INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Bachelorarbeit

Gamification zur Wissensvermittlung für Linux-Serveradministratoren am Beispiel des Leibniz-Rechenzentrums

Veronika Sonntag

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Bachelorarbeit

Gamification zur Wissensvermittlung für Linux-Serveradministratoren am Beispiel des Leibniz-Rechenzentrums

Veronika Sonntag

Aufgabensteller: Prof. Dr. Wolfgang Hommel

Betreuer: Tanja Hanauer
Daniela Pöhn

Abgabetermin: 7. Juni 2016

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

München, den 7. Juni 2016

. . . . . .

(Unterschrift des Kandidaten)

Zusammenfassung

Gamification ist die Nutzung von Spielelementen in Kontexten, die nichts mit Spielen zu tun haben. Lernplattformen wie Codeacademy, ein Massive Open Online Course zum Erlernen von Programmiersprachen, nutzen solche Spielelemente zur Motivation ihrer Nutzer und zur Steigerung des Lernerfolgs.

In dieser Arbeit werden verschiedene Gamfication-Ansätze evaluiert. Außerdem wird untersucht, ob Gamification auch zur Wissensvermittlung in Bereichen wie der Serveradministration, die sehr viel Fachwissen erfordert, eingesetzt werden kann. Zu diesem Zweck wird eine Lernanwendung für die Linux-Serveradministratoren des Leibniz-Rechenzentrums entwickelt, die ihnen die richtige Konfiguration der Server vermitteln soll.

Die Webanwendung informiert über die Nutzung von Administrationswerkzeugen wie SSH und TCP Wrapper. Dabei stellt sie den Nutzern auch Aufgaben und ermöglicht es so, die erlernten Fähigkeiten praktisch einzusetzen. Außerdem setzt sie Spieldesignelemente wie Badges, Fortschrittsbalken oder Leaderboards ein, um die Nutzer zu motivieren und den Lernstoff interessanter zu machen.

Die Evaluation der Anwendung mit Hilfe einer Umfrage unter Testpersonen führt zu dem Schluss, dass Gamification-Elemente auf die Wissensvermittlung in der Serveradministration übertragbar sind. Da es sich dabei allerdings um ein sehr umfangreiches Themengebiet handelt, kann eine Lernanwendung das benötigte Wissen nicht vollständig vermitteln. Sie kann aber als Begleitung für Fachbücher oder ähnliche Informationsquellen genutzt werden, um den Lernstoff zu veranschaulichen.

Inhaltsverzeichnis

1. Einleitung

1.1. Motivation

Massive Open Online Courses wie *Codeacademy* [cod 15] gewinnen in der Wissensvermittlung in den letzten Jahren zunehmend an Bedeutung. Die Website bietet interaktive Kurse für verschiedene Programmiersprachen an, die die Mitglieder nach eigenen Interessen belegen können.
Das Ziel der Plattform ist es, die Bildung zu reformieren und interessanter zu machen. Dazu orientiert sie sich nach eigener Aussage auch am sozialen Netzwerk *Facebook* und *Zynga*, einem Unternehmen, das in Facebook integrierte Browserspiele entwickelt [abo 15]. Diese Integration von sozialen und spielerischen Aspekten scheint *Codeacademy* auch zu gelingen, im Dezember 2015 hat die Seite mehr als 25 Millionen Nutzer weltweit [cod 15].

Codeacademy benutzt Gamification, um seine Mitglieder an sich zu binden und ihnen zu helfen, ihre Lernziele zu erreichen. Beispielsweise wird jede Lektion in kleine, einfach zu lösende Einzelaufgaben unterteilt, auf deren Bearbeitung sofortiges Feedback folgt. Außerdem können sich die Nutzer in einer Gemeinschaft mit ähnlichen Zielen austauschen, Wissen weitergeben und Kontakte knüpfen.

Auch an bayerischen Schulen wird das Erlernen von Programmiersprachen in den Unterricht integriert. Dabei soll den Schülern die Programmierumgebung *Robot Karol* [KrFr 13] helfen. Diese bringt eine eigene Programmiersprache namens *Karol* mit, die bewusst einfach gehalten und auf einen kleinen Sprachumfang begrenzt ist.
Der Roboter, der auf einer Idee von Richard E. Pattis [Patt 81] basiert, kann sich in einer einfachen Bildschirmwelt bewegen und programmiert werden, um bestimmte Aufgaben zu erledigen. Ein Beispiel dafür ist das Bauen eines Hauses mit Ziegelsteinen, die der Roboter ablegen kann. Dadurch wird die Ausführung einer eingegebenen Befehlsfolge sichtbar, so dass die Schüler lernen, ein Problem mit einem bestimmten Befehlssatz strukturiert zu lösen.

Damit bewegt sich *Robot Karol* auf einer abstrakteren Ebene als *Codeacademy*, da es die Grundkonzepte der imperativen Programmierung mithilfe einer speziell für diesen Zweck entwickelten Programmiersprache lehrt.

Wie bei *Codeacademy* bekommen die Schüler auch bei *Robot Karol* sofortiges Feedback, ob sie die Aufgabenstellung richtig gelöst haben, wenn der Roboter die eingegebene Befehlsfolge ausführt. Dabei können sie ihre eigenen Lösungsstrategien ausprobieren und aus ihren Fehlern lernen. Obwohl der Lehrer bestimmte Aufgaben wie den Bau eines Hauses vorgeben kann, haben die Schüler durch die offene Gestaltung des Spiels die Möglichkeit, selbst kreativ zu werden und sich eigene Ziele zu setzen.

Beide Anwendungen integrieren Gamification-Elemente in den Bildungskontext, um die Motivation und damit den Lernerfolg ihrer Nutzer zu steigern. Deterding et al. definieren *Gamification* als

„the use of game design elements in non-game contexts “[DDKN 11a, S. 1],

also die Anwendung von Elementen des Spieldesigns in Kontexten, die nicht mit Spielen zusammenhängen. Das können Fortschrittsbalken und das Vergeben von Badges für erfolgreich abgeschlossene Lektionen sein, aber auch abstraktere Konzepte wie schnelles Feedback oder das Aufteilen von Lernprozessen in kurze, klar definierte Aufgaben. Dazu kommen soziale Faktoren wie die Position in einem Leaderboard als Statussymbol oder das gegenseitige Feedback in einer Community [ZiCu 11].

Der Einsatz von Spielen und Spielelementen in der Bildung eröffnet einen neuen Lernstil, der die Struktur der klassischen formalen Instruktion ignoriert. Er erlaubt, Lösungen auszuprobieren und aus eigenen Fehlern zu lernen und sich Wissen *just in time* anzueignen, also kurz bevor es benötigt wird. Außerdem wird das Lernmaterial nicht fest vorgegeben, sondern kann auch von anderen Lernenden kommen und erlaubt Eigeninitiative [Beck 04].

1.2. Ziel der Arbeit

Trotz des großen Erfolges in der Programmierung gibt es in der Informatik Bereiche, in denen Gamification bisher kaum genutzt worden ist. Ein Beispiel für einen solchen Bereich ist die Systemadministration, in der Aufgaben wie die sichere Konfiguration eines Servers viel Fachwissen und Erfahrung erfordern. Können gamifizierte Anwendungen auch die hier notwendigen tiefergehenden Fähigkeiten vermitteln?

Ist es möglich für einzelne Bereiche der Informationssicherheit wie Accountverwaltung, Updatemanagement oder Firewallkonfiguration gezielt Wissen zu vermitteln und so die Konfiguration von Servern in diesem Bereich signifikant zu verbessern?

Das Ziel dieser Arbeit ist die Evaluation verschiedener Gamification-Ansätze und die Übertragung dieser Ansätze auf die Serveradministration. Als Praxisbeispiel dient dabei das Leibniz-Rechenzentrum der Bayerischen Akademie für Wissenschaften (LRZ) [LR b] in Garching bei München. Das LRZ stellt unter anderem die Kommunikationsstruktur sowie E-Mail- und Webdienste für die Münchner Hochschulen zur Verfügung und nutzt dafür Linux-Server [LR a].

In dieser Arbeit wird eine gamifizierte Anwendung entwickelt, die den Systemadministratoren des LRZ die Konfiguration der Server gemäß den Anforderungen der Informationssicherheit vermitteln soll.

1.3. Struktur der Arbeit

Im Folgenden wird die Struktur der Arbeit geschildert.

In Kapitel 2 werden die theoretischen Grundlagen behandelt. Der Begriff Gamification und ähnliche Konzepte wie Serious Games, Playful Design und Edutainment werden zunächst näher definiert. Darauf folgt ein Abschnitt über verschiedene Motivationstheorien, um zu erklären, warum Menschen gerne spielen und wie gamifizierte Anwendungen diesen Spieldrang für sich nutzen. Außerdem werden die Spieldesignkonzepte aufgezeigt, die eingesetzt werden um eine Anwendung zu gamifizieren. Diese sind auf verschiedenen Abstraktionsstufen zu finden. Danach werden die verschiedenen Anwendungsbereiche von Gamification, die außer der Bildung auch die Bereiche Militär, Gesundheit, Marketing und Unternehmensführung einschließen, dargestellt. Auch auf kritische Meinungen, die in der Gamification auch Gefahren sehen, wird eingegangen. Abschließend werden einige Anwendungsbeispiele, wie Codeacademy, Robot Karol und ein Tower Defense Spiel, das Schülern die Grundkonzepte der Softwarewartung beibringen soll [RRBF 11], behandelt.

Im darauf folgenden Abschnitt werden die Grundlagen der Serveradministration, die für die Administratoren des LRZ von besonderer Bedeutung sind, erläutert. Dabei werden allgemeine Administrationswerkzeuge wie Cronjobs zur Automatisierung wiederkehrender Aufgaben, Secure Shell (SSH) zum Arbeiten an entfernten Maschinen und Network File System (NFS) zum Einhängen entfernter Verzeichnisse in den Dateibaum erklärt. Ein weiterer Schwerpunkt liegt auf der Absicherung von Servern gegen Angriffe. Dazu werden unter anderem TCP Wrapper zur Beschränkung des Zugriffs auf bestimmte Serverdienste und Firewalls zur Zugriffsbeschränkung auf ganze Systeme behandelt. Die auf die Absicherung von Webservern spezialisierten Web Application Firewalls stellen dabei einen Sonderfall dar. Zur Erkennung und Behandlung von Sicherheitsvorfällen dienen Intrusion Detection Systems (IDS) und Intrusion Prevention Systems (IPS), die anhand der Beispiele OSSEC und AppArmor erläutert werden.

Danach wird in Kapitel 3 zuerst die Systemadministration des LRZ, für die die Anwendung entwickelt wird, und die dort verwendete spezifische Serverkonfiguration vorgestellt. Aus den Herausforderungen in diesem Kontext wie der hohen Mitarbeiterfluktuation ergeben sich die Anforderungen an die Anwendung. Diese werden unterteilt in gamificationspezifische, inhaltliche, funktionale und nichtfunktionale Anforderungen.

Diese Anforderungen werden in Kapitel 4 im Design der Anwendung umgesetzt. Dabei werden die verwendeten Technologien geschildert. Außerdem wird auf die eingesetzten Spieldesignprinzipien, den zu vermittelnden Lerninhalt und das Design, das sich aus den funktionalen und nichtfunktionalen Anforderungen aus dem vorhergehenden Kapitel ergibt, eingegangen. Zudem wird das Oberflächendesign thematisiert.

Darauf folgt in Kapitel 5 die Implementierung der Lernanwendung. Diese orientiert sich am MVC-Modell, wird also in Model, View und Controller unterteilt. Dabei wird auf das Datenmodell (Model) und die Präsentation (View) näher eingegangen.

In Kapitel 6 wird die Anwendung evaluiert. Zuerst wird geprüft, ob die Anforderungen aus Kapitel 3 umgesetzt worden sind. Danach wird mit Hilfe einer Umfrage unter einigen Testpersonen festgestellt, ob die Anwendung den Lerninhalt vermitteln kann und benutzerfreundlich ist.

Abschließend werden in Kapitel 7 die wichtigsten Ergebnisse der Arbeit zusammengefasst.

2. Grundlagen

Dieses Kapitel definiert den Begriff Gamification und behandelt verschiedene Theorien zur Spielermotivation, die erklären, warum Spiele als anregend empfunden werden. Außerdem werden einige Designprinzipien für gamifizierte Anwendungen erläutert. Zusätzlich werden verschiedene Einsatzbereiche von Gamification und auch Kritikpunkte daran beschrieben. Der Abschnitt schließt mit mehren Anwendungsbeispielen in der Informatik.
Im zweiten Teil werden einige Grundlagen der Linux-Serveradministration erläutert, die mit Hilfe von Gamification vermittelt werden sollen. Dazu werden zu Beginn einige Administratorwerkzeuge wie Cronjobs, Secure Shell oder das Network File System vorgestellt. Anschließend richtet sich der Fokus auf die Absicherung von Linux-Servern, die für größere Organisationen wie das LRZ, die potenzielle Angriffsziele darstellen, besonders von Bedeutung sind. Dazu gehören die Bibliothek TCP Wrapper, aber auch Firewalls, bei denen besonders auf Web Application Firewalls eingegangen wird, die zum Schutz von Webservern ausgelegt sind. Abschließend werden Intrusion Detection Systems am Beispiel *OSSEC* und Intrusion Prevention Systems anhand von *AppArmor* behandelt.

2.1. Gamification

Im folgenden Abschnitt wird der Begriff Gamification und ähnliche Konzepte, wie *Playful Design*, *Serious Games* und *Edutainment*, definiert. Außerdem werden verschiedene Theorien zur Spielermotivation vorgestellt und die Prinzipien, die beim Design von gamifizierten Anwendungen zum Einsatz kommen, erläutert. Danach wird auf die verschiedenen Anwendungsbereiche von Gamification eingegangen sowie kritische Meinungen zum Thema Gamification dargestellt. Abschließend werden einige Beispiele für Gamification in der Informatik analysiert.

2.1.1. Definition

Es existieren unterschiedliche Definitionen für den Begriff Gamification, die gebräuchlichste stammt aber von Deterding et al. [DDKN 11a, DDKN 11b]. Sie definieren *Gamification* als

> „the use of game design elements in non-game contexts “[DDKN 11a, S. 1],

also die Anwendung von Elementen des Spieldesigns in Kontexten, die nichts mit Spielen zu tun haben.

Game wird hierbei in Kontrast zu *Play* gesetzt, wobei *Game* ein Spiel mit festen Zielen und Regeln meint, während *Play* eine breitere Überkategorie ist, die auch das freie, improvisierte Spielen beinhaltet (vgl. Abb. 2.1). Dabei wird auch Jane McGonigals [McGo 11] Definition von *Gamefulness* in Kontrast zu *Playfulness* übernommen. Während *Playfulness*

> „the experiential and behavioral qualities of playing "[DDKN 11a, S. 3]

bezeichnet, also das Erleben und Verhalten beim Spielen im Sinne von Playing, bezieht sich *Gamefulness* auf das Erleben und Verhalten beim Gaming.

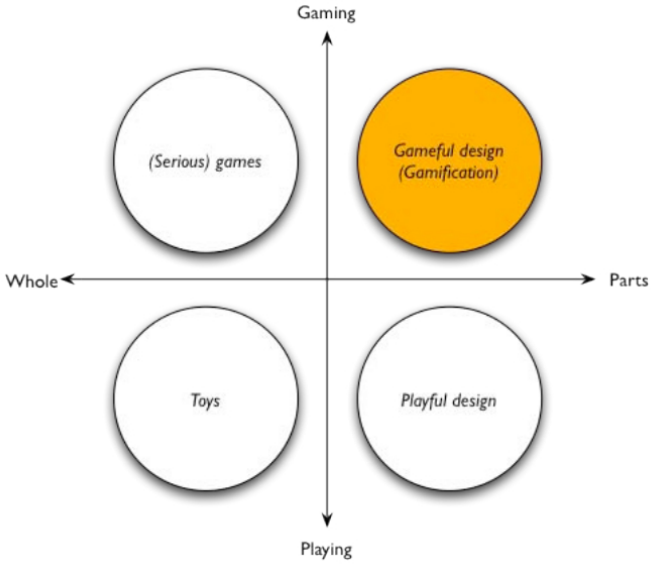

Abbildung 2.1.: Abgrenzung von Gamification zu Serious Games und Playful Design nach Deterding [DDKN 11a, S. 5]

Deterding et al. betonen, dass der Begriff Gamification nicht auf digitale Medien beschränkt werden sollte, auch wenn die meisten Beispiele digital sind, da Spiele selbst transmedial sind [DDKN 11a].
Dadurch, das die Definition alle Kontexte einschließt, die nicht direkt mit Spielen zusammenhängen, wird Gamification auch explizit nicht auf bestimmte Anwendungsbereiche eingeschränkt. Die verschieden Anwendungsgebiete der Gamification erläutert Abschnitt 2.1.4.

Gamification bezieht sich nach dieser Definition allerdings nur auf Designelemente, also beispielsweise nicht auf technische Elemente wie Controller, die in der Mensch-Maschine-Interaktion schon oft als Eingabegeräte außerhalb von Spielen umfunktioniert wurden [DDKN 11a]. Mit diesen Spieldesignelementen, die verschiedene Grade der Abstraktion haben können, beschäftigt sich Abschnitt 2.1.3.

Außerdem grenzen Deterding et al. den Begriff *Gamification* gegenüber *Serious Games* ab. *Serious Games* sind demnach

> „full-fledged games for non-entertainment purposes "[DDKN 11a, S. 3],

also vollständige Spiele, die nicht nur die Unterhaltung des Spielers zum Ziel haben. *Gamification* wird dagegen auf Anwendungen eingeschränkt, die nur einzelne Spielelemente

benutzen, aber kein komplettes Spiel sind (vgl. Abb. 2.1).
Allerdings ist der Übergang fließend, genauso wie *Spielelemente* nicht unbedingt eindeutig identifizierbar sind. Nach Deterding et al. sind das Elemente, die charakteristisch für Spiele sind, also in vielen, aber nicht unbedingt in allen Spielen vorkommen [DDKN 11a].

Ritterfeld et al. dagegen definieren *Serious Games* als

> „any form of interactive computer-based game software for one or multiple players to be used on any platform and that has been developed with the intention to be more than entertainment“[RCV 09, S. 6],

also als jegliche Form von interaktiver Spielsoftware für einen oder mehrere Spieler auf einer beliebigen Plattform, die mehr als nur die Unterhaltung des Spielers zum Ziel hat. Damit beschränken sie den Begriff zwar nicht auf eine bestimmte Plattform, aber auf den digitalen Bereich.

Dennis Charsky [Char 10] unterscheidet zudem zwischen *Serious Games* und *Edutainment*. Er sieht *Serious Games* als Weiterentwicklung von *Edutainment* (vgl. Kapitel 2.1.4). *Edutainment* habe den schlechten Ruf, Übungen und Drill nur als Spiele zu verkleiden und Unterhaltung als reine Belohnung für das Erfüllen von Aufgaben zu missbrauchen. Der Spieler kommt zum Beispiel in der Handlung nur voran, wenn er vorher eine Übung macht. Dadurch kann Edutainment zwar reines Faktenwissen vermitteln, aber kein tiefergehendes Wissen.

Serious Games dagegen gewähren dem Spieler mehr Freiheiten, wie die Möglichkeit, eigene Ziele zu definieren. Sie verbinden außerdem Lernen und Unterhaltung, so dass der Spieler besser motiviert wird und das Gelernte leichter übertragen kann. Dadurch können diese Spiele auch höhere Fähigkeiten implizit anhand von Beispielen oder durch eigene Fehler vermitteln. Trotzdem hat Edutainment seine Berechtigung, falls nur Faktenwissen vermittelt werden soll [Char 10].

Charsky unterscheidet auch zwischen *Spielen* und *Simulationen*. Während viele *Spiele* Simulationen enthalten, fehlen den meisten *Simulationen* bestimmte Spielelemente wie *Fantasy* (vgl. Kapitel 2.1.2). Meist soll der Spieler kontextbezogene Trainings absolvieren, wie zum Beispiel in einem Flugsimulator. Trotzdem sind Flugsimulatoren durch das Hinzufügen dieser Spielelemente in Spiele umgewandelt worden, die Konzepte sind also sehr ähnlich [Char 10].

In dieser Arbeit wird *Gamification* als Überbegriff, der auch ähnliche Begriffe wie *Serious Games* einschließt, behandelt. Gamification umfasst also sowohl Anwendungen, die nur Spielelemente nutzen, als auch vollständige Spiele, die mehr zum Ziel haben als Unterhaltung.
Außerdem werden beide Konzepte wie bei Deterding et al. nicht auf den digitalen Bereich beschränkt, sondern sind transmedial. Allerdings ist Gamification ein Synonym für *Gameful Design* und bezieht sich damit nur auf das Erleben und Verhalten der Spieler bei Spielen mit festen Zielen und Regeln. *Playful Design* schließt dagegen jede Art von Spielen, also auch das freie, improvisierte Spielen ein.
Auch *Edutainment* fällt unter diesen Begriff, da es ebenfalls Spielelemente zur Motivation seiner Nutzer einsetzt. Allerdings ist Edutainment besser geeignet, reines Faktenwissen zu lehren, während Serious Games auch komplexere Fähigkeiten vermitteln können.

Simulationen und *Spiele* sind sehr ähnliche Konzepte, da sowohl Spiele Simulationen enthalten können als auch Simulationen zu Spielen erweitert werden. Allerdings fehlen Simulationen bestimmte Spielelemente wie Fantasy.

2.1.2. Motivationstheorie

Der folgende Abschnitt beschäftigt sich mit verschiedenen Ansätzen der Motivationstheorie, um zu verstehen, wie Spiele Menschen zu bestimmten Verhaltensweisen motivieren und wie Gamification diese Motivation für andere Bereiche nutzen kann.

Flow-Theorie

Mihaly Csikszentmihalyi [CsCs 91] bezeichnet den Zustand, den ein Spieler optimalerweise erreichen soll, als *Flow*. Dabei muss der Spieler genau so stark gefordert werden, dass er werder über- noch unterfordert ist, sondern die Aufgabe mit seinen Fähigkeiten gerade noch meistern kann. Er befindet sich also in einem Zustand zwischen Angst und Langeweile, in dem er die Zeit vergisst und optimal motiviert ist [ZiCu 11] (vgl. Abb. 2.2). BLAH.

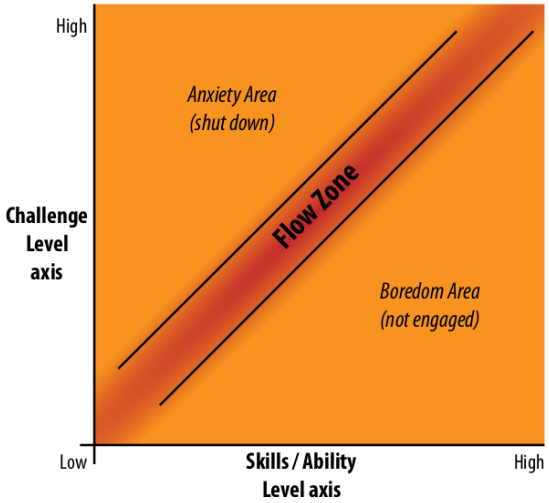

Abbildung 2.2.: Der optimale Zustand zwischen Angst und Langeweile wird als Flow bezeichnet. [ZiCu 11, S. 18]

Eine wichtige Vorraussetzung, um diesen Zustand zu erreichen, ist demnach eine Aktivität, die den Spieler herausfordert, für die er aber auch die passenden Fähigkeiten hat. Das Ziel muss sein, diese Fähigkeiten zu perfektionieren, Konkurrenzdenken sollte dagegen nur eine untergeordnete Rolle spielen. Die Ziele sollten klar definiert sein und der Spieler sollte Feedback erhalten, um das Gefühl zu bekommen, etwas erreicht zu haben. Außerdem sollte er das Gefähl haben, die Aktivität kontrollieren zu können.
Idealerweise konzentriert sich der Spieler dann vollständig auf seine Aufgabe, verliert das Zeitgefühl und geht in seiner Tätigkeit auf [CsCs 91].

Intrinsische vs. extrinsische Motivation

Motivation kann grundsätzlich intrinsisch oder extrinsisch sein. *Intrinsische Motivation* kommt von innen heraus, man führt also eine Handlung aus, weil sie Spaß macht oder einen interessiert. *Extrinsische Motivation* wird dagegen von äußeren Reizen ausgelöst, z.B. durch Belohnungen wie Geld und sozialen Status, oder Bestrafung [ZiCu 11].

Deci und Ryan [DeRy 11] definieren in ihrer *Self-Determination Theory* drei wichtige Faktoren für intrinsische Motivation. *Relatedness* beschreibt das Bedürfnis nach sozialer Interaktion und Zugehörigkeit, *Competence* das Bedürfnis, eine Aufgabe zu meistern und *Autonomy* das Bedürfnis nach Selbstbestimmung.

Wenn intrinsische Motivation durch extrinsische Anreize verdrängt wird, tritt der sogenannte *Korrumpierungseffekt* (engl. *Overjustification Effect*) auf. Alfie Kohn [Kohn 99] zeigt, dass Kinder, die für das Zeichnen von Bildern bezahlt werden, zwar mehr Bilder, aber in schlechterer Qualität als vorher zeichnen. Wenn die Belohnung abgesetzt wird, zeichnen sie nicht mehr so gerne wie vorher, durch die extrinsische Belohnung haben die Kinder also ihre intrinsische Motivation verloren.

Nach Daniel H. Pink [Pink 11] motiviert Geld als Belohnung zwar bei einfachen Aufgaben, bei komplexen Herausforderungen, bei denen kreatives Denken erforderlich ist, sinkt aber die Leistung ab. Folglich ist Geld ungeeignet, um kreatives Denken hervorzurufen. Allerdings sind andere extrinsische Belohnungen wie dauerhafter sozialer Status durchaus erfolgreich, um Kreativität und Spiel zu fördern [ZiCu 11].

Trotz der möglichen negativen Auswirkungen von extrinsischen Belohnungen betonen Zicherman und Cunningham [ZiCu 11], dass extrinsische Belohnungen nicht immer schlecht sind. Wenn sie gut in ein Spiel integriert werden, können sie dem Spieler als Einstiegshilfe dienen, um Spaß an dem Spiel zu entwickeln und damit seine intrinsische Motivation zu entdecken.
Manchmal braucht der Spieler auch einen zusätzlichen Anreiz zu seiner intrinsischen Motivation. Zum Beispiel will der Nutzer einer Anwendung, die beim Abnehmen helfen soll, meist von sich aus sein Gewicht reduzieren, es fällt ihm aber schwer, dauerhaft motiviert zu bleiben. Ein gamifiziertes System kann ihn dabei mit Belohnungen für seine Erfolge unterstützen.
Außerdem hat extrinsische Motivation den praktischen Vorteil, dass sie im Gegensatz zur intrinsischen Motivation vom Spieldesigner beeinflussbar ist.

Intrinsische Motivation, bei der die betreffende Person aus Interesse oder aus Spaß handelt, entsteht demnach also durch das Bedürfnis nach sozialer Interaktion, Selbstbestimmung oder danach, eine Aufgabe zu meistern. Sie ist zur Motivation von komplexen und kreativen Leistungen besser geeignet als extrinsische Motivation, die von äußeren Reizen ausgelöst wird. Dafür ist extrinsische Motivation im Gegensatz zur intrinsischen durch den Spieldesigner beeinflussbar. Der Korrumpierungseffekt kann aber dazu führen, dass die intrinsische Motivation des Nutzers durch die Belohnung verloren geht, was vor allem durch Belohnungen in Form von Geld begünstigt wird.

Malones Theorie der intrinsisch motivierenden Instruktion

Mit dem Konzept der intrinsischen Motivation beschäftigt sich auch Thomas W. Malone [Malo 81]. Er ist wie Kohn und Pink der Meinung, dass intrinsische Motivation vor allem bei komplexeren Aufgaben zu einem besseren Lernerfolg führt. Um ein Serious Game so zu gestalten, dass der Spieler intrinsisch motiviert ist, sind nach Malone einige Faktoren zu beachten.

Challenge beschreibt die Herausforderung, die die Aufgaben in einem Spiel darstellen. Ähnlich wie bei der Flow-Theorie soll diese Herausforderung den Fähigkeiten des Spielers angemessen sein, also weder zu schwierig noch zu einfach gestaltet werden. Auch hier spielen die Konzentration auf eine Aufgabe und Feedback eine wichtige Rolle.

Mit *Curiosity*, also Neugier, wird der intrinsische Drang, eine neue Umgebung zu erforschen und überrascht zu werden, beschrieben.

Fantasy macht ein Spiel interessanter und aufregender. Nach Malone und Lepper [MaLe 87] wird *Exogenous Fantasy* als Belohnung eingesetzt, zum Beispiel erfährt der Spieler mehr über die Handlung des Spiels, wenn er eine Aufgabe gelöst hat. *Endogenous Fantasy* ist dagegen mit der Aufgabe verbunden, es gibt also keine klare Unterscheidung zwischen Aufgabe und Spielhandlung.
Charsky [Char 10] greift diesen Aspekt in seiner Unterscheidung zwischen *Edutainment* und *Serious Games* auf (vgl. Kapitel 2.1.1). Während *Edutainment* vorwiegend Exogenous Fantasy nutzt, um seine Spieler zu belohnen, verbinden *Serious Games* Fantasy und Challenge mit Endogenous Fantasy. Dadurch fällt es dem Spieler leichter, Gelerntes zu behalten und zu übertragen.

Malone definiert außerdem *Choice* als weiteres Mittel, um Spieler intrinsisch zu motivieren. Die Wahl, eigene Ziele festzulegen, den Schwierigkeitsgrad anzupassen, oder zu bestimmen, welches Level zuerst gespielt wird, gibt dem Spieler Kontrolle über sein Handeln. Dabei lassen sich Parallelen zum Grundbedürfnis der Selbstbestimmung in der Self-Determination Theory von Deci und Ryan (vlg. Kapitel 2.1.2) feststellen. Werden die Aufgaben und Regeln eines Spiels allerdings zu frei gehalten, verliert es den Charakter eines Games und wird zum Play [Malo 81].

Um die Spieler intrinsisch zu motivieren muss ein Spiel nach Malone also den Spieler herausfordern, seine Neugier wecken und Fantasieelemente enthalten, die das Spiel interessant machen und die Übertragung des Gelernten erleichtern. Außerdem sollte der Spieler das Gefühl haben, sein Handeln selbst bestimmen zu können, zum Beispiel durch Anpassung des Schwierigkeitsgrades.

Fogg's Behavior Model for Persuasive Design

B.J. Fogg [Fogg 09] hat ein eigenes Modell entwickelt, das erklärt, wie menschliches Verhalten beeinflusst werden kann. Wie in Abbildung 2.3 zu sehen, besteht das Modell aus drei Hauptelementen: *Motivation*, *Ability* und *Triggers*.

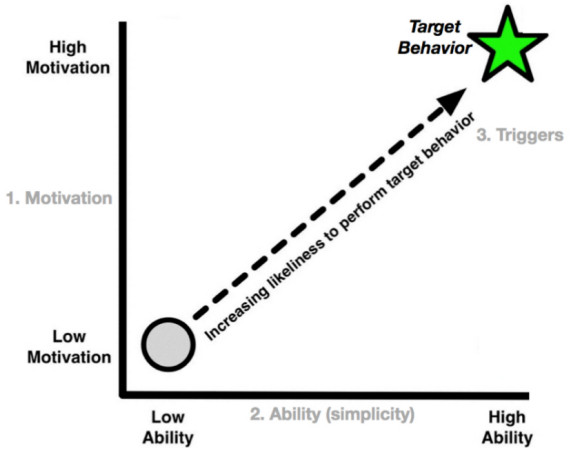

Abbildung 2.3.: Fogg's Behavior Model [Fogg 09, S. 2]

Motivation kann demnach durch drei Paare von Gegensätzen entstehen: Lust und Schmerz, Hoffung und Angst, und soziale Akzeptanz und Zurückweisung. Während Lust und Schmerz als Motivation in gamifizierten Anwendungen aus ethischen Gründen eher ungeeignet sind, wird soziale Akzeptanz häufig eingesetzt, zum Beispiel in sozialen Netzwerken. Auch Hoffnung, zum Beispiel bei Abnehmprogrammen, und Angst, wie bei abschreckenden Bildern auf Zigarettenschachteln, die vom Rauchen abhalten sollen, beeinflussen das menschliche Verhalten.

Um eine Aufgabe zu lösen, braucht ein Spieler die passenden Fähigkeiten (*Ability*). Das schließt genügend Zeit und die finanziellen Resourcen ein, um die Aufgabe zu erledigen. Die Notwendigkeit großer körperlicher oder geistiger Anstrengung oder Tätigkeiten, die nicht routiniert sind, können eine Aufgabe dagegen erschweren. Zudem sind Abweichungen von der sozialen Norm hinderlich für die Ausführung einer Aufgabe, weil sie soziale Zurückweisung zur Folge haben können. Beispielsweise würden die meisten Menschen ungern im Schlafanzug zu einem Geschäftsessen gehen.
Diese individuellen Fähigkeiten sind bei jedem Menschen unterschiedlich und hängen von der knappsten Ressource ab. Wenn also eine Handlung viel Zeit erfordert, die die betreffende Person aber nicht hat, wird sie sie auch nicht ausführen, obwohl alle anderen Fähigkeiten vorhanden sind. Je höher Motivation und Ability, desto wahrscheinlicher wird eine Handlung ausgeführt.

Zusätzlich braucht ein bestimmtes Verhalten einen *Trigger*, also einen Auslöser. Das kann ein *Spark*, ein *Facilitator* oder ein *Signal* sein.

- Besitzt die Person zwar die passenden Fähigkeiten für eine Aufgabe, ist aber nicht motiviert, kann ein *Spark* eingesetzt werden, um die Motivation zu erhöhen. Dabei können ein oder mehrere der zuvor genannten motivierenden Faktoren genutzt werden.

- Ist die Motivation dagegen hoch, aber die Ability niedrig, kann ein *Facilitator* verwendet werden, um das Verhalten zu erleichtern. Zum Beispiel werben viele Anwendungen damit, mit nur einem Klick installierbar zu sein, also weder Zeit noch Fachwissen

zu erfordern.

- Sind sowohl Motivation als auch Ability vorhanden, kann ein *Signal* ein Verhalten auslösen, indem es den Nutzer daran erinnert oder signaliert, dass dieses Verhalten im Moment angemessen ist. Zum Beispiel signalisiert eine Ampel, die auf grün schaltet, dass die Autofahrer losfahren können, motiviert diese aber nicht dazu.

Nach Fogg wird ein bestimmtes Verhalten also ausgeführt, wenn die betreffende Person motiviert ist und die nötigen Fähigkeiten und Ressourcen besitzt. Außerdem muss das Verhalten entweder durch ein Signal ausgelöst werden oder, falls die nötige Motivation oder die Fähigkeiten fehlen, die Motivation erhöht beziehungsweise die Ausführung der Handlung erleichtert werden.

Bartles Spielertypen

Richard Bartle [Bart 96] erklärt die Motivation hinter Spielen durch die Aufteilung der Spieler in vier Typen.

- *Explorers*, also Entdecker, wollen die Spielwelt erforschen und ihrer Gemeinschaft ihre Entdeckungen mitteilen.
- *Achievers*, also Leistungsmenschen, wollen dagegen Herausforderungen bestehen und sich mit anderen messen.
- Das Hauptziel von *Socializers* ist das Pflegen von sozialen Kontakten mit anderen Spielern.
- Die letzte Gruppe sind *Killers*, die andere Spieler attackieren, um über sie öffentlich zu triumphieren.

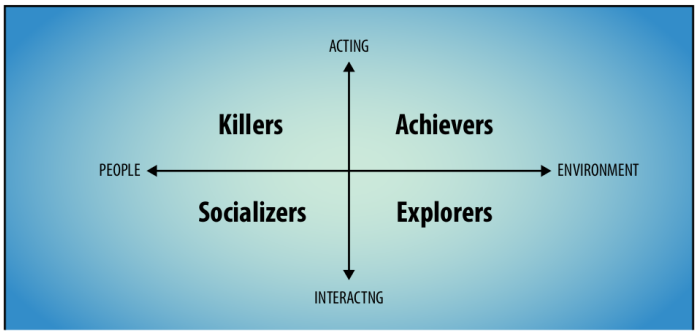

Abbildung 2.4.: Die vier Spielertypen nach Bartle [ZiCu 11, S. 22]

Wie in Abbildung 2.4 zu sehen, setzt Bartle die Spielertypen nach ihren Interessen in Bezug zueinander. Während sowohl Achiever als auch Explorer stärker an der Spielumgebung als an sozialen Interaktionen interessiert sind, wollen Achiever aktiv handeln, während Explorer lieber mit der Umwelt passiv interagieren und entdecken. Socializer interagieren, anders als Explorer, lieber mit den anderen Spielern als mit der Spielwelt. Killer sind, ebenso wie Socializer, an ihren Mitspielern interessiert, wollen sie aber besiegen statt mit ihnen zu interagieren, und ähneln in diesem Punkt den Achievern.

Diese Spielertypen schließen sich aber nicht gegenseitig aus, die meisten Spieler sind eine Mischung aus allen vier Typen, verändern sich im Lauf der Zeit und verhalten sich sogar von Spiel zu Spiel unterschiedlich. Könnte man allerdings jeden Spieler eindeutig einer Gruppe zuordnen, würden die Socializer den weitaus größten Teil der Spieler ausmachen, gefolgt von Explorern und Achievern, wobei Killer die kleinste Gruppe stellen würden. Deshalb ist es beim Spieldesign wichtig, nach Möglichkeit auch soziale Elemente wie Nachrichten oder Foren zu integrieren, um die Socializer anzusprechen [ZiCu 11].

Zusammengefasst ist für die Motivation der Spieler wichtig, dass sie weder unter- noch überfordert sind, sich also weder langweilen noch gestresst fühlen.
Für Aufgaben, die höhere kognitive Fähigkeiten erfordern, ist es außerdem wichtig, dass die Motivation intrinsisch ist, also von innen kommt. Allerdings kann dabei der Korrumpierungseffekt dazu führen, dass die vorhandene intrinsische Motivation der Nutzer durch extrinsische verdrängt wird.
Außerdem sollte ein Spiel den Spieler fordern, seine Neugier wecken und ihm erlauben, sich eigene Ziele zu setzen. Fantasieelemente machen ein Spiel interessanter und erleichtern die Übertragung des Gelernten.
Spieler sind an der sozialen Interaktion mit anderen Spielern, der Erforschung der Spielwelt und den Herausforderungen im Spiel interessiert. Manche Spieler wollen auch über andere Spieler triumphieren und greifen diese an. Der Großteil ist aber vor allem an sozialer Interaktion interessiert.

2.1.3. Spieldesign

Deterding et al. [DDKN 11a] bezeichnen mit Gamification die Benutzung von Spieldesignelementen in Kontexten, die nichts mit Spielen zu tun haben. Diese Spieldesignelemente, die in vielen, aber nicht in allen Spielen vorkommen müssen, unterteilen sie in fünf verschiedene Abstraktionsniveaus. Von konkret bis abstrakt geordnet sind das *Interface design patterns*, *Game design patterns and mechanics*, *Game design principles and heuristics*, *Game models* und *Game design methods* [DDKN 11a] (vgl. Tabelle 2.1).

Interface design patterns

Interface design patterns sind konkrete Oberflächendesignelemente, die in einem bestimmten Kontext für ein bestimmtes Problem eingesetzt werden und auch prototypische Implementierungen einschließen. Dazu gehören *Levels*, *Badges* und *Leaderboards* [DDKN 11a].

In den meisten Spielen zeigen *Levels* dem Spieler seinen Fortschritt an und geben Feedback. Da sich der Spieler normalerweise möglichst lange mit dem Spiel beschäftigen soll, ist es sinnvoll, die ersten Levels möglichst einfach zu gestalten. Mit der Erfahrung des Spielers sollte auch der Schwierigkeitsgrad steigen. Allerdings ist dieser Anstieg nicht linear, sondern bogenförmig, das heißt, dass manche Levels wesentlich schwieriger zu bewältigen sind als ihre Vorgänger. Das kann zwar dazu führen, dass einige Spieler sich überfordert fühlen und aufgeben, dafür haben die Spieler, die es schaffen, das Gefühl, ein Ziel erreicht

Abstraktionsniveau	**Beschreibung**	**Beispiel**
Game interface design patterns	Häufig genutzte Oberflächendesignkomponenten für ein bekanntes Problem in einem bekannten Kontext, einschließlich prototypische Implementierungen	Levels, Badges, Leaderboards
Game design patterns and mechanics	Häufig genutzte Designkomponenten, die den Spielablauf beeinflussen	Zeitbeschränkungen, beschränkte Ressourcen, Spielrunden
Game design principles and heuristics	Richtlinien, um ein Designproblem zu lösen oder eine gegebene Lösung zu evaluieren	Klar definierte Ziele
Game models	Konzeptuelle Modelle von Spielkomponenten oder der Spielerfahrung	MDA; CEGE; Challenge, Fantasy, Curiosity
Game design methods	Spieldesignspezifische Prozesse und Praktiken	Playcentric design, Playtesting

Tabelle 2.1.: Die Abstraktionsniveaus von Spieldesignelementen mit ausgewählten Beispielen nach Deterding et al. [DDKN 11a]

zu haben. Im Zusammenhang mit Levels werden oft *Fortschrittsbalken* genutzt, die den Fortschritt innerhalb eines Levels anzeigen [ZiCu 11].

Badges werden an Spieler für bestimmte Leistungen verteilt und sind Statussymbole. Viele Spieler sammeln sie aber auch gerne oder freuen sich einfach, in einem gamifizierten System von einem Badge überrascht zu werden. Schöne, aufwendig designte Badges können auch einen ästhetischen Wert haben oder Levels als Fortschrittsanzeige ersetzen [ZiCu 11].

Leaderboards, also Ranglisten, helfen Nutzern, ihre Leistungen untereinander zu vergleichen. Dabei wird zwischen zwei Typen unterschieden.
No-disincentive leaderboards, die oft in sozialen Netzwerken verwendet werden, sollen Spieler nicht demotivieren, falls sie weit unten in der Rangliste auftauchen. Stattdessen motivieren sie ihre Nutzer, indem sie sie immer in die Mitte der Liste setzen, also immer nur den Abstand zu den nächstbesseren und den nächstschlechteren Spielern anzeigen.
Infinite leaderboards sorgen dafür, dass kein Spieler vom Leaderboard absteigt oder zu lange ohne Änderung auf einer bestimmten Position bleibt. Sie schränken die Sicht ihrer Spieler ein und zeigen zum Beispiel nur Freunde oder Spieler in der Nähe an, was vor allem bei einer sehr großen Zahl von Spielern sinnvoll ist [ZiCu 11].

Punkte können, ähnlich wie Levels oder Badges, den Fortschritt anzeigen. Zichermann und Cunningham [ZiCu 11] betonen aber, dass Punkte nicht nur oberflächlich verwendet werden können. Auch intern kann dem Nutzer einer gamifizierten Anwendung für jede Handlung Punkte zugewiesen werden. Diese sogenannten *Experience Points* können, selbst wenn sie für den Nutzer nicht sichtbar sind, aufzeigen, wie der Spieler mit dem System interagiert und wie aktiv er ist. Für den Spieler sichtbare Experience Points werden zum Beispiel bei

Vielfliegerprogrammen vergeben, Punkte können aber auch abstrakter repräsentiert werden, wie zum Beispiel die Zahl der Follower im Nachrichtendienst Twitter. Außerdem können Punkte als virtuelle Währung oder als virtuelle Geschenke dienen [DDKN 11a].

Game design patterns and mechanics

Game design patterns sind Designelemente, die typischerweise immer wieder in einem Spiel vorkommen und das Spielgeschehen beeinflussen. Sie schließen aber keine prototypischen Implementierungen mit ein, sondern können durch verschiedene *Interface design patterns* implementiert werden. Das sind zum Beispiel Zeitbegrenzungen, beschränkte Ressourcen und Spielrunden, bei denen sich die Spieler abwechseln [DDKN 11a].

Bergström et al. [BBL 10] beschreiben *Game Design Patterns* nach Björk und Holopainen [BjHo 05] als

> „a part of the interaction possible in games “[BBL 10, S. 18],

also als ein Teil der möglichen Interaktion in Spielen.
Sie unterteilen diese Patterns nach dem *MDA Framework* (vgl. Kapitel 2.1.3 Game models) in *Mechanical Patterns*, *Dynamical Patterns* und *Aesthetical Patterns*.

- *Mechanical Patterns* sind die Komponenten eines Spiels, also Regeln und Aktionen, die der Spieler ausführen kann.
- *Dynamical Patterns* resultieren daraus, wie der Spieler mit den Mechanical Patterns interagiert.
- *Aesthetical Patterns* entstehen aus den ersten beiden Patterns und beschreiben die Erfahrung des Spielers, also was er beim Spielen fühlt.

Bergström et al. analysieren mehrere Spiele auf Patterns, die Kameradschaft begünstigen, unter anderem das Massively Multiplayer Online Roleplaying Game *World of Warcraft* [Bliz 04]. In dem in einer Fantasiewelt angesiedelten Spiel kreieren die Spieler Avatare mit bestimmten Fähigkeiten, die auf Schlachtzüge gehen oder in Dungeons in Gruppen Monster töten, um ihre Ausrüstung zu verbessern.
Die Schlachtzüge bestehen nach Bergström et al. aus den Mechanical Patterns *Player versus Environment*, also Spieler gegen Umgebung und *Mutual Enemies*, also gemeinsame Feinde. Daraus entstehen die Dynamical Patterns *Cooperation* und *Teamplay*, woraus wiederum das Aesthetical Pattern *Team Strategy Identification* zwischen den Spielern entsteht.
Das liegt auch daran, dass die Spielfiguren verschiedenen Klassen und Völkern angehören und dadurch unterschiedliche Fähigkeiten haben (*Asymetrical Abilities*). Dadurch müssen sie im Team mit diesen Fähigkeiten kompatible Rollen übernehmen (*Selectable Functional Roles*), um als Gruppe erfolgreich zu sein (*Team Combo*).

Game design heuristics and principles

Game design heuristics sind Richtlinien zur Lösung von Designproblemen, die auf Erfahrungswerten basieren und auch zur Evaluation einer Designlösung benutzt werden können

[DDKN 11a].

Noah Schaffer [Scha 07] nennt einige konkrete Heuristiken für die Usability von Spielen. Generell empfiehlt er, das Design für den Spieler intuitiv zu gestalten, zum Beispiel mit klar definierten Zielen und einfach zu bedienenden Steuerelementen.
Die grafischen Benutzeroberflächen sollten wichtige Informationen wie Punkte oder Munition persistent anzeigen, so dass der Spieler sie immer im Blick hat, ihn aber nicht durch unwichtige Informationen überfordern. Der Spielablauf sollte so gestaltet sein, dass der Spieler klar definierte Ziele hat und Hindernisse sowie Gegner leicht identifizieren kann. Steuerelemente sollten intuitiv oder nach einmaligem Lesen der Anleitung bedienbar sein, zum Beispiel indem Industriestandards übernommen werden. Die Levels sollten so konstruiert sein, dass der Spieler sich nicht verirrt oder frustriert wird, weil er an einer Stelle nicht mehr weiter kommt. Nach dem Lösen einer schwierigen Aufgabe sollte er seinen Spielstand speichern können, damit er diese nicht noch einmal erledigen muss.

Game models

Game models sind konzeptuelle Modelle von Spielkomponenten oder der Spielerfahrung, wie zum Beispiel das *MDA Framework* oder *CEGE*. Auch Malones *Challenge, Fantasy, Curiosity* (vgl. Kapitel 2.1.2) können als Spielkomponenten betrachtet werden, die ein Spiel enthalten muss, um Spaß zu machen [DDKN 11a].

Das *MDA Framework* von Hunicke et al. [HLZ 04] beschreibt ein Spiel als Interaktion von drei Komponenten: Mechanics, Dynamics und Aesthetics. Dabei werden Aesthetics von den anderen beiden Komponenten beeinflusst.

- *Mechanics* sind die Komponenten eines Spiels, also die verschiedenen Aktionen, die ein Spieler ausführen kann und die Kontrollmechanismen eines Spiels.
- *Dynamics* ist die Art und Weise, wie ein Spieler mit den Mechanics interagiert.
- *Aesthetics* beschreibt, was der Spieler dabei erlebt und warum ihm ein Spiel Spaß macht.

Beim Pokern wären Mechanics zum Besipiel mischen, Karten ziehen, und wetten. Dynamics entstehen durch die Interaktion des Spielers mit diesen Mechanics, beispielsweise indem er blufft. Aesthetics sind dabei soziale Interaktion, der Wettkampf zwischen den Spielern oder das Spiel als Herausforderung.

CEGE steht für *Core Elements of the Gaming Experience* und beschreibt nach Cálvillo-Gomez et al. [CGCC 10] die notwendigen, aber nicht hinreichenden Vorraussetzungen für eine gute Spielerfahrung, die sich sowohl auf das Spiel selbst als auch auf die Interaktion mit dem Nutzer beziehen.
Nach diesem Framework besteht ein Spiel aus dem *Game-play* und dem *Environment*. Das *Game-play* ist der Inhalt des Spiels, seine Regeln und Szenarien. Das *Environment* ist die Art, wie das Spiel dem Spieler präsentiert wird, also die tatsächliche grafische und akustische Umsetzung des Spiels. Die Interaktion des Nutzers mit dem Spiel wird als *Puppetry*

bezeichnet. Dieser Prozess wird von drei Faktoren bestimmt: *Control*, *Ownership* und *Facilitators*.

- Spieler übernimmt die Kontrolle über das Spiel, indem er die zur Verfügung stehenden Aktionen und Ereignisse des Spiels nutzt und lernt, es zu bedienen (*Control*).
- Damit bringt er es dazu, auf ihn zu reagieren und nimmt die durch ihn ausgelösten Aktionen des Spiels als seine eigenen wahr, macht sich also das Spiel zueigen (*Ownership*), was das Spiel wiederum durch Belohnungen anerkennt. Control führt also zu Ownership, woraus wiederum der Spaß am Spiel entsteht.
- *Facilitators* sind Faktoren wie vorherige Erfahrung mit ähnlichen Spielen oder ästhetische Aspekte, die diesen Prozess erleichtern.

Game design methods

Game design methods sind spieldesignspezifische Prozesse wie *Playcentric design* oder *Playtesting* [DDKN 11a].

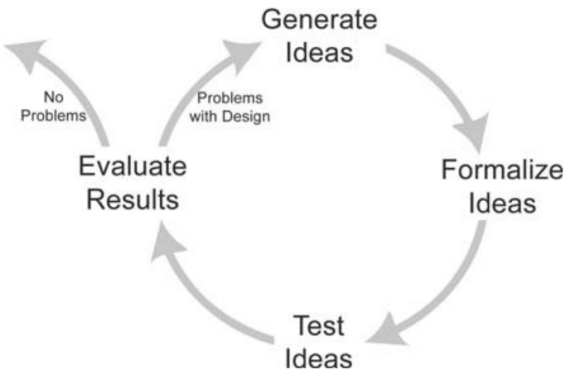

Abbildung 2.5.: Der iterative Designprozess beim Playcentric design [Full 14, S. 15]

Beim *Playcentric design* steht die Erfahrung des Spielers im Mittelpunkt des gesamten Designprozesses. Am Anfang des Prozesses werden *Player experience goals* gesetzt, es wird also definiert, welche Erfahrungen der Spieler während des Spielens haben soll. Zum Beispiel soll der Spieler die Freiheit haben, Aktivitäten in beliebiger Reihenfolge auszuführen, oder sich eher glücklich und entspannt fühlen anstatt einem Wettkampf ausgesetzt zu werden.
Außerdem sollte vor der Implementierung ein physischer Prototyp entstehen, der die wichtigsten Spielmechaniken enthält. Indem der Designer dieses Spiel ausprobiert, kann er die Reaktionen der Spieler testen und erhält erstes Feedback über die Spielererfahrungen.
Wie in Abbildung 2.5 zu sehen, ist der Designprozess iterativ, das heißt neue Ideen werden generiert, formalisiert, getestet und evaluiert. Werden bei der Evaluation Probleme erkannt, beginnt der Prozess wieder von vorne, indem neue Ideen generiert werden. Dadurch wird die Anwendung kontinuierlich verbessert [Full 14].

Damit hängt auch *Playtesting* zusammen. Beim Playtesting geht es darum, Feedback von Spielern zu bekommen, um die Spielerfahrung zu verbessern. Dabei bezieht es sich aber nur

auf die Spieldesignelemente, schließt also beispielsweise die Fehlerfreiheit der Programmierung oder die Benutzerfreundlichkeit der Oberfläche aus. Playtesting sollte in jedem iterativen Designzyklus durchgeführt werden, wobei die Tester idealweise aus der Zielgruppe des Spiels stammen sollten [Full 14].

Obwohl Deterding et al. noch weitere Beispiele für die verschiedenen Ebenen nennen, ist eine vollständige Aufzählung im Rahmen dieser Arbeit nicht möglich. Zusammengefasst unterteilen Deterding et al. Spieldesignelemente in fünf verschiedene Abtraktionsebenen.
Interface design patterns wie Levels oder Badges sind oberflächliche Designelemente, die auch prototypische Implementierungen einschließen. *Game design patterns* sind abstraktere Muster, die charakteristisch für Spiele sind und wiederum durch Interface design patterns implementiert werden könen, wie Zeitbeschränkungen oder Spielrunden. *Game design heuristics* sind Richtlinien, wie Designprobleme angegangen oder gegebene Designlösungen evaluiert werden können, zum Beispiel klar definierte Ziele. *Game models* sind konzeptuelle Modelle von Spielkomponenten oder der Spielerfahrung, wie das MDA Framework oder CEGE. *Game design methods* beschreiben Spieldesignprozesse wie Playcentric design oder Playtesting.

2.1.4. Anwendungsbereiche

Obwohl der Schwerpunkt dieser Arbeit auf dem Einsatz von Gamification in der Bildung liegt, ist die Anwendung nicht auf diesen Bereich begrenzt. Im Folgenden wird die Nutzung von Gamification im militärischen Bereich, in der Unternehmensführung, im Marketing, in der Politik, im Gesundheitswesen und in der Bildung erläutert.

Militär

Das Militär nutzt seit Jahrtausenden Spiele und Simulationen zum Training. Schach, das ursprünglich Offizieren in der Ausbildung militärische Kampfstrategien vermitteln sollte, ist wohl eines der bekanntesten Beispiele für derartige Spiele im nichtdigitalen Bereich. Im 19. Jahrhundert erfand das preußische Militär *Kriegsspiel*, das auf einer Landkarte mit Metallstreifen Truppenbewegungen abbildete.
Als in den 20er Jahren des 20. Jahrhunderts die ersten rein mechanischen Flugsimulatoren aufkamen, begann das Militär Simulationen zu nutzen, um seine Soldaten gefahrlos zu trainieren. Mit der Entwicklung der computergestützten 3D-Technik wurden diese Simulationen digitalisiert und auch bei der Rekrutierung genutzt.
America's Army [arm 02], das 2002 entwickelt wurde ist und einen möglichst realistischen Eindruck davon vermitteln soll, was einen Soldaten in der Armee erwartet, hatte 2004 vier Millionen registrierte Mitglieder. Es soll potenzielle Rekruten dazu motivieren der Armee beizutreten, aber auch ungeeignete Bewerber aussortieren und bei ausländischen Spielern amerikanische Werte bewerben [MiCh 05].

Unternehmensführung

Da der Anteil der Arbeitnehmer, die mit Computerspielen aufgewachsen sind, in den nächsten Jahren immer weiter zunehmen wird, nutzen auch Unternehmen vermehrt gamifizierte Anwendungen, um ihre Mitarbeiter zum Beispiel in Onlinekursen weiterzubilden [MiCh 05]. Es gibt aber auch Ansätze, mit gamifizierten Systemen die Mitarbeiterzufriedenheit zu erhöhen, indem die Leistungen der Mitarbeiter festgehalten und mit schnellem Feedback versehen oder Wettkampfelemente in den Arbeitsalltag integriert werden. Besonders gute Leistungen können zum Beispiel mit Leaderboards hervorgehoben oder mit besonderen Privilegien belohnt werden. Der Wissensaustausch unter den Arbeitnehmern kann mit Foren und Wikis, deren Pflege wiederum durch das System belohnt wird, gefördert werden [PrRa 15].

Marketing

Auch im Marketing werden gamifizierte Systeme eingesetzt, um Kunden zu binden. Ein Beispiel dafür sind Kundenkarten, bei denen der Kunde für seine Einkäufe mit Punkten belohnt wird, die er später gegen neue Ware einlösen kann. Dadurch wird die Stammkundschaft gepflegt, während neue Kunden mit Einstiegsrabatten zum Mitmachen motiviert werden. Dabei werden aber auch Daten über die Kunden und ihr Kaufverhalten gesammelt, mit denen das Angebot verbessert wird [ZiCu 11].

Politik

Sogenannte *Persuasive Games* sind Spiele, die informieren und eine politische, kulturelle oder religiöse Nachricht vermitteln sollen [MiCh 05]. Sie werden sogar im Wahlkampf eingesetzt, wie das Spiel *„Howard Dean for Iowa“* [BoFr 03], in dem der Spieler für den Politiker Howard Dean Wahlkampf betreibt und das seine Unterstützer dazu bringen soll, sich aktiv zu engagieren [Bogo 07].
Das Spiel *Peacemaker* [BuBr] soll Teenager über den Nahostkonflikt informieren und zeigen, wie Frieden durch Zusammenarbeit erreichbar ist. Dabei kann der Spieler zwischen einer Rolle als israelischer Premierminister oder palästinensischer Präsident wählen und versuchen, eine friedliche Lösung zu finden, bevor seine Amtszeit vorbei ist [MiCh 05].

Gesundheitswesen

Im Gesundheitswesen werden Spiele einerseits eingesetzt, um eine gesunde Lebensweise zu propagieren oder beim Abnehmen zu helfen. *Exergames* wie *Wii Fit* [Nint 08], bei dem der Nutzer sein eigenes Fitnessprogramm zusammenstellen kann, motivieren zum Sporttreiben. Serious Games trainieren aber auch Ärzte oder dienen als Therapieform. Zum Beispiel werden Simulationen, in denen ein Patient in einer sicheren Umgebung mit seinen Ängsten konfrontiert werden kann, zur Behandlung von Angststörungen eingesetzt. Ein Patient mit Höhenangst fährt darin beispielsweise in einem ersten Schritt virtuell Aufzug und wird dann nach und nach mit angsteinflößenderen Szenarien konfrontiert [MiCh 05, Mold 08].

Bildung

In der Bildung gewannen Computerspiele in den 80er Jahren als *Edutainment* an Bedeutung. Ein bekanntes Beispiel dafür ist das Spiel *„Where in the world is Carmen Sandiego?“* [Brod 85]. Der Spieler muss dabei als Detektiv die gerissene Diebin Carmen Sandiego fangen, die sich an verschiedenen Orten auf der ganzen Welt versteckt. Dabei hinterlässt sie Spuren, die der Spieler entschlüsseln muss, um sie zu finden. Diese Spuren sind echte geographische Fakten, die der Spieler erlernen soll.

Wie schon in Kapitel 2.1.1 erwähnt, sieht Dennis Charsky [Char 10] Serious Games als Weiterentwicklung von Edutainment.
Edutainment verkleidet demnach nur Übungen und Drill als Spiele und missbraucht Unterhaltung als reine Belohnung für das Erfüllen von Aufgaben. Der Spieler kommt zum Beispiel in der Handlung von *„Where in the world is Carmen Sandiego?“* nur voran, wenn er vorher ihre Spuren entschlüsselt. Dadurch kann das Spiel nach Charsky zwar reines geographisches Faktenwissen vermitteln, aber kein tiefergehendes Wissen.
Serious Games gewähren dagegen dem Spieler mehr Freiheiten, wie die Möglichkeit, eigene Ziele zu definieren, und verbinden Lernen und Unterhaltung.
Als Beispiel dafür nennt er *Virtual U* [Virt 03]. Der Spieler leitet dabei eine virtuelle Universität und kann verschiedene Variablen manipulieren, die sich auf den Zustand der Universität auswirken. Was eine gute Universität ausmacht, ob Forschungsgelder oder zufriedene Studenten am wichtigsten sind, kann jeder für sich entscheiden und seine Strategie frei ausprobieren. Durch das Lernen an Beispielen und durch eigene Fehler kann das Spiel auch höhere Fähigkeiten wie im Management erforderlich implizit vermitteln.
Trotzdem kann der Einsatz von Edutainment durchaus sinnvoll sein, wenn hauptsächlich Faktenwissen vermittelt werden soll [Char 10].

Die Anwendungsgebiete von Gamification überschneiden sich meist mit der Bildung. Das Militär trainiert damit Soldaten, Unternehmen nutzen sie zur Fortbildung ihrer Mitarbeiter, in der Politik soll sie informieren und im Gesundheitswesen wird sie zur Ausbildung von Ärzten genutzt.
Allerdings gibt es auch Bereiche, die nicht direkt damit verknüpft sind. Zum Beispiel geht es im Marketing darum, mit Gamification Kunden zu binden. Aber auch in den zuvor genannten Gebieten wird Gamification zusätzlich für andere Zwecke eingesetzt, zum Beispiel zur Erhöhung der Mitarbeiterzufriedenheit in Unternehmen oder als Therapieform in der Medizin.

2.1.5. Kritik

Trotz vieler Befürworter der Gamification, die die Erfolge und vielseitige Anwendbarkeit von Gamification herausstellen, gibt es auch kritische Meinungen.

- Im Bildungskontext zweifeln noch viele Lehrer am tatsächlichen Lerneffekt von Spielen. Diese müssen außerdem speziell geschult werden, um Spiele richtig in den Unterricht zu integrieren [MiCh 05].

- Zudem könnten gamifizierte Anwendungen bei älteren Nutzern, die nicht mit Computerspielen aufgewachsen sind, auf Ablehnung stoßen. 2014 waren allerdings 32 Prozent aller Social Gamer 55 oder älter [thi 14], was darauf schließen lässt, dass Gamification auch ältere Generationen ansprechen kann [STT 14].
- Ein weiterer wichtiger Punkt ist der sogenannte *Korrumpierungseffekt*, der in Abschnitt 2.1.2 näher beschrieben wurde. Dabei geht die intrinsische Motivation des Nutzers durch Belohnungen der gamifizierten Anwendung verloren. In einem Betrieb, der Gamification zur Mitarbeitermotivation nutzt, besteht die Gefahr, dass die Mitarbeiter den Spaß an der Arbeit verlieren und nur noch arbeiten, weil sie dafür durch das System belohnt werden [STT 14].
- Außerdem können, wenn die Belohnung in Form von Punkten oder Geld höher eingeschätzt wird als die eigentliche Arbeit, unbeabsichtigte Verhaltensweisen entstehen. Gibt es zum Beispiel Punkte für das Kommentieren eines Beitrags in einem Forum, könnte das die Benutzer dazu verleiten, möglichst viele, aber sinnlose Kommentare zu posten. Allerdings kann dieser Gefahr durch Designmaßnahmen entgegengewirkt werden, zum Beispiel mit einer täglichen Obergrenze für Kommentare [ZiCu 11].
- Vor allem bei Persuasive Games, die ja dafür gedacht sind, die Einstellung der Spieler zu einem bestimmten Thema zu beeinflussen, besteht auch die Gefahr, dass sie zur Manipulation der Spieler oder zur verdeckten Werbung missbraucht werden.
- Im Marketing werden gamifizierte Anwendungen wie Kundenkarten zum Sammeln von Kundendaten genutzt, um Angebot und Werbung besser an die Kunden anzupassen [ZiCu 11]. Den Kunden ist das aber nicht immer bewusst. Außerdem könnten diese Daten verkauft oder missbraucht werden.
- Ein ähnliches Problem besteht bei gamifizierten Anwendungen im Unternehmenskontext. Durch das Festhalten des Verhaltens der Mitarbeiter im System kann eine Firma versuchen, die Handlungen ihrer Mitarbeiter zu überwachen oder auch zu manipulieren. Zudem kann ein hoher Konkurrenzdruck unter den Mitarbeitern entstehen, die dadurch das eigentliche Ziel, die Firma voranzubringen, aus den Augen verlieren. Außerdem besteht bei zu verspielten Anwendungen die Gefahr, unprofessionell zu wirken [STT 14].
- Spielentwickler kritisieren auch, dass durch den Zwang, zum Beispiel in der Arbeit an solchen gamifizierten Systemen teilzunehmen, die Freiheit verloren geht, die Spiele eigentlich bieten. Außerdem werden oberflächliche Spielelemente wie Badges oder Levels oft eingesetzt, ohne auf die tatsächlichen Bedürfnisse der Nutzer einzugehen, wodurch der motivierende Effekt ausbleibt [STT 14, ZiCu 11].
- Zusätzlich ist die Entwicklung, aber auch der Betrieb und die Weiterentwicklung gamifizierter Systeme sehr teuer. Auch der finanzielle Aufwand für Belohnungen muss mit einkalkuliert werden [STT 14, ZiCu 11].

Trotz dieser Kritikpunkte wächst die Nachfrage nach Gamification stetig. Während die Umsätze 2012 noch bei 242 Millionen lagen, sollen sie bis 2018 5,5 Milliarden erreichen [Edwi 15]. Die Vorteile von gamifizierten Anwendungen wie die Fähigkeit, ihre Nutzer effektiv zu motivieren, scheinen also mögliche Nachteile und Gefahren zu überwiegen.

2.1.6. Anwendungsbeispiele in der Informatik

Im Folgenden werden einige Beispiele für gamifizierte Anwendungen und Serious Games in der Informatik erläutert. Die Plattform *Codeacademy* und die Programmierumgebung *Robot Karol* sind Anwendungen, die ihre Nutzer beim Erlernen von Programmiersprachen unterstützen. Außerdem wird ein Tower Defense Spiel, das Schülern und Studenten die Grundlagen der Softwarewartung beibringen soll, vorgestellt.

Codeacademy

Codeacademy [cod 15] ist ein Massive Open Online Course zum Erlernen von Programmiersprachen. Die Nutzer können interaktive Kurse für gängige Programmiersprachen wie Java belegen oder lernen, wie man eine Webseite entwickelt. Die Anmeldung ist kostenlos, es werden aber kostenpflichtige Zusatzangebote wie persönliche Beratung zur Kursauswahl oder zusätzliche Übungsprojekte angeboten.
Nach eigener Aussage ist es das Ziel der Plattform, die Bildung zu reformieren und interessanter zu machen [abo 15]. Als Vorbilder dienen dabei auch das soziale Netzwerk *Facebook* und *Zynga*, ein Unternehmen, das in Facebook integrierte Browserspiele entwickelt. Die Seite versucht also, soziale Aspekte und Spielelemente in ihr Angebot zu integrieren.
Codeacademy wurde 2011 gegründet und hat im Dezember 2015 mehr als 25 Millionen Nutzer weltweit [cod 15], es scheint mit diesem Konzept also sehr erfolgreich zu sein.

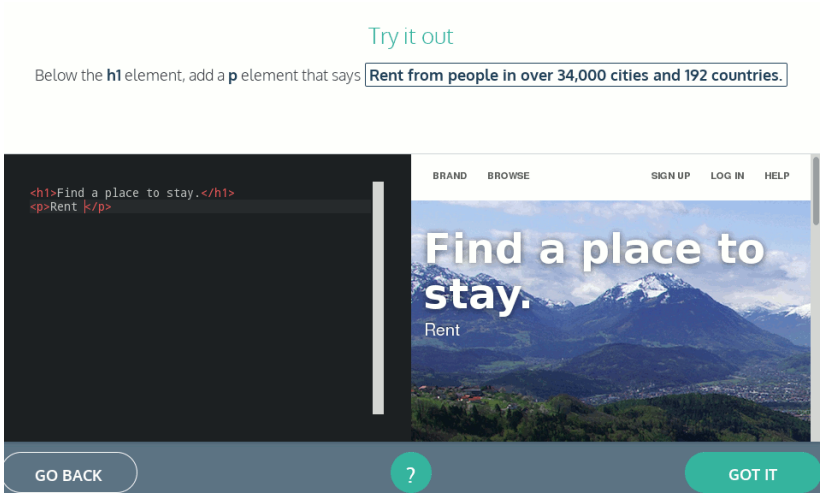

Abbildung 2.6.: Der HTML-Kurs auf codeacademy.com

Um ihre Nutzer an sich zu binden und sie zu motivieren, ihre selbst gesteckten Ziele zu erreichen, nutzt die Plattform Gamification. Die Teilnehmer können selbst entscheiden, welche Kurse sie belegen und ob und wann sie sie abschließen, Codeacademy ermöglicht ihnen also einen hohen Grad an Selbstbestimmung. Das eigentliche Ziel der Kursteilnehmer ist der Erwerb neuer Fähigkeiten, ihre Motivation ist also intrinsisch. Die Mitglieder können sich in einem Forum mit Menschen, die ähnliche Interessen und Ziele haben, austauschen und soziales Feedback erhalten. Dadurch werden auch soziale Faktoren eingebunden.

Die Aufgaben, die ein Teilnehmer abschließen muss, um sein Ziel zu erreichen, werden mit Punkten bewertet, so dass der Nutzer ein Gefühl für seine Fortschritte bekommt. Dieser Effekt wird durch weitere Spielememente in der Bedienoberfläche wie Fortschrittsbalken und Badges für erreichte Kursziele verstärkt. Die Kurse sind in kurze Aufgaben aufgeteilt, bei denen der Nutzer sein Wissen anwenden kann und nach deren Bearbeitung er sofortiges Feedback erhält [STT 14].

Robot Karol

An bayerischen Schulen soll die Programmierumgebung *Robot Karol* [KrFr 13] helfen, den Schülern die Grundkonzepte der imperativen Programmierung zu vermitteln. Diese basiert auf einer Idee von Richard E. Pattis [Patt 81] und bringt eine einfache Programmiersprache namens *Karol* mit. Der Funktionsumfang dieser Sprache umfasst Kontrollstrukturen, wie Verzweigungen und Schleifen, und Steuerbefehle für einen Roboter. Er kann sich zum Beispiel nach rechts oder links drehen, einen Schritt nach vorne gehen oder einen Ziegelstein ablegen. Damit kann der Roboter durch eine virtuelle Welt gesteuert werden, um Aufgaben wie das Bauen eines Hauses mit Ziegelsteinen zu erledigen.

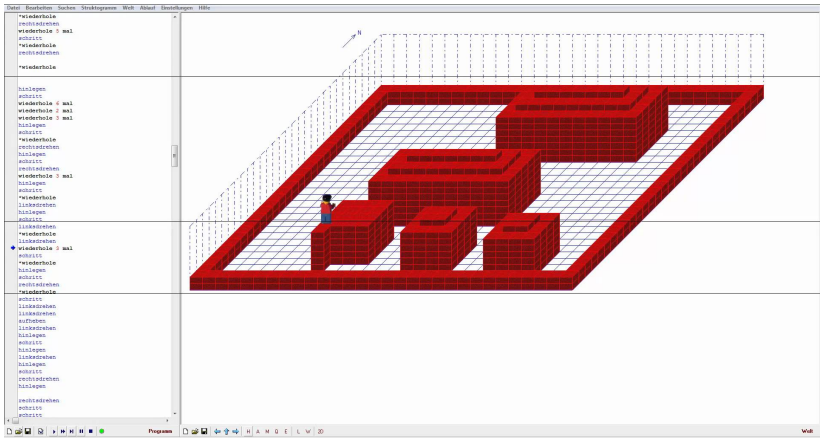

Abbildung 2.7.: Eine Stadt, gebaut mit Robot Karol [rob 11]

Robot Karol macht die Ausführung eines Befehlssatzes sichtbar, so dass die Schüler sofortiges Feedback bekommen, ob sie die Aufgabe richtig gelöst haben. Das Programm erlaubt es den Schülern, ihr Wissen praktisch anzuwenden und während des Spielens durch Erfahrung zu lernen. Dadurch lernen die Schüler, ein Problem mit einem bestimmten Befehlssatz strukturiert zu lösen.
Die Programmierumgebung ist so konzipiert, dass sie keine Aufgaben oder Levels mitbringt, sondern die Aufgaben vom Lehrer gestellt werden. Allerdings können die Schüler durch dieses Spielkonzept auch eigene Ideen umsetzen, auch hier erlaubt das Programm also eine gewisse Autonomie.

Ein ähnliches Konzept verfolgt das Spiel *Lightbot* [lig 16]. Es ist als kostenloses Flashgame oder als App erhältlich und nutzt ebenfalls einen Roboter in einer virtuellen Welt, um Schülern die Grundlagen der Programmierung beizubringen. Im Gegensatz zu Robot Karol

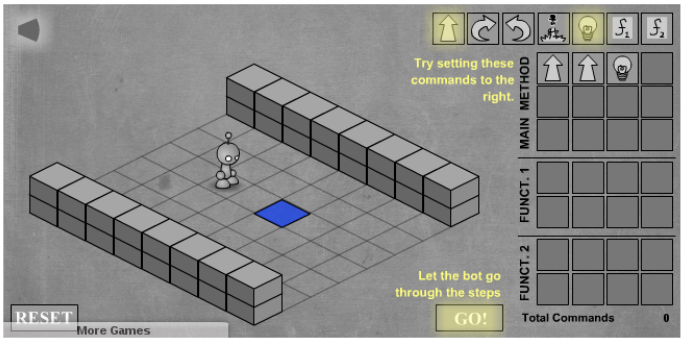

Abbildung 2.8.: Die Flash-Variante von Lightbot auf armorgames.com

ist Lightbot in Levels aufgeteilt, in denen der Spieler den Roboter durch die jeweilige Spielwelt navigieren und zu einer Kachel führen muss, die der Roboter dann beleuchtet. Damit kann das Spiel auch ohne Anleitung durch einen Lehrer gespielt werden.
Lightbot lässt den Spieler die als Grafiken dargestellten Befehle per Drag and Drop zusammenfügen. Der Spieler kann aber in jedem Level nur eine begrenzte Zahl von Befehlen einsetzen, die zur Verfügung stehenden Ressourcen sind also begrenzt.

Tower Defense Spiel zum Erlernen von Softwarewartung

Rusu et al. [RRBF 11] haben ein Tower Defense Spiel entwickelt, das Schülern und Studienanfängern die Grundlagen der Softwarewartung beibringen soll. Der Spieler wird als Metapher für Software Bugs mit Ameisen konfrontiert und kann auf dem Spielfeld eine begrenzte Zahl von Türmen platzieren, um diese abzuschießen. Die vier Arten der Softwarewartung, die im Spiel vermittelt werden, sind *adaptiv*, *korrektiv*, *perfektionierend* und *präventiv*.

Die *adaptive Wartung*, also die Anpassung des Systems an veränderte technische Umgebungsbedingungen, wird dadurch simuliert, dass der Nutzer sein System im Spielverlauf an unterschiedlich große und schnelle Ameisen anpassen muss. Die begrenzte Zahl an Türmen repräsentiert dabei die Wiederverwendung von Codeelementen.
Dadurch, dass der Spieler Türme wieder abreißen und an anderer Stelle aufbauen kann, betreibt er *korrektive Wartung* an seinem System, behebt also Fehler.
Perfektionierende Wartung ist im Gegensatz zu den ersten beiden Formen proaktiv und soll das System schneller, effizienter oder flexibler machen, bevor ein Fehler entsteht. Im Spiel entspricht das der Vorgabe, dass keine Ameise ihr Ziel erreichen soll.
Präventive Wartung soll bisher unbekannte Fehler im System finden und korrigieren, bevor sie zum Problem werden. Diese wird durch das strategische Platzieren von Türmen umgesetzt, um die Ameisen, die fehlerhafte Eingabezustände repräsentieren, vor dem Erreichen ihres Ziels zu zerstören.

In diesem Beispiel ist ein Spielklassiker angepasst worden, um die Grundlagen der Softwarewartung abstrahiert und implizit zu vermitteln. Das Spiel erlaubt dem Nutzer einen hohen Grad der Selbstbestimmung. Der Spieler kann seine eigenen Strategien ausprobieren, aus seinen Fehlern lernen und so mehr als reines Faktenwissen erlernen.

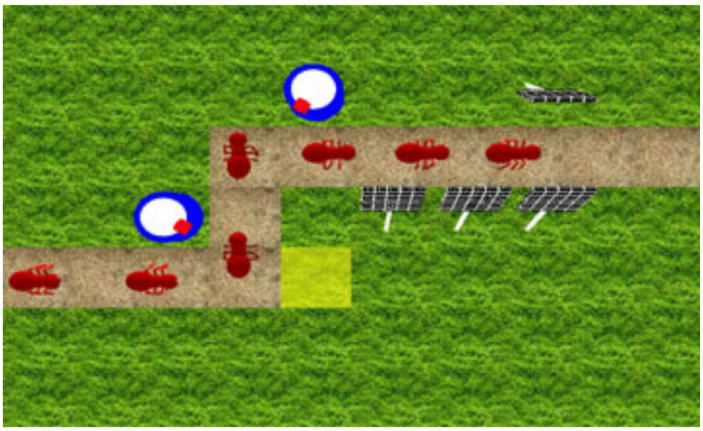

Abbildung 2.9.: Der Spieler muss die Ameisen mit Türmen abwehren. [RRBF 11, S. 177]

In einer Nutzerstudie gaben die meisten Versuchspersonen an, dass ihnen das Spiel Spaß gemacht hat. Außerdem konnte der Großteil der Schüler und Studenten das im Spiel Gelernte auch auf die Softwarewartung übertragen [RRBF 11].

Obwohl sowohl *Codeacademy* als auch *Robot Karol* die Vermittlung von Programmierkenntnissen zum Ziel haben, bewegen sie sich auf unterschiedlichen Abstraktionsgraden. Während *Codeacademy* Kurse für verschiedene verbreitete Programmiersprachen anbietet, bringt *Robot Karol* eine eigene einfache Sprache mit, die die Grundkonzepte der imperativen Programmierung vermitteln soll. Diese ist zwar außerhalb der Programmierumgebung nicht direkt anwendbar, dafür aber weniger komplex als die meisten gängigen Sprachen und daher für Programmieranfänger gut geeignet.
Schüler ohne Programmiererfahrung sind auch die Zielgruppe von *Robot Karol*, während *Codeacademy* eher Erwachsene, die zum Beispiel zusätzliche Qualifikationen für ihren Beruf sammeln wollen, anspricht. Auch hier wird aber keine Programmiererfahrung vorausgesetzt.
Ein weiterer Unterschied ist, dass *Codeacademy* im strengeren Sinn eine gamifizierte Anwendung ist, aber kein vollständiges Serious Game. *Codeacademy* setzt Spielelemente wie schnelles Feedback oder Badges ein, um die Nutzer zu motivieren. *Robot Karol* und auch *Lightbot* sind dagegen vollständige Spiele.
Auch das Tower Defense Spiel zur Vermittlung der Grundprinzipien der Softwarewartung ist ein vollständiges Serious Game und hat eine ähnliche Zielgruppe wie *Robot Karol*, nämlich Schüler und Studienanfänger. Wie bei *Robot Karol* werden hier höhere Fähigkeiten abstrahiert vermittelt, indem die Techniken der Softwarewartung durch die Verteidigung gegen Ameisen simuliert werden.

2.2. Linux-Serveradministration

Im folgenden Abschnitt werden einige Grundlagen der Linux-Serveradministration erläutert. Ziel dieser Arbeit ist es zu zeigen, ob dieses oft als trocken empfundenene Fachwissen mit Hilfe von Gamification interessant vermittelt werden kann.

Zuerst werden einige grundlegende Administrationswerkzeuge behandelt, wie die relativ einfachen Cronjobs, mit denen regelmäßig wiederkehrende Aufgaben automatisiert werden können. Auch das Secure Shell Protokoll zum verschlüsselten Arbeiten auf entfernten Rechnern und das Network File System, mit dem Verzeichnisse auf entfernten Systemen lokal eingehängt werden können, fallen in diese Kategorie.

Danach wird auf sicherheitsspezifische Anwendungen eingegangen, die für größere Organisationen wie das LRZ besonders von Bedeutung sind, da sie potenzielle Angriffsziele darstellen. Ein einfaches Werkzeug für eine Basisabsicherung ist TCP Wrapper, mit dem der Zugriff auf Serverdienste für bestimmte Clients eingeschränkt werden kann. Firewalls erweitern diesen Schutz, indem sie den Verkehr für ganze Netzwerke nach genaueren Kriterien filtern. Hierbei werden Web Application Firewalls, die auf die Absicherung von Webservern spezialisiert sind, hervorgehoben. Als nächstes werden die Intrusion Detection Systems (IDS) und Intrusion Prevention Systems (IPS) vorgestellt, die ein System überwachen und sicherheitsrelevante Vorfälle erkennen und im Falle eines Intrusion Prevention Systems auch verhindern. Im Anschluss folgt mit OSSEC ein Beispiel für ein IDS und mit AppArmor ein Beispiel für ein IPS. Beide dieser Systeme sind host-based, überwachen also jeweils einen Rechner.

2.2.1. Cronjobs

Ein *Cronjob* ist eine Aufgabe, die vom Betriebssystem regelmäßig zu einer bestimmten Zeit ausgeführt werden soll [PlWe 12]. Dadurch können Tätigkeiten automatisiert und ohne die Aufsicht des Administrators ausgeführt werden, was eine erhebliche Arbeitserleichterung darstellen kann. Werden zum Beispiel in einer Firma täglich alle Daten auf einem internen Server gesichert, passiert das am besten in den frühen Morgenstunden, wenn keine Benutzer darauf arbeiten. Dadurch ändert niemand gerade die Daten oder wird durch die längere Antwortzeit während der Sicherung bei der Arbeit behindert. Durch die Verwendung von Cronjobs muss der Administrator dabei nicht anwesend sein, sondern kann die Ausführung am nächsten Werktag überprüfen.

Cronjobs werden von einem *Daemon*, also einem Dienst, der im Hintergrund ohne direkte Verbindung zu Maus und Tastatur läuft, namens `cron` ausgeführt. Dieser überprüft einmal pro Minute, ob abzuarbeitende Aufträge anstehen [PlWe 12, Kofl 14].

Diese Aufträge sind in speziellen Tabellen, den *Crontabs*, auflistet, die angeben zu welcher Zeit an welchem Datum welcher Befehl ausgeführt werden soll. Diese können sich an verschiedenen Orten im System befinden [PlWe 12]:

- `/etc/crontab` ist die Crontab des Administrators, in der normalerweise alle systemweiten Cronjobs eingetragen werden.
- In `/var/spool/cron/crontabs` liegen alle lokalen Crontabs der verschiedenen Benutzer, benannt nach den jeweiligen Benutzernamen.

In den Verzeichnissen `/etc/cron.hourly`, `/etc/cron.daily`, `/etc/cron.weekly` und `/etc/cron.monthly` können Skripte, die jeweils täglich,

wöchentlich, monatlich oder in einem anderen Rythmus ausgeführt werden sollen, abgelegt werden [Kofl 14].

Aufbau einer Crontab-Datei

Jede Zeile in einer Crontab-Datei ist entweder ein Kommentar, der mit # eingeleitet werden muss, oder die Definition eines Cronjobs. Bei der Crontab-Datei des Administrators bestehen diese Definitionen aus sieben, bei den anderen Benutzern aus sechs durch Leerzeichen oder Tabs getrennten Feldern. Diese Felder haben in der Reihenfolge der Angabe die folgende Bedeutung [Kofl 14]:

- *Minute* der Ausführung als Zahl von 0 bis 59
- *Stunde* als Zahl zwischen 0 und 23
- *Tag des Monats* als Zahl von 1 bis 31
- *Monat*, kann als Zahl zwischen 1 und 12 oder mit Namen angegeben werden
- *Tag der Woche*, wird als Zahl von 0 bis 7 angegeben, wobei 0 und 7 beide den Sonntag bezeichen.
- *Nutzer*, unter dessen Namen das Kommando ausgeführt wird. Dieses Feld existiert nur in der Tabelle des Superusers, da bei den anderen Benutzern alle Befehle unter ihren eigenen Rechten ausgeführt werden.
- *Kommando*, das ausgeführt werden soll.

Die Wildcard `*` wird als *jeder Zeitpunkt* interpretiert. Ein Eintrag mit dem Ausführungszeitpunkt `* * * * *` wird also zu jeder Minute zu jeder Stunde an jedem Tag ausgeführt. `* 3 * * *` würde dagegen täglich um 3 Uhr ausgeführt werden. Außerdem sind Listen wie `2,5,6` und Intervalle wie `3-5` erlaubt. Bei der Wildcard und bei Intervallen kann eine Schrittweite angegeben werden, `*/2` heißt also *alle zwei Stunden*, `0-9/2` enspricht `0,2,4,6,8`. Wird sowohl ein Tag des Monats als auch ein Wochentag angegeben, wird der Job ausgeführt, wenn eine Bedingung von beiden zutrifft, als zum Beispiel zum ersten des Monats und an jedem Montag wie in Listing 2.1 [PlWe 12]. Im Beispiel werden alle Kommandos mit `root`-Rechten ausgeführt.

Listing 2.1: Beispiel für eine `/etc/crontab`-Datei nach [PlWe 12]

```
# wird jeden Tag 5 Minuten nach Mitternacht ausgefuehrt
5 0 * * *     root /root/bin/daily.job >> /root/tmp/out
# wird am 1. jedes Monats und jeden Montag um 14:15 ausgefuehrt
15 14 1 * 1   root /bin/monthly
```

Zusätzlich können in den Crontabs Umgebungsvariablen als Paare `name=wert` definiert werden. Diese werden entweder den ausgeführten Programmen zur Verfügung gestellt oder haben eine Bedeutung für `cron` selbst. Einige Variablen werden auch vorbelegt. Dazu gehören [PlWe 12]:

- `HOME` wird mit dem Wert aus `/etc/passwd` belegt.

- `SHELL` wird ebenfalls mit einem Wert aus dieser Datei vorbelegt und ist für `cron` selbst wichtig, da die Befehle vom in der Variable angegebenen Interpreter ausgeführt werden.
- `PATH` dient dazu, Programme ohne absoluten Pfadnamen zu finden und ist mit `/usr/bin:/bin` vorbelegt.
- `MAILTO` legt die E-Mail-Adresse fest, an die die eventuellen Ausgaben des im Hintergrund ausgeführten Programms gesendet werden sollen. Damit können Fehlermeldungen nachvollzogen werden. Ist die Variable nicht gesetzt, wird die Mail an den Eigentümer der Tabelle geschickt.

Der Befehl `crontab`

Die verschiedenen Crontabs können zwar auch direkt editiert werden, sollten aber mit dem Befehl `crontab` bearbeitet werden. Dieser kennt folgende Parameter [IEEE 03]:

- `-e`: die Crontab-Datei des Benutzers mit einem Editor bearbeiten oder, falls noch keine existiert, eine neue erstellen
- `-l`: den Inhalt der entsprechenden Crontab-Datei ausgeben
- `-r`: den Inhalt der Crontab-Datei löschen

Mit dem optionalen Parameter `-u`, gefolgt von einem Benutzernamen, wählt `crontab` die Tabelle des jeweiligen Benutzers an [PlWe 12]. Wird dieser nicht angegeben, wird die Tabelle des Aufrufers angewählt.

`crontab` ändert die Timestamps der jeweiligen Crontabs bzw. des Verzeichnisses `/var /spool /cron /crontabs`, woran `cron` erkennt, dass die Dateien neu eingelesen werden müssen. Daher ist kein Neustart des Prozesses notwendig [PlWe 12].

Mithilfe von Cronjobs können also wiederkehrende Aufgaben automatisiert werden. Cronjobs werden durch den Daemon `cron` ausgeführt, der mit `crontab` konfiguriert werden kann. Dieser Befehl bearbeitet die Crontab-Dateien, die für jeden Nutzer auflisten, wann welche Aufgaben ausgeführt werden sollen.

2.2.2. Secure Shell (SSH)

Secure Shell (*SSH*) ist ein Protokoll, mit dem auf der Kommandozeile eines enfernten Systems gearbeitet werden kann. Dadurch ist es unerlässlich für die Administration von Servern, auf die kein physischer Zugriff möglich ist. Bei älteren Diensten wie `telnet` (*Teletype Network* [Merr 16]) oder `rlogin` erfolgt die Übertragung unverschlüsselt, wodurch Passwörter und persönliche Daten abgefangen werden können. SSH verschlüsselt dagegen die Verbindung und wird deswegen als sichere Alternative empfohlen [DKK+ 14, Kofl 14].

Die Authentifizierung der Kommunikationspartner erfolgt dabei über einen öffentlichen Schlüssel. Client und Server haben also jeweils einen öffentlichen Schlüssel, der dem Kommunikationspartner über einen unverschlüsselten Kanal mitgeteilt werden kann, und einen

geheimen privaten Schlüssel. Mit dem privaten Schlüssel wird eine Signatur erzeugt, die mit Hilfe des öffentlichen Schlüssels verifiziert werden kann [Open 16c].
SSH wurde ursprünglich von Tatu Ylönen entwickelt. Da die ursprüglich quelloffene Lizenzierung immer restriktiver wurde, wird seit 1999 vom *OpenBSD Project* [Open 16a] die quelloffene Abspaltung *OpenSSH* entwickelt [oss 16]. Im Folgenden wird diese Variante, die unter Linux standardmäßig installiert ist, beschrieben.

OpenSSH umfasst unter anderem folgende Programme [Open 16c, DKK+ 14]:

- `ssh`, ein Secure Shell Client
- `scp`, ein Client zum verschlüsselten Kopieren von Dateien von und auf entfernte Systeme mit *Secure Copy* (*SCP*)
- `sftp`, ein Client zum verschlüsselten Übertragen von Dateien mit dem *Secure File Transfer Protocol* (*SFTP*)
- `sshd`, der SSH-Daemon
- `sshkeygen` zum Erzeugen von Authentifizierungsschlüsseln
- `sshkeyscan` zum Einlesen öffentlicher Schlüssel
- `ssh-agent` zum Speichern des privaten Schlüssels im Arbeitsspeicher

SSH läuft standardmäßig über den TCP-Port 22 [DKK+ 14]. *Ports* werden zur Identifikation verschiedener Dienste verwendet. Die *privilegierten* oder *well-known* Ports bis einschließlich 1023, für deren Betrieb `root`-Rechte benötigt werden, sind dabei für verschiedene Server-Dienste reserviert. Ein Client, der sich mit einem Server verbinden will, schickt eine Anfrage also an einen solchen reservierten Port auf einer bestimmten IP-Adresse. Er wählt dazu normalerweise dynamisch einen höheren Port, an den der Server seine Antwort schickt [Kofl 14, Harr 07].

Konfiguration des Clients

Mit dem Befehl `ssh` kann sich ein Nutzer auf einem Client unter Angabe des Hostnamens oder der IP-Adresse des Zielservers auf einem SSH-Server einloggen. Unterscheidet sich der Nutzername auf dem Server von dem auf dem Client, muss auch dieser angegeben werden (vgl. Listing 2.2). Bei der ersten Verbindung mit einem Server muss dessen *Fingerprint*, also der Fingerabdruck seines öffentlichen Schlüssels, akzeptiert werden. Dieser wird dann in der Datei `~/.ssh/known_hosts` abgelegt. Anhand dessen kann der Server bei der nächsten Verbindung identifiziert werden, um zu verhindern, dass sich ein Angreifer als dieser ausgibt. Danach muss das Passwort für das Konto auf dem Server angegeben werden [DKK+ 14].

Listing 2.2: Login auf dem Server mit SSH nach [DKK+ 14]

```
# ssh username@server.lrz.de
```

In der Datei `~/.ssh/config` können Parameter wie der Benutzername oder ein vom Standard abweichender Port abgespeichert werden, so dass beim Verbindungsaufbau nur

noch der Host angegeben werden muss. In Listing 2.3) wird der Nutzer bei Angabe von `lrz.de` also automatisch auf dem Server mit der IP-Adresse `10.10.47.11` über den Port `2222` als Nutzer `testuser` eingeloggt [DKK+ 14].

Listing 2.3: Mögliche Konfiguration in `~/.ssh/config` nach [DKK+ 14]

```
Host lrz.de
        HostName 10.10.47.11
        Port 2222
        User testuser
```

Der Befehl `scp` funktioniert ähnlich wie der Standard-Kopierbefehl `cp`, ist aber zum Kopieren von Dateien und Verzeichnissen vom Clientsystem auf den Server und umgekehrt gedacht. In Listing 2.4 wird die Datei `test.txt` aus dem aktuellen Verzeichnis in `/usr/local/texte` auf `lrz.de` kopiert. `scp` implementiert dabei nur den Dateitransfer selbst, die Anmeldung läuft über `ssh`.

Listing 2.4: Kopieren einer Datei mit `scp` nach [DKK+ 14]

```
# scp test.txt lrz.de:/usr/local/texte
```

`sftp` erweitert `scp` und bietet zusätzliche Dateioperationen an, zum Beispiel um den Besitzer einer Datei zu ändern. Die Anmeldung mit `sftp` funktioniert analog zu `ssh` (vgl. Listing 2.5) [DKK+ 14].

Listing 2.5: Aufbau einer Verbindung mit `sftp` nach [DKK+ 14]

```
# sftp testuser@server.lrz.de
```

Um über SSH auf dem Server ein grafisches Programm zu starten, das auf dem Display des Clients eingeblendet wird, wird *X11-Forwarding* verwendet. Dabei wird `ssh` auf dem Client mit dem Schalter `-X` aufgerufen. In der Datei `/etc/ssh/ssh_config` müssen dafür die Parameter `ForwardX11` und `ForwardX11Trusted` auf `yes` gesetzt werden (vlg. Listing 2.6).

Listing 2.6: Aktivieren von X11-Forwarding in `/etc/ssh/ssh_config` nach [DKK+ 14]

```
ForwardX11 yes
ForwardX11Trusted yes
```

Erzeugen eines Schlüsselpaares

Nach einer Analyse des Hasso-Plattner-Instituts von mehr als 215 Millionen geleakter Identitätsdaten verschiener Onlinedienste ist das beliebteste Passwort im Jahr 2015 „123456“und macht mehr als ein Prozent der untersuchten Datensätze aus [Hass 15]. Das zeigt, dass viele Nutzer sehr einfache oder kurze Passwörter wählen, die von einem Angreifer leicht gebrochen werden können und dadurch ein Sicherheitsrisiko darstellen.

Um dieses Risiko zu minimieren, kann ein Benutzer zur Authentisierung an einem SSH-Server statt eines selbstgewählten Passworts auch einen zufällig generierten Schlüssel verwenden, der mindestens 768 Bit lang sein muss. Mit `ssh-keygen` wird ein Paar aus privatem und öffentlichem Schlüssel erzeugt (vgl. Listing 2.7). Der Schalter `-t` legt dabei den verwendeten Algorithmus fest, im Beispiel RSA . Der Schalter -b gibt die Länge der Schlüssel in Bit an, bei Verwendung von RSA ist der Standard 2048 bit, was aktuell als sicher angesehen wird. Dabei muss eine *Passphrase*, also eine Art Passwort angegeben werden, mit der der private Schlüssel des Nutzers verschlüsselt wird, um unberechtigte Zugriffe zu verhindern. Diese Passphrase kann auch leer sein [Open 16b].

Der private Schlüssel wird bei Verwendung von RSA standardmäßig in der Datei `~/.ssh/id_rsa` abgelegt, der öffentliche in `~/.ssh/id_rsa.pub` [DKK+ 14].

Listing 2.7: Erzeugung eines Schlüsselpaares nach [PlWe 12]

```
# ssh-keygen -t rsa -b 2048
```

Um sich mit SSH auf einem Server ohne Passwort einzuloggen, muss diesem der öffentliche Schlüssel mit `ssh-copy-id` unter Angabe des Hostnamens und Benutzers bekannt gemacht werden. Dieser wird dann auf dem Server der Datei `/home/username/ssh/authorized_keys` hinzugefügt. Danach fragt der Server nicht mehr nach einem Passwort, dafür muss die Passphrase für den Schlüssel eingegeben werden. Der `ssh-agent` kann die Passphrase speichern, so dass der Nutzer sie nicht bei jeder Anmeldung eingeben muss [DKK+ 14].

Konfiguration des Servers

Der Daemon `sshd` kann über die Datei `/etc/ssh/sshd_config` konfiguriert werden (vgl. Listing 2.8).

Im Beispiel wird der Port, auf dem Verbindungen entgegengenommen werden, auf den Standardport 22 festgelegt. Da die Sicherheitseinstellung `UsePrivilegeSeparation` auf `yes` gesetzt ist, läuft die Kommunikation nach erfolgreichem Login über einen Kindprozess des Servers, der mit den Rechten des entsprechenden Nutzers gestartet wird [PlWe 12].

Mit `AllowUsers` bzw. `AllowGroups` kann nur bestimmten Benutzern oder Gruppen der Zugriff erlaubt werden. Analog kann mit `DenyUsers` bzw. `DenyGroups` betimmten Benutzern oder Gruppen der Login verweigert werden. Die Benutzer- und Gruppenbeschränkungen werden in der Reihenfolge `DenyUsers`, `AllowUsers`, `DenyGroups`, `AllowGroups` geprüft. Mit der Zeile `PermitRootLogin no` können außerdem Logins als root verhindert werden. Von einem normalen Konto kann aber immer noch in den root-Modus gewechselt werden.

Außerdem kann mit einer Änderung der Zeile `Protocol 2` in `Protocol 2,1` erreicht werden, dass zusätzlich zum aktuellen SSHv2 das veraltete SSHv1 angeboten wird, zum Beispiel aus Kompatibilitätsgründen. Um X11-Forwarding zu ermöglichen, muss der Eintrag `X11Forwarding` existieren und auf `yes` gesetzt sein [DKK+ 14].

Listing 2.8: Beispielkonfiguration in `/etc/ssh/sshd_config` nach [PlWe 12, DKK+ 14]

```
Port 22
UsePrivilegeSeparation yes
AllowUsers meier, mueller
AllowGroups backup
PermitRootLogin no
Protocol 2
X11Forwarding yes
```

Durch das Bearbeiten der Datei `/etc/ssh/authorized_keys`, in der der Server die öffentlichen Schlüssel aller bekannten Nutzer ablegt, kann der verfügbare Befehlssatz für Nutzer eingeschränkt werden, die auf dem Server nur bestimmte Kommandos ausführen dürfen. Dabei werden die erlaubten Befehle vor den Schlüssel des jeweiligen Nutzers geschrieben [DKK+ 14].

Mit `ssh` kann also verschlüsselt auf einem entfernten System gearbeitet werden. Dateien und Verzeichnisse können dabei mit `scp` und `sftp` zwischen dem Server und dem Client übertragen werden. Um die Eingabe eines oft kryptographisch schwachen Passworts zu umgehen, kann ein Schlüssel generiert werden, der dem Server anstatt dessen zur Authentifizierung übergeben wird.

2.2.3. Network File System (NFS)

Während mit `scp` und `sftp` nur die Übertragung von Daten von und auf entfernte Systeme möglich ist, können mit dem *Network File System* (*NFS*) entfernte Verzeichnissen und Festplattenpartitionen direkt ins lokale System eingebunden werden. Das entfernte Dateisystem wird dabei direkt in den lokalen Dateibaum eingehängt. Dadurch können Daten auf einem Server zentral verwaltet und auf Clientsystemen bereitgestellt werden. Außerdem wird die Sicherung von Anwenderdaten auf einem Server vereinfacht [DKK+ 14].

In der aktuellen Version *NSFv4* laufen alle Daten unter Verwendung des TCP-Protokolls über den Standardport 2049. Es werden also im Gegensatz zur Vorgängerversion die Protokolle zum Einhängen, Sperren von Dateien und Registrieren von auf dem Client geöffneten Dateien zusammengefasst. Die eigentlichen Freigaben werden in ein Pseudodateisystem eingebunden [DKK+ 14]. Die folgenden Ausführungen beziehen sich auf diese NFS-Version.

Sowohl auf dem Client als auch auf dem Server muss der Dienst `rpc.imapd` laufen. Dieser sorgt dafür, dass die Benutzernamen auf dem NFS-Server auf die lokalen Benutzernamen gemappt werden [Kofl 14]. Das ist relevant, weil unter Linux jede Datei einen Besitzer hat, auf den sich die Zugriffsrechte beziehen.

Konfiguration des Servers

Zur Konfiguration eines NFS muss auf dem Server nach der Installation der entsprechenden Pakete zunächst ein Verzeichnis für das Pseudodateisystem erstellt werden. Danach müssen in diesem Pseudodateisystem *bind-Mountpoints*, also Einhängepunkte, an die später die eigentlichen Verzeichnisse mit den freizugebenden Daten gebunden werden, erzeugt werden. Anschließend werden die eigentlichen Datenverzeichnisse in die erzeugten bind-Mountpoints eingebunden [DKK+ 14]. Das heißt, dass auf diese Datenverzeichnisse auch vom Mountpoint aus zugegriffen werden kann [util 14].

In Listing 2.9 wird zuerst das Verzeichnis `/nfsexport` für das Pseudodateisystem erstellt. Danach wird darin der bind-Mountpoint `/nfsexport/fotos` für das Verzeichnis `/data/fotos` erzeugt. Anschließend wird das Verzeichnis `/data/fotos` an diesen Mountpoint gebunden. Auf dieses Verzeichnis kann jetzt also auch unter `/nfsexport/fotos` zugegriffen werden. Der `mount`-Befehl kann normalerweise nur von `root` ausgeführt werden [util 14].

Listing 2.9: Erzeugen eines bind-Mountpoints und Binden eines Verzeichnisses nach [Kofl 14, util 14]

```
root@server:~# mkdir /nfsexport
root@server:~# mkdir nfsexport/fotos
root@server:~# mount --bind /data/fotos /nfsexport/fotos
```

Damit das Einbinden des NFS-Verzeichnisses zukünftig automatisch erfolgt, wird die Datei `/etc/fstab` wie in Listing 2.10 ergänzt [Kofl 14]:

Listing 2.10: Ergänzen der Datei `/etc/fstab` nach [util 14]

```
/data/fotos     /nfsexport/fotos  none  bind
```

Die Freigaben für den NFS-Server werden in die Datei `/etc/exports` eingetragen (vgl. Listing 2.11). Im Beispiel bezieht sich der erste Eintrag auf das Pseudodateisystem und der zweite auf die eigenliche Freigabe. Die Parameter haben dabei folgenden Bedeutungen [DKK+ 14]:

- `/nfsexport` bzw. `/nfsexport/fotos` bezeichnen die Freigabe, die hier verwaltet wird, also das Pseudodateisystem und den Freigabeordner.
- `192.168.123.0/24` ist das Netzwerk, innerhalb dessen alle Hosts Zugriff auf die Daten haben. Dieses kann durch die Subnetzmaske bis auf einen Host genau eingeschränkt werden.
- `ro`, also *read-only* und `rw`, also *read-write* sind Zugriffsberechtigungen. Während das Pseudodateisystem *read-only* eingehängt wird, so dass die Clients nicht direkt in das Dateisystem schreiben dürfen, werden die Freigaben *read-write* eingehängt, die Clients dürfen also auch in die Freigabe schreiben. Die Standardeinstellung ist *read-only*.

- Der Parameter `sync` bzw. `async` beeinflusst das Verhalten des Servers beim Speichern von Daten, die vom Client gesendet werden. Ist `sync` gesetzt, meldet der Server dem Client erst, dass ein Schreibversucht erfolgreich war, wenn er eine positive Rückmeldung vom lokalen Dateisystem hat. Mit `async` teilt der Server dem Client den Erfolg des Schreibversuchs schon nach Empfang der Schreibanfrage mit, also schon bevor er sie an das lokale Dateisystem weitergeleitet hat. Das kann zum Beispiel bei einer vollen Festplatte zum Datenverlust führen.
- Mit `root_squash` hat der lokale Benutzer `root` auf dem Client an der Freigabe nur die Rechte, die der Benutzer `nobody` auf dem Server hat. Das ist auch die Voreinstellung. Soll der lokale Benutzer `root` dagegen volle Dateisystemrechte erhalten, muss der Parameter auf `no_root_squash` gesetzt werden.
- `subtree_check` bzw. `no_subtree_check` wird relevant, wenn nur ein Verzeichnis und keine ganze Partition freigegeben wird. Ist `subtree_check` gesetzt, prüft der Server bei jedem Zugriff nicht nur, ob sich die Datei in der Partition befindet, sondern auch, ob die Datei im entsprechenden Verzeichniszweig ist. Diese zusätzliche Sicherheitsprüfung ist allerdings für den Server aufwendig und kann mit `no_subtree_check` deaktiviert werden, was auch standardmäßig eingestellt ist.
- `fsid=0` kennzeichnet das Wurzelverzeichnis des Pseudodateisystems für das System.

Nachdem die Änderungen an `/etc/exports` durchgeführt worden sind, muss der NFS-Server neu geladen werden [DKK+ 14].

Listing 2.11: Beispielhafte Freigaben in `/etc/exports` nach [DKK+ 14]

```
# Eintrag für das Pseudodateisystem
/nfsexport 192.168.123.0/24/(ro,sync,root_squash,\
no_subtree_check,fsid=0)
# Eintrag für die Freigabe
/nfsexport/fotos 192.168.123.0/24/(ro,sync,root_squash,\
no_subtree_check)
```

Konfiguration des Clients

Um die Daten auf den Clients verfügbar zu machen, muss das entfernte Dateisystem wie in Listing 2.12 gemountet werden. Dabei kann entweder das gesamte Pseudodateisystem oder einzelne freigebene Verzeichnisse angegeben werden, wobei nicht der Pfad innerhalb des Pseudodateisystems, sondern das eigentliche Verzeichnis angegeben wird. Im Beispiel wird das freigegebene Verzeichnis `daten` auf dem Client als `/daten` eingehängt. Der Schalter `-t nfsv4` erzwingt dabei die Verwendung von NFSv4. Um das entfernte Verzeichnis beim Start automatisch einzuhängen, muss es in `/etc/fstab` eingetragen werden [DKK+ 14].

Listing 2.12: Einhängen des entfernten Dateisystems auf dem Client nach [DKK+ 14]

```
root@client:~# mount -t nfsv4 server:/fotos /fotos
```

Mit einem NFS können also von einem Server bereitgestellte Dateisysteme lokal auf einem Client gemountet werden. Auf dem Server muss dazu ein Pseudo-Dateisystem erzeugt und die darin freizugebenden Verzeichnisse an einen bind-Mountpoint gebunden werden. Danach werden sie in die Datei `/etc/exports` eingetragen. Auf dem Client kann das entfernte Dateisystem dann gemountet werden.

2.2.4. TCP Wrapper

Während in den vorhergehenden Abschnitten grundlegende Administrationswerkzeuge behandelt wurden, liegt der Fokus im Folgenden auf der Sicherheit. Als erstes wird das in Linux integrierte Werkzeug *TCP Wrapper* behandelt, das eine Grundabsicherung darstellt.

TCP Wrapper (`tcpd`) ist eine Bibliothek, also eine Sammlung von Hilfsmodulen für andere Programme, die Access Control Lists (ACLs), also Zugriffssteuerungslisten verwaltet. Dadurch kann unberechtigten Clients der Zugriff auf bestimmte Serverdienste verweigert oder umgekehrt bei einem kritischen System nur ausgesuchten Nutzern der Zugang erlaubt werden. So kann ein Dienst zum Beispiel nur im lokalen Netzwerk verfügbar gemacht werden. TCP Wrapper wird von Netzwerkdiensten wie SSH und NFS genutzt oder von einem *Superserver* aufgerufen [Kofl 14].

Ein *Superserver* (`inetd` oder `xinetd`) wartet darauf, dass bestimmte Netzwerkverbindungen hergestellt werden. Ist zum Beispiel eine FTP-Funktionalität (*File Transfer Protocol*) konfiguriert, wird der eigentliche FTP-Server erst gestartet, wenn tatsächlich eine Verbindungsanfrage vorliegt. Dadurch, dass selten genutzte Dienste nicht immer laufen, können Ressourcen eingespart werden [The 15].

Superserver verwalten dabei sowohl *interne* als auch *externe Dienste*. *Interne Dienste*, für die keine externen Binaries verwendet werden, werden vom Superserver selbst bereitgestellt. *Externe Dienste* wie FTP-Server werden dagegen in einem eigenen Prozess gestartet und kommunizieren dann indirekt über den Superserver mit dem Client [PlWe 12]. `xinetd` ist dabei die erweiterte Version des inzwischen veralteten `inetd` [Kofl 14].

Konfiguration von `tcpd`

Um die Autorisierung einer Verbindung zu überprüfen, nutzt TCP Wrapper die beiden Konfigurationsdateien `/etc/hosts.allow` und `/etc/hosts.deny` [PlWe 12].

- Zuerst wird dabei geprüft, ob der Client in der `/etc/hosts.allow` steht. Ist das der Fall, wird die Verbindung zugelassen.
- Ansonsten wird nach dem Client in `/etc/hosts.deny` gesucht. Ist er in dieser Datei gelistet, wird die Verbindung verweigert.
- Falls in keiner der beiden Dateien ein Eintrag zu finden ist, wird die Verbindung erlaubt.

Die Einstellungen in den beiden Konfigurationsdateien gelten dabei für alle Dienste, die auf die Bibiothek zugreifen [Kofl 14].

In Listing 2.13 wird der Zugriff standardmäßig verweigert, da in `/etc/hosts.deny` die Wildcard `ALL`, die auf alle Hosts zutrifft, auf `ALL` gesetzt ist. In `/etc/hosts.allow` wird der Zugriff nur lokalen Hosts, deren Namen also keine Punkte enthält (`LOCAL`) erlaubt. Außerdem dürfen alle Mitglieder der `lrz.de` Domain außer `terminalserver.lrz.de` zugreifen [Vene 16]. Der Zugriff auf den SSH-Server wird im lokalen Netzwerk `192.168.0.*` erlaubt [Kofl 14].

Listing 2.13: Beispielhafte Konfiguration von `tcpd` nach [Vene 16, Kofl 14]

```
#/etc/hosts.deny
ALL:ALL

#/etc/hosts.allow
ALL: LOCAL
ALL: .foobar.edu EXCEPT terminalserver.foobar.edu
sshd: 192.168.0.0/24
```

TCP Wrapper kontrolliert also den Zugriff auf bestimmte Serverdienste. `tcpd` wird dabei von einem Superserver aufgerufen, der auf Verbindungsanfragen wartet und die einzelnen Netzwerkdienste erst startet, wenn sie benötigt werden. Über die Konfigurationsdateien `/etc/hosts.allow` und `/etc/hosts.deny` kann einem Client die Verbindung erlaubt bzw. verweigert werden.

2.2.5. Firewalls

TCP Wrapper kann zwar bestimmten Clients oder Nutzern den Zugriff auf einen Server verweigern, stellt aber nur eine Basisabsicherung dar. Um einzelne Pakete nach genaueren Kriterien zu filtern und den Zugriff auf ganze Netzwerke zu verhindern, wird aber eine *Firewall* benötigt.

Firewalls schränken den Zugriff auf einen Rechner oder ein Netzwerk ein. Meist werden sie eingesetzt, um lokale Netzwerke wie Firmennetze vor Zugriffen aus dem Internet zu schützen. Der eingehende Verkehr wird dazu untersucht und anhand eines Regelwerks überprüft. In diesem Regelwerk wird definiert, ob auf bestimmte Ports oder Dienste zugegriffen werden darf und welche IP-Adressen oder Netzwerke beschränkt werden sollen [Harr 07, PlWe 12]. Die Ausführungen in diesem Abschnitt beziehen sich auf den *CISSP All-in-One Exam Guide* von Shon Harris [Harr 07].

Es gibt verschiedene Typen und Entwicklungsstufen von Firewalls, die unterschiedliche Anforderungen erfüllen. Dabei wird zwischen *Packet Filtering Firewalls*, *Stateful Firewalls*, *Dynamic Packet Filtering*, *Proxy Firewalls* und *Kernel Proxy Firewalls* differenziert.

Packet Filtering Firewalls

Packet Filtering Firewalls sind die einfachsten Firewalls und gehören zur ersten Generation. Sie filtern Pakete anhand von ACLs. Dabei inspizieren sie nur die Headerinformation auf Netzwerkprokollebene, sie haben also nur Zugriff auf die Quell- und Ziel-IP-Adressen, Quell- und Ziel-Portnummern, Protokolltypen und die Richtung des Verkehrs, also eingehend oder ausgehend. Für jedes Paket wird überprüft, ob es eine Regel gibt, die darauf zutrifft und der Zugriff entsprechend erlaubt oder verweigert. Ist keine Regel vorhanden, wird der Zugriff verweigert.

Da sie nur die Headerinformationen einzelner Pakete isoliert betrachten, werden Angriffe, die beispielsweise Sicherheitslücken in Anwendungen ausnutzen, nicht erkannt. Dafür sind die einfachen Überprüfungen schnell und anwendungsunabhängig. Daher werden sie oft eingesetzt, um einfache Angriffe herauszufiltern, und durch andere Firewalltypen ergänzt.

Stateful Firewalls

Da Packet Filtering Firewalls jedes Paket einzeln betrachten, kennen sie den jeweiligen Kontext nicht. Verbindungsorientierte Protokolle wie das Netzwerkprotokoll TCP sind aber zustandsabhängig. Das heißt, dass beispielsweise beim Verbindungsaufbau bestimmte Pakete in bestimmter Reihenfolge zwischen den Kommunikationspartnern ausgetauscht werden und während der Kommunikation gültige Sequenznummern enthalten sein müssen. Ein Angreifer kann durch ein Paket, das in einem Protokoll im aktuellen Zustand ungültig ist, das Zielsystem zum Absturz bringen.

Solche ungültigen Pakete versuchen *Stateful Firewalls* zu erkennen, indem sie die Headerdaten ein- und ausgehender Pakete in Zustandstabellen akkumulieren und diese dann einzelnen Sitzungen zuordnen. Da die Zustandtabelle aber eine endliche Ressource ist, kann eine solche Firewall durch Überlastung mit sinnloser Information zum Absturz gebracht werden.

Dynamic Packet Filtering

Wie bereits in Abschnitt 2.2.2 erwähnt, wählt ein Client zur Verbindung mit einem Server normalerweise dynamisch einen Port höher als 1023. Da dieser Port bei jeder Verbindung unterschiedlich sein kann, müssten entweder alle eingehende Verbindungen an Ports größer als 1023 erlaubt werden oder dynamische Regeln erstellt werden, die jeweils für eine Verbindung die Antwort durch einen Server von außen zulassen.

Dynamic Packet Filtering ermöglicht das, indem bei Registrierung einer Anfrage von innen dynamisch eine ACL erzeugt wird, die eine Antwort des Servers von außen an die Quell-IP-Adresse und den Quellport zulässt. Wenn im zu schützenden System keine Server laufen, die von außen zugänglich sein müssen, kann so auch jeder Verkehr von außen verboten werden, der keine Antwort auf eine Anfrage ist.

Proxy Firewalls

Während die zuvor genannten Firewalltypen Pakete nur filtern, agiert eine *Proxy Firewall* als Mittelsmann zwischen einem sicheren und einem unsicheren Netzwerk. Die Kommunikation läuft also nicht mehr direkt zwischen den Teilnehmern, sondern über die Firewall, die jeweils eine eigene Verbindung zu jedem Kommunikationspartner aufbaut und nur als sicher eingestufte Informationen weitergibt.

In Abbildung 2.10 fordert ein Nutzer, dessen Browser konfiguriert ist, einen Firewall Proxy Server zu nutzen, eine Webseite aus dem Internet an. Die Firewall akzeptiert diese Anfrage und leitet sie an den Webserver weiter. Dieser antwortet darauf, ohne zu wissen, dass die Anfrage nicht vom Proxy, sondern von einem Nutzer dahinter kommt. Nachdem der Proxy die Webseite vollständig empfangen und überprüft hat, gibt er sie an den Nutzer weiter. Dadurch wird der Nutzer vor Angriffen aus dem Internet geschützt.

Arbeitet eine solche Firewall nur auf Netzwerkebene, wird sie als *Circuit-Level Proxy* bezeichnet. Sie inspiziert also wie ein Paketfilter nicht den Inhalt der Pakete, sondern nur die Headerinformation. Das hat den Vorteil, dass sie anwendungsunabhängig ist, allerdings werden bestimmte Angriffe nicht erkannt.

Ein *Application-Level Proxy* versteht dagegen auch den Inhalt eines Pakets, also auch anwendungsspezifische Protokolle wie HTTP . Dadurch kann der Netzwerkverkehr wesentlich feiner gefiltert werden, dafür wird innerhalb der Firewall für jedes Protokoll ein eigener Proxy benötigt und die Überprüfung ist sehr aufwendig und verzögert unter Umständen die Kommunikation.

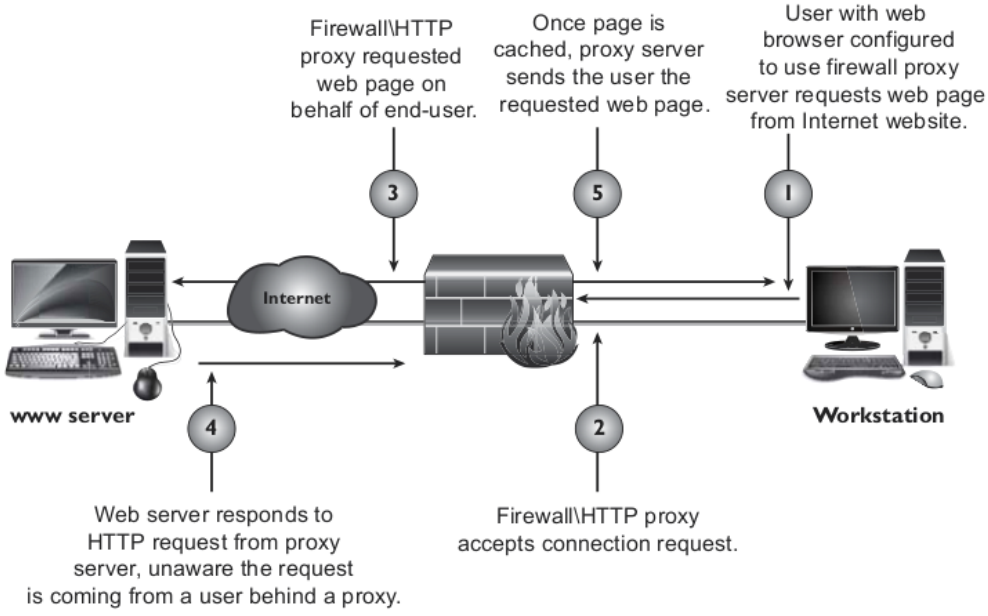

Abbildung 2.10.: Eine Proxy Firewall unterbricht die Verbindung zwischen Client und Server [Harr 07, S. 637]

Kernel Proxy Firewalls

Kernel Proxy Firewalls unterscheiden sich von den zuvor genannten Technologien und gehören zur fünften Generation. Wie Application Level Proxies unterbrechen sie die Kommunikation zwischen dem zu schützenden internen und dem externen Netzwerk. Für jedes Paket werden dynamisch die spezifischen Proxies geladen, die für die Analyse des betreffenden Pakets auf den unterschiedlichen Protokollebenen benötigt werden. Da Kernel Proxy Firewalls im Kernel anstatt auf höherer Softwareebene arbeiten, sind sie schneller als Application Layer Proxy Firewalls. Außerdem können sie mehrere unterschiedliche Protokolle und Dienste untersuchen, allerdings können sie diese nicht so genau kontrollieren.

Firewalls lassen sich also in Typen aufteilen, die unterschiedlich komplex sind und zu unterschiedlichen Generationen zählen.

Einfache Firewalls analysieren nur Headerinformationen und betrachten jedes Paket isoliert. Weiter entwickelte Systeme speichern diese Headerinformationen und ordnen einzelne Pakete so Sitzungen und Protokollzuständen zu. Dadurch werden im Protokoll ungültige Pakete erkannt. Durch dynamisches Erstellen von ACLs wird die Antwort eines externen Servers auf die Anfrage eines internen Clients ermöglicht.

Proxy Firewalls unterbrechen die Kommunikation zwischen zwei Netzwerken und leiten nur als sicher eingestuften Verkehr weiter. Dabei können sie wie die Paketfilter auf Netzwerkebene oder auf Anwendungsebene arbeiten. Dabei wird auch der Inhalt eines Pakets untersucht und für jedes Anwendungsprotokoll ein eigener Proxy bereitgestellt.
Kernel Proxy Firewalls unterscheiden sich von diesen dadurch, dass sie im Kernel arbeiten statt auf höherer Softwareebene arbeiten und dadurch schneller sind. Sie können ein Paket auf allen Protokollebenen untersuchen, indem sie dynamisch alle spezifischen Proxies laden, die dafür benötigt werden.

Die meisten modernen Firewalls kombinieren allerdings mehrere dieser Techniken. Außerdem gibt es spezialisierte Produkte wie *Web Application Firewalls*, die auf den Schutz von Webservern ausgelegt sind. Diese werden im nächsten Abschnitt beschrieben.

2.2.6. Web Application Firewalls

Webserver müssen aus dem Internet erreichbar sein, haben aber oft auch Zugriff auf Datenbanken, die sensible Daten enthalten. Dadurch stellen sie ein lohnendes Ziel für Angriffe dar. *Web Application Firewalls* sind Application-Level Proxy Firewalls, die auf die Absicherung von Webservern ausgelegt sind [Harr 07]. Wie Intrusion Detection Systems und Intrusion Prevention Systems, die in Abschnitt 2.2.7 erläutert werden, untersuchen sie jedes ein- und ausgehende Paket auf bekannte Angriffssignaturen oder Anomalien. Sie unterscheiden sich von diesen aber dadurch, dass sie auf der Anwendungsebene arbeiten [KSBG 13]. Sie können den HTTP-Verkehr zwischen Browser und Server analysieren und verwalten den Zugriff auf angefragte URLs, Authentisierungsinformationen und Informationen über die aktuelle Sitzung [DPJV 06].

Erkennung und Schutz vor Angriffen

Die Daten, die der Clientbrowser an den Webserver in Form von Eingabeparametern für Formulare oder Auswahlfelder übermittelt, können manipuliert werden. Zum Beispiel werden bei einer *SQL-Injection* SQL-Anweisungen über Eingabeparameter an den Datenbankserver geschleust. Dort werden sie ausgeführt und können zum Auslesen privater Daten benutzt werden, wie am Beispiel des LRZ Matrikelnummern und Kontoverbindungen von Studenten [KSBG 13]. Nach dem *Open Web Application Security Project* (*OWASP*) gehörten Injection-Angriffe im Jahr 2013 zu den zehn kritischsten Sicherheitsrisiken bei Webanwendungen [The 13].

Um solche Manipulationen zu erkennen, kann eine Web Application Firewall ein *positives* oder ein *negatives Sicherheitsmodell* anwenden [KSBG 13]:

- Bei einem *positiven Sicherheitsmodell* werden die Daten mit *Whitelists* gefiltert, auf denen für jede URL der Webanwendung die jeweiligen Parameter mit den dafür gültigen Werten angegeben sind. Es ist also alles verboten, was nicht explizit durch eine Regel erlaubt ist. Dadurch können ungewöhnliche Anfragen und damit auch bisher unbekannte Angriffe erkannt werden.

- Ein *negatives Sicherheitsmodell* nutzt dagegen *Blacklists*, die bekannte Angriffsmuster enthalten. Dabei wird alles verboten, was explizit auf den Listen vorkommt, alles andere ist dagegen standardmäßig erlaubt. Dadurch können bereits bekannte Angriffe erkannt werden. Da bei einer SQL-Injection zwangsläufig SQL-Anweisungen eingesetzt werden, kann eine Anfrage zum Beispiel auf Schlüsselwörter wie `SELECT FROM` untersucht werden.

Um eine Website zuverlässig zu schützen, sollten beide Sicherheitsmodelle kombiniert werden, also eingehende Anfragen sowohl auf bekannte Angriffe als auch auf Anomlalien überprüft werden [KSBG 13].

Während Blacklists meist vorgefertigt vom Hersteller geladen werden können, müssen Whitelists individuell an die URLs und Parameter der zu schützenden Anwendung angepasst werden. Dazu können drei verschiedene Techniken angewendet werden [KSBG 13]:

- Eine Web Application Firewall kann selbst lernen, welche Anfragen ungefährlich sind, indem der normale Betrieb einer Website passiv überwacht und daraus Zugriffsregeln erstellt werden. Dieser Prozess kann aber einige Monate dauern, zudem müssen die generierten Regeln manuell überprüft werden. Außerdem können Angriffe, die während der Lernphase stattfinden, irrtümlich als Normalzustand angenommen und in das Regelwerk integriert werden.

- Außerdem können die Regeln als Resultat des *Vulnerability Assessment Process* entstehen, also eines fortlaufenden Prozesses, bei dem die Schwächen und Angriffsflächen einer Webanwendung ermittelt werden. Dadurch wird die Firewall auf den Schutz dieser spezifischen Angriffsflächen fokussiert.

- Die Regeln können aber auch manuell erstellt werden. Das ist allerdings die aufwendigste Methode, da jede URL und die möglichen Werte jedes Parameters einzeln defi-

niert werden müssen. Das macht sie für die meisten Anwendungsfälle impraktikabel.

Ein Problem, dass bei allen angewendeten Techniken auftritt, sind *False Positives*, also legitime Anfragen, die fälschlicherweise als Angriff gewertet werden. Deshalb muss eine Web Application Firewall eine einfache und schnelle Möglichkeit anbieten, um eine Regel aufzuweichen [Web 06]. Außerdem müssen die Regeln an alle Änderungen in der zu schützenden Anwendung angepasst werden, was einen fortlaufenden Arbeitsaufwand erforderlich macht [RoMa 13].

Eine Web Application Firewall analysiert nicht nur eingehenden, sondern auch ausgehenden Verkehr, um Datenlecks zu vermeiden. Beispielsweise sind Fehlermeldungen eines Webservers oft sehr detailliert und können einem Nutzer unbeabsichigt Informationen über die darunter liegende Infrastruktur geben [KSBG 13].

Ein weiterer häufiger Angriff auf Webanwendungen ist das sogenannte *Forceful Browsing*. Dabei werden URLs direkt im Browser angegeben, so dass auch Dateien und Dokumente zugänglich sind, auf die nicht über Hyperlinks zugegriffen werden kann [DPJV 06]. Dadurch können sensible Daten gestohlen werden, die auf dem Webserver liegen. Eine Web Application Firewall kann das verhindern, indem sie Links verschlüsselt und nur noch Zugriffe über einen verschlüsselten Link akzeptiert [Web 06].

Integration

Eine Web Application Firewall kann *host-based*, das heißt auf einem einzelnen Server installiert, oder *network-based* sein, also den Verkehr für ein ganzes Netzwerk überwachen [KSBG 13]. Dabei sind folgende Betriebsarten möglich [Web 06]:

- Bei der Integration als *Bridge* wird sie in die Verbindung eingeschaltet und liest den Netzverkehr mit. Dafür müssen keine Änderungen an der Konfiguration des Routings, des Webservers oder der bestehenden Firewall vorgenommen werden.
- Sie kann auch als *Router* fungieren. Dann muss das Netzwerk so konfiguriert werden, dass der Verkehr durch die Web Application Firewall geleitet wird.
- Beim Betrieb als *Reverse Proxy* (vgl. Abb. 2.11) wird sie zwischen Client und Webserver geschaltet. Nachdem die Anfrage den Paketfilter passiert hat, kommuniziert der Browser also mit der Web Application Firewall, die die Anfragen überprüft. Danach baut sie eine eigene Verbindung zum Server auf, leitet die Anfragen weiter und liefert die als Antwort erhaltenen Webseiten aus. Im Gegensatz zu einem Proxy bleibt hier dem Client statt dem Webserver die echte Adresse des Servers verborgen, dadurch muss er auch seinen Browser nicht entsprechend konfigurieren. Dafür muss die interne Netzwerkkonfiguration an die Nutzung eines Proxies angepasst werden.
- Im *embedded* Modus wird die Firewall auf dem Web Server als Plug-in installiert und erfordert daher keine Umstellung des Netzwerks.

Eine Web Application Firewall ist also eine Firewall, die auf der Anwendungsebene einzelne HTTP-Anfragen bezogen auf die URLs und Parameter einer Anwendung analysiert. Dabei werden Whitelists eingesetzt, bei denen ein bestimmtes Verhalten explizit erlaubt werden

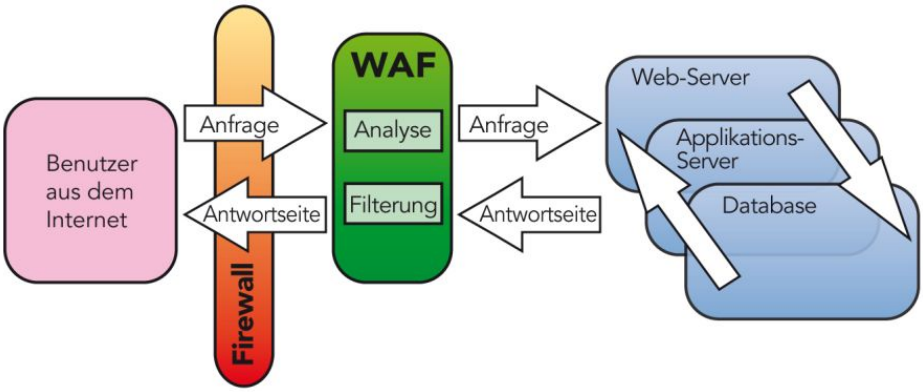

Abbildung 2.11.: Integration einer Web Application Firewall im Reverse Proxy Modus [Marx 09, S. 3] von Cirosec

muss. Dadurch werden ungewöhnliche Anfragen erkannt und als mögliche bisher unbekannte Angriffe identifiziert. Bekannte Angriffe wie SQL-Injection werden mit Hilfe von Blacklist blockiert, die bestimmte Anfragemuster explizit verbieten. Whitelists müssen an die zu schützende Anwendung angepasst und aktualisiert werden, falls sich diese verändert, wodurch ein hoher Arbeitsaufwand entsteht.
Web Application Firewalls können host-based oder network-based sein und haben unterschiedliche Betriebsmodi, von denen abhängt, ob die Netzwerkkonfiguration angepasst werden muss.

2.2.7. Intrusion Detection Systems (IDS) und Intrusion Prevention Systems (IPS)

Eine Firewall wendet ein Regelwerk an, um bestimmte Zugriffe zu blockieren. Ein System kann aber auch durch Malware, also Schadprogramme, durch Angriffe von außen oder durch autorisierte Nutzer, die ihre Privilegien missbrauchen, gefährdet werden. Um bekannte, aber auch unbekannte Gefahren zu entdecken, muss eine Firewall durch ein weiteres Sicherheitssystem ergänzt werden.

Ein *Intrusion Detection System* (*IDS*) überwacht die Vorgänge in einem System und analysiert sie auf mögliche *Security Incidents*, also Verletzungen oder drohende Verletzungen von im System geltenden Sicherheitsrichtlinien. Während ein IDS diese Sicherheitsvorfälle nur registriert, kann ein *Intrusion Prevention System* (*IPS*) diese auch verhindern. Das kann zum Beispiel durch das Beenden einer vom Angreifer genutzten Netzwerkverbindung oder einer Nutzersitzung geschehen, aber auch durch die Rekonfiguration der Firewall [ScMe 07].

Die folgenden Ausführungen beziehen sich auf Empfehlungen des *National Institute of Standards and Technology* nach Karen Scarfone und Peter Mell [ScMe 07]. Da diese sowohl auf Intrusion Detection Systems als auch auf Intrusion Prevention Systems zutreffen, wird der Begriff *Intrusion Detection and Prevention System* (*IDPS*) definiert, der beide dieser Systeme einschließt.

Methoden zur Erkennung von Security Incidents

Ein IDPS kann zur Erkennung von Security Incidents unterschiedliche Methoden nutzen. Dazu gehören *Signature Based Detection*, *Anomaly-Based Detection* und *Stateful Protocol Analysis*:

- *Signature Based Detection* ist ein Verfahren zur Erkennung bekannter Angriffe. Dabei werden *Signaturen*, also charakteristische Merkmale bekannter Security Incidents genutzt, um diese zu identifizieren. Damit können aber weder bisher unbekannte noch stark abgewandelte Angriffe erkannt werden.

- Die *Anomaly-Based Detection* erkennt auch bisher unbekannte Angriffe, indem sie den Normalzustand eines Systems mit dem tatsächlichen Zustand vergleicht und nach Anomalien sucht. Meist wird dieser Normalzustand in einer Trainingsphase erfasst und daraus Profile für verschiedene Anwendungen generiert, die mit Regelsätzen deren normales Verhalten definieren. Ein Problem dabei ist aber, dass Security Indidents, die in der Trainingsphase stattfinden, als normal betrachtet und in die Profile integriert werden können.

- Bei der *Stateful Protocol Analysis* werden vom Hersteller des IDPS vordefinierte Profile genutzt. Diese enthalten für jeden Zustand eines Protokolls generell akzeptierte Definitionen von gutartigem Verhalten. Eine FTP-Sitzung befindet sich zum Beispiel vor der Anmeldung im nicht autorisierten und danach im autorisierten Zusand. Während der Nutzer nach der Anmeldung einige unterschiedliche Befehle verwenden kann, wäre vor der Anmeldung lediglich die Verwendung von Kommandos wie das Aufrufen der Hilfe oder die Angabe von Nutzername und Passwort unverdächtig.
  Die Stateful Procol Analysis kann auch verdächtige Befehlssequenzen identifizieren und zwischen einzelnen Nutzern unterscheiden. Außerdem wird die angemessene Nutzung der verschiedenen Befehle überprüft, zum Beispiel die Länge der angegenen Parameter.
  Die Komplexität der Prüfungen und die Überwachung verschiedener Sitzungen ist allerdings sehr aufwendig. Zudem werden Angriffe nicht erkannt, die zwar unverdächtige Befehle verwenden, aber diese beispielsweise in sehr schneller Abfolge angeben und dadurch das System überlasten. Dazu kommt, dass Protokolle auf unterschiedlichen Systemen oft unterschiedlich implementiert sind und sich auch im Lauf der Zeit verändern. An diese Unterschiede und Veränderungen muss die Stateful Protocol Analysis angepasst werden.

Bei allen IDPS besteht zusätzlich das Problem der *False Positives* und *False Negatives*. *False Positives* sind normale Aktivitäten, die fälschlicherweise als Angriffe erkannt werden. *False Negatives* sind dagegen Angriffe, die nicht als solche erkannt werden. Werden False Positives zum Beispiel durch eine weniger strenge Definition des Normalzustandes reduziert, steigen die False Negatives dafür an und umgekehrt.

Typen von IDPS

IDPS werden anhand dessen, welche Ereignisse sie überwachen und wie sie installiert sind, in vier Typen unterteilt. Das sind *Network-Based*, *Wireless*, *Network Behavior Analysis* und *Host-Based*:

- Ein *Network-Based* IDPS überwacht den Verkehr für einen Netzwerkteil oder ein Gerät und untersucht Netzwerk- und Protokollaktivitäten auf verdächtiges Verhalten. Meist wird es an der Grenze zwischen verschiedenen Netzen installiert.

- Ein *Wireless* IDPS analysiert den Verkehr in drahtlosen Netzwerken. Allerdings kann es keine verdächtigen Aktivitäten in Protokollen der Anwendungsebene oder in höheren Protokollen wie TCP identifizieren, sondern nur in den Protokollen, die die drahtlose Übertragung selbst steuern.

- *Network Behavior Analysis* (*NBA*) untersucht ein Netzwerk auf ungewöhnliche Datenströme. Diese kommen zum Beispiel bei bestimmten Formen von Malware oder *DDoS*-Angriffen (*Distributed Denial of Service*) vor. Bei DDoS-Angriffen werden von verschieden Hosts aus Anfragen in hoher Frequenz an ein System geschickt, das dadurch überlastet wird und seine Dienste nicht mehr zur Verfügung stellen kann. Oft werden Systeme, die diese Technik anwenden, innerhalb des internen Netzwerkes einer Organisation installiert.

- Ein *Host-Based* IDPS ist auf einem einzelnen Host installiert, um die Ereignisse auf diesem zu überwachen. Dazu gehören der Netzwerkverkehr für diesen Host, Logdateien, laufende Prozesse, Dateizugriffe und Aktivitäten von Anwendungen. Meist werden solche Systeme auf kritischen Hosts wie Servern, die öffentlich zugänglich sind oder sensible Informationen verwalten, installiert. Beispiele für solche IDPS sind *AppArmor* und *OSSEC*, die in den nächsten beiden Abschnitten erläutert werden.

Zusammengefasst überwachen sowohl Intrusion Detection Systems als auch Intrusion Prevention Systems die Vorgänge in einem System. Ein IDS protokolliert und meldet Security Incidents aber nur, während ein IPS diese auch aktiv verhindert.
Die Erkennung bereits bekannter Angriffe erfolgt dabei über von Hersteller gelieferte Signaturen. Bisher unbekannte Security Incidents werden vom IDPS dagegen als Anomalien registriert, also Abweichungen vom Normalzustand des geschützten Systems, der zuvor definiert werden muss. Außerdem werden vom Hersteller vordefinierte Regeln genutzt, um Protokolle bezogen auf deren Zustände zu analysieren und ungewöhnliches Verhalten bestimmter Nutzer zu erkennen.
Außerdem werden IDPS danach unterschieden, ob sie einen ganzen Netzwerkteil überwachen oder nur auf einem einzelnen Host installiert sind. Außerdem gibt es spezialisierte Systeme für drahtlose Netzwerke und solche, die ungewöhnliche Datenflüsse in einem Netzwerk erkennen.

In den folgenden beiden Abschnitten werden *OSSEC* als Beispiel für ein Host-based Intrusion Detection System und *AppArmor* als Implementierung eines Host-based Intrusion Prevention System erläutert.

2.2.8. OSSEC (Hosted Intrusion Detection System)

OSSEC (*Open Source Security*) [Team 15] ist ein Beispiel für ein quelloffenes Host-based Intrusion Detection System, das auf den meisten gängigen Betriebssystemen, Linux eingeschlossen, läuft. Die Seite sectools.org listet *OSSEC* im Januar 2016 als eines der meistgenuzten Intrusion Detection Systems [Lyon 16]. Das Programm wurde ursprünglich von Daniel B. Cid entwickelt und 2009 von *Trend Micro* übernommen. Die folgenden Ausführungen beziehen sich auf das Buch *OSSEC Host-based Intrusion Detection Guide* von Bray et al. [BCH 08].

Das IDS bietet unter andererem die Analyse von Logdateien, die Integritätsüberprüfung von Dateien und die Erkennung von Rootkits. Rootkits sind Werkzeuge, die nach dem Einbruch in ein System installiert werden, um Malware vor der Entdeckung durch Antivirenprogramme zu schützen. Bekannte Rootkits können mit Hilfe von Signaturen identifiziert werden. Durch das Feature *Active Response* können, ausgelöst durch Security Incidents, bestimmte Kommandos ausgeführt werden, um den Angriff zu blockieren. Dadurch werden also auch Elemente eines IPS integriert.

OSSEC besteht aus mehreren Teilen. Das eigentliche IDS befindet sich auf dem *Manager* oder *Server*, so dass alle Einstellungen und Auswertungen zentral verwaltet werden können. Außerdem können so gleichzeitig mehrere Systeme überwacht werden. Dafür wird auf jedem zu schützenden System ein *Agent* installiert, der Informationen wie Logdateien sammelt und an den Manager weiterleitet. Manche Daten werden dabei in Echtzeit gesammelt, andere nur periodisch. Die Agents sind kleine Programme, die unter einem speziellen Nutzer mit geringen Privilegien laufen, um die Sicherheit nicht zu gefährden und darauf ausgelegt sind, die Ressourcen des Gastsystems nicht zu belasten. Falls kein Agent installiert werden kann, ist auch eine Variante ohne Agents möglich.

Erkennen von Security Incidents

Um zu erkennen, ob ein Ereignis ein Security Incident ist und ein Alarm ausgelöst werden sollte, analysiert OSSEC das Log zuerst in mehreren Schritten (vgl. Abb. 2.12).

Als erstes extrahiert der *Predecoder* die statischen Informationen aus bekannten Feldern eines Logs wie Zeit, Datum, Hostname, Programmname und Lognachricht. Listing 2.14 zeigt ein Log vor dem Predecoding. Auf dem Host `linux-server` hat am 14. April um 17:33 jemand versucht, sich mit dem ungültigen Nutzernamen `lcid` am SSH-Server anzumelden.

Listing 2.14: Log vor dem Predecoding nach [BCH 08]

```
Apr 14 17:33:09 linux_server sshd[1231]: Failed password for
invalid user lcid from 192.168.2.180 port 1799 ssh2
```

Aus diesem Log wird nach dem Predecoding die Tabelle 2.2. Als nächstes extrahiert der *Decoder* nicht-statische Informationen wie die IP-Adresse und den Nutzernamen, von denen die Anfrage kam, aus der Ausgabe des Predecodings. Aus dem vorher genannten Beispiel wird dann Tabelle 2.3.

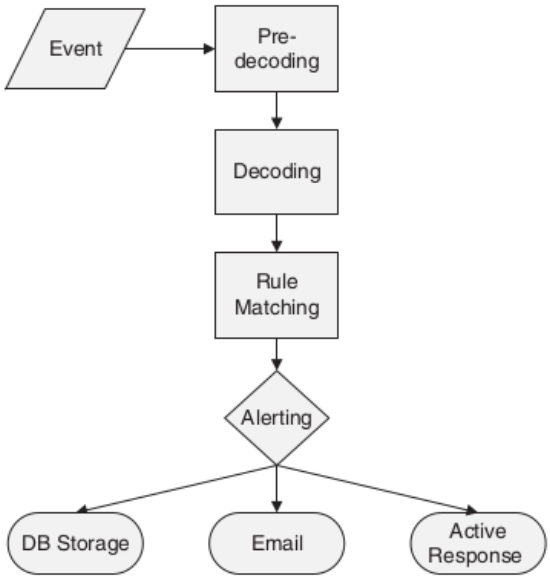

Abbildung 2.12.: Einordnung eines Ereignisses als Security Incident nach Bray et al. [BCH 08, S. 105]

Feld	**Beschreibung**
hostname	linux_server
program_name	sshd
log	Failed password for invalid user lcid from 192.168.2.180 port 1799 ssh2
time/date	Apr 14 17:33:09

Tabelle 2.2.: Das Log nach dem Predecoding nach Bray et al. [BCH 08]

Feld	**Beschreibung**
scrip	192.168.2.180
user	lcid

Tabelle 2.3.: Das Log nach dem Decoding nach Bray et al. [BCH 08]

Diese Informationen werden dann mit den *Regeln* verglichen, die für das System gelten. Diese Regeln sind XML-Dateien, die meist in `/var/ossec/rules/` liegen und in denen definiert ist, wann ein *Alert* ausgelöst wird. Das heißt, das eine Benachrichtigung in das Alert Log in `/var/ossec/logs/alerts.log` geschrieben oder zum Beispiel als E-Mail an den Administrator verschickt wird. Einige vordefinierte Regeln werden vom Hersteller mitgeliefert, es können aber auch Regeln vom Benutzer anhand der vom Decoder analysierten Logdateien erstellt werden. Die Erkennung von Security Incidents erfolgt also anomaly-based.

Listing 2.15 zeigt eine Regel, bei der für einen fehlgeschlagenen Loginversuch am SSH-Server wie im vorher genannten Beispiel ein Alert ausgelöst wird. Die Gruppe umfasst Regeln für `syslog` und `sshd`. Jede Regel braucht eine eindeutige ID und ein Level. Ein Level

ist eine vordefinierte Alarmstufe von 0 bis 15, zeigt also an, wie schwerwiegend der Vorfall ist. Das Programm muss vom Decoder als `sshd` erkannt worden sein, außerdem muss die Lognachricht die Zeichenkette `Failed password` enthalten. Zusätzlich kann eine kurze Beschreibung der Regel angegeben werden.

Listing 2.15: Beispiel für eine nutzerdefinierte Regel nach [BCH 08]

```
<group name="syslog,sshd,">
<rule id="100123" level="7">
<decoded_as>sshd</decoded_as>
<match>^Failed password</match>
<description>Failed SSHD password attempt</description>
</rule>
</group>
```

OSSEC ist als ein Host-based Intrusion Detection System, das mehrere Hosts zentral von einem Manager aus überwachen kann. Die Erkennung von Security Incidents erfolgt dabei anomaly-based, es werden also vom Hersteller gelieferte und benutzerdefinierte Regeln kombiniert, um ungewöhnliche Vorfälle im System zu erkennen. Dabei werden aus Logdateien zuerst statische Informationen wie Zeit, Hostname und Lognachricht, danach nichtstatische Informationen wie Nutzername und Quell-IP-Adresse extrahiert. Falls eine Regel mit diesen Informationen übereinstimmt, wird ein Alert ausgelöst.
Im folgenden Abschnitt wird AppArmor vorgestellt, das im Gegensatz zu OSSEC ein Intrusion Prevention System ist.

2.2.9. AppArmor (Hosted Intrusion Prevention System)

Linux implementiert standardmäßig eine *Discretionary Access Control* (*DAC*) [app 11]. Dabei werden jeder Datei Zugriffsrechte zugewiesen, also welche Nutzer oder Nutzergruppen eine Datei lesen, bearbeiten oder ausführen dürfen. Ein Prozess, der von einem Nutzer aufgerufen wird, arbeitet allerdings auch mit dessen Zugriffsrechten. Dadurch kann zwar der Zugriff bestimmter Nutzer auf eine Datei beschränkt, aber nicht zwischen einzelnen Programmen unterschieden werden. Das kann zum Problem werden, wenn ein Angreifer beispielsweise über Fehler im Code Kontrolle über die Anwendung erlangt. Er kann dann Schadcode mit den Rechten, unter denen das Programm läuft, ausführen, im schlechtesten Fall mit Superuser-Rechten [Kofl 14].

Um das zu verhindern, erweitert *AppArmor* (*Application Armor*) [app 15] das Sicherheitskonzept von Linux um eine *Mandatory Access Control* (*MAC*) . Das heißt, dass der Zugriff einzelner Anwendungen auf bestimmte Ressourcen wie Dateien oder das Netzwerk eingeschränkt werden kann. AppArmor ist also ein Host-based Intrusion Prevention System, bei dem ein Incident eine Regelüberschreitung einer Applikation ist.

Das IPS ist wurde ursprünglich von der Firma Immunix entwickelt. Inzwischen ist es in den Linux-Kernel integriert und läuft über die *Linux Security Module* Schnittstelle als Kernelmodul und ist in einigen Distributionen schon vorinstalliert [DKK$^+$ 14, app 11]. Im Vergleich

zu ähnlichen Systemen wie *SELinux* [sel 13] kommt AppArmor mit wesentlich weniger Regeln aus. Außerdem können eigene Regeln vom Nutzer einfacher erstellt werden, daher gilt AppArmor als leichter bedienbar. Allerdings werden die absoluten Pfade in den Regeln als Sicherheitsrisiko kritisiert, genauso wie das relativ kurze Regelwerk, bei dem standardmäßig weniger Programme geschützt werden [Kofl 14].

Profile und Betriebsmodi

Die Zugriffsregeln für die einzelnen Anwendungen werden in Profilen festgelegt. Zusätzlich zu den vordefinierten Regeln kann der Nutzer auch eigene hinzufügen. Für jedes Profil kennt AppArmor zwei Modi, den *Enforce Mode* und den *Complain Mode* [DKK+ 14, Cano 10b]:

- Im *Enforce Mode* werden die geladenen Profile durchgesetzt, also Anwendungen an bestimmten Zugriffen gehindert und der Zugriffsversuch registriert.
- Im *Complain Mode* werden Verletzungen der Regeln nur im `syslog` prokolliert. Dieser Modus kann zur Entwicklung verwendet werden, da hier das normale Verhalten eines Programms beobachtet und die Regeln entsprechend angepasst werden können.

Um AppArmor zu starten, stoppen oder neuzustarten kann das *init*-Skript `/etc/init.d/apparmor` genutzt werden [Cano 10b]. Mit dem Befehl `apparmor_status` oder `aa_status` kann der Status, also der aktuelle Betriebsmodus der geladenen Profile abgefragt werden. In Listing 2.16 gibt es 110 geladene vordefinierte Profile, davon sind 102 im Enforce- und 8 im Complain Mode. Insgesamt laufen 129 Prozesse, von denen 13 benutzerdefinierte Profile haben. Davon werden die Regeln bei 8 Prozessen durchgesetzt, bei 5 werden Übertretungen nur protokolliert [DKK+ 14, Cano 10a].

Listing 2.16: Beispielhafte Ausgabe von `aa_status` nach [Cano 10a]

```
module is loaded.
110 profiles are loaded.
102 profiles are in enforce mode.
8 profiles are in complain mode.
Out of 129 processes running:
13 processes have profiles defined.
8 processes have profiles in enforce mode.
5 processes have profiles in complain mode.
```

Über `aa-enforce <profil>` bzw. `aa-complain <profil>` kann ein bestimmtes Profil in den Enforce- oder den Complain-Modus geschaltet werden. Das Werkzeug `unconfined` ermittelt die bisher von AppArmor ungeschützten Dienste, indem es für alle Dienste mit offenen TCP-Ports überprüft, ob schon ein Profil vorhanden ist [DKK+ 14].

Erstellen von Profilen

Die einzelnen Profile befinden sich im Verzeichnis `/etc/apparmor.d/`. Die darin definierten Regeln sind standardmäßig Befugnisse, es wird dem Programm also etwas explizit

erlaubt. Versucht die Anwendung dagegen, auf eine Ressource zuzugreifen, für die keine Regel im Profil definiert ist, wird je nach Modus das Verhalten registriert bzw. der Zugriff verweigert. Diese Regeln können sich in Form von *Capabilities* auf Superuserprivilegien, aber auch auf Netzwerkzugriffe oder Dateien beziehen.

Listing 2.17 zeigt Auszüge aus dem vordefinierten Profil für den Druckserver `cupsd`. Das gesetzte Flag `complain` zeigt an, dass das Profil im Complain Mode ist. Die Capability `chown` erlaubt dem Server, den Eigentümer von Dateien zu ändern. Außerdem darf er Bluetooth nutzen und das Verzeichnis `/var/spool/cups` lesend und schreibend zugreifen [pro 13].

Listing 2.17: Auszug aus dem vordefinierten Profil für `cupsd` nach [Pitt 07]

```
/usr/sbin/cupsd flags=(complain) {
   capability chown,
   network bluetooth,
   /var/spool/cups/ rw,
}
```

Mit dem Werkzeug `aa_genprof` kann der Nutzer ein Profil für ein bestimmtes Programm erstellen. Dabei wird ein Profil erstellt und in den Complain Mode gesetzt. Danach muss der Nutzer das jeweilige Programm ausführen, so dass die Aktivitäten geloggt werden können. Dieses Verhalten wird dann als Standard angenommen. Die ausgeführten Aktivitäten werden dann ins Profil als Regeln übernommen, so dass die Anwendung alles tun darf, was ihrem Standardverhalten entspricht, ungewöhnliches Verhalten aber registriert bzw. verhindert wird. Abschließend kann der Nutzer die erstellten Regeln überprüfen. Zu wenige Befugnisse können dabei zu Fehlern im geschützten Programm führen, so dass nachgebessert werden muss. Zu viele Befugnisse machen die Sicherheitsvorkehrungen dagegen wirkungslos [DKK+ 14].

AppArmor erweitert also die auf Dateien und Nutzer bezogene Discretionary Access Control von Linux und implementiert eine Mandatory Access Control. Dadurch können einzelne Anwendungen überwacht und Zugriffe verweigert werden, wenn sie nicht im Profil der Anwendung explizit erlaubt sind. Dabei können vordefinierte Profile geladen und vom Nutzer ergänzt werden. Während die Regeln im Enforce Mode auch durchgesetzt werden, werden Verstöße im Complain Mode nur protokolliert, was unter anderem zum Erstellen neuer Profile nützlich sein kann. Durch die Aufzeichnung des normalen Verhaltens eines Programms wird klar, welche Zugriffsrechte es für den fehlerfreien Betrieb braucht.

2.3. Zusammenfassung

In diesem Kapitel wurden zuerst die Grundlagen der Gamification erläutert. Dabei wurde der Begriff definiert, verschiedene Theorien zur Spielermotivation und Designprinzipien beleuchtet und einige Einsatzbereiche von Gamification, aber auch Kritikpunkte vorgestellt. Zusätzlich wurden mehrere Anwendungsbeispiele in der Informatik beschrieben.

Im zweiten Teil wurden die Grundlagen der Serveradministration behandelt, die mit Hilfe von Gamification vermittelt werden sollen. Dazu gehören

- Cronjobs zur Automatisierung wiederkehrender Aufgaben,
- das Network File System zum Einhängen entfernter Systeme in den Dateibaum und
- Secure Shell zum Arbeiten in der Kommandozeile auf einem entfernten System.

Zur Absicherung eines Servers werden zusätzlich folgende Werkzeuge verwendet:

- Die TCP-Wrapper-Bibliothek kontrolliert den Zugriff für bestimmte Hosts.
- Firewalls filtern den Verkehr für ganze Netzwerke.
- Web Application Firewalls schützen speziell Webanwendungen, indem sie den ein- und ausgehenden Verkehr auf Anwendungsebene analysieren.
- Intrusion Detection Systems wie OSSEC ergänzen die Firewalls, indem sie bekannte und bisher unbekannte Angriffe erkennen.
- Intrusion Prevention Systems wie AppArmor können diese zusätzlich verhindern.

3. Szenario LRZ

Im folgenden Kapitel wird zuerst das Leibniz-Rechenzentrum beschrieben, wobei der Fokus auf den Tätigkeitsbereichen liegt, die mit der Administration von Linux-Servern zusammenhängen. Es dient in dieser Arbeit als Anwendungsbeispiel für Wissensvermittlung mit Hilfe von Gamification, indem eine Webanwendung entwickelt wird, die den Serveradministratoren für ihre Aufgabe nötige Grundlagen vermitteln soll. Danach wird auf die Herausforderungen beim Serverbetrieb am LRZ eingegangen. Außerdem werden die Anforderungen aufgezeigt, die sich aus diesen Herausforderungen an eine solche Anwendung ergeben.

3.1. Beschreibung des LRZ

Das *Leibniz-Rechenzentrum der Bayerischen Akademie für Wissenschaften* (*LRZ*) [LR b] in Garching bei München wurde 1964 zur Forschung auf dem Feld der Informatik gegründet [LR 14]. Es betreibt unter anderem einen Hochleistungsrechner, auf dem wissenschaftliche Arbeiten aller Fachrichtungen ausgeführt werden. Außerdem stellt es für die Ludwig-Maximilians-Universität (LMU) und die Technische Universität München (TUM) Dienste wie E-Mail und Webservices, aber auch digitale Lehrangebote zur Verfügung. Zudem ist es für die Pflege des Münchner Wissenschaftsnetzes zuständig. Dieses verbindet die Münchner Hochschulen und Forschungseinrichtungen wie die Max-Planck-Institute, aber auch Studentenwohnheime untereinander und mit dem Internet [LR a].

Um diese Angebote zur Verfügung stellen zu können müssen unter anderem Mail- und Webserver, aber auch DNS-Server und Firewalls betrieben werden. Im März 2016 nutzt das LRZ für diesen Zweck mehr als 1000 virtuelle Linux-Server, die von mehr als 70 Administratoren betreut werden. Diese sind aber meist keine hauptberuflichen Serveradministratoren, sondern wissenschaftliche Mitarbeiter, die auch forschen oder für die Lehre an LMU und TUM zuständig sind [LR 15].

3.2. Herausforderungen

Da das LRZ die Daten und Webauftritte mehrerer Hochschulen verwaltet und viele Netzwerkdienste anbietet, stellt es ein potenzielles Angriffsziel dar. Ein Ausfall dieser Dienste würde sehr viele Nutzer betreffen, zum Beispiel waren in München im Wintersemester 2014/15 mehr als 115.000 Studenten eingeschrieben [Stat 16]. Da auch personenbezogene Daten gespeichert werden, müssen Sicherheitsanforderungen erfüllt werden, um diese zu

schützen. Zusätzlich wird oft eine spezifische, auf die Anforderungen des LRZ zugeschnittene Serverkonfiguration benötigt. Da sich die technischen Möglichkeiten ständig weiterentwickeln, müssen die Server auch regelmäßig gewartet und die Konfiguration angepasst werden.

Allerdings werden die Server am LRZ meist von wissenschaftlichen Mitarbeitern verwaltet, die zusätzlich noch weitere Aufgaben haben. Diese haben normalerweise auf zwei Jahre befristete Verträge, müssen also schnell eingearbeitet werden. Außerdem wird ein Linux-Server oft deutlich länger betrieben, so dass die Betreuung eines Servers auch zwischen den Mitarbeitern weitergegeben wird. Dabei muss sichergestellt werden, dass die Server ordentlich übergeben und neue Administratoren geschult werden, da es sonst zu Konfigurationsfehlern kommen kann, die im schlechtesten Fall zu Ausfällen oder Sicherheitslücken führen können.

Zusammengefasst muss das LRZ aus Datenschutzgründen bestimmte Sicherheitsanforderungen erfüllen, die sich mit den technischen Möglichkeiten verändern. Zudem bietet das LRZ viele Dienste an, die eine spezifische Konfiguration erfordern. Die Erfüllung dieser Aufgaben wird aber durch eine hohe Fluktuation bei den Mitarbeitern erschwert.

3.3. Anforderungen

Um diesen Herausforderungen gerecht zu werden, wird in dieser Arbeit eine Anwendung entwickelt, die am LRZ mit Hilfe von Gamification Wissen über Linux-Serveradministration vermitteln soll. Im Folgenden werden die gamificationspezifischen, inhaltlichen, funktionalen und nichtfunktionalen Anforderungen an diese Anwendung beschrieben.
Die gamificationspezifischen Anforderungen erschließen sich aus den Designprinzipien der Gamification aus Abschnitt 2.1, während sich die inhaltlichen Anforderungen auf Abschnitt 2.2 beziehen. Darauf folgen die funktionalen und nichtfunktionalen Anforderungen, die sich unter anderem aus den Herausforderungen im vorhergehenden Abschnitt ergeben.

3.3.1. Gamificationspezifische Anforderungen

Das oft als trocken wahrgenommene Fachwissen soll mit Hilfe von Gamification interessanter und anschaulicher gemacht werden. Dadurch können Mitarbeiter schneller eingearbeitet werden und die Übergabe eines Servers an einen anderen Administrator wird erleichert. Die im Folgenden genannten Anforderungen ergeben sich aus den Designprinzipien der Gamification aus Abschnitt 2.1.

[GAM 1] Personalisierung Jeder Nutzer soll ein Profil haben, das er personalisieren kann. Dadurch hat der Nutzer das Gefühl, selbstbestimmt handeln und die Anwendung an seine Bedürfnisse anpassen zu können.

[GAM 2] Übersichtlichkeit Um die Anwendung übersichtlicher zu gestalten und den eigenen Fortschritt für den Nutzer ersichtlicher zu machen, soll das zu vermittelnde Wissen in Themengebiete aufgeteilt werden.

[GAM 3] Sofortiges Feedback Die Anwendung soll Aufgaben enthalten, in denen der Nutzer sein erworbenes Wissen anwenden kann. Nach dem Lösen einer Aufgabe soll der Nutzer sofortiges Feedback erhalten.

[GAM 4] Lernfortschritte Der Lernfortschritt eines Nutzers soll für ihn auf seinem Profil durch Oberflächenelemente wie Levels, Badges, Punkte oder Fortschrittsbalken sichtbar sein. Auch an den Units und Lessons soll klar erkennbar sein, ob sie vom Nutzer[1] schon bearbeitet worden sind. Außerdem sollen die Nutzer eine Möglichkeit haben, ihre Leistungen untereinander zu vergleichen.

[GAM 5] Selbstbestimmung Die einzelnen Themengebiete sollen in beliebiger Reihenfolge bearbeitbar und die Bearbeitung unterbrechbar sein. Das erlaubt dem Nutzer einen hohen Grad der Selbstbestimmung.

3.3.2. Inhaltliche Anforderungen

Die Anwendung soll den Linux-Serveradministratoren des LRZ die spezifischen Konfigurationen vermitteln, die benötigt werden, um verschiedene Dienste und Kommunikationsinfrastruktur zur Verfügung zu stellen und abzusichern. Die Grundlagen dafür wurden in Abschnitt 2.2 erläutert.

Administration

Die Anforderungen in diesem Abschnitt beschäftigen sich mit dem Erlernen allgemeiner Werkzeuge für die Linux-Serveradministration.

[CON 1] Cronjobs Cronjobs sind Aufgaben, die vom Betriebssystem regelmäßig zu einer bestimmten Zeit ausgeführt werden. Sie sind bei der Automatisierung von einfachen, wiederkehrenden Aufgaben in der Administration hilfreich.

[CON 2] Secure Shell (SSH) Secure Shell (SSH) ist ein Protokoll, mit dessen Hilfe auf der Kommandozeile eines entfernten Systems gearbeitet werden kann. Es gilt als sichere Alternative zu vergleichbaren Diensten wie telnet, da es die Verbindung verschlüsselt.

[1] Die Mitarbeiter des LRZ werden im Folgenden im Zusammenhang mit der Anwendung als Nutzer bezeichnet.

[CON 3] Network File System (NFS) Network File System (NFS) ermöglicht die Einbindung eines entfernten Verzeichnisses in den lokalen Dateibaum. Dadurch können Daten zentral auf einem Server bereit gestellt, aber wie in einem lokalen Verzeichnis genutzt werden.

Sicherheit

In diesem Abschnitt liegt der Fokus auf der Absicherung von Linux-Servern gegen unberechtigte Zugriffe und Angriffe.

[CON 4] TCP Wrapper TCP Wrapper ist eine Bibliothek, die Zugriffslisten verwaltet. Damit kann bestimmten Clients der Zugriff auf einen Serverdienst verweigert oder umgekehrt nur ausgesuchten Nutzern der Zugriff erlaubt werden.

[CON 5] Firewalls Firewalls schränken den Zugriff auf ein Netzwerk oder einen einzelnen Rechner ein. Meist schützen sie ein lokales Netz vor Zugriffen aus dem Internet. Der eingehende Verkehr wird dazu anhand eines Regelwerkes überprüft.

[CON 6] Web Application Firewalls Web Application Firewalls sind spezielle Firewalls, die auf den Schutz von Webservern vor Angriffen ausgelegt sind. Sie arbeiten auf der Anwendungsebene und untersuchen jedes eingehende Paket auf bekannte Angriffssignaturen oder Anomalien.

[CON 7] Intrusion Detection Systems (IDS) und Intrusion Prevention Systems (IPS) Ein Intrusion Detection System (IDS) überwacht alle Vorgänge in einem System und analysiert sie auf mögliche sicherheitsrelevante Vorfälle. Ein Intrusion Prevention System (IPS) kann diese Vorfälle im Gegensatz zum IDS nicht nur melden, sondern auch verhindern.

[CON 8] OSSEC (Hosted Intrusion Detection System) OSSEC (Open Source Security) ist ein Beispiel für ein Intrusion Detection System, das auf einem Host installiert ist und diesen überwacht. Es analysiert beispielsweise Logdateien, überprüft die Integrität von Dateien und kann bekannte Rootkits erkennen.

[CON 9] AppArmor (Hosted Intrusion Prevention System) AppArmor (Application Armor) ist ein Beispiel für ein Hosted IPS und erweitert die Sicherheitsarchitektur von Linux um eine Mandatory Access Control. Das heißt, dass nicht nur der Zugriff einzelner Nutzer, sondern auch der Zugriff bestimmter Anwendungen auf Ressourcen wie Dateien oder das Netzwerk eingeschränkt werden kann.

3.3.3. Funktionale Anforderungen

Zusätzlich zu den zuvor genannten inhaltlichen und gamificationspezifischen Anforderungen sollen folgende technischen Anforderungen erfüllt werden:

[FUN 1] Login Die Nutzer sollen sich mit ihrer LRZ-SIM-Kennung einloggen können, so dass sie sich nicht registrieren und auch kein eigenes Passwort vergeben müssen. Durch den geringeren Aufwand für den Nutzer soll die Akzeptanz der Anwendung erhöht werden.

[FUN 2] Profil Jeder Nutzer soll außerdem ein Profil mit Pseudonym haben, das spezifisch für die Anwendung ist. Dadurch kann abgespeichert werden, welche Lerneinheiten die Nutzer bereits absolviert haben, so dass die Nutzer ihren Leistungsstand überprüfen können. Zudem kann dadurch die gamificationspezifische Anforderung **[GAM 1] Personalisierung** umgesetzt werden.

[FUN 3] Administratorkonto Der Administraror des Programms soll ein eigenes Konto haben, von dem aus er die Möglichkeit hat, auf alle Nutzerdaten zuzugreifen. So kann er zum Beispiel die tatsächliche Nutzung und Akzeptanz der Anwendung überprüfen oder Lerneinheiten hinzufügen oder ändern.

Nach dem Durcharbeiten der Lerneinheiten sollen die Nutzer im Stande sein, einen Linux-Server nach den Anforderungen des LRZ zu konfigurieren. Die Motivation der Nutzer und dadurch auch der Lernerfolg soll mit Hilfe von Gamification erhöht werden.

3.3.4. Nichtfunktionale Anforderungen

Zusätzlich zu den funktionalen Anforderungen sollen die folgenden nichtfunktionale Anforderungen erfüllt werden, die sich ebenfalls aus den zuvor genannten Herausforderungen ergeben.

[NFT 1] Korrektheit Die Nutzereingaben sollen korrekt verarbeitet werden. So soll die Lösung eines Nutzers korrekt bewertet werden und auf eine gleiche Eingabe immer die gleiche Rückmeldung folgen.

[NFT 2] Umgebung und Wartbarkeit Die Anwendung soll webbasiert sein und im Intranet des LRZ verfügbar sein, damit die Nutzer mit einem Browser darauf zugreifen können, ohne spezielle Technologien installieren zu müssen. Dadurch, dass das Programm zentral auf einem Webserver läuft, kann es außerdem leichter gewartet und aktualisiert werden.

[NFT 3] Sicherheit Zum Schutz der Nutzerdaten muss die Kommunikation zwischen Browser und Anwendung verschlüsselt ablaufen. Außerdem soll das Programm zum Schutz vor Angriffen nur im Intranet des LRZ verfügbar sein.

[NFT 4] Vertraulichkeit Aus Datenschutzgründen sollen die Nutzer keinen Zugriff auf die Profile anderer Nutzer haben. Daten, die andere Nutzer einsehen können, sollen außerdem nur mit einem Pseudonym angezeigt werden.

[NFT 5] Erweiterbarkeit Um an technische Weiterentwicklungen und Veränderungen angepasst werden zu können, soll der vermittelte Inhalt der Anwendung zudem erweiterbar sein. Es soll also einfach sein, neue Lerneinheiten hinzuzufügen und veraltete Lernmaterialien zu aktualisieren.

[NFT 6] Bedienbarkeit Das Programm soll intuitiv bedienbar sein, ohne vorher eine Anleitung zu lesen.

3.4. Übersicht der Anforderungen

In der folgenden Tabelle werden die in diesem Kapitel genannten Anforderungen aufgelistet und in Muss- und Kann-Anforderungen unterteilt. Die Gewichtung ist dabei folgende:

- + für Muss-Anforderungen, die für die Nutzung der Anwendung unabdingbar sind
- 0 für Kann-Anforderungen, die zwar wünschenswert, aber optional sind

Da es sich bei der Anwendung um einen Prototypen handelt, liegt der Fokus vorerst auf den Anforderungen, die für die Administratoren des LRZ den größten Nutzen haben. Da die Anwendung nach **[NFT 5]** erweiterbar sein soll, kann der Administrator zusätzliche Aufgaben hinzufügen. Daher enthalten die inhaltlichen Anforderungen viele Kann-Anforderungen, während die restlichen Anforderungen eine höhere Priorität besitzen und daher als Muss-Anforderungen eingestuft werden.

Im nächsten Kapitel werden die genannten gamificationspezifischen, inhaltlichen, funktionalen und nichtfunktionalen Anforderungen in das Design der Anwendung integriert.

Tabelle 3.1.: Anforderungsübersicht mit Gewichtungen

ID	Anforderung	Gewichtung
Gamificationspezifische Anforderungen		
[GAM 1]	Personalisierung	+
[GAM 2]	Übersichtlichkeit	+
[GAM 3]	Sofortiges Feedback	+
[GAM 4]	Lernfortschritt	+
[GAM 5]	Selbstbestimmung	+
Inhaltliche Anforderungen		
[CON 1]	Cronjobs	0
[CON 2]	Secure Shell (SSH)	+
[CON 3]	Network File System (NFS)	0
[CON 4]	TCP Wrapper	+
[CON 5]	Firewalls	0
[CON 6]	Web Application Firewalls	0
[CON 7]	Intrusion Detection/Prevention Systems	0
[CON 8]	OSSEC (Hosted Intrusion Detection System)	0
[CON 9]	AppArmor (Hosted Intrusion Prevention System)	+
Funktionale Anforderungen		
[FUN 1]	Login	+
[FUN 2]	Profil	+
[FUN 3]	Administratorkonto	+
Nichtfunktionale Anforderungen		
[NFT 1]	Korrektheit	+
[NFT 2]	Umgebung und Wartbarkeit	+
[NFT 3]	Sicherheit	+
[NFT 4]	Vertraulichkeit	+
[NFT 5]	Erweiterbarkeit	+
[NFT 6]	Bedienbarkeit	+

4. Design der Anwendung

Im folgenden Kapitel wird das Design der webbasierten Anwendung beschrieben, die die Serveradministratoren des LRZ bei der Konfiguration unterstützen soll. Dabei werden die Spieldesignprinzipien erläutert, die eingesetzt werden, um die Wissensvermittlung zu erleichtern und unterhaltsamer zu machen. Darauf folgt der Inhalt der gamifizierten Anwendung, also die Lektionen, die die Nutzer erlernen sollen, und das Design als Umsetzung der funktionalen und nichtfunktionalen Anforderungen aus Kapitel 3. Abschließend werden das Oberflächendesign der Anwendung und das Administratorinterface beschrieben.

4.1. Verwendete Technologien

Nach **[NFT 2]** soll die Anwendung webbasiert sein. Sie wird mit dem auf der Programmiersprache Python basierenden Framework Django implementiert und läuft auf einem Apache-Webserver unter Debian. Die Authentifizierung der Nutzer erfolgt über LDAP, zur Verschlüsselung der Verbindung wird TLS verwendet.

4.1.1. Python

Die quelloffene Skriptsprache *Python* [Pyth] läuft auf allen gängigen Betriebssystemen und weist ein hohes Abstraktionsniveau auf. Dadurch ist der Programmcode oft kürzer als bei vergleichbaren Programmiersprachen. Außerdem hat Python eine große Standardbibliothek und unterstützt mehrere Paradigmen, darunter auch das objektorientierte [Pyth 16].

4.1.2. Django

Django [Djan 16a] ist ein quelloffenes Web Application Framework, das in Python geschrieben ist. Der Fokus des Frameworks liegt auf schneller Entwicklung, Skalierbarkeit und Sicherheit.

Die Absicherung einer Webanwendung unterstützt Django beispielsweise dadurch, dass es automatisch vor SQL-Injection Angriffen, Cross-Site-Scripting und Cross-Site-Request-Forgery schützt [Djan 16c, Djan 16d]:

- Vor *SQL-Injection* Angriffen, also dem Einschleusen von SQL-Befehlen in die Datenbank über Eingabeparameter, schützt Django, indem es SQL-Befehle in Eingabedaten

automatisch maskiert.

- Beim *Cross-Site-Scripting* schleußt ein Angreifer clientseitige Scripts in eine Website ein. Wird diese aufgerufen, werden die Scripts vom Browser auf dem Clientsystem ausgeführt, wodurch der Angreifer zum Beispiel sensible Daten stehlen kann. Deshalb maskiert Django in Variablen, die solche Scripts enthalten könnten, automatisch bestimmte Zeichen, so dass sie vom Browser nicht mehr interpretiert werden.

- Ein Angreifer nutzt *Cross-Site-Request-Forgery*, um einen authentisierten Nutzer einer Webanwendung dazu zu bringen, bestimmte Handlungen auszuführen. Zum Beispiel schickt der Angreifer per E-Mail einen Link an den Nutzer, der beim Anklicken das Passwort des Nutzers ändert. Um dies zu verhindern, akzeptiert Django Formulareingaben nur, wenn vom Clientbrowser ein CSRF-Token mit angegeben wird, also eine Zahl, die zufällig generiert und diesem über einen Cookie mitgeteilt wird.

Durch diesen Schutz werden häufige Fehler bei der Programmierung vermieden, die eine Webseite angreifbar machen können.

Die Entwicklung einer Webanwendung wird auch dadurch vereinfacht, dass in der Installation eine SQLite-Datenbank und ein für die Entwicklung geeigneter Webserver enthalten ist. Außerdem existieren viele Erweiterungen, die eingebunden werden können. In dieser Arbeit werden das Authentisierungs-Backend `django-auth-ldap` [Sage 09] zur Authentisierung der Nutzer über LDAP und das Modul `django-tinymce` [Cass 16] verwendet. Letzteres integriert den quelloffenen Editor *TinyMCE* [Epho 16] in Django. TinyMCE vereinfacht das Formatieren von Texten im Browser für den Benutzer.

In dieser Arbeit wurde Django aufgrund seines integrierten Schutzes vor häufigen Angriffen und der schnellen, einfachen Entwicklung für die Implementierung der Webanwendung ausgewählt. Außerdem wird Django am LRZ bereits genutzt, die Anwendung lässt sich also gut in die bestehende Infrastruktur integrieren.

4.1.3. LDAP

Da sich die Mitarbeiter mit iher LRZ-SIM-Kennung einloggen sollen und keine Registrierung erforderlich sein soll, erfolgt die Authentisierung über *LDAP* (*Lightweight Directory Access Protocol*) . Mit dem Protokoll können Informationen aus entfernten Verzeichnissen abgefragt und verändert werden. Das LRZ stellt LDAP-Server zur Authentifizierung zur Verfügung.

4.1.4. SQLite

SQLite [Hipp 16b] ist eine Programmbibliothek, die eine Datenbank-Engine implementiert. Im Gegensatz zu anderen Datenbanken wird SQLite also direkt in ein Programm eingebunden, statt in einem eigenen Serverprozess zu laufen. Durch die Beschränkung auf notwen-

dige Funktionalitäten ist die quelloffene Bibliothek zudem sehr kompakt. Im März 2016 ist SQLite die am weitesten verbreitete Datenbank der Welt [Hipp 16a].

4.1.5. Bootstrap

Das freie CSS-Framework *Bootstrap* [OtTh 15] wurde ursprünglich von Twitter entwickelt und ist eines der beliebtesten Frameworks seiner Art. Es vereinfacht die Entwicklung von Weboberflächen mit HTML, CSS und JavaScript, wobei der Fokus auf der Entwicklung von responsiven Webseiten, die auf allen Geräten nutzbar sind, liegt. Dazu werden unter anderem häufig verwendete Oberflächenkomponenten wie Navigationsmenüs zur Verfügung gestellt.

4.1.6. Apache HTTP-Server

Der *Apache HTTP-Server* [Apac 16] ist ein quelloffener Webserver, der im März 2016 der meistgenutzte Webserver auf dem Markt ist [Netc 16]. Der flexible, erweiterbare HTTP-Server läuft auf allen gängigen Betriebssystemen und ist in den meisten Linux-Distributionen wie auch Debian standardmäßig enthalten.

4.1.7. TLS

Der Webserver ist konfiguriert, die Verbindung mit *TLS* (*Transport Layer Security*) zu verschlüsseln. In diesem Fall wird dazu die quelloffene Implementierung *OpenSSL* (*Open Secure Socket Layer*) verwendet, die auf dem Apache Server standardmäßig installiert ist.

4.1.8. Debian GNU/Linux

Debian GNU/Linux [Soft 16] ist eine freies Betriebssystem, das den Linux-Kernel verwendet und im Serverbereich am LRZ weit verbreitet ist. Es ist auf Kompaktheit und Stabilität ausgelegt, bietet eine große Auswahl an installierbaren Paketen und unterstützt die gängige Hardware.

4.2. Eingesetzte Spieldesignprinzipien

Aus den gamificationspezifischen Anforderungen aus Abschnitt 3.3.1 ergeben sich die folgenden Spieldesignprinzipien, die eingesetzt werden, um die Anwendung interessanter machen und die Nutzer zu motivieren.

[GAM 1] Pseudonym Für jeden Nutzer wird bei der ersten Anmeldung ein Profil erstellt. Der Nutzer kann dieses personalisieren, indem er sich einen Nickname, also ein Pseudonym, aussucht.

[GAM 2] Lerneinheiten Um die Anwendung übersichtlicher zu gestalten und den Lernerfolg für den Nutzer ersichtlich zu machen, werden die einzelnen Aufgaben (*Lessons*) in Lerneinheiten (*Units*) aufgeteilt. Diese Units beschäftigen sich jeweils mit einem Themenbereich aus Kapitel 2.2, der auch eine inhaltliche Anforderung darstellt, wie zum Beispiel **[CON 2] SSH**.

[GAM 3] Rückmeldung Jede Lesson hat einen Einführungstext mit einer Erklärung und einer kleinen Aufgabe, die dem Nutzer gestellt wird. Dadurch kann er sein erworbenes Wissen direkt anwenden und überprüfen. Löst der Nutzer die Aufgabe, erhält er eine Meldung mit positivem Feedback. Falls seine Lösung falsch ist, erhält er eine entsprechende Meldung, in der auch ein Lösungshinweis enthalten ist.

[GAM 4] Oberflächenelemente Hat der Nutzer alle Lessons einer Unit richtig gelöst, erhält er ein Badge. Dieses wird in seinem Profil angezeigt und unterstützt ihn dabei, seinen Fortschritt einzuschätzen und motiviert zu bleiben. Während vollständige Units als Badges angezeigt werden, wird der Fortschritt innerhalb einer unvollständig bearbeiteten Unit als Fortschrittsbalken angezeigt. Dadurch kann der Nutzer seinen Wissensstand auch innerhalb einer Unit nachvollziehen. Auch in den Listen- und Einzelansichten der Lessons und Units wird angezeigt, ob diese schon bearbeitet worden sind.
Außerdem gibt es ein Leaderboard, auf dem die Nicknames der Nutzer mit ihren Badges angezeigt werden. So können die Nutzer ihre Leistungen vergleichen und sich miteinander messen.

[GAM 5] Badges Gegenüber Levels als Fortschrittsanzeige haben Badges den Vorteil, dass sie keine festgelegte Reihenfolge haben. Die Nutzer können also selbst entscheiden, welche Units sie zuerst bearbeiten und müssen auch eine angefangene Unit nicht abschließen, bevor sie die nächste beginnen können. Dadurch erlaubt die Anwendung den Nutzern einen hohen Grad an Selbstbestimmung.

Diese Designelemente sollen das Erlernen von Fachwissen über die Konfiguration von Linux-Servern erleichtern und interessanter gestalten.

4.3. Inhalt

Der folgende Abschnitt beschäftigt sich mit dem Inhalt der Anwendung, der auf Kapitel 2.2 aufbaut und in den inhaltlichen Anforderungen 3.3.2 festgelegt ist. Wie im vorhergehenden Abschnitt 4.2 erläutert, ist der Inhalt in Lessons und Units aufgeteilt, wobei sich die Units jeweils mit einer inhaltlichen Anforderung beschäftigen.

[CON 1] Secure Shell (SSH) Das Thema SSH wird in zwei Units aufgeteilt, wobei die erste Unit allgemeine Informtationen zum Thema abdeckt und die zweite für das LRZ

spezifische Aufgaben enthält. Die erste Unit beschäftigt sich vor allem mit der Konfiguration des Clients über Kommandozeilenparameter und Konfigurationsdateien. Abbildung 4.1 zeigt eine Lesson aus dieser Unit.

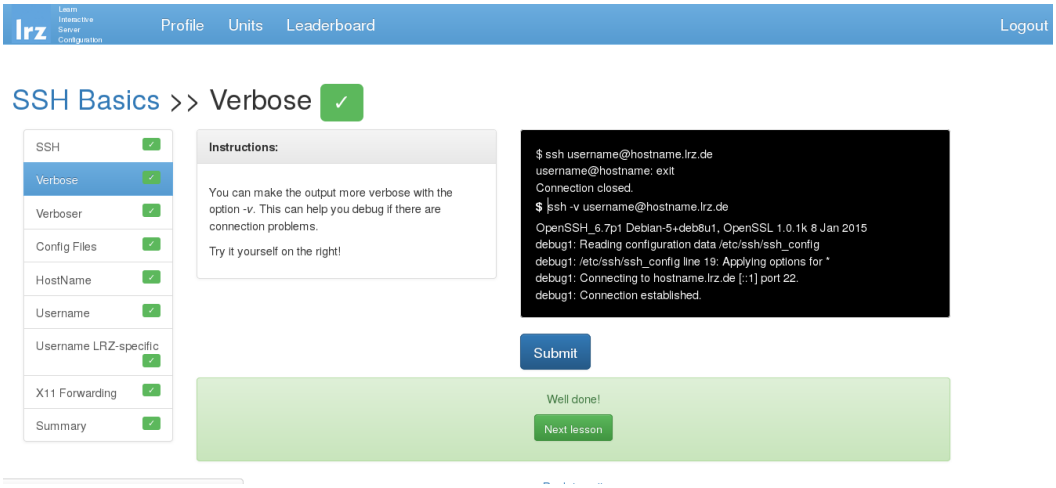

Abbildung 4.1.: Eine Lesson zum Thema SSH

[CON 3] TCP Wrapper Die Unit zum Thema TCP Wrapper dreht sich vor allem um das Zulassen und Blockieren von Zugriffen auf Serverdienste bestimmter Clients mit Hilfe der beiden Konfigurationsdateien. Außerdem liegt der Fokus auf der Vermittlung einer sicheren Grundkonfiguration, die nicht explizit erlaubte Zugriffe grundsätzlich unterbindet.

[CON 15] AppArmor (Hosted Intrusion Prevention System) Die Lerneinheit, die sich mit AppArmor beschäftigt, enthält Aufgaben zur Erstellung von Profilen für bestimmte Anwendungen. Auch hier wird Wert auf das Erlernen einer sicheren Grundkonfiguration eines Systems gelegt.

Da es sich bei der Anwendung um einen Prototypen handelt, werden vorerst nur die Muss-Anforderungen umgesetzt. Das Hinzufügen neuer Units und Lessons ist aber wie in **[NFT 5]** definiert durch den Administrator jederzeit möglich.

4.4. Design

Zusätzlich zu den gamificationspezifischen und inhaltlichen Designelementen werden im Folgenden die technischen Anforderungen in Abschnitt 4.1 und die nichtfunktionalen Anforderungen in Abschnitt 3.3.4 umgesetzt.

[FUN 1] Login Wie bereits in Abschnitt 4.1 erläutert, erfolgt die Authentisierung über LDAP. In diesem Fall liegen die Nutzerdaten auf einem zentralen Server statt in der Datenbank der Anwendung, müssen also nicht vom Nutzer eingegeben werden.

[FUN 2] Profil Bei der ersten Anmeldung wird für jeden Nutzer ein eigenes Profil erstellt. Dabei kann er sich ein eindeutiges Pseudonym aussuchen und entscheiden, ob seine Leistungen im Leaderboard angezeigt werden sollen. Diese Einstellungen kann er später jederzeit ändern.

[FUN 3] Administratorkonto Zusätzlich zu den normalen Benutzerkonten gibt es Konten für die Administratoren, die sich über ein separates Interface einloggen können. Die Administratoren können Nutzerprofile verwalten, neue Lessons und Units hinzufügen und veraltete Lektionen löschen. Es gibt auch ein lokales Administratorkonto, über das sich der Administrator einloggen kann, falls die Authentisierung über LDAP ausfällt.

[NFT 1] Umgebung und Wartbarkeit Die Anwendung ist webbasiert und läuft auf einem Webserver im Intranet des LRZ. Genauere Angaben zu den verwendeten Technologien finden sich in Abschnitt 4.1.

[NFT 2] Sicherheit Wie bereits in Abschnitt 4.1 erläutert, wird zur Verschlüsselung der HTTP-Verbindung OpenSSL verwendet. Da sich der Server im Intranet des LRZ befindet, ist er außerhalb dieses Netzwerks standardmäßig nicht zugänglich.

[NFT 3] Vertraulichkeit Die Nutzer haben keinen Zugriff auf die Profile anderer Nutzer. Bei der ersten Anmeldung können sie einstellen, ob ihre Fortschritte im Leaderboard angezeigt werden sollen. Standardmäßig ist das nicht der Fall. Sie können diese Einstellung in ihrem Profil jederzeit ändern.

[NFT 4] Erweiterbarkeit Über die Administratorkonten können die Administratoren der Anwendung jederzeit neue Lessons und Units hinzufügen und bereits vorhandene aktualisieren oder löschen.

[NFT 5] Bedienbarkeit Die Benutzeroberfläche wird mit dem Ziel gestaltet, übersichtlich und einfach bedienbar zu sein. Eine genauere Beschreibung des Oberflächendesigns erfolgt im nächsten Abschnitt 4.5.

4.5. Oberflächendesign

Das Design der Oberfläche soll übersichtlich und ohne vorherige Einarbeitung bedienbar sein (**[NFT 6]**). Im folgenden wird das Designkonzept der Oberfläche erläutert. Dabei wird auf den Kontrollfluss, die Aufteilung der Oberfläche und das Administratorinterface eingegangen.

4.5.1. Kontrollfluss

Abbildung 4.3 stellt den Kontrollfluss der Oberfläche dar. Nach dem Login mit der LRZ-SIM-Kennung wird der Nutzer zu seinem Profil weitergeleitet. Beim ersten Login wird er aufgefordert, einen Profilnamen auszusuchen und anzugeben, ob seine Leistungen im Leaderboard angezeigt werden sollen. Aus diesen Daten wird anschließend sein Profil generiert. Die hier angegebenen Daten können jederzeit vom Profil aus geändert werden. *First Login* und *Change Profile* sind also nur Hilfsoberflächen, die die Eingabe von Profildaten ermöglichen. Falls der Nutzer ein Administrator ist, kann er von seinem Profil aus auch über einen Link auf die Administratoroberfläche zugreifen. Dieser ist für andere Nutzer nicht sichtbar.

Abbildung 4.2.: Designkonzept des Menüs

Über das Menü (vgl. Abb. 4.2) hat der Nutzer Zugriff auf sein Profil, die Unit-Übersicht und das Leaderboard. Außerdem kann er sich darüber ausloggen. Die Unit-Übersicht listet alle verfügbaren Units auf. Die Einzelansichten der Units enhalten wiederum eine Übersicht über alle Lessons, die zur jeweiligen Unit gehören. In der Lesson-Einzelansicht können diese bearbeitet werden.

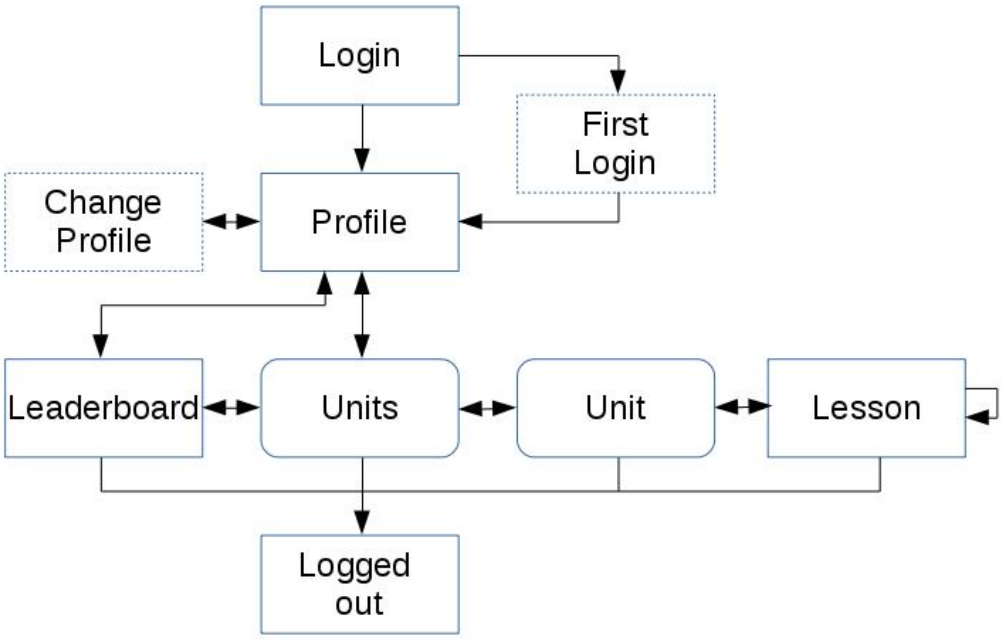

Abbildung 4.3.: Darstellung des Kontrollflusses der Oberfläche

4.5.2. Administratorinterface

Das Administratorinterface wird von Django automatisch generiert und muss lediglich konfiguriert werden. Die Administratoren können sich über eine eigene Seite einloggen und die Benutzer und ihre Profile verwalten. Außerdem können sie Lessons und Units hinzufügen, ändern und löschen.

Abbildung 4.4 zeigt einen Screenshot der Detailansicht einer Unit im Administratorinterface. Standardmäßig ermöglicht Django in dieser Ansicht die Anpassung einer einzelnen Unit. Um das Hinzufügen und Bearbeiten einzelner Lessons zu erleichtern, wurde die Standarddarstellung so konfiguriert, dass alle Lessons einer Unit als Liste in der Detailansicht angezeigt werden. Dadurch kann über die Unit direkt auf die untergeordneten Lessons zugegriffen werden.

Units und Lessons werden in der Anwendung erst ab ihrem in der Datenbank gespeicherten Erstellungszeitpunkt angezeigt. Setzt der Administrator diesen Erstellungszeitpunkt in der Zukunft, kann er unvollständige Lessons oder Units noch einmal bearbeiten, bevor sie für die Nutzer der Anwendung sichtbar werden.

Die Beschreibungen der Units und die Anweisungen in den Lessons können mit dem Editor TinyMCE bearbeitet werden. Dadurch können die Texte ohne direkte Eingabe von HTML formatiert werden, zum Beispiel können Dateipfade einfacher kursiv gesetzt werden.

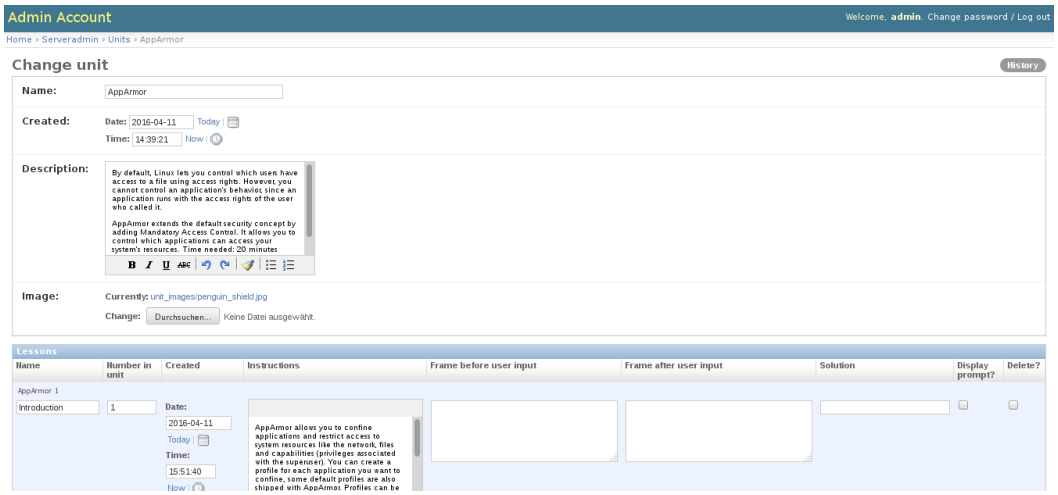

Abbildung 4.4.: Bearbeitung einer Unit über das Administratorinterface

4.5.3. Aufteilung

Abbildung 4.5 zeigt das Designkonzept der Nutzeroberfläche. Nach dem Login (**Bild 8**) wird der Nutzer zu seinem Profil weitergeleitet. Bei der Erstanmeldung muss er vorher einen Nickname angeben und kann angeben, ob er im Leaderboard angezeigt werden soll (**Bild 6**).

Auf dem Profil (**Bild 1**) werden die Profilinformationen wie Name, Nutzername und Anzeige im Leaderboard dargestellt. Außerdem hat der Nutzer einen Überblick über seine Badges in Form von Bildern, die der Administrator für jede Unit hochladen kann, und über unvollständige Units in Form von Fortschrittsbalken. Erlaubt der Nutzer das in seinem Profil, wird er mit seinem Nickname und der Anzahl seiner Badges im Leaderboard (**Bild 3**) angezeigt. Seine Profildaten kann der Nutzer nachträglich ändern (**Bild 7**).

Über die Listenansicht aller verfügbaren Units (**Bild 2**) kann der Nutzer auf die Einzelansichten (**Bild 4**) zugreifen, die wiederum alle der Unit zugeordneten Lessons auflisten. In der Detailansicht der Lessons (**Bild 5**) kann der Nutzer über das Menü auf der linken Seite zu den anderen Lessons dieser Unit wechseln. In der Mitte befindet sich ein Einleitungstext, rechts daneben ein Eingabefeld für die Lösung, das von einem Rahmen für eventuelle Vorgaben umgeben ist. Dieser Rahmen kann beim Erstellen der Lesson angegeben werden und wird vor und nach dem Eingabefeld für die Lösung angezeigt, um den Nutzer bei der Lösung zu unterstützen. Das Lösungsfeld soll optisch an eine Linux-Shell erinnern. Im unteren Teil der Seite bekommt der Nutzer eine grün markierte Rückmeldung für eine richtige Lösung und eine rot markierte Rückmeldung mit Tipps für die Lösung für eine falschen Lösung.

Nach dem Logout erhält der Nutzer eine Bestätigung und einen Link, der zurück zum Login führt (**Bild 9**).

Zusammengefasst wurde in diesem Kapitel das Design der Lernanwendung für die Serveradministratoren des LRZ beschrieben. Dabei wurden die verwendeten Technologien erläutert und auf die Umsetzung der gamificationspezischen, inhaltlichen, funktionalen und nichtfunktionalen Anforderungen aus Abschnitt 3.3 eingegangen. Außerdem wurden der Kontrollfluss und die Aufteilung des Oberflächendesigns der Benutzer- und Administratorkonten erläutert.

Abbildung 4.5.: Designkonzept der Nutzeroberfläche

5. Implementierung

Das folgende Kapitel beschäftigt sich mit der Implementierung des in Kapitel 4 beschriebenen Designs der Webanwendung. Das Framework Django, mit dem die Implementierung erfolgt, lehnt sich an das *MVC-Modell* (*Model, View, Controler*) an. Dabei wird ein Programm in das Datenmodell (*Model*), die Präsentation (*View*) und die Programmsteuerung (*Controller*) aufgeteilt. Durch diese klare Trennung sind die einzelnen Programmteile weitgehend unabhängig voneinander. Zum Beispiel kann die Oberfläche ohne große Änderungen an der Programmsteuerung oder dem Datenmodell erweitert oder ausgetauscht werden.

In Django wird das Datenmodell definiert, indem die Modelle als Python-Klassen deklariert und dann automatisch in die Datenbank übertragen werden. Auf die Daten kann dann über eine API zugegriffen werden, die ebenfalls diese Klassen nutzt. Die Präsentation übernehmen die Views, also Funktionen, die für jede Seite einer Webanwendung definiert werden, zusammen mit den zugehörigen HTML-Templates, in denen die genaue Darstellung in HTML und CSS definiert werden kann.
Die Views beschreiben also, welche Daten präsentiert werden, während die Templates beschreiben, wie sie präsentiert werden. Die Programmsteuerung übernimmt dabei das Framework selbst [Djan 16b].

Zusätzlich bietet Django einen Rahmen für automatisierte Unittests. Dadurch kann bestehender Code einfach getestet und Änderungen überprüft werden.

Das folgende Kapitel orientiert sich an dieser Aufteilung und beschäftigt sich daher zuerst mit dem Datenmodell der Anwendung. Anschließend wird die Implementierung der Präsentation, also der Views und Templates besprochen. Außerdem werden die automatischen Tests behandelt. Da Django selbst in diesem Modell den Controller darstellt, wird er hier nicht näher erläutert, um im Rahmen der Arbeit zu bleiben.

5.1. Datenmodell

Django bietet eine API an, in der Datenmodelle als Python-Klassen deklariert werden können. Diese werden dann als Tabellen in die Datenbank übertragen. Über diese API kann dann auch auf die Daten zugegriffen werden, indem Daten aus der Datenbank als Objekte dieser Klassen abgebildet werden. Tabelle 5.1 bietet eine Übersicht über das Datenmodell der Lernanwendung, das im Folgenden erläutert wird.

Abbildung 5.1 auf Seite 70 stellt die Beziehungen zwischen den Klassen mit Hilfe eines Entity-Relationship-Modells dar. Zur besseren Übersichtlichkeit sind dabei die Attribute der einzelnen Klassen vernachlässigt worden.

Tabelle 5.1.: Übersicht der Klassen des Datenmodells der Anwendung

Name	Funktion
`User`	Automatisch erzeugte Klasse für allgemeine Nutzerinformationen wie Anmeldename und Passwort
`Profile`	Nutzerprofile mit anwendungsspezifischen Informationen wie Nickname und bearbeiteten Lessons
`Lesson`	Aufgaben, die den Nutzern gestellt werden, mit Lösungen und Einführungstexten
`Unit`	Themengebiete, die als Badges dargestellt werden und jeweils mehrere Lessons enthalten
`LessonProtocol`	Klasse zur Speicherung aller Nutzereingaben in die Formulare der Lessons

Die Klasse `User` wird von Django automatisch erzeugt und enthält Nutzerinformationen wie Anmeldename und Passwort. In diesem Fall frägt die Anwendung diese Nutzerdaten über LDAP ab.

Jedem dieser Nutzer ordnet die Anwendung bei der ersten Anmeldung genau ein Profil zu (`Profile`). Wie in Listing 5.1 zu sehen, werden anwendungsspezifische Informationen wie der Nickname oder ob der Nutzer im Leaderboard angezeigt werden möchte gespeichert. Außerdem wird darin gespeichert, welche Lessons vom Nutzer bearbeitet worden sind.

Zudem sind mehrere Funktionen implementiert, die aus den Daten über bearbeitete Lessons aus der Datenbank zusätzliche Informationen berechnen. Dazu gehören die bearbeiteten Units mit der Zahl der zugehörigen bearbeiteten Lessons in Zeile 10 und die vollständig bearbeiteten Units, also die Badges, in Zeile 14. In Zeile 18 werden die unvollständig bearbeiteten Units berechnet.
Diese Informationen werden unter anderem für die Darstellung des Lernstandes im Profil, aber auch für das Leaderboard und die Lessons und Units benötigt. Um redundanten Code zu vermeiden, werden sie bereits im Datenmodell zur Verfügung gestellt.

Listing 5.1: Definition der Klasse Profile

```
1  class Profile(models.Model):
2          user = models.OneToOneField(User)
3          lessons = models.ManyToManyField(Lesson)
4          name = models.CharField(max_length=200, unique=True)
5          display_leaderboard = models.BooleanField('published␣in␣
               leaderboard?', default=False)
6          created = models.DateTimeField()
7          def __unicode__(self):
8                  return self.name
9          # all units for this profile with number of lessons
               completed
10         def unit_count(self):
11         return Unit.objects.filter(lesson__profile=self).annotate(
               num_lessons=Count('lesson')).order_by('id')
12         units = property(unit_count)
```

```
13          # complete units
14          def list_badges(self):
15                  return Unit.objects.filter(lesson__profile=self).
                       exclude(lesson__in=Lesson.objects.exclude(
                       profile=self)).distinct()
16          badges = property(list_badges)
17          # all incomplete units with number of lessons completed
18          def list_in_progress(self):
19                  return self.units.exclude(id__in=self.badges)
20          in_progress = property(list_in_progress)
```

Diesem Profil werden auch die bearbeiteten `Lessons` zugeordnet. Jede Lesson hat einen Namen und einen kurzen Einführungstext. Falls es sich nicht um eine theoretische Lesson handelt, bei der der Nutzer keine Aufgabe lösen muss, hat sie auch eine Lösung und optional einen Vorgaberahmen.

Jede Lesson gehört zu genau einer `Unit`. Jede Unit hat einen Namen, einen kurzen Einführungstext und ein Bild, das zum Beispiel zur Darstellung des Badges, das der Nutzer für die Lösung einer Unit bekommt, verwendet wird.

Die Klasse `LessonProtocol` wird zur Speicherung aller Nutzereingaben in den Lessons verwendet. So kann für jede Lesson die letzte Eingabe des betreffenden Nutzers angezeigt werden, egal ob richtig oder falsch.

Da in der dem `LessonProtocol` zugrunde liegenden Tabelle die Eingaben aller Nutzer in die Lesson-Formulare gespeichert werden, also auch Falsch- und Mehrfacheingaben, wird sie im laufenden Betrieb stark anwachsen. Daher existiert zusätzlich eine von Django erzeugte Zwischentabelle, um die Lessons abzuspeichern, die ein Profil erfolgreich bearbeitet hat. Obwohl diese Information ist auch im `LessonProtocol` enthalten ist, würde die Extraktion der bearbeiten Lessons aus dem `LessonProtocol` möglicherweise das Programm verlangsamen.

5.2. Präsentation

Die Verarbeitung der für die Oberfläche benötigten Daten übernehmen in Django die Views. Jede Seite der Anwendung hat eine View, zu der ein Template gehört, das die Daten in HTML-Code an den Browser des Nutzers ausgibt. Zusätzlich können CSS-Stylesheets und Medien wie Bilder eingebunden werden. Die einzelnen Views werden dann in einer eigenen Datei mit URL-Pfaden versehen, über die sie dann im Browser erreichbar sind. Im folgenden Abschnitt werden die wichtigsten Oberflächenelemente erläutert.

Bei der Entwicklung der Oberfläche wurde besonders auf auf die Umsetzung der nichtfunktionalen Anforderung **[NFT 6] Bedienbarkeit** und der gamificationspezifischen Anforderung **[GAM 4] Lernfortschritt** geachtet. Wie bereits im vorhergenden Kapitel 4 erläutert, wird diese Anforderung durch Oberflächenelemente wie das Leaderboard umgesetzt. Im

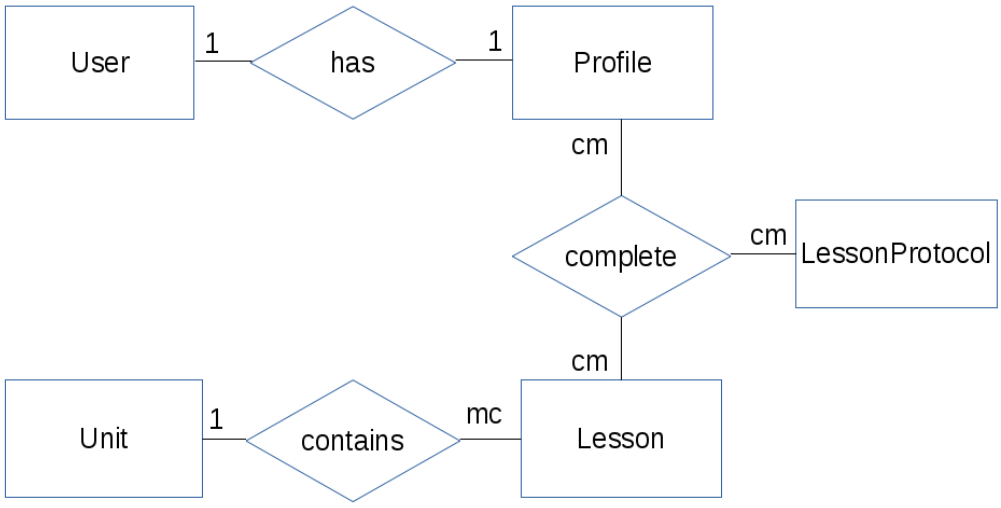

Abbildung 5.1.: Entity-Relationship-Modell der Anwendung

Folgenden werden die wichtigsten Views erläutert. Hilfsformulare, zum Beispiel zur Ersteingabe oder Änderung der Profildaten, werden ausgespart.

5.2.1. Profil

Sofern ein Nutzer bereits ein Profil hat, wird er direkt nach dem Login dorthin weitergeleitet. Die zugehörige View fragt den Profilnamen, die Sichtbarkeitseinstellungen im Leaderboard und die bereits bearbeiteten Lessons und Units aus den Profildaten ab.

Die vollständig bearbeiteten Units werden dann im Template als Badges angezeigt. Unvollständig bearbeitete Units werden dagegen mit einem Fortschrittsbalken versehen, der den prozentualen Fortschritt innerhalb einer Unit anzeigt. Dadurch hat der Nutzer eine klare Übersicht über seine Lernfortschritte. Über einen Link zu einer anderen View können der Profilname und die Sichtbarkeitseinstellungen geändert werden.

5.2.2. Units

Die vorhandenen Units werden mit ihren Bildern und ihrer Kurzbeschreibung in einer Übersicht aufgelistet. Dabei frägt die zuständige View die Units aus der Datenbank ab. Außerdem ermittelt sie aus den Profildaten des eingeloggten Nutzers, welche dieser Units schon bearbeitet worden sind.

Im Template wird diese Information neben den Namen und dem zugeordneten Badge der jeweiligen Unit angezeigt. Klickt der Nutzer auf den Namen der Unit, wird die Kurzbeschreibung mit weiteren Informationen wie den zugehörigen Man-Pages, also der Dokumentation von Linux, angezeigt. Darunter führt ein Link zur Detailansicht der Unit. Dadurch entsteht

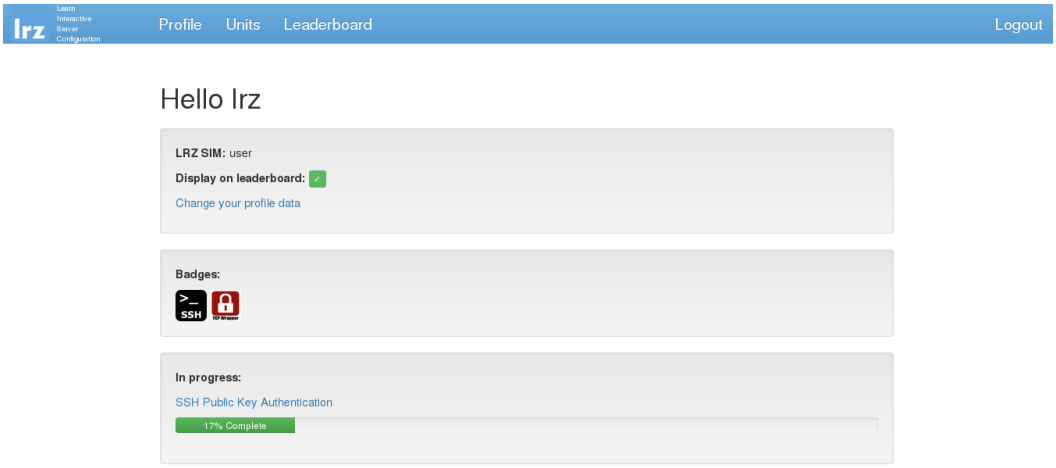

Abbildung 5.2.: Profil eines Nutzers

ein übersichtliches Design, das leicht zu bedienen ist, aber bei Bedarf genauere Informationen liefert.

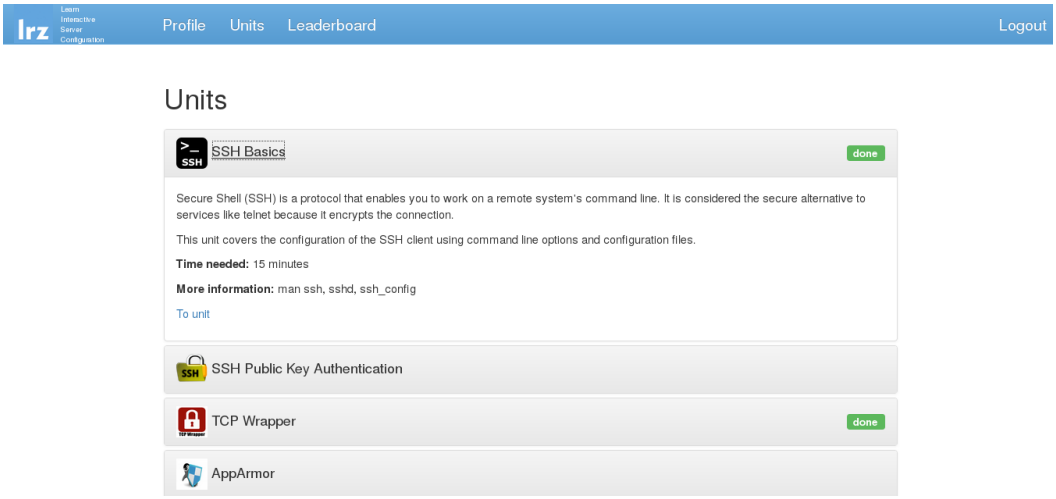

Abbildung 5.3.: Übersicht der vorhandenen Units

Die Detailansicht der Unit zeigt den Namen und das zugehörige Badge der Unit an und listet alle enthaltenen Lessons auf. Lessons, die schon bearbeitet worden sind, werden mit einem Häkchen entsprechend markiert.

5.2.3. Lessons

Die Detailansicht der einzelnen Lessons enthält einen kurzen Einführungstext mit Anweisungen und ein Eingabefenster für die Lösung des Nutzers. Der Administrator hat beim Erstellen der Aufgaben die Möglichkeit, einen Rahmen einzugeben, der vor und nach dem Eingabefeld angezeigt wird und dem Nutzer eine Hilfestellung bieten soll. Außerdem kann

Abbildung 5.4.: Übersicht der Lessons einer Unit

er wählen, ob ein Prompt angezeigt werden soll, also die Benutzeroberfläche der Linux-Shell simuliert werden soll. Das wäre bei einem einzugebenden Befehl sinnvoll, bei einer Konfigurationsdatei dagegen weniger.

In Abbildung 5.6 gibt die Zeile `capability sys_admin` ein Beispiel für eine Regel in AppArmor, die der Nutzer für seine Lösung nur entsprechend anpassen muss.
Ist die Eingabe des Nutzers falsch, erhält er eine Rückmeldung der, wenn möglich, eine Hilfestellung beigefügt ist. Das ist zum Beispiel der Fall, wenn die Eingabe des Nutzers bis auf Leerzeichen oder Groß- und Kleinschreibung korrekt ist.

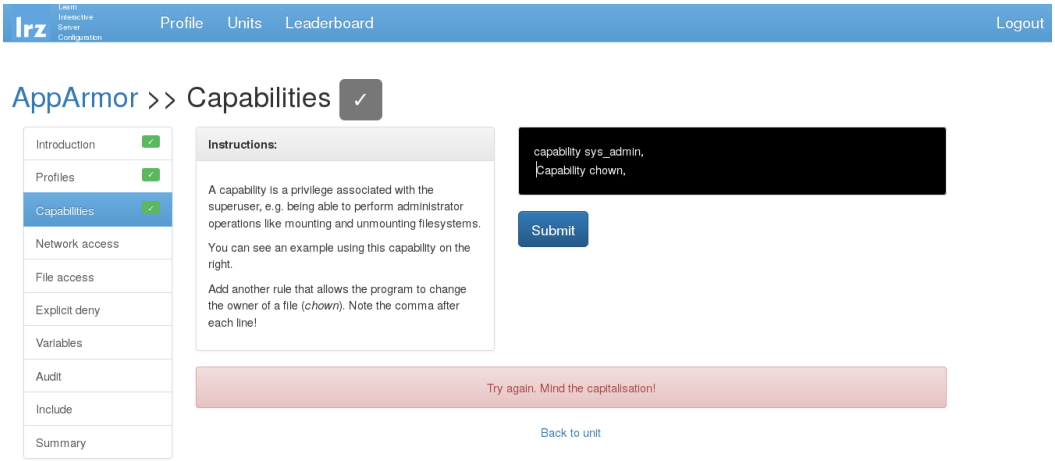

Abbildung 5.5.: Rückmeldung mit Verbesserungsvorschlag bei Falscheingabe

Listing 5.2 zeigt einen Ausschnitt aus der Lesson View, in dem in Zeile 4 Nutzereingabe und Lösung der Lesson verglichen werden. Ist die Eingabe korrekt, wird dem Profil in Zeile 8 die Lesson und falls zutreffend auch das Badge hinzugefügt.

Ist die Eingabe falsch, erhält der Nutzer eine entsprechende Meldung. Die Eingabe wird wortweise mit der Musterlösung verglichen, das heißt führende oder abschließende Leer-

zeichen und mehrere Leerzeichen in Folge werden toleriert (vgl. Zeilen 2 und 3). Andere Unterschiede bei den Leerzeichen oder abweichende Groß- und Kleinschreibung führen zu einer entsprechenden Fehlermeldung wie in den Zeilen 18 und 21, da bei Konfigurationsdateien solche Fehler ungewollte Effekte auslösen oder zumindest gängigen Konventionen widersprechen können. Außerdem wird der Nutzer in Zeile 15 zur erneuten Eingabe aufgefordert, falls eine leere Eingabe oder eine Eingabe, die nur aus Leerzeichen besteht, erkannt wurde.

Listing 5.2: Eingabevalidierung in der Lesson View

```
1  user_solution_whole = form.cleaned_data['solution']
2  user_solution = user_solution_whole.split() #slice string into
       words
3  lesson_solution = lesson.solution.split()
4  if user_solution == lesson_solution:
5          message = "Well done!"
6          correct = True
7          if lesson not in profile_lesson_list:
8                  profile.lessons.add(lesson)
9                  lesson.done = "done"
10                 if lesson.unit in profile.badges:
11                         badge = True
12                 else:
13                         correct = False
14                         if not user_solution: #user_solution is
                               empty or contains only white spaces
15                                 message = "Please enter your
                                       solution."
16                         # case is the problem
17                         elif [u.lower() for u in user_solution] ==
                                [l.lower() for l in lesson_solution]:
18                                 message = "Try again. Mind the
                                       capitalisation!"
19                         # blanks are the problem
20                         elif "".join(user_solution) == "".join(
                               lesson_solution):
21                                 message = "Try again. Mind the
                                       blank spaces!"
22                         else:
23                                 message = "Try again. "
24 LessonProtocol.objects.create(profile=profile, lesson=lesson,
       user_solution=user_solution_whole, message=message, timestamp=
       timezone.now(), correct=correct)
25 context = {'unit': unit, 'lesson': lesson, 'lesson_list':
       lesson_list, 'form': form, 'message': message, 'next': next, '
       correct': correct, 'badge': badge}
26 return render(request, 'serveradmin/lesson.html', context)
```

Während die Aufgaben dem Nutzer die Gelegenheit geben, das erworbene Wissen praktisch

anzuwenden, kann der Administrator auch Lessons erstellen, die keine Lösungseingabe vom Nutzer erwarten. Dies ist zum Beispiel sinnvoll, wenn komplexere theoretische Sachverhalte erklärt werden sollen.

Abbildung 5.6 zeigt eine solche Lesson, bei der die Eingabemöglichkeit für den Nutzer entfällt. Der Administrator erstellt eine Theorie-Lesson, indem er keine Lösung für die Aufgabe angibt. Während die View keine eigene Implementierung für solche Lessons hat, wird im Template überprüft, ob der Administrator eine Lösung angegeben hat. Ist das nicht der Fall, wird die Oberfläche entsprechend angepasst.

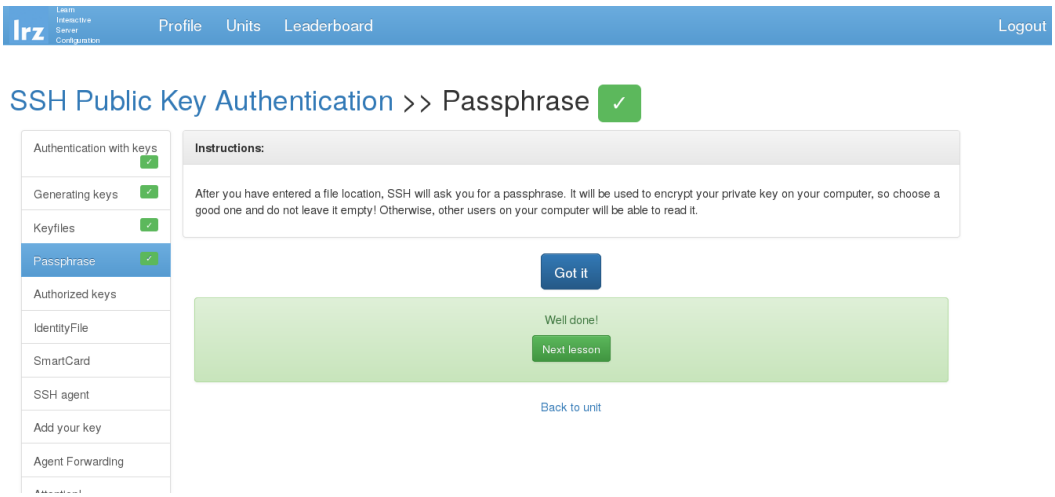

Abbildung 5.6.: Detailansicht einer Theorie-Lesson

5.2.4. Leaderboard

Das Leaderboard (vgl. Abb 5.7) soll den Nutzern helfen, ihre Leistungen untereinander zu vergleichen und dadurch motivieren. Alle Nutzer, die mindestens ein Badge haben, werden unter ihrem Nutzernamen und der Anzahl ihrer Badges darin angezeigt. Sie müssen diese Funktion aber explizit bei der ersten Anmeldung oder in ihrem Profil aktivieren, so dass sichergestellt ist, dass ihr Lernstand nicht unbeabsichtigt für andere sichtbar ist **[NTF 4]**.

Die View frägt dafür aus der Datenbank alle Nutzer ab, die auf dem Leaderboard angezeigt werden wollen und die alle Lessons einer oder mehrerer Units richtig gelöst haben. Die Nutzerdaten werden dann mit der Anzahl der jeweiligen Badges absteigend sortiert an das Template weitergegeben.

5.3. Tests

Zusätzlich zur Aufteilung in Präsentation und Datenmodell bietet Django einige Funktionen für automatisierte Unittests. Dadurch kann die Reaktion auf bestimmte Eingaben automatisch getestet werden. Das ist besonders bei Änderungen am Code sinnvoll, da getestet werden kann, ob bestehende Funktionen unbeabsichtigt verändert wurden.

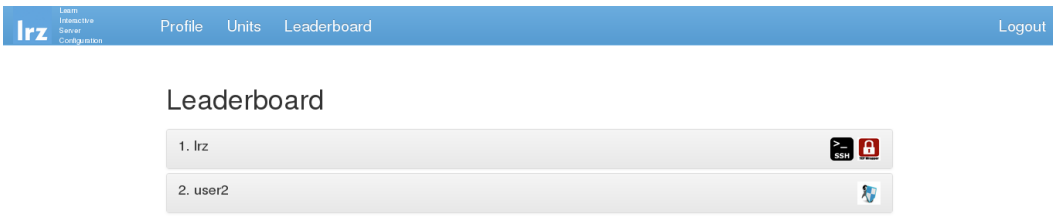

Abbildung 5.7.: Das Leaderboard

Werden die Tests ausgeführt, wird eine neue Testdatenbank erzeugt, die danach wieder zerstört wird. Dadurch werden die Tests nicht auf der Produktivdatenbank ausgeführt, so dass keine Fehler entstehen können. Um die Testdatenbank mit passenden Daten zu füllen können Datenbankdumps, also Kopien der Daten aus der Produktivdatenbank, genutzt oder im Code Daten eingefügt werden.

Listing 5.3 zeigt eine Funktion, die überprüft, ob die `Units` View alle Units in der Datenbank anzeigt. Dabei werden in den Zeilen 3 und 4 neue Units erzeugt. In Zeile 5 wird die URL der View aufgerufen und in den Zeilen 6 und 7 überprüft, ob es bei der Anfrage einen Fehler gab und ob die Units in der Antwort enthalten sind.

Listing 5.3: Test für die Units-View

```
1 def test_units_view_with_two_units(self):
2                 """ Should display both questions. """
3                 unit1 = Unit.objects.create(name="unit1", image="
                      test.jpg", created=timezone.now())
4                 unit2 = Unit.objects.create(name="unit2", image="
                      test.jpg", created=timezone.now())
5                 response = self.client.get(reverse('serveradmin:
                      units'))
6                 self.assertEqual(response.status_code, 200)
7                 self.assertEqual(set(response.context['unit_list'
                      ]), set(Unit.objects.all()))
```

5.4. Zusammenfassung

Zusammengefasst orientieren sich Django-Projekte grob am MVC-Modell, unterteilen sich also in Datenmodell, Präsentation und Programmsteuerung. Die Datenmodelle werden in Django als Python-Klassen deklariert und können über eine API abgefragt werden. Die

Präsentation wird durch Views, die beschreiben welche Daten dargestellt werden, und HTML-Templates, die beschreiben, wie diese Daten dargestellt werden, definiert. Die Programmsteuerung übernimmt das Framework selbst.

Das Datenmodell definiert Modelle für die Nutzer und ihre Profile, die Units und ihre zugeordneten Lessons und ein Protokoll, in dem alle Nutzereingaben in die Lesson-Formulare abgespeichert werden. Die Präsentation enthält Views und Templates für jede Webseite, wie zum Beispiel das Nutzerprofil, die Listen- und Detailansicht der Units, die Detailansicht der Lessons und das Leaderboard.

Django bietet außerdem ein Framework für automatisierte Unittests an. Vor allem bei Änderungen im Code kann damit sichergestellt werden, dass bereits bestehende Funktionen nicht beeinträchtigt sind.

6. Evaluation

Das folgende Kapitel behandelt die Evaluation der in dieser Arbeit entwickelten Webanwendung. Dadurch soll eingeschätzt werden, ob das Lernprogramm die in Abschnitt 3.3 definierten Anforderungen erfüllt. Wichtig ist dabei vor allem, ob der Lernstoff vermittelt wird und ob die Nutzer motiviert sind und Spaß an der Bearbeitung der Aufgaben haben.

Zu diesem Zweck haben sowohl zwei Mitarbeiter des LRZ als auch zwei Versuchspersonen, die nicht aus der Informatik kommen, das Programm getestet. Zudem wurden sie zu ihrem Wissensstand vor und nach dem Bearbeiten der Aufgaben und zur Benutzerfreundlichkeit befragt. Mit dem dafür verwendeten Fragebogen können in der Zukunft noch weitere Nutzer Feedback zur Anwendung geben.

6.1. Anforderungen

Die Anforderungen aus Abschnitt 3.3 gliedern sich in gamificationspezifische, inhaltliche, funktionale und nichtfunktionale Anforderungen. Tabelle 6.1 listet diese auf und ergänzt Anforderungen, die umgesetzt wurden, mit einem **+**, während Anforderungen, die nicht umgesetzt wurden, mit einer **0** versehen sind. Während alle Muss-Anforderungen (in der Tabelle mit der Gewichtung + dargestellt) erfüllt wurden, wurden die Kann-Anforderungen (mit der Gewichtung 0) vorerst zurückgestellt. Dabei handelt es sich um inhaltliche Anforderungen, die nachträglich ergänzt werden können.

6.2. Umfrage

Zusätzlich zur Implementierung der Anforderungen soll überprüft werden, ob die Anwendung ihren Zweck erfüllt, also den Lernstoff vermittelt und ihre Nutzer dabei mit Gamification-Elementen motiviert. Da dies am Besten durch eine Nutzerstudie ermittelbar ist, wurde eine Umfrage erstellt, die einigen Mitarbeitern des LRZ und einigen Testpersonen, die kein Vorwissen in dem Bereich haben, vorgelegt wurde. Die Umfrage besteht aus zwei Teilen, von denen der erste vor der Bearbeitung der Lerneinheiten in der Anwendung beantwortet werden soll und der zweite danach. Die Units, die bearbeitet werden, beschäftigen sich mit der TCP-Wrapper-Bibliothek und AppArmor.

Zu Beginn beantworten die Testpersonen einige Fragen zu ihrer Erfahrung mit Linux und ihrer Arbeitsstelle. Dadurch können die Ergebnisse nach den befragten Personengruppen unterschieden werden. Dabei kann es sich um Administratoren des LRZ handeln, die sich in

Tabelle 6.1.: Anforderungsübersicht mit Gewichtungen und der Angabe, ob sie implementiert wurden

ID	Anforderung	Gewichtung	Implementiert?
Gamificationspezifische Anforderungen			
[GAM 1]	Personalisierung	+	+
[GAM 2]	Übersichtlichkeit	+	+
[GAM 3]	Sofortiges Feedback	+	+
[GAM 4]	Lernfortschritt	+	+
[GAM 5]	Selbstbestimmung	+	+
Inhaltliche Anforderungen			
[CON 1]	Cronjobs	0	0
[CON 2]	Secure Shell (SSH)	+	+
[CON 3]	Network File System (NFS)	0	0
[CON 4]	TCP Wrapper	+	+
[CON 5]	Firewalls	0	0
[CON 6]	Web Application Firewalls	0	0
[CON 7]	Intrusion Detection/Prevention Systems	0	0
[CON 8]	OSSEC (Hosted Intrusion Detection System)	0	0
[CON 9]	AppArmor (Hosted Intrusion Prevention System)	+	+
Funktionale Anforderungen			
[FUN 1]	Login	+	+
[FUN 2]	Profil	+	+
[FUN 3]	Administratorkonto	+	+
Nichtfunktionale Anforderungen			
[NFT 1]	Korrektheit	+	+
[NFT 2]	Umgebung und Wartbarkeit	+	+
[NFT 3]	Sicherheit	+	+
[NFT 4]	Vertraulichkeit	+	+
[NFT 5]	Erweiterbarkeit	+	+
[NFT 6]	Bedienbarkeit	+	+

dem Bereich bereits gut auskennen, Mitarbeiter, die weniger Erfahrung mit der Serverkonfiguration haben, und Personen, die kaum Vorwissen im Bereich Informatik haben.

Der erste Teil der Umfrage soll den Wissensstand der Testpersonen vor der Bearbeitung der Lerneinheiten in der Webanwendung testen. Er besteht aus jeweils vier Aufgaben zu TCP Wrapper und AppArmor. Im zweiten Teil der Umfrage sollen die Testpersonen diese Aufgaben nach der Bearbeitung der relevanten Units erneut lösen, um zu testen, ob sich ihr Wissensstand verändert hat.

Im Anschluss werden die Testpersonen zur Bedienbarkeit der Anwendung befragt und ob ihnen das Lernen mit der Anwendung Spaß gemacht hat. Dadurch soll aufgezeigt werden, ob die verwendeten Spieldesignelemente die Nutzer tatsächlich motivieren.

Zum Erstellen der Umfrage wurde *Google Formulare* [Goog 16] genutzt. Das Werkzeug ermöglicht die einfache, schnelle Erstellung von Umfragen, die automatisch ausgewertet werden. Das erstellte Formular kann den Testpersonen über einen Link oder per E-Mail zugeschickt werden. Nach dem Abschluss einer Umfrage kann diese wieder gelöscht werden.

Abbildung 6.1 zeigt einen Screenshot des Umfrageformulars.

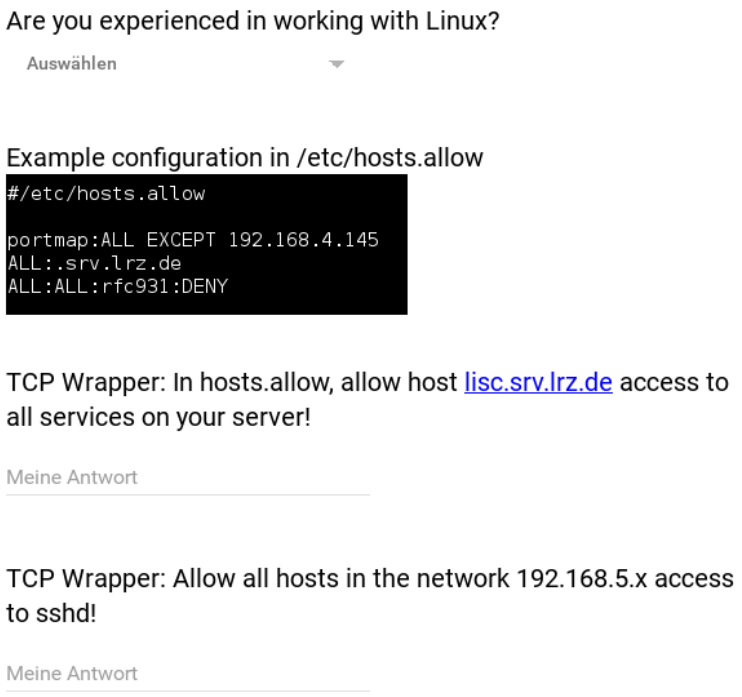

Abbildung 6.1.: Screenshot des Umfrageformulars

6.3. Wissensvermittlung

Durch die Umfrage soll geprüft werden, ob die Lernanwendung Fachwissen im Bereich der Linux-Serveradministration vermitteln kann, also ihren Zweck erfüllt. Deshalb wird in der Umfrage der Wissensstand zu TCP Wrapper und AppArmor vor und nach der Bearbeitung der Aufgaben getestet. Tabelle 6.2 zeigt die Ergebnisse abhängig von den befragten Personengruppen, die im Folgenden näher erläutert werden.

Tabelle 6.2.: Ergebnisse der Evaluation

	Anzahl befragte Personen	**Richtig beantwortete Fragen vor Bearbeitung der Aufgaben**	**Richtig beantwortete Fragen nach Bearbeitung der Aufgaben**
LRZ	2	2/8	4/8
Sonstige	2	0/8	2/8

Der Fragebogen wurde an einige Mitarbeiter des LRZ geschickt, in der gesetzen Frist nahmen aber nur zwei daran teil, wobei eine Antwort verwertbar war. Die Testperson gab darin an, mehrere Jahre Erfahrung mit Linux zu haben. Sie hatte kein Vorwissen zu AppArmor,

konnte aber vor der Bearbeitung der Lerneinheiten in der Anwendung zwei von vier Aufgaben zu TCP Wrapper richtig lösen. Danach konnte sie alle Fragen zu TCP Wrapper richtig beantworten, machte aber Fehler bei den Aufgaben zu AppArmor.

Eine Ursache dafür könnte sein, dass die Unit zu TCP Wrapper einfacher oder leichter verständlich war, möglicherweise ist aber ein gewisses Vorwissen für einen optimalen Lernerfolg notwendig.

Die anderen beiden Testpersonen, die nur wenig Erfahrung im Umgang mit Linux und kein Vorwissen zu AppArmor oder TCP Wrapper hatten, konnten zwar einen Großteil der Aufgaben in der Anwendung lösen, machten aber im zweiten Teil des Fragebogens viele Fehler und konnten jeweils nur zwei von acht Aufgaben richtig beantworten.

Daraus kann man ebenfalls schließen, dass für den Lernerfolg bei der Nutzung ein bestimmtes Vorwissen zum Thema Linux erforderlich ist. Zum Beispiel interpretierten manche Testpersonen die Kommentare in den Beispielen als Befehlssequenzen. Diese Kommentare kommen aber in den meisten Konfigurationsdateien unter Linux vor und werden in der Anwendung deshalb nicht extra erwähnt.

Zusammengefasst ist für ein tiefergehendes Verständnis der Lerneinheiten in der Anwendung wohl eine gewisse Erfahrung mit Linux und Vorwissen im Bereich der Serverkonfiguration notwendig. Da die Aufgaben in der Anwendung auch die zu vermittelnden Bereiche nicht vollständig abdecken können, müssen für einen optimalen Lernerfolg auch zusätzliche Informationsquellen hinzugezogen werden.

6.4. Bedienbarkeit

Die Anwendung soll das Fachwissen nicht nur vermitteln, sondern ihre Nutzer auch motivieren. Aus diesem Grund wurden beim Design Gamification-Elemente eingesetzt, um den Lernstoff für die Nutzer interessanter zu machen. Um zu testen, ob die Anwendung angenehm zu bedienen ist und das Lernen damit Spaß macht, wurden den Testpersonen nach der Bearbeitung der Aufgaben zwei offene Fragen zu ihren Erfahrungen gestellt. Die Ergebnisse werden im Folgenden erläutert.

Die Versuchspersonen wurden gefragt, ob sie die Anweisungen verstanden haben und die Aufgaben einfach zu bearbeiten waren. Außerdem sollten sie angeben, ob ihnen das Lernen mit der Anwendung gefallen hat.

Insgesamt gaben alle Testpersonen an, dass das Lernen mit der Anwendung Spaß gemacht habe. Die Testpersonen ohne Erfahrungen mit Linux hatten allerdings, wie im vorherigen Abschnitt beschrieben, einige inhaltliche Verständnisschwierigkeiten. Das zeigt auch hier, dass die Lerneinheiten ohne Vorwissen nur schwer zu bearbeiten sind.

Eine Testperson gab an, dass es eine Möglichkeit geben sollte, die Musterlösung einer Lesson anzeigen zu lassen. Dies wurde aber bei der Planung der Anwendung nicht ermöglicht, weil die Nutzer dadurch darin bestärkt werden sollen, eigenständig Informationen zu suchen und sich mit ihren Mitarbeitern zu beraten.

Die Testperson mit Vorwissen gab an, dass die Aufgaben relativ einfach zu bearbeiten waren. Sie kritisierte, dass die Anweisungen teilweise etwas unübersichtlich formatiert waren. Diese wurden daraufhin noch einmal überarbeitet.

Die Bewertung der Bedienbarkeit fiel also überwiegend positiv aus. Für das inhaltliche Verständnis waren aber, wie schon im vorhergehenden Abschnitt beschrieben, gewisse Vorkenntnisse nötig.

6.5. Zusammenfassung

Zusammengefasst wurden alle Muss-Anforderungen aus Abschnitt 3.3 implementiert. Die Kann-Anforderungen wurden dagegen vorerst zurückgestellt. Sie können aber, da es sich um inhaltliche Anforderungen handelt, durch den Adminstrator der Anwendung ergänzt werden. Um die Wissensvermittlung und Bedienbarkeit der Anwendung zu überprüfen, wurde ein Fragebogen erstellt, der die Vorkenntnisse im Umgang mit Linux und der Systemadministration mit den Kenntnissen nach Bearbeitung der Lerneinheiten in der Anwendung vergleicht. Außerdem können die Testpersonen die Bedienbarkeit der Anwendung bewerten.

Eine wichtige Erkenntnis daraus war, dass Vorkenntnisse im Umgang mit Linux und in der Serveradministration für ein tiefergehendes Verständnis notwendig sind. Da die Anwendung die zu vermittelnden Themengebiete auch nicht vollständig behandeln kann, ist sie wohl mehr als Begleitung zu Informationsquellen wie Büchern geeignet. Die Bedienbarkeit der Anwendung wurde positiv bewertet und die Testpersonen gaben an, dass das Lernen mit der Anwendung Spaß macht. Sie schafft es also, den Lernstoff anschaulich darzustellen und ihre Nutzer zu motivieren.

Obwohl die Umfrage an weitere Mitarbeiter des LRZ geschickt wurde, nahmen in der gesetzen Frist nur zwei Testpersonen aus dem LRZ daran teil, was die Ergebnisse wenig aussagekräftig macht. Der Fragebogen kann aber auch in Zukunft verwendet werden, um Feedback von weiteren Nutzern zu erhalten. Außerdem ist eine weitere Evaluation sinnvoll, in der die Nutzer nach circa einem Jahr Anwendungsdauer befragt werden. Dabei kann auch anhand der Protokolle in der Anwendung geprüft werden, wie die Anwendung angenommen wird und wie viel sie tatsächlich genutzt wird.

7. Zusammenfassung

Im folgenden Kapitel wird die Arbeit zusammengefasst und im Fazit auf die Fragestellungen in der Motivation in Abschnitt 1.1 eingegangen. Abschließend folgt ein Ausblick auf mögliche weitere Arbeiten.

In Kapitel 2 wurde das Konzept der Gamification als

> „the use of game design elements in non-game contexts "[DDKN 11a, S. 1],

also die Anwendung von Elementen des Spieldesigns in Kontexten, die nicht mit Spielen zusammenhängen, definiert und erläutert, wie gamifizierte Anwendungen ihre Nutzer mit Spieldesignelementen motivieren. Diese Spieldesignelemente wurden auf verschiedene Abstraktionsebenen aufgeteilt und erläutert. Außerdem wurden die vielfältigen Anwendungsbereiche von Gamification, zu denen auch die Bildung gehört, aber auch kritische Meinungen zu dem Thema aufgezeigt. Abschließend wurden einige Anwendungsbeispiele in der Informatik vorgestellt, wie zum Beispiel die Lernplattform Codeacademy, die auch als Inspiration für die in dieser Arbeit entwickelte Anwendung diente.

Im Anschluss wurden in Abschnitt 2.2 einige Grundkonzepte der Linux-Serverkonfiguration behandelt, die für die Administratoren des LRZ von besonderer Bedeutung sind. Dazu gehören allgemeine Administrationswerkzeuge wie Cronjobs zur Automatisierung wiederkehrender Aufgaben, SSH zum Arbeiten auf entfernten Systemen und NFS zum Einbinden entfernter Verzeichnisse in den Dateibaum.

Außerdem wurden sicherheitsrelevante Anwendungen wie die TCP-Wrapper-Bibliothek, mit der der Zugriff auf Serverdienste für bestimmte Clients eingeschränkt werden kann, thematisiert. In diese Kategorie gehören auch Firewalls, die den Zugriff auf ganze Systeme einschränken können, und die Web Application Firewalls, die auf den Schutz von Webservern spezialisiert sind. Für die Erkennung von Sicherheitsvorfällen können zudem Instrusion Detection Systems und Intrusion Prevention Systems verwendet werden.

Um zu zeigen, dass die zum Beispiel bei Codeacademy verwendeten Gamification-Prinzipien auch auf die Serveradministration übertragbar sind, wurde in der Arbeit eine Lernanwendung für die Serveradministratoren des LRZ entwickelt. Die Anforderungen, die sich daraus an diese Anwendung ergeben, wurden in Kapitel 3 in gamificationspezifische, inhaltliche, funktionale und nichtfunktionale Anforderungen unterteilt. Dabei beziehen sich die gamificationspezifischen Anforderungen auf den Einsatz der Spieldesignprinzipien aus Kapitel 2. Die inhaltlichen Anforderungen sind die Grundlagen der Serverkonfiguration, die in Abschnitt 2.2 erläutert wurden, während sich die funktionalen und nichtfunktionalen Anforderungen aus den Herausforderungen am LRZ aus Abschnitt 3.2 ergeben.

Diese Anforderungen flossen in Kapitel 4 in das Design der Anwendung ein. Die Anwen-

dung wurde als Webanwendung in Python unter Nutzung des Frameworks Django implementiert. Für die Oberfläche wurde das CSS-Framework Bootstrap genutzt. Beim Design der Oberfäche lag der Fokus auf der Bedienbarkeit und Übersichtlichkeit.

Bei der Implementierung in Kapitel 5 wurde das MVC-Modell von Django übernommen, ist also grob unterteilt in Model, View und Controller. Während das Framework selbst der Controller ist, also die Programmlogik enthält, wurden für das Datenmodell der Anwendung Python-Klassen definiert, die die Schnittstelle zur Datenbank bilden. Die Oberfläche wird durch die Views modelliert, die definieren, was angezeigt wird. Außerdem gibt es HTML-Templates sowie CSS-Stylesheets, die bestimmen, wie die Daten angezeigt werden.

In Kapitel 6 wurde die Anwendung anhand der Anforderungen aus Kapitel 3 und eines Fragebogens evaluiert. Während die Muss-Anforderungen vollständig implementiert wurden, wurden die Soll-Anforderungen vorerst zurückgestellt. Der Fragebogen sollte überprüfen, ob die Anwendung das Fachwissen vermitteln kann. Die Ergebnisse zeigen, dass für ein tieferes Verständnis der Lerninhalte Vorwissen nötig ist. Außerdem wurden die Testpersonen befragt, ob die Anwendung leicht zu bedienen war und die Nutzung Spaß macht. Beide Aspekte wurden überwiegend positiv bewertet. Da die Beteiligung an der Umfrage am LRZ allerdings sehr niedrig war, sind die Ergebnisse nur bedingt aussagekräftig.

Aufbauend auf dieser Zusammenfassung der Arbeit folgt im nächsten Abschnitt ein Fazit, das auf die Fragestellungen in der Motivation in Abschnitt 1.1 zurückkommt.

7.1. Fazit

Ziel der Arbeit war es, verschiedene Gamificationansätze zu evaluieren und auf die Serveradministration zu übertragen. Dabei sollte festgestellt werden, ob Gamification bei der Wissensvermittlung in diesem Bereich ebenfalls zur Motivation der Lernenden eingesetzt werden kann. Zu diesem Zweck wurde eine Lernanwendung für die Linux-Serveradministratoren des Leibniz-Rechenzentrums entwickelt, die ihnen dabei helfen soll, Fachwissen leichter zu erlernen und neue Mitarbeiter schneller einzuarbeiten.

Die Anwendung ermöglicht es ihren Nutzern ähnlich wie Codeacademy, ihr erworbenes Wissen praktisch anzuwenden und setzt zur Motivation Spieldesignelemente wie Badges und Fortschrittsbalken ein. Zusätzlich zu diesen Oberflächenelementen hat jeder Spieler ein Profil mit Nickname und kann seine Leistungen im Leaderboard mit anderen Nutzern vergleichen. Die Lektionen sind in Lerneinheiten aufgeteilt, so dass der Lernstoff übersichtlicher und der Lernfortschritt leicher erkennbar wird.

In der Evaluation gaben alle Testpersonen an, dass ihnen das Lernen mit der Anwendung Spaß gemacht hat. Das Programm schafft es also, seine Nutzer zu motivieren und den Inhalt interessant darzustellen. Die Evaluation zeigt jedoch auch, dass die Anwendung nur bedingt für Nutzer ohne Vorkenntnisse geeignet ist. Das kann mit der Vermittlung des Lerninhalts zusammenhängen, liegt aber wohl auch daran, dass die Serveradministration ein Aufgabengebiet ist, das sehr viel Fachwissen erfordert. Da dieses Themengebiet sehr umfangreich ist, kann es auch kaum vollständig in wenigen Lerneinheiten vermittelt werden. Allerdings

kann die Anwendung begleitend zu anderen Informationsquellen wie Fachbüchern genutzt werden, ob als Einstieg in das Themengebiet oder um die bereits erworbenen Kenntnisse praktisch anzuwenden.

Insgesamt kann Gamification also durchaus zur Wissensvermittlung in der Serveradministration verwendet werden. Da auf dem Gebiet aber sehr umfangreiches, aber auch detailliertes Fachwissen erforderlich ist, können gamifizierte Anwendungen den Lernstoff nicht alleine vermitteln. Sie können aber als Unterstützung zu den klassischen Lehrmethoden betrachtet werden.

7.2. Ausblick

Im folgenden Abschnitt wird ein Ausblick auf mögliche zukünftige Arbeiten gegeben, die auf dieser Arbeit aufbauen.

In Kapitel 3 wurden sowohl Muss-Anforderungen als auch einige Kann-Anforderungen an die Lernanwendung definiert. Während die Muss-Anforderungen vollständig umgesetzt wurden, wurden einige inhaltliche Kann-Anforderungen zurückgestellt. Dabei handelt es sich um zusätzliche Lerneinheiten, die in der Administratoroberfläche ergänzt werden könnten.

Außerdem ist der Vergleich der Nutzereingaben mit den Musterlösungen in den Lessons aktuell noch generisch. Zum Beispiel werden Leerzeichen, die in bestimmten Konfigurationsdateien optional sind, nicht erkannt. Statt dessen wird ein Eingabe gefordert, die der Musterlösung entspricht. Hier könnte der Vergleich verfeinert und eventuell auf den Inhalt der Aufgaben abgestimmt werden. Ebenso sind die Hilfestellungen bei einer Falscheingabe im Moment ohne Aufgabenbezug und sehr einfach. Diese könnten ebenfalls spezifischer und abhängig vom Inhalt werden.

Zudem kann die Anwendung, da es sich um eine Webanwendung handelt, zwar auch mit einem Smartphone oder Tablet genutzt werden, ist aber für den PC optimiert. Hier könnte man die Oberfläche zum Beispiel mit Hilfe von Bootstrap responsiv gestalten oder eine App entwickeln, mit deren Hilfe auf die Anwendung zugegriffen werden kann.

Zusätzlich ist die Evaluation in dieser Arbeit wenig aussagekräftig, weil sehr wenige Testpersonen befragt wurden. Da die Mitarbeiter des LRZ die Zielgruppe der Anwendung darstellen, müssten ihre Bewertungen auch stärker einfließen. Daher wäre es sinnvoll, diese noch einmal zu befragen nachdem die Anwendung eine Zeit lang im Produktivbetrieb eingesetzt wurde. Zu diesem Zeitpunkt, zum Beispiel nach einem Jahr, hätten sie schon mehr Erfahrungen mit dem Programm gesammelt und könnten auch bewerten, ob es ihnen bei der Serverkonfiguration geholfen hat. Außerdem könnten die Protokolle der Anwendung ausgewertet werden, um herauszufinden, wie viel sie tatsächlich genutzt wird.

Falls die Anwendung am LRZ gut angenommen wird, könnte die Bearbeitung der Aufgaben für die Serveradministratoren auch verpflichend werden. In diesem Fall wären zusätzliche Funktionen nötig, mit denen die Teilnahme anhand der Anwendungsprotokolle über-

prüft werden können. Außerdem wären Benachrichtigungsfunktionen wie automatische Erinnungsmails von Vorteil.

Zusammengefasst könnten also die zusätzlichen inhaltlichen Anforderungen umgesetzt werden, der Vergleich der Lösungen und die Hilfestellungen in den Lessons verfeinert und die Anwendung für die Verwendung auf Mobilgeräten optimiert werden. Außerdem würde eine weitere Evaluation nach etwa einem Jahr genauere Aufschlüsse über den Erfolg der Anwendung geben. Falls die Bearbeitung der Aufgaben am LRZ verpflichtend werden soll, muss das Programm um weitere Funktionen zur Überprüfung der Teilnahme und für Benachrichtigungen erweitert werden.

Abkürzungsverzeichnis

ACL Access Control List

API Application Programming Interface

DAC Discretionary Access Control

DDoS Distributed Denial of Service

DNS Domain Name Service

FTP File Transfer Protocol

HTTP Hypertext Transfer Protocol

IDPS Intrusion Detection and Prevention System

IDS Intrusion Detection System

IPS Intrusion Prevention System

LDAP Lightweight Directory Access Protocol

LMU Ludwig-Maximilians-Universität

LRZ Leibniz-Rechenzentrum der bayerischen Akademie für Wissenschaften

MAC Mandatory Access Control

MVC Model View Controler

NBA Network Behavior Analysis

NFS Network File System

OpenSSL Open Secure Socket Layer

OWASP Open Web Application Security Project

RSA Rivest, Shamir and Adleman - Kryptosystem

SCP Secure Copy

SFTP Secure File Transfer Protocol

SQL Structured Query Language

SSH Secure Shell

TCP Transmission Control Protocol

telnet Teletype Network

TLS Transport Layer Security

TUM Technische Universität München

URL Uniform Resource Locator

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[abo 15] *About Codeacademy*, 2015. https://www.codecademy.com/about, Zugriff am 18.12.2015.

[Apac 16] APACHE SOFTWARE FOUNDATION : *Apache HTTP Server Project*, 2016. https://httpd.apache.org/, Zugriff am 15.3.2016.

[app 11] *Getting Started*. AppArmor Wiki, 2011. http://wiki.apparmor.net/index.php/GettingStarted, Zugriff am 18.1.2016.

[app 15] *AppArmor Project Wiki*, 2015. http://wiki.apparmor.net/index.php/Main_Page, Zugriff am 27.1.2016.

[arm 02] *America's Army*, 2002. Computerspiel, www.americasarmy.com, Zugriff am 3.1.2016.

[Bart 96] BARTLE, RICHARD: *Hearts, clubs, diamonds, spades: Players who suit MUDs*. Journal of MUD research, 1(1):19, 1996. http://mud.co.uk/richard/hcds.htm, Zugriff am 11.5.2016.

[BBL 10] BERGSTRÖM, KARL, STAFFAN BJÖRK und SUS LUNDGREN: *Exploring aesthetical gameplay design patterns: camaraderie in four games*. In: *Proceedings of the 14th International Academic MindTrek Conference: Envisioning Future Media Environments*, Seiten 17–24. ACM, 2010.

[BCH 08] BRAY, RORY, DANIEL CID und ANDREW HAY: *OSSEC host-based intrusion detection guide*. Syngress, 2008.

[Beck 04] BECK, JOHN C: *Got game: How the gamer generation is reshaping business forever*. Harvard Business Press, 2004.

[BjHo 05] BJÖRK, S. und J. HOLOPAINEN: *Patterns in Game Design*. Charles River Media, Hingham MA, 2005.

[Bliz 04] BLIZZARD ENTERTAINMENT: *World of Warcraft*, 2004. Computerspiel, http://eu.battle.net/wow/de/, Zugriff am 5.1.2015.

[BoFr 03] BOGOST, IAN und GONZALO FRASCA: *Howard Dean for Iowa*, 2003. Computerspiel, http://www.deanforamericagame.com/, Zugriff am 3.1.2016.

[Bogo 07] BOGOST, IAN: *Persuasive games: The expressive power of videogames*. MIT Press, 2007.

[Brod 85] BRODERBUND SOFTWARE: *Where in the World is Carmen Sandiego?*, 1985. Computerspiel.

[BuBr] BURAK, ASI und ERIC BROWN: *Peacemaker*. Computerspiel, `http://peacemakergame.com/`, Zugriff am 3.1.2016.

[Cano 10a] CANONICAL LTD.: *aa_status Manpage Ubuntu 14.04*. Ubuntu Manuals, 2010. `http://manpages.ubuntu.com/manpages/trusty/man8/apparmor_status.8.html`, Zugriff am 18.1.2016.

[Cano 10b] CANONICAL LTD.: *apparmor Manpage Ubuntu 14.04*. Ubuntu Manuals, 2010. `http://manpages.ubuntu.com/manpages/trusty/man7/apparmor.7.html`, Zugriff am 18.1.2016.

[Cass 16] CASSEE, JOOST: *TinyMCE integration for Django*, 2016. `https://github.com/aljosa/django-tinymce`, Zugriff am 16.5.2016.

[CGCC 10] CALVILLO-GÁMEZ, EDUARDO H, PAUL CAIRNS und ANNA L COX: *Assessing the core elements of the gaming experience*. In: *Evaluating user experience in games*, Seiten 47–71. Springer, 2010.

[Char 10] CHARSKY, DENNIS: *From edutainment to serious games: A change in the use of game characteristics*. Games and Culture, 5(2):177–198, 2010.

[cod 15] *Learn to Code - Codeacademy*, 2015. `www.codeacademy.com`, Zugriff am 16.12.2015.

[CsCs 91] CSIKSZENTMIHALYI, MIHALY und MIHALY CSIKZENTMIHALY: *Flow: The psychology of optimal experience*, Band 41. HarperPerennial New York, 1991.

[DDKN 11a] DETERDING, SEBASTIAN, DAN DIXON, RILLA KHALED und LENNART NACKE: *From game design elements to gamefulness: defining gamification*. In: *Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments*, Seiten 9–15. ACM, 2011.

[DDKN 11b] DETERDING, SEBASTIAN, DAN DIXON, RILLA KHALED und LENNART NACKE: *Gamification: Toward a Definition*. In: *Proceedings of the CHI Conference on Human Factors in Computing Systems 2011*, Seiten 9–15. ACM, 2011.

[DeRy 11] DECI, EDWARD L und RICHARD M RYAN: *Self-determination theory*. Handbook of theories of social psychology, 1:416–433, 2011.

[Djan 16a] DJANGO SOFTWARE FOUNDATION: *Django - The web framework for perfectionists with deadlines.*, 2016. `https://www.djangoproject.com/`, Zugriff am 15.3.2016.

[Djan 16b] DJANGO SOFTWARE FOUNDATION: *FAQ: General*, 2016. `https://docs.djangoproject.com/en/1.9/faq/general/#django-appears-to-be-a-mvc-framework-but-you-call-the-controller-the-view-and-the-view-the-template-`

how-come-you-don-t-use-the-standard-names, Zugriff am 14.4.2016.

[Djan 16c] DJANGO SOFTWARE FOUNDATION: *Security in Django*, 2016. https://docs.djangoproject.com/en/1.9/topics/security/, Zugriff am 31.3.2016.

[Djan 16d] DJANGO SOFTWARE FOUNDATION: *Why Django?*, 2016. https://www.djangoproject.com/start/overview/, Zugriff am 15.3.16.

[DKK+ 14] DEIMEKE, DIRK, STEFAN KANIA, CHARLY KÜHNAST, DANIEL VAN SOEST und PEER HEINLEIN: *Linux-Server - Das umfassende Handbuch*. Galileo Computing, 2014.

[DPJV 06] DESMET, LIEVEN, FRANK PIESSENS, WOUTER JOOSEN und PIERRE VERBAETEN: *Bridging the gap between web application firewalls and web applications*. In: *Proceedings of the fourth ACM workshop on Formal methods in security*, Seiten 67–77. ACM, 2006.

[Edwi 15] EDWIN, HEERA: *4 Reasons to use gamification in 2016*, 2015. http://24x7learning.com/blog/4-reasons-to-use-gamification-in-2016/, Zugriff am 10.1.2016.

[Epho 16] EPHOX: *TinyMCE — The Most Advanced WYSIWYG HTML Editor*, 2016. https://www.tinymce.com/, Zugriff am 16.5.2016.

[Fogg 09] FOGG, BRIAN J: *A behavior model for persuasive design*. In: *Proceedings of the 4th international Conference on Persuasive Technology*, Seite 40. ACM, 2009.

[Full 14] FULLERTON, TRACY: *Game design workshop: a playcentric approach to creating innovative games*. CRC press, 2014.

[Goog 16] GOOGLE INC.: *Professionelle Formulare leicht gemacht*, 2016. https://www.google.com/intl/de_de/forms/about/, Zugriff am 2.5.2016.

[Harr 07] HARRIS, SHON: *CISSP All-in-One Exam Guide*. Logical Security, 2007.

[Hass 15] HASSO-PLATTNER-INSTITUT: *Statistiken*, 2015. https://sec.hpi.uni-potsdam.de/leak-checker/statistics, Zugriff am 25.1.16.

[Hipp 16a] HIPP, RICHARD: *Most Widely Deployed and Used Database Engine*, 2016. https://www.sqlite.org/mostdeployed.html, Zugriff am 19.3.2016.

[Hipp 16b] HIPP, RICHARD: *SQLite*, 2016. https://www.sqlite.org/, Zugriff am 19.3.2016.

[HLZ 04] HUNICKE, ROBIN, MARC LEBLANC und ROBERT ZUBEK: *MDA: A formal approach to game design and game research*. In: *Proceedings of the AAAI Workshop on Challenges in Game AI*, Band 4, 2004.

[IEEE 03] IEEE/THE OPEN GROUP: *POSIX 1003.1 - man page for crontab*, 2003. `http://www.unix.com/man-page/posix/1posix/crontab`, Zugriff am 13.1.2016.

[Kofl 14] KOFLER, MICHAEL: *Linux - Das umfassende Handbuch*. Galileo Computing, 2014.

[Kohn 99] KOHN, ALFIE: *Punished by rewards: The trouble with gold stars, incentive plans, A's, praise, and other bribes*. Houghton Mifflin Harcourt, 1999.

[KrFr 13] KRŠKO, ONDREJ und ULLI FREIBERGER: *Robot Karol*, 2013. Computerspiel, `https://www.mebis.bayern.de/infoportal/faecher/mint/inf/robot-karol/`, Zugriff am 18.12.2015.

[KSBG 13] KHANDELWAL, SHASHANK, PARTHIV SHAH, MR KAUSHAL BHAVSAR und SAVITA GANDHI: *Frontline techniques to prevent web application vulnerability*. International Journal of Advanced Research in Computer Science and Electronics Engineering (IJARCSEE), 2(2):pp–208, 2013.

[lig 16] *Lightbot*, 2016. Computerspiel, `https://lightbot.com/`, Zugriff am 5.1.2016.

[LR a] LEIBNIZ-RECHENZENTRUM: *Das Leibniz-Rechenzentrum - Image-Broschüre*. `https://www.lrz.de/wir/lrz-flyer/lrz_Image_broschuere.pdf`, Zugriff am 31.1.2016.

[LR b] LEIBNIZ-RECHENZENTRUM: *Leibniz-Rechenzentrum der Bayerischen Akademie der Wissenschaften*. `www.lrz.de`, Zugriff am 8.1.2016.

[LR 14] LEIBNIZ-RECHENZENTRUM: *Geschichte des Leibniz-Rechenzentrums*, 2014. `https://www.lrz.de/wir/geschichte/`, Zugriff am 31.1.2016.

[LR 15] LEIBNIZ-RECHENZENTRUM: *Jahresbericht 2014*, 2015. `https://www.lrz.de/wir/berichte/JB/JBer2014.pdf`, Zugriff am 31.1.2016.

[Lyon 16] LYON, GORDON: *SecTools.Org: Top 125 Network Security Tools*, 2016. `http://sectools.org/tag/ids/`, Zugriff am 21.1.2016.

[MaLe 87] MALONE, THOMAS W und MARK R LEPPER: *Making learning fun: A taxonomy of intrinsic motivations for learning*. Aptitude, learning, and instruction, 3(1987):223–253, 1987.

[Malo 81] MALONE, THOMAS W: *Toward a theory of intrinsically motivating instruction*. Cognitive science, 5(4):333–369, 1981.

[Marx 09] MARX, STEFAN: *Web Application Firewalls - Grundlagen und Marktübersicht*. TecChannel - IT im Mittelstand, 2009. `http://www.tecchannel.de/webtechnik/webserver/2019855/web_application_firewall_waf_uebersicht_grundlagen/`, Zugriff am 19.1.2016.

[McGo 11] MCGONIGAL, JANE: *Reality is broken: Why games make us better and how they can change the world*. Penguin, 2011.

[Merr 16] MERRIAM WEBSTER: *telnet*, 2016. `http://www.merriam-webster.com/dictionary/telnet`, Zugriff am 16.1.2016.

[MiCh 05] MICHAEL, DAVID R und SANDRA L CHEN: *Serious games: Games that educate, train, and inform*. Muska & Lipman/Premier-Trade, 2005.

[Mold 08] MOLDENHAUER, JÖRG: *Serious Games*. In: STRENG, SARA, DOMINIKUS BAUR, GREGOR BROLL, ALEXANDER DE LUCA, RAPHAEL WIMMER und ANDREAS BUTZ (Herausgeber): *Trends in E-Learning*, 2008.

[Netc 16] NETCRAFT: *March 2016 Web Server Survey*, 2016. `http://news.netcraft.com/archives/2016/03/18/march-2016-web-server-survey.html`, Zugriff am 19.3.2016.

[Nint 08] NINTENDO: *Wii Fit*, 2008. `https://www.nintendo.de/Spiele/Wii/Wii-Fit-283894.html#_bersicht`, Zugriff am 3.1.2016.

[Open 16a] OPENBSD: *OpenSSH - keeping your communiqués secret*, 2016. `http://www.openssh.com/`, Zugriff am 16.1.2016.

[Open 16b] OPENBSD: *ssh-keygen Manual Page*, 2016. `http://www.openbsd.org/cgi-bin/man.cgi/OpenBSD-current/man1/ssh-keygen.1?query=ssh-keygen&sec=1`, Zugriff am 17.1.2016.

[Open 16c] OPENBSD: *ssh Manual Page*, 2016. `http://www.openbsd.org/cgi-bin/man.cgi/OpenBSD-current/man1/ssh.1?query=ssh&sec=1`, Zugriff am 16.1.2016.

[oss 16] *OpenSSH Project History and Credits*, 2016. `http://www.openssh.com/history.html`, Zugriff am 27.1.2016.

[OtTh 15] OTTO, MARK und JACOB THORNTON: *Bootstrap: The world's most popular mobile-first and responsive front-end framework.*, 2015. `http://getbootstrap.com/`, Zugriff am 19.3.2016.

[Patt 81] PATTIS, RICHARD E: *Karel the robot: a gentle introduction to the art of programming*. John Wiley & Sons, Inc., 1981.

[Pink 11] PINK, DANIEL H: *Drive: The surprising truth about what motivates us*. Penguin, 2011.

[Pitt 07] PITT, MARTIN: *AppArmor Profil für cupsd*, 2007.

[PlWe 12] PLÖTNER, JOHANNES und STEFFEN WENDZEL: *Linux - Das umfassende Handbuch*. Rheinwerk Computing, 2012.

[pro 13] *A quick guide to AppArmor profile Language*. AppArmor Wiki, 2013. `http://wiki.apparmor.net/index.php/QuickProfileLanguage`, Zugriff am 18.1.2016.

[PrRa 15] PRAKASH, EDMOND C und MADHUSUDAN RAO: *Transforming Learning and IT Management through Gamification*. Springer, 2015.

[Pyth] PYTHON SOFTWARE FOUNDATION: *Welcome to Python.org*. https://www.python.org/, Zugriff am 15.3.2016.

[Pyth 16] PYTHON SOFTWARE FOUNDATION: *Whetting Your Appetite*, 2016. https://docs.python.org/3/tutorial/appetite.html, Zugriff am 15.3.2016.

[RCV 09] RITTERFELD, UTE, MICHAEL CODY und PETER VORDERER: *Serious games: Mechanisms and effects*. Routledge, London, 2009.

[rob 11] *Building a city with Robot Karol*, 2011. https://i.ytimg.com/vi/GEFpSsfeouA/maxresdefault.jpg.

[RoMa 13] RODRIGUEZ, CHRIS und RICHARD MARTINEZ: *The Growing Hacking Threat to Websites: An Ongoing Commitment to Web Application Security*. Frost & Sullivan. Retrieved, 13, 2013.

[RRBF 11] RUSU, ADRIAN, ROBERT RUSSELL, EDWARD BURNS und ANDREW FABIAN: *Employing software maintenance techniques via a tower-defense serious computer game*. In: *Edutainment Technologies. Educational Games and Virtual Reality/Augmented Reality Applications*, Seiten 176–184. Springer, 2011.

[Sage 09] SAGERSON, PETER: *Django Authentication Using LDAP*, 2009. https://pythonhosted.org/django-auth-ldap/, Zugriff am 15.3.2016.

[Scha 07] SCHAFFER, NOAH: *Heuristics for usability in games*. White Paper, Seiten 1–30, 2007.

[ScMe 07] SCARFONE, KAREN und PETER MELL: *Guide to intrusion detection and prevention systems (idps)*. NIST special publication, 800(2007):94, 2007.

[sel 13] *SELinux Project Wiki*, 2013. http://selinuxproject.org/page/Main_Page, Zugriff am 27.1.2016.

[Soft 16] SOFTWARE IN THE PUBLIC INTEREST, INC. : *Debian - Das universelle Betriebssystem*, 2016. https://www.debian.org/index.de.html.

[Stat 16] STATISTISCHES AMT, LANDESHAUPTSTADT MÜNCHEN: *München in Zahlen*, 2016. http://www.muenchen.de/sehenswuerdigkeiten/muenchen-in-zahlen.html#einwohner, Zugriff am 31.1.2016.

[STT 14] SHAUCHENKA, NATALLIA, ANABEL TERNÈS und IAN TOWERS: *Gamification*. In: *Internationale Trends in der Markenkommunikation*, Seiten 33–50. Springer, 2014.

[Team 15] TEAM, OSSEC PROJECT: *Home - OSSEC*, 2015. http://ossec.github.io/, Zugriff am 21.1.2016.

[The 13] THE OWASP FOUNDATION: *OWASP Top 10 - 2013*, 2013. https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project, Zugriff am 30.1.2016.

[The 15] THE FREEBSD DOCUMENTATION PROJECT: *The inetd Super-Server*. The FreeBSD Handbook, 1995-2015. `https://www.freebsd.org/doc/handbook/network-inetd.html`, Zugriff am 17.1.2016.

[thi 14] *Social Gaming Statistic 2014 - Revenues, Purchasing & Demographics*. thinkmanagement.com, 2014. `http://www.thinkmanagement.com/social-gaming-statistics-2014-revenues-purchasing-demographics/`, Zugriff am 30.12.2015.

[util 14] UTIL-LINUX PROJECT: *mount Manual Page*, 2014. `http://man7.org/linux/man-pages/man8/mount.8.html`, Zugriff am 27.1.2016.

[Vene 16] VENEMA, WIETSE: *hosts.allow(5) - Linux man page*, 2016. `http://linux.die.net/man/5/hosts.allow`, Zugriff am 17.1.2016.

[Virt 03] VIRTUAL U PROJECT: *Virtual U*, 2003. Computerspiel.

[Web 06] WEB APPLICATION SECURITY CONSORTIUM: *Web Application Firewall Evaluation Criteria*, 2006. `http://projects.webappsec.org/f/wasc-wafec-v1.0.pdf`, Zugriff am 29.1.2016.

[ZiCu 11] ZICHERMANN, GABE und CHRISTOPHER CUNNINGHAM: *Gamification by design: Implementing game mechanics in web and mobile apps*. O'Reilly Media, Inc., 2011.

A. Installationsanleitung

Im Folgenden wird die Installation der Anwendung erläutert.

- Benötigte Pakete für Django inklusive LDAP-Backend: `libapache2-mod-wsgi`, `python-django`, `python-auth-ldap`, `python-pip`
- Konfiguration von Django als Daemon in `/etc/apache2/apache2.conf` : vgl. Listing A.1

Listing A.1: Konfiguration von Django als Daemon in apache2.conf

```
1 WSGIDaemonProcess lisc.srv.lrz.de python-path=/var/www/mysite
2 WSGIProcessGroup lisc.srv.lrz.de
3 WSGIScriptAlias / /var/www/mysite/mysite/wsgi.py
4
5 <Directory /var/www/mysite/mysite>
6 <Files wsgi.py>
7 Require all granted
8 </Files>
9 </Directory>
```

- `django-tinymce` über `pip` nachinstallieren: `pip install django-tinymce`
- Django-Projekt liegt in `/var/www/mysite`
- Konfiguration LDAP-Backend, Verzeichnis für Static Files etc. in `/var/www/mysite/mysite/settings.py` (vgl. Listing A.2)

Listing A.2: Konfiguration des Django-Projekts in settings.py

```
1  """
2  Django settings for mysite project.
3
4  For more information on this file, see
5  https://docs.djangoproject.com/en/1.7/topics/settings/
6
7  For the full list of settings and their values, see
8  https://docs.djangoproject.com/en/1.7/ref/settings/
9  """
10
11 # Build paths inside the project like this: os.path.join(BASE_DIR,
       ...)
12 import os
```

```
13 BASE_DIR = os.path.dirname(os.path.dirname(__file__))
14 TEMPLATE_DIRS = [os.path.join(BASE_DIR, 'templates')]
15
16
17 # Quick-start development settings - unsuitable for production
18 # See https://docs.djangoproject.com/en/1.7/howto/deployment/
      checklist/
19
20 # SECURITY WARNING: keep the secret key used in production secret!
21 SECRET_KEY = '!um5oe-4-_b$c&#tgrin#%+u-w(-9e^mgj!llf-+tveuhmzr#_'
22
23 # SECURITY WARNING: don't run with debug turned on in production!
24 DEBUG = False
25
26 TEMPLATE_DEBUG = False
27
28 ALLOWED_HOSTS = ['127.0.0.1','lisc.srv.lrz.de']
29
30
31 # Application definition
32
33 INSTALLED_APPS = (
34     'django.contrib.admin',
35     'django.contrib.auth',
36     'django.contrib.contenttypes',
37     'django.contrib.sessions',
38     'django.contrib.messages',
39     'django.contrib.staticfiles',
40     'serveradmin',
41     'tinymce',
42 )
43
44 MIDDLEWARE_CLASSES = (
45     'django.contrib.sessions.middleware.SessionMiddleware',
46     'django.middleware.common.CommonMiddleware',
47     'django.middleware.csrf.CsrfViewMiddleware',
48     'django.contrib.auth.middleware.AuthenticationMiddleware',
49     'django.contrib.auth.middleware.
          SessionAuthenticationMiddleware',
50     'django.contrib.messages.middleware.MessageMiddleware',
51     'django.middleware.clickjacking.XFrameOptionsMiddleware',
52 )
53
54 import ldap
55 from django_auth_ldap.config import LDAPSearch, LDAPSearchUnion,
      GroupOfNamesType
56
57
```

```
58 # Baseline configuration.
59 AUTH_LDAP_SERVER_URI = "<URI␣LDAP-Server>"
60
61 AUTH_LDAP_BIND_DN = "CN=<Kennung>,OU=Kerberos,OU=Kennungen,o=lrz-
      muenchen,c=de"
62 AUTH_LDAP_BIND_PASSWORD = "<Passwort>"
63 AUTH_LDAP_USER_SEARCH = LDAPSearch("ou=Kerberos,ou=Kennungen,o=lrz
      -muenchen,c=de",
64     ldap.SCOPE_SUBTREE, "(uid=%(user)s)")
65
66 # This is the default, but I like to be explicit.
67 AUTH_LDAP_ALWAYS_UPDATE_USER = True
68
69 # Cache group memberships for an hour to minimize LDAP traffic
70 AUTH_LDAP_CACHE_GROUPS = True
71 AUTH_LDAP_GROUP_CACHE_TIMEOUT = 3600
72
73 AUTH_LDAP_CONNECTION_OPTIONS = {
74         ldap.OPT_DEBUG_LEVEL: 0,
75         ldap.OPT_REFERRALS: 0,
76 }
77
78 # Keep ModelBackend around for per-user permissions and maybe a
      local
79 # superuser.
80 AUTHENTICATION_BACKENDS = (
81     'django_auth_ldap.backend.LDAPBackend',
82     'django.contrib.auth.backends.ModelBackend',
83 )
84
85 ROOT_URLCONF = 'mysite.urls'
86
87 WSGI_APPLICATION = 'mysite.wsgi.application'
88
89 LOGIN_URL = 'login'
90
91 LOGIN_REDIRECT_URL = 'serveradmin:profile'
92
93 # Database
94 # https://docs.djangoproject.com/en/1.7/ref/settings/#databases
95
96 DATABASES = {
97     'default': {
98         'ENGINE': 'django.db.backends.sqlite3',
99         'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
100     }
101 }
102
```

```
103 # Internationalization
104 # https://docs.djangoproject.com/en/1.7/topics/i18n/
105
106 LANGUAGE_CODE = 'en-us'
107
108 TIME_ZONE = 'Europe/Berlin'
109
110 USE_I18N = True
111
112 USE_L10N = True
113
114 USE_TZ = True
115
116
117 # Static files (CSS, JavaScript, Images)
118 # https://docs.djangoproject.com/en/1.7/howto/static-files/
119
120 STATIC_URL = '/static/'
121 STATIC_ROOT = '/var/www/mysite/static'
122
123 # Uploaded Images
124 MEDIA_URL = '/media/'
125 MEDIA_ROOT = '/var/www/mysite/media/'
```

- Static files in passenden Ordner kopieren: `python /var/www/mysite/manage.py collectstatic` (bei Kopie des kompletten Projekts ohne Änderungen nicht notwendig)
- Änderungen an der Datenbank übernehmen: `python /var/www/mysite/manage.py makemigrations`, `python /var/www/mysite/manage.py migrate` (bei Kopie des kompletten Projekts ohne Änderungen nicht notwendig)
- Änderungen an der Anwendung übernehmen: `touch /var/www/mysite/mysite/wsgi.py`

B. Umfrage

Im Folgenden wird die Umfrage für die Evaluation der Anwendung wiedergegeben.

B.1. Learning Interactive Server Configuration - Evaluation Part 1

Please answer these questions on your level of skills before starting the lessons in the application.

- Where do you work?
- What is your position there?
- Are you experienced in working with Linux?

```
#/etc/hosts.allow

portmap:ALL EXCEPT 192.168.4.145
ALL:.srv.lrz.de
ALL:ALL:rfc931:DENY
```

Abbildung B.1.: Example configuration in /etc/hosts.allow

- **TCP Wrapper**: In hosts.allow, allow host lisc.srv.lrz.de access to all services on your server!
- **TCP Wrapper**: Allow all hosts in the network 192.168.5.x access to sshd!
- **TCP Wrapper**: Allow access to all services from hosts in the local network!
- **TCP Wrapper**: In your hosts.allow, DENY access to all services to host lisc.srv.lrz.de!
- **AppArmor**: In its AppArmor profile, allow tcpdump to change the owner (chown) of a file!
- **AppArmor**: Allow tcpdump TCP access to the network!
- **AppArmor**: Grant tcpdump read/write access to the files in the directory /home/user/*!
- **AppArmor**: Explicitly deny write access to /home/user/test.txt!

```
#/etc/apparmor.d/usr.sbin.tcpdump

capability setuid,
capability setgid,

network raw,

audit deny @{HOME}/.*/ rw,
```

Abbildung B.2.: Example configuration of an AppArmor profile

B.2. Learning Interactive Server Configuration - Evaluation Part 2

Please answer these questions after completing the lessons in the application. They will retest your level of skills and ask about your experiences.

```
#/etc/hosts.allow

portmap:ALL EXCEPT 192.168.4.145
ALL:.srv.lrz.de
ALL:ALL:rfc931:DENY
```

Abbildung B.3.: Example configuration in /etc/hosts.allow

- **TCP Wrapper**: In hosts.allow, allow host lisc.srv.lrz.de access to all services on your server!
- **TCP Wrapper**: Allow all hosts in the network 192.168.5.x access to sshd!
- **TCP Wrapper**: Allow access to all services from hosts in the local network!
- **TCP Wrapper**: In your hosts.allow, DENY access to all services to host lisc.srv.lrz.de!

```
#/etc/apparmor.d/usr.sbin.tcpdump

capability setuid,
capability setgid,

network raw,

audit deny @{HOME}/.*/ rw,
```

Abbildung B.4.: Example configuration of an AppArmor profile

- **AppArmor**: In its AppArmor profile, allow tcpdump to change the owner (chown) of a file!
- **AppArmor**: Allow tcpdump TCP access to the network!
- **AppArmor**: Grant tcpdump read/write access to the files in the directory /home/user/*!
- **AppArmor**: Explicitly deny write access to /home/user/test.txt!
- Was the application easy to use and did you understand the instructions? Were there any problems?
- Did you enjoy learning about Linux Server Configuration with the application?