - XSS exploit countermeasure? Filtering user input: http://www.nabble.com/XSS-exploit-countermeasure--Filtering-user-input-td16364314.html#a16364314 (Stand 2008-04-21)

[A-37] Quality Assurance Agent (XUL-Applikation für Firefox): http://labs.libre-entreprise.org/projects/qaa/ (Stand 2008-06-09)

Aus Kontakt:

[A-38] Nabble - Assuring Security by testing: http://www.nabble.com/Assuring-Security-by-testing-tt17000341.html#a17000341 (Stand 2008-05-10)

## Apache Tomcat

Homepage: http://tomcat.apache.org/

Aus Homepage/Wiki:

[A-39] Red Hat JBoss: http://en.wikipedia.org/w/index.php?title=JBoss_application_server&oldid=217280820 (Stand 2008-06-09)

[A-40] Sun GlassFish: http://en.wikipedia.org/w/index.php?title=GlassFish&oldid=217430610 (Stand 2008-06-09)

[A-41] Apache Tomcat - Apache Tomcat -- Reporting Security Problems: http://tomcat.apache.org/security.html (Stand 2008-04-21)

Aus Repository/Buildskripte:

[A-42] JUnit-Testcases: http://svn.apache.org/repos/asf/tomcat/trunk/test/ Revision 649785 (Stand 2008-04-21)

Aus Kontakt:

[A-43] Nabble - Assuring Security by testing: http://www.nabble.com/Assuring-Security-by-testing-to16979750.html#a16979750 (Stand 2008-04-21)

[A-44] Fortify Software: Java Open Review Project: https://opensource.fortify.com/teamserver/welcome.fhtml (Stand 2008-04-21)

[A-45] Fortify Software und FindBugs bieten Sicherheits - und Qualitätsanalyse für Java-Code: http://www.fortify.com/news-events/releases/de_DE/2006/2006-12-11.jsp (Stand 2008-04-21)

## Bugzilla

Homepage: http://www.bugzilla.org/

Aus Homepage/Wiki:

[A-46] Security Advisories :: Bugzilla :: bugzilla.org: http://www.bugzilla.org/security/ (Stand 2008-04-22)

[A-47] Features :: Bugzilla :: bugzilla.org: http://www.bugzilla.org/features/#security (Stand 2008-04-22)

[A-48] Developer's Guide :: Bugzilla :: bugzilla.org: http://www.bugzilla.org/docs/developer.html#security (Stand 2008-04-22)

[A-49] Developer's Guide :: Bugzilla :: bugzilla.org: http://www.bugzilla.org/docs/developer.html#testsuite (Stand 2008-04-22)

Aus Repository/Buildskripte:

[A-50] runtests.pl: http://lxr.mozilla.org/bugzilla/source/runtests.pl (Stand 2008-04-22)

Aus Kontakt:

[A-51] Assuring Security by testing: http://bugzilla.org/cgi-bin/mj_wwwusr?user=ossipov%40inf.fu-berlin.de&passw=&list=developers&brief=on&func=archive-get-

part&extra=200804/41 (Stand 2008-05-10)
Antwort 1: http://bugzilla.org/cgi-bin/mj_wwwusr?user=ossipov%40inf.fu-berlin.de&passw=&list=developers&brief=on&func=archive-get-part&extra=200805/1 (Stand 2008-05-10)
Antwort 2: http://bugzilla.org/cgi-bin/mj_wwwusr?user=ossipov%40inf.fu-berlin.de&passw=&list=developers&brief=on&func=archive-get-part&extra=200805/2 (Stand 2008-05-10)

## Gallery

Homepage: http://gallery.menalto.com/

Aus Homepage/Wiki:

[A-52] Gallery2:Code Overview - Gallery Codex: http://codex.gallery2.org/index.php?title=Gallery2:Code_Overview&oldid=16684 (Stand 2008-04-22)

[A-53] Gallery2:Development Concepts - Gallery Codex: http://codex.gallery2.org/index.php?title=Gallery2:Development_Concepts&oldid=9994 (Stand 2008-04-22)

[A-54] Gallery2:Test Matrix - Gallery Codex: http://codex.gallery2.org/index.php?title=Gallery2:Test_Matrix&oldid=17276 (Stand 2008-04-22)

Aus Repository/Buildskripte:

[A-55] Modules: http://gallery.svn.sourceforge.net/viewvc/gallery/trunk/gallery2/modules/?pathrev=17588 (Stand 2008-04-22)

Aus Mailinglisten/Foren:

[A-56] Are there updates available to address the SQL Injection... | Gallery: http://gallery.menalto.com/node/38002 (Stand 2008-04-22)

[A-57] security unit tests? | Gallery: http://gallery.menalto.com/node/30374 (Stand 2008-04-22)

Aus Kontakt:

[A-58] [Gallery-devel] Assuring Security by testing: http://sourceforge.net/mailarchive/forum.php?thread_name=4819A2D9.2020800%40inf.fu-berlin.de&forum_name=gallery-devel (Stand 2008-05-10)

## JAMWiki

Homepage: http://jamwiki.org/

Aus Homepage/Wiki:

[A-59] User:Wrh2 – JAMWiki: http://jamwiki.org/wiki/en/Special:History?topicVersionId=8535&topic=User:wrh2 (Stand 2008-05-10)

Aus Repository/Buildskripte:

[A-60] XSS Unit-Tests: http://jamwiki.svn.sourceforge.net/viewvc/jamwiki/wiki/trunk/jamwiki-core/src/test/resources/data/topics/?pathrev=2197 (Stand 2008-04-23)

Aus Kontakt:

[A-61] Feedback – JAMWiki: http://jamwiki.org/wiki/en/Special:History?topicVersionId=8750&topic=Feedback (Stand 2008-05-10)

## Joomla!

Homepage: http://www.joomla.org/

Aus Homepage/Wiki:

[A-62] How to add CSRF anti-spoofing to forms - Joomla! Documentation: http://docs.joomla.org/index.php?title=How_to_add_CSRF_anti-spoofing_to_forms&oldid=2376 (Stand 2008-04-23)

[A-63] Joomla Administrators Security Checklist - Joomla! Documentation: http://docs.joomla.org/index.php?title=Joomla_Administrators_Security_Checklist&oldid=4643#Ongoing_Site_Administration (Stand 2008-04-23)

[A-64] Joomla! Developer Network: http://dev.joomla.org/component/option,com_jd-wiki/Itemid,/id,tips:make_secure/&s=xss (Stand 2008-04-23)

[A-65] Joomla! Extensions Directory - Joomla! Tools Suite : JTS-sa, HISA-sa, JTS-post: http://extensions.joomla.org/component/option,com_mtree/task,viewlink/link_id,1734/Itemid,35/ (Stand 2008-04-23)

[A-66] Projekte > Joomla! Tools Suite > Home: http://joomlacode.org/gf/project/jts/ (Stand 2008-04-23)

[A-67] Security and Performance FAQs - Joomla! Documentation: http://docs.joomla.org/index.php?title=Security_and_Performance_FAQs&oldid=4408 (Stand 2008-04-23)

[A-68] Top 10 Stupidest Administrator Tricks - Joomla! Documentation: http://docs.joomla.org/index.php?title=Top_10_Stupidest_Administrator_Tricks&oldid=4210 (Stand 2008-04-23)

[A-69] Vulnerable Extensions List - Joomla! Documentation: http://docs.joomla.org/index.php?title=Vulnerable_Extensions_List&oldid=4409 (Stand 2008-04-23)

[A-70] Security - Joomla! Documentation: http://docs.joomla.org/index.php?title=Security&oldid=4632 (Stand 2008-04-23)

Aus Mailinglisten/Foren:

[A-71] Joomla! Community Forum • View topic - Is there a script recommended to test security?: http://forum.joomla.org/viewtopic.php?f=267&t=288535 (Stand 2008-04-23)

Aus Kontakt:

[A-72] Joomla! Community Forum • View topic - Assuring Security by testing: http://forum.joomla.org/viewtopic.php?f=326&t=288634 (Stand 2008-05-10)

## MediaWiki

Homepage: http://www.mediawiki.org/

Aus Homepage/Wiki:

[A-73] Security – MediaWiki: http://www.mediawiki.org/w/index.php?title=Security&oldid=160018 (Stand 2008-04-24)

[A-74] MediaWiki - Nick Jenkins: http://nickj.org/index.php?title=MediaWiki&oldid=2921 (Stand 2008-04-24)

Aus Repository/Buildskripte:

[A-75] [mediawiki] Index of /tags/REL1_12_0/phase3/tests: http://svn.wikimedia.org/viewvc/mediawiki/tags/REL1_12_0/phase3/tests/?pathrev=32264 (Stand 2008-04-24)

[A-76] [mediawiki] View of /trunk/phase3/maintenance/fuzz-tester.php: http://svn.wikimedia.org/viewvc/mediawiki/trunk/phase3/maintenance/fuzz-tester.php?revision=33124&view=markup (Stand 2008-04-24)

Aus Kontakt:

[A-77] Assuring Security by testing: http://thread.gmane.org/gmane.science.linguistics.wikipedia.technical/37926 (Stand 2008-05-10)

## Openbravo ERP

Homepage: http://www.openbravo.com/

Aus Homepage/Wiki:

[A-78] Category:QualityAssurance - Developer at Openbravo: http://wiki.openbravo.com/wiki/index.php?title=Category:QualityAssurance&oldid=10101 (Stand 2008-04-24)

Aus Kontakt:

[A-79] [Openbravo-development] Assuring Security by testing: http://sourceforge.net/mailarchive/forum.php?thread_name=4819A5F1.5040405%40inf.fu-berlin.de&forum_name=openbravo-development (Stand 2008-05-10)

## OpenCms

Aus Repository/Buildskripte:

[A-80] OpenCms Referenzseiten: http://www.opencms.org/de/support/references/index.html (Stand 2008-04-25)

[A-81] [OpenCms] Index of /opencms/test/org/opencms: http://cvs.opencms.org/viewvc.cgi/opencms/test/org/opencms/ (Stand 2008-04-25)

Aus Kontakt:

[A-82] [opencms-dev] Assuring Security by testing: http://lists.opencms.org/pipermail/opencms-dev/2008q2/029862.html (Stand 2008-05-10)

## phpBB

Homepage: http://www.phpbb.com/

Aus Homepage/Wiki:

[A-83] phpBB • Comparison of phpBB to other Popular Forum Solutions (Paid Security Audit): http://www.phpbb.com/about/features/compare.php (Stand 2008-04-25)

Aus Mailinglisten/Foren:

[A-84] Area51 @ phpBB.com • View topic - phpbb's test framework?: http://area51.phpbb.com/phpBB/viewtopic.php?f=4&t=29037 (Stand 2008-04-25)

[A-85] phpBB • View topic - Automated Testing: http://www.phpbb.com/community/viewtopic.php?f=64&t=591219&p=3259094&hilit=automated+testing#p3259094 (Stand 2008-04-25)

[A-86] phpBB • View topic - More secure PHPBB3: http://www.phpbb.com/community/viewtopic.php?f=46&t=829705&p=4714765&hilit=security+testing#p4714765 (Stand 2008-04-25)

Aus Kontakt:

[A-87] Area51 @ phpBB.com • View topic - Assuring Security by testing: http://area51.phpbb.com/phpBB/viewtopic.php?f=3&t=29576 (Stand 2008-05-10)

[A-88] Transkript einer privaten Nachricht aus Area 51:

Re: Assuring Security by testing

Sent at: Tue May 06, 2008 11:08 am

by Kellanved

Hi Michael,

Donnerstag ist bei mir generell schlecht, da dort der ganze Tag mit Terminen zugepflastert ist. Von daher würde ich entweder Morgen irgendwann zwischen 12 und 14 Uhr, oder am Montag vorschlagen. Ich bin auch nur kurz für zwei Wochen in Deutschland, daher ist das Fenster recht eng (Bin aber zum Linuxtag wieder zurück).

ja - ich habe sie überflogen. Ich bin etwas verwundert über die "10+ Sicherheitslücken" - mir sind eigentlich keine bekannt, sieht man von Secunias seltsamer Einstufung von Changelogeinträgen ab.

Ich kann Dir schon eine kurze Zusammenfassung geben: unser Sicherheitsansatz ist "formale Methoden", genauer gesagt verwenden wir typisierte Zuweisungsoperatoren und typabhängige Db-queries. Dies allein bringt bereits die automatisierten Tools aus dem Tritt. Accetunix z.B. lieferte einfach nur Schwachsinn als Ergebnis.

Typisierung fängt einen beträchtlichen Teil der möglichen unangenehmen Überraschungen. Darüber hinaus haben wir ein internes "Audit" Team, dessen Mitglieder kontinuierlich Änderungen an der Codebasis untersuchen; ebenso tut es das grössere QA Team. Das Audit Team - z.T. recht bekannte Sicherheitsleute - fährt zudem Penetration Tests und gibt frühe Warnungen über "0days".

Automatisierung heißt für uns idR "Grep" . Wir verwenden außerdem für phpBB 3.2 Unit Tests mit PHPUnit.

Auch sind viele Sicherheitsprobleme im Web auch bei Entwicklern nicht hinreichend bekannt; bei der Konkurrenz kann man z.B. einen fahrlässigen Umgang mit Uploads etc beobachten. Tools finden solche Sachen aber praktisch nie. Know-How und rigorose (Penetration-) Tests scheinen mir für Sicherheit entscheidend zu sein.

Sieht man sich die Reports auf Secunia an, wird schnell klar dass 90% der Sicherheitslücken der Vergangenheit in MODs zu finden waren. Dafür bietet das MOD Team kostenlose Sicherheitsaudits an. Für diese Audits verwenden wir eine selbstgeschriebende Software, die insbesondere die Verwendung von Typen kontrolliert.

MFG

~H

[A-89] Mitschrift eines persönliches Interviews mit Henry Sudhof (Stand 2008-05-08):

- Security Audit von SektorEins/Stefan Esser (PHP-Gott) durchgeführt
- Nutzt kein explizites Securitytestwerkzeug
- Halbautomatische Tests identifizieren Sicherheitsprobleme
- Automatische Sicherheitstestwerkzeuge sind sehr minderwertig
- Sie waren nicht im Stande eine Sicherheitslücke zu finden und lieferten sehr

viele false positives

- Sie haben keine der bekannten Verwundbarkeiten gefunden
- Manuelles Audit ist besser
- Sicherheitsansatz: Formale Methoden
- typisierte Zuweisungsoperationen wichtig
- typanhängige DB-Anfragen
- Die meisten automatischen Tools kommen mit sowas nicht klar, Acunetix (1000 €+) liefert nur schwachsinnige Ergebnisse (meldete mehr als 160 Lücken, alles false positive)
- Internes Audit Team übeprüft regelmäßig die Codebasis
- QA-Team testet regelmäßig
- Audit Team führt manuelle Penetrationstests durch
- Automatisierung bedeutet „grep“
- Know-how und ausgiebige Penetrationstests sind Schlüssel für gute Sicherheit
- Bieten kostenlose Audits für MODs an
- Für Audits wird ein selbst geschriebenes Tool auf basis von „grep“ benutzt, insbesondere werden Typen kontrolliert
- Sicherheit ist Erfahrungssache
- Jeder Commit/Patch geht durch ein mehrstufiges Testsystem von Werkzeugen und Teams
- Fuzzing liefert nicht das, was es soll
- PHP ist selbst das Problem, schwache Typisierung ist 80 % der Probleme
- Größtes PHP-Skript mit 300 000 Zeilen Code
- Wichtige Sicherheitsschritte: Typüberprüfung; Code aufbrechen in lesbare, kleine Teile; Sich mit der Problematik beschäftigen

[A-90] Website Security - Acunetix Web Security Scanner: http://www.acunetix.com/ (Stand 2008-06-09)

## phpMyAdmin

Homepage: http://www.phpmyadmin.net/

Aus Homepage/Wiki:

[A-91] phpMyAdmin > Security | MySQL Database Administration Tool | www.phpmyadmin.net: http://www.phpmyadmin.net/home_page/security.php

(Stand 2008-04-26)

[A-92] XSS – PmaWiki: http://wiki.cihar.com/wiki/index.php?title=XSS&oldid=4235 (Stand 2008-04-26)

Aus Repository/Buildskripte:

[A-93] [phpmyadmin] Index of /trunk/phpMyAdmin/test: http://phpmyadmin.svn.sourceforge.net/viewvc/phpmyadmin/trunk/phpMyAdmin/test/?pathrev=11302 (Stand 2008-04-26)

Aus Kontakt:

[A-94] [Phpmyadmin-devel] Assuring Security by testing: http://sourceforge.net/mailarchive/forum.php?thread_name=48199E5D.7040402%40inf.fu-berlin.de&forum_name=phpmyadmin-devel (Stand 2008-05-10)

## WebCalendar

Homepage: http://www.k5n.us/webcalendar.php

Aus Repository/Buildskripte:

[A-95] [webcalendar] Index of /webcalendar/tests: http://webcalendar.cvs.sourceforge.net/webcalendar/webcalendar/tests/?pathrev=REL_2_0 (Stand 2008-04-26)

Aus Kontakt:

[A-96] Assuring Security by testing: http://sourceforge.net/forum/message.php?msg_id=4938878 (Stand 2008-05-10)

## XOOPS

Homepage: http://www.xoops.org/

Aus Homepage/Wiki:

[A-97] XoopsWiki - QA and Testing : XOOPS Project: http://www.xoops.org/modules/mediawiki/index.php?title=QA_and_Testing&oldid=3542 (Stand 2008-04-26)

[A-98] CN-Techniker - Xoops Protector: http://www.cn-techniker.de/blog/cms/xoops-protector.html (Stand 2008-04-26)

[A-99] PEAK XOOPS - Protector 3.04a: http://xoops.peak.ne.jp/md/mydownloads/singlefile.php?cid=1&lid=94 (Stand 2008-04-26)

Aus Kontakt:

[A-100] Assuring Security by testing: http://sourceforge.net/forum/message.php?msg_id=4938688 (Stand 2008-05-10)