INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Masterarbeit

Managementsystem für Sicherheitstests in einem Hochschulrechenzentrum

Maximilian Wirtz

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Masterarbeit

Managementsystem für Sicherheitstests in einem Hochschulrechenzentrum

Maximilian Wirtz

Aufgabensteller: Prof. Dr. Helmut Reiser

Betreuer: Dipl.-Inform. Stefan Metzger
M.A. Tanja Hanauer

Abgabetermin: 19. Oktober 2016

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

München, den 19. Oktober 2016

. . . . . .

(Unterschrift des Kandidaten)

Das Thema Informationssicherheit gewinnt täglich an Bedeutung. Auch an Hochschulrechenzentren nimmt die Bedeutung, z.B. aufgrund gesetzlicher Anforderungen, zu. Um den Stand der Informationssicherheit und die Wirksamkeit der implementierten Maßnahmen zu prüfen, sind Sicherheitstests das Mittel der Wahl.

Sicherheitstests ermöglichen es Schwachstellen zu identifizieren, um sie anschließend beheben zu können. Tests an produkiven Systemen stellen allerdings eine Gefahr für den reibungslosen Betrieb der Systeme dar. Aus diesem Grund müssen Sicherheitstests detailiert geplant und während der Ausführung begleitet werden. Auch die Nachverfolgung der gefundenen Schwachstellen muss gewährleistet sein.

In dieser Arbeit wurde dazu ein Managementsystem konzipiert, um regelmäßig Sicherheitstests an Hochschulrechenzentren durchzuführen und die gefundenen Schwachstellen zu managen. Um die Prozesse zu unterstützen wurde des Weiteren eine Anwendung dazu konzipiert und implementiert.

Dieses Managementsystem für Sicherheitstests kann von Hochschulrechenzentren angewendet werden, um die Dienste des Hochschulrechenzentrums regelmäßig auf ihre Sicherheit hin zu überprüfen. Durch die prozessunterstützende Anwendung ist es sogar möglich, einige Aktivitäten zu automatisieren, sodass eine Einführung ressourcenschonend möglich ist.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

Das Thema IT-Sicherheit gewinnt immer mehr an Bedeutung. Beinahe wöchentlich erfährt man in den Medien von erfolgreichen Hacker Angriffen auf Unternehmen, Behörden oder Privatpersonen. So wurden der Zentralbank von Bangladesch im März 2016 knapp eine Milliarde US Dollar entwendet [Sok16]. Auch beim deutschen Bundestag wurde im Mai 2015 bekannt, dass dieser Opfer eines Angriffs war, bei dem die Angreifer mehr als 16 Gigabytes an E-Mails kopierten. Das ganze Ausmaß ist bis heute noch nicht klar [Sch15].

Item	2014 Cost	Uses
1,000 Stolen Email Addresses	$0.50 to $10	Spam, Phishing
Credit Card Details	$0.50 to $20	Fraudulent Purchases
Scans of Real Passports	$1 to $2	Identity Theft
Stolen Gaming Accounts	$10 to $15	Attaining Valuable Virtual Items
Custom Malware	$12 to $3500	Payment Diversions, Bitcoin Stealing
1,000 Social Network Followers	$2 to $12	Generating Viewer Interest
Stolen Cloud Accounts	$7 to $8	Hosting a Command-and-Control (C&C) Server
1 Million Verified Email Spam Mail-outs	$70 to $150	Spam, Phishing
Registered and Activated Russian Mobile Phone SIM Card	$100	Fraud

Abbildung 1.1.: Datenpreise auf Schwarzmärkten [SYM15]

Mittlerweile hat sich ein echter Markt für gestohlene Daten entwickelt. So sind z.B. 1000 gestohlene E-Mailadressen laut Symmantec, einem großen Softwarehersteller mit Fokus auf IT-Sicherheit, 0,50 bis 10 US Dollar wert (Abb. 1.1). Bitcoins dienen meist als Zahlungsmittel und mittels TOR-Netzwerk können anonym Marktplätze betrieben werden [SYM15]. Die gestohlene Daten können dann von anderen Kriminellen gekauft und weiterverwendet werden. Gestohlenen E-Mailadressen können zum Beispiel als Ziele für Spam und Phishing dienen, um an weitere Daten beispielsweise Bankdaten zu gelangen.

Des Weiteren trägt der Trend zur Digitalisierung dazu bei, immer mehr Daten zum einen zu speichern und zum anderen über Webschnittstellen zu nutzen. So gibt es nun mehr Webanwendungen als potentielle Ziele für Angreifer und mehr Daten zu kopieren bzw. zu stehlen. Mussten früher einige Webanwendungen erfolg-

reich angegriffen werden um mehrere Tausend E-Mailadressen zu erhalten, reicht heute ein Angriff. Auch durch die immense Anzahl an Anwendungen in Unternehmen, wird es für diese deutlich schwerer bei allen Anwendungen für deren Sicherheit zu sorgen. Dies steigert die Attraktivität von Angriffen enorm.

Auch an Hochschulen und Universitäten hat die Digitalisierung längst Einzug gehalten. Studenten bekommen bei der Immatrikulation sofort mindestens eine E-Mailadresse. Noten können online eingesehen werden und Stundenpläne können angelegt und bearbeitet werden. Auch in der Forschung, gerade bei Kooperationen zwischen mehreren Forschungseinrichtungen, ist die Digitalisierung essentiell. So sind Hochschulen und Universitäten für Spammer und Kreditkartenbetrüger ein rentables Ziel. Forschungsergebnisse oder besonders Zwischenergebnisse können ein attraktives Ziel darstellen. Gerade im Hinblick auf Rüstungsforschung könnten nicht nur Kleinkriminelle, sondern auch Fremdstaaten eine Rolle als Angreifer spielen.

Um Schwachstellen, die Angriffe erst ermöglichen, im Vorfeld zu erkennen und zu beheben, kann man Anwendungen auf ihre Sicherheit hin prüfen. Dies kann mittels Sicherheitstests, etwa automatisiertem Schwachstellenscanner oder manuell mit Hilfe eines Penetrationtests stattfinden.

Das Leibniz Rechenzentrum (LRZ) als zentraler Dienstleister für die beiden Eliteuniversitäten Technische Universität München und Ludwig-Maximilians Universität, sowie für weiterer Hochschulen im Münchner Wissenschaftsnetz, z.B. Hochschule München, ist bestrebt, seine Dienste und Anwendungen möglichst sicher zu halten und vor Angriffen und Datendiebstahl zu schützen.

1.1. Motivation

Bei der Cyber-Sicherheits-Umfrage 2015 durch die Allianz für Cyber-Sicherheit wurden Unternehmen nach häufigen Ursachen von erfolgreichen Angriffen befragt. Dabei wurden u.a. folgende Ursachen identifiziert. (Abb. 1.2)

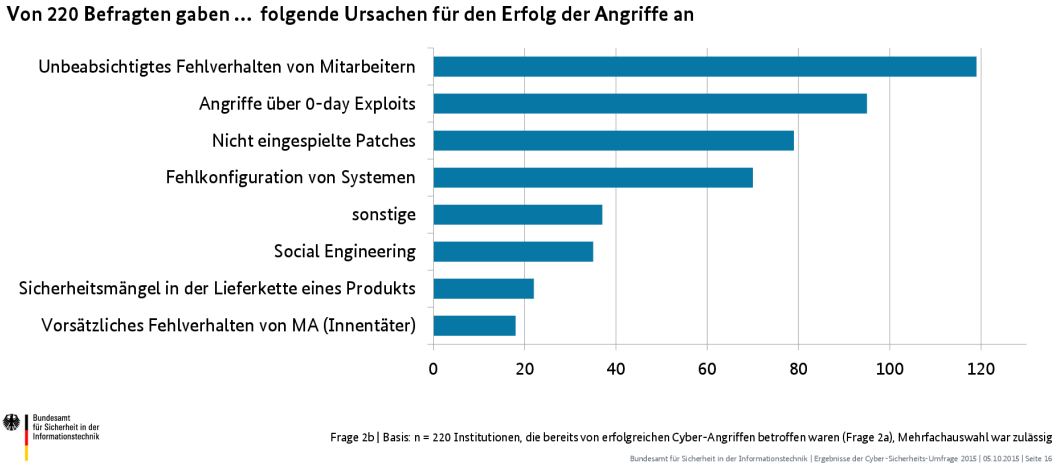


Abbildung 1.2.: Ursachen erfolgreicher Angriffe [All15]

Eine Vielzahl dieser Ursachen (bspw. nicht eingespielte Patches, Fehlkonfigurationen, etc.) können mit Hilfe von Sicherheitstests entdeckt und anschließend behoben werden. So gaben in dieser Befragung auch circa 150 von 424 befragten Unternehmen an regelmäßige Penetrationtests bzw. Revisionen durchzuführen.

Da es sich bei Penetrationtests um simulierte Angriffe handelt, brauchen sie ein spezielles Vorgehen damit sie nicht fälschlicherweise als Angriff gewertet werden. Zumdem sollten die Ergebnisse bzw. die gefundenen Schwachstellen nachverfolgt und sichergestellt werden, dass sie nachhaltig behoben wurden.

In einem so großen und heterogenen Netzwerk wie dem Münchener Wissenschaftsnetz, kommen weitere Herausforderungen hinzu. So ist das Leibniz-Rechenzentrum zwar oftmals Betreiber von Infrastrukturkomponenten, jedoch werden dort nicht ausschließlich LRZ eigene Anwendungen ausgeführt. Somit erweitert sich der

Kreis der Beteiligten, die bei einem Sicherheitstest involviert sind. Zudem fehlt bei Anwendungsverantwortlichen teilweise Detailwissen, um mit allen Ergebnissen eines solchen Tests umgehen zu können. Es bietet sich an solche Sicherheitstests zentral zu managen und den Anwendungsverantwortlichen Unterstützung zu geben. Um solch einen Prozess und der Unterstützung durch eine Anwendung geht es in dieser Arbeit.

1.2. Zielsetzung

In dieser Arbeit soll der Fokus auf technische Schwachstellen liegen. Bei Angriffen auf Unternehmen und Behörden spielt jedoch nicht nur die Technik eine Rolle. Auch Mitarbeiter können i.d.R. ungewollt hilfreich für einen Angriff sein. Als Stichwort sei hier Social-Engineering genannt. Beim Social-Engineering werden Mitarbeiter dazu gebracht Handlungen für einen Angreifer auszuführen, die einzig Interesse des Angreifers stehen. Bspw. die Herausgabe von Passwörtern oder die Installation von Schadsoftware. Dies kann und sollte man auch regelmäßig testen um die Mitarbeiter dann bedarfsgerecht zu schulen. Derartige Tests sind nicht Gegenstand dieser Arbeit. In dieser Arbeit liegt der Fokus auf technischen Schwachstellen.

Ziel dieser Arbeit ist es, einen Prozess zur Durchführung von Sicherheitstests von Anwendungen am LRZ zu entwickeln. Also, wie eine Planung dazu aussehen kann, können interne Ressourcen beispielsweise automatisierte Scanner genutzt werden, oder muss eine Ausschreibung für einen externen Dienstleister durchgeführt werden? Wer muss informiert werden? Welche Vorlaufzeiten werden benötigt? Wer bekommt die Ergebnisse? Des Weiteren sollen auf die jeweiligen Zielgruppen zugeschnittene Berichte konzipiert werden. So benötigen die Entwickler Informationen wie solche Schwachstellen verhindert werden können mit Tipps für eine Implementierung. Ein Abteilungsleiter benötigt eventuell eine Zusammenfassung der aus den Schwachstellen resultierenden Risiken usw.

Weiterhin sollen Anwendungen mittels Risikomanagement identifiziert werden, bei denen Sicherheitstests dringend anzuraten sind. Eine statische Webseite mit ausschließlich öffentlichen Informationen, wie zum Beispiel Öffnungszeiten oder Kontaktadressen, stellt nur ein geringes Risiko für Datendiebstahl dar. Sie könnte aber als Einstiegspunkt ins interne Netz des LRZs dienen. Auch eine Webmail-Anwendung mit Zugang zu allen E-Mailpostfächern stellt ein großes Risiko für Datendiebstahl dar.

Um die Prozesse der Planung, Benachrichtigung, Berichtserstellung und Berichtsverteilung, sowie das Risikomanagement zu unterstützen, soll weiterhin eine Anwendung entworfen und implementiert werden. Dabei sollen vorhandene Schnittstellen und Prozesse, etwa zentrales Configuration Management und Schwachstellenmanagement, so gut wie möglich einbezogen werden.

Aufgrund der vielen Kunden des LRZs soll auch überlegt werden, Sicherheitstests als Service zu installieren. So könnten z.B. Lehrstühle mit eigenen Webauftritten, mit relativ geringem Aufwand ihre Anwendungen testen lassen und somit ein besseres Sicherheitsniveau erreichen.

1.3. Aufbau der Arbeit

Im folgenden Kapitel werden die Grundlagen für den weiteren Verlauf der Arbeit erläutert. So wird zunächst der Begriff Sicherheit im Zusammenhang mit Sicherheitstests, wie ein Informationssicherheits Managementsystem aufgebaut sein kann, welche Arten von Schwachstellen existieren und mit welchen Möglichkeiten getestet werden kann, erläutert.

Nachdem die Grundlagen gelegt wurden, werden die Herausforderungen im Hochschulkontext, insbesondere im Hinblick auf Sicherheitstests aufgezeigt und es werden die Anforderungen an ein Managementsystem für Sicherheitstests aufgestellt.

Anschließend werden Arbeiten und Konzepte gesucht und diskutiert, denen ähnliche Anforderungen zu Grunde liegen. Diese Konzepte werden dann dahingehend untersucht, ob sie die Anforderungen an ein Managementsystem für Sicherheitstests erfüllen können.

Nachfolgend soll mithilfe existierender Arbeiten ein Konzept für ein Managementsystem für Sicherheitstests entwickelt und beschrieben werden. Zur Unterstützung dieses Konzepts soll des Weiteren eine Anwendung konzipiert werden, die das Konzept unterstützt.

Im Anschluss darauf soll das Konzept für eine solche Anwendung implementiert werden und die Implementierung beschrieben werden.

Nachdem das Konzept entwickelt und eine Anwendung implementiert wurde, soll das Konzept und die Anwendung einmal prototypisch angewendet werden.

Abschließend wird das Ergebnis kritisch bewertet, ein Fazit gezogen und ein Ausblick gegeben.

2. Grundlagen

2.1. Sicherheit

Im Englischen gibt es für das deutsche Wort Sicherheit eigentlich zwei Worte: Security und Safety. Safety bedeutet, dass ein System sicher ist, also das keine Gefahr vom System aus geht. Das Wort Security meint, dass das System sicher vor externen Gefahren ist. Das deutsche Wort Sicherheit beinhaltet beide Begriffe. Bei Sicherheitstests werden allerdings die Security-Eigenschaften eines Systems geprüft, also die Gefahren die von außen kommen [Men15].

Die potentiellen Gefahren für ein System zielen im Allgemeinen auf die Vertraulichkeit, die Integrität und die Verfügbarkeit ab. Man spircht dabei auch oft von den Schutzzielen der Informationssicherheit.

Systeme beinhalten häufig vertrauliche Informationen (**Vertraulichkeit**). Um dies beurteilen zu können muss zuvor eine Datenklassifizierung vorgenommen werden. Im Allgemeinen sind z.B. Passwörter vertraulich und sollten unter keinen Umständen unbefugten Personen zugänglich gemacht werden. Eine Schwachstelle, die es einem Angreifer ermöglichen würde auf Passwörter eines Systems zuzugreifen, gefährdet somit die Vertraulichkeit der Daten.

Computersysteme dienen zur Datenverarbeitung. Hierbei ist es wichtig, dass man sich darauf verlassen kann, dass die Daten integer sind (**Integrität**). Sind die Daten nicht mehr integer, muss dies möglichst schnell erkennbar sein. Man stelle sich eine Bank vor, bei der Überweisungsdaten manipuliert wurden. Merkt die Bank dies frühzeitig, kann sie die Überweisung stoppen. Andernfalls wird die Überweisung mit beispielsweise einem falschen Betrag oder einer falschen IBAN ausgeführt, was zu einem finanziellen Schaden für die Bank führen kann.

Auch die **Verfügbarkeit** ist ein Schutzziel, da durch nicht erreichbare Systeme ebenfalls Schaden entstehen kann. Man stelle sich ein Onlineversandhaus vor, dass für mehrere Stunden keine Bestellungen aufnehmen kann. In diesem Fall entfallen die Umsätze mehrerer Stunden. Auch die Nichtverfügbarkeit einer Telefonanlage kann ein Problem darstellen, da man zum Beispiel bei einem Brand ggf. die Feuerwehr nicht mehr rechtzeitig rufen kann.

Ein Sicherheitstest soll nun zeigen, ob ein System anfällig für bestimmte externe Einflüsse ist. Dabei ist hervorzuheben, dass ein solcher Test lediglich das Vorhandensein einer Schwachstelle aufzeigen kann. Welche Schwachstellen ein System haben kann ist vielfältig und hängt davon ab, um was für eine Art von System es sich handelt.

2.2. Risiko

Eine Gefahr birgt immer einen möglichen Schaden. Das kann zum einen der erwähnte Umsatzausfall sein, zum anderen sind auch nicht direkt monetäre Schäden denkbar. Zum Beispiel bei einem Unfall, wenn es aufgrund eines Ausfalls der Telefonanlage nicht möglich ist Hilfe zu holen. Auch muss ein Schaden nicht in vollem Umfang eintreten, daher spricht man von Schadenspotential [Sto02].

Neben dem Schadenspotential hat eine Gefahr auch eine Eintrittswahrscheinlichkeit. Sie beschreibt wie häufig eine Gefahr eintreten und somit ein Schaden entstehen kann.

Ein Risiko setzt sich aus den beiden Größen, Schadenspotential und Eintrittswahrscheinlichkeit, zusammen. Um die Risiken und die daraus resultierenden Schäden zu verwalten und zu vermindern gibt es das Risikomanagement. Das Risikomanagement besteht hauptsächlich aus drei iterierenden Schritten (Abb. 2.1).

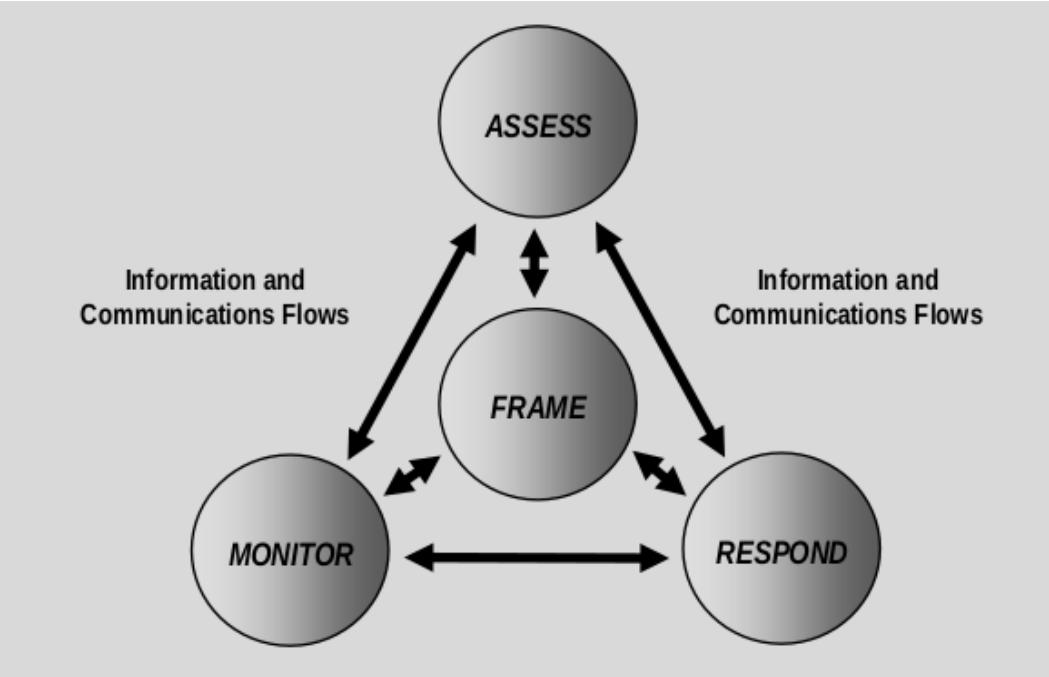


Abbildung 2.1.: Die Risikoprozess Schritte [Sto02]

2.2.1. Risiko Identifizierung und Analyse

Bei der Risiko Identifizierung und Analyse wird zunächst versucht Gefahren zu erkennen und das daraus resultierende Risiko zu ermitteln. Im Falle der Informationssicherheit kann dies unter anderem durch Sicherheitstests erfolgen. Andere Informationssicherheits Risiken, etwa der Ausfall eines Rechenzentrums aufgrund einer Umweltkatastrophe, muss anhand von Szenarien ermittelt werden. Dazu existieren Vorlagen zum Beispiel vom Bundesamt für Sicherheit in der Informationstechnologie [BSI16]. Anschließend müssen die potentiellen Schäden und die Eintrittswahrscheinlichkeit ermittelt werden.

2.2.2. Risikobewältigung oder Risikobeherrschung

Nachdem die Risiken identifiziert und analysiert wurden, muss mit diesen Risiken umgegangen werden. Man spricht auch von Risikobewältigung oder Risikobeherrschung. Dazu gibt es mehrere Möglichkeiten.

Man kann versuchen das Risiko zu mindern (**Risikominderung**) indem man risikomindernde Maßnahmen umsetzt. Durch Schließen einer Schwachstelle oder durch den Einsatz bestimmter Filter und Regeln (z.B. durch eine Web-Application Firewall) kann eine Ausnutzung verhindert werden. Durch eine Minderung bleiben jedoch oft noch Restrisiken bestehen. So kann man sich zum Beispiel gegen Denial-of-Service (DoS) Angriffe, also das Überlasten eines Systems, mit mehr Ressourcen besser schützen. Diese Angriffe haben jedoch prinzipiell keine Begrenzung. Das heißt, man kann sich nicht gegen beliebig große Angriffe schützen. Die Eintrittswahrscheinlichkeit eines solchen großen Angriffs sinkt jedoch mit größerem Einsatz von Ressourcen und somit sinkt auch das Risiko.

Auch kann man ein Risiko akzeptieren (**Risikoakzeptanz**). Dabei nimmt man das Risiko in Kauf, da Gegenmaßnahmen entweder nicht existieren oder weil diese in keinem Verhältnis zum Nutzen stehen. So kann man das Risiko eines Ausfalls des einzigen Rechenzentrums zwar erkannt haben, aber es für nicht wirtschaftlich halten, ein zweites Rechenzentrum parallel an einem entfernten Ort zu betreiben.

Eine weitere Möglichkeit ist der **Risikotransfer**. Dabei wird das Risiko auf einen Dritten transferiert. So kann man sich beispielsweise gegen Betriebsunterbrechungsschäden aufgrund Hacker-Angriffen versichern lassen. Auch andere Maßnahmen wie ein Content Delivery Network (CDN) auszulagern und als Dienstleitung

einzukaufen ist denkbar. Einige Anbieter haben sich auf die Abwehr von DoS Angriffen spezialisiert und können deutlich größere Ressourcen anbieten.

2.2.3. Risikokontrolle

Ist ein Risiko identifiziert und bewältigt worden, sollte es dennoch weiterhin kontrolliert werden. Oftmals ändern sich die Eintrittsvoraussetzungen oder es entstehen neue Maßnahmen, die ein Risiko mindern können, zum Beispiel durch neue Sicherheitssysteme. Nicht nur durch eine Veränderung der Eintrittswahrscheinlichkeit kann sich ein Risiko ändern. Auch das Schadenspotential kann sich ändern und sollte regelmäßig überprüft werden. Durch die Digitalisierung werden immer mehr Geschäftsprozesse online verfügbar. Diese digitalen Prozesse gewinnen an Wichtigkeit, stellen aber im Angriffsfall ein höheres Schadenspotential dar. Neben der Änderung eines Risikos, kann ein Risiko auch wegfallen, indem zum Beispiel ein System nicht mehr benötigt und daher abgeschaltet wird. Diese Änderungen spielen bei der Risikoanalyse und der anschließenden Risikobewältigung eine entscheidende Rolle und sollten somit regelmäßig kontrolliert werden.

2.3. IT-Grundschutz

Um Informationen angemessen zu schützen, bedarf es eines Informationssicherheits Managementsystems (ISMS). Zur Implementierung eines solchen ISMS gibt es verschiedene Möglichkeiten. International anerkannt ist z.B. ein ISMS auf Basis von ISO 27001. Die Normen der ISO 27001 werden von der International Organisation for Standardization (ISO) in Zusammenarbeit mit Expertengremien erstellt und herausgegeben. Auch die IT Infrastructure Libary (ITIL) und Control Objectives for Information and related Technology (COBIT) haben Standards für ein ISMS. Dabei sind ITIL, COBIT und ISO 27001 weitestgehend untereinander kompatibel.

Die ISO 27001 beschreibt hauptsächlich Standards die es einzuhalten gilt, weniger eine Anleitung zur Implementierung eines ISMS. Das Bundesamt für Sicherheit in der Informationstechnologie (BSI) hat auf Basis der ISO 27001 einen eigenen Standard geschaffen, den IT-Grundschutz. Der IT-Grundschutz ist dabei komplett kompatibel zur ISO 27001 und beschreibt Schritt für Schritt, was ein erfolgreiches Informationssicherheitsmanagement ausmacht und welche Aufgaben daraus folgen.

Nach dem IT-Grundschutz besteht ein ISMS aus vier Komponenten, den Managementprinzipien, Ressourcen, Sicherheitsprozessen und den Mitarbeitern (Abb. 2.2) [BSI08].

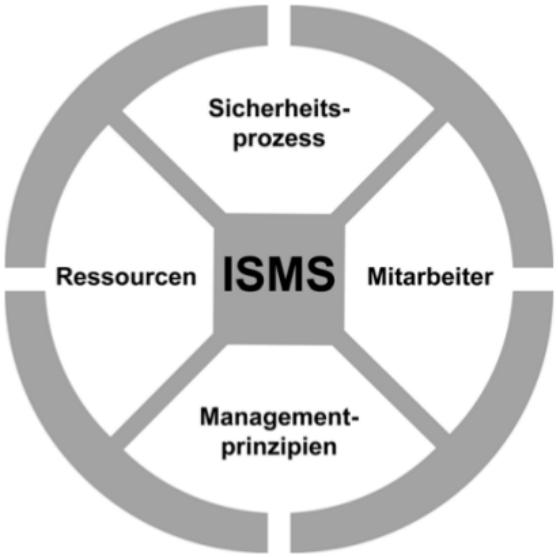


Abbildung 2.2.: Bestandteile eines ISMS

2.3.1. Managementprinzipien

Zunächst ergeben sich Aufgaben und Pflichten für die Unternehmensführung. So soll die Unternehmensführung sich zunächst zu ihrer Verantwortung für Informationssicherheit eindeutig bekennen. Auch sollte die Informationssicherheit in alle Bereiche des Unternehmens integriert werden und bei jedem Prozess bedacht werden. Die Informationssicherheit wird von der Unternehmensführung gesteuert und aufrechterhalten. Dazu gehört z.B. das Bereitstellen von ausreichenden Mitteln und das Einsetzen von Richtlinien. Auch sollten erreichbare Ziele gesetzt werden, um eine kontinuierliche Verbesserung zu ermöglichen. Des Weiteren sollte sich die Unternehmensführung ihrer Vorbildfunktion klar werden und auch bei der Informationssicherheit mit gutem Beispiel voran gehen.

Um die Informationssicherheit aufrecht zu erhalten und kontinuierlich zu verbessern ist es zum einen nötig, die Strategie immer wieder zu überdenken und ggf. anzupassen und zum anderen das Prüfen des aktuellen Stands. Dazu bieten sich interne und externe Audits an.

Um es der Unternehmensführung zu ermöglichen Entscheidungen zu treffen, muss diese ausreichend informiert werden. Dies kann z.B. mit regelmäßigen Berichten und über Ergebnisse aus Audits erfolgen. Um die Kontinuität der Prozesse zu gewährleisten, ist es notwendig diese Prozesse zu Dokumentieren.

2.3.2. Ressourcen

Für ein angemessenes Sicherheitsniveau sind auch finanzielle, personelle und zeitliche Ressourcen notwendig. So kann es sein, dass neue technische Systeme gekauft und installiert werden müssen. Diese müssen anschließend von Mitarbeitern bedient werden. Diese Mitarbeiter müssen in diesem Zusammenhang auch für die eingesetzten Produkte geschult werden. Auch zusätzliche oder neue Mitarbeiter können notwendig werden, wenn es keine vorhandenen Mitarbeiter mit entsprechender Qualifikation gibt. Für die Bereitstellung ist die Unternehmensführung verantwortlich. Bei der Bereitstellung ist stets auf ein angemessenes Kosten-Nutzen-Verhältnis zu achten.

2.3.3. Mitarbeiter

Neben dem Mitarbeiter als Ressource, können Mitarbeiter auch direkt Schäden verhindern. Dazu müssen den Mitarbeitern die Grundlagen für Informationssicherheit vermittelt werden. Etwa warum man Informationssicherheit benötigt und wie man sie strategisch umsetzen will. Für das Schaffen dieser Grundlagen sind Schulungen notwendig in denen man eine gemeinsame Wertvorstellung entwickeln und das Engagement der Mitarbeiter fördern kann.

2.3.4. Sicherheitsprozess

Nachdem das Management die Rahmenbedingungen geschaffen hat, muss ein Prozess etabliert werden. Dabei soll die Informationssicherheit ein Prozess sein, der kontinuierlich verbessert werden kann. Der Sicherheitsprozess soll dabei nach dem Plan-Doc-Check-Act (PDCA) Modell verlaufen (Abb. 2.3).

Planung des Sicherheitsprozesses

Zunächst soll der Sicherheitsprozess geplant werden. Dazu legt man zunächst den Geltungsbereich fest. Dies kann entweder das ganze Unternehmen sein, aber auch nur ein Teil davon, etwa die IT-Abteilung. Auch die Rahmenbedingungen gilt es abzuklären. Das können gesetzliche Anforderungen wie der Datenschutz , aber auch das Unternehmensziel, Kundenanforderungen, Bedrohungen durch Sicherheitsrisiken z.B. Reputationsverlust oder Datenverlust von Forschungsergebnissen sein. Nachdem der Geltungsbereich und die Rahmenbedingungen geklärt sind, werden die Ziele für die Informationssicherheit und Vorgaben für die Erfüllung dieser Ziele festgelegt. Dieser Sicherheitsprozess soll die Informationssicherheit in einem Unternehmen gewährleisten.

Abbildung 2.3.: Das PDCA-Modell

2.4. Schwachstellen

Die Informationssicherheit von Unternehmen hat oft Schwachstellen, denen es zu begegnen gilt. Das kann z.B. die Tatsache sein, dass es nur ein Rechenzentrum gibt und dieses im Falle einer Naturkatastrophe nicht mehr zur Verfügung steht oder aber eine technische Schwachstelle in einem Dienst, die es ermöglicht Passwörter auszulesen.

Allgemein kann man Schwachstellen anhand zahlreicher Eigenschaften kategorisieren. Im folgenden soll verstärkt auf technische Schwachstellen in Diensten und Anwendungen eingegangen werden um einen groben Überblick über solche Kategorien zu geben. Das Prinzip lässt sich auch auf andere Schwachstellen anwenden, würde aber den Rahmen dieser Arbeit übersteigen.

Die wohl umfangreichste Liste an technischen Schwachstellenkategorien ist die Common Weakness Enumeration (CWE). Ziel der CWE ist es, einheitliche Begriffe zu definieren und zu nutzen. CWE entstammt der MITRE Corporation, einer gemeinnützigen Organisation, die eine Ausgründung aus dem Massachusetts Institute of Technology (MIT) ist. Sie wird hauptsächlich von der US-Regierung unterstützt [MIT].

Im Rahmen dieser Arbeit wird im Folgenden auf die OWASP Top 10 eingegangen.

2.4.1. OWASP Top 10

Das Open Web Application Security Project (OWASP) hat einen Fokus auf Webanwendungen. Das OWASP ist eine gemeinnützige Organisation und finanziert sich hauptsächlich aus seinen Konferenzen und Spenden aus der Privatwirtschaft [OWA16]. Es betreibt einige Projekte wie den *OWASP Dependency Check*, das *OWASP Zed Attack Proxy Project* und das *OWASP Top Ten Project*. Bei letzterem handelt es sich um eine Auflistung der zehn kritischsten Web-Applikation Schwachstellen (Tablle 2.1). Die letzte Aktualisierung fand im Jahr 2013 statt, die Liste ist aber bis heute aktuell [OWA13].

ID	Titel
A1	Injection
A2	Broken Authentication and Session Management
A3	Cross-Site Scripting (XSS)
A4	Insecure Direct Object References
A5	Security Misconfiguration
A6	Sensitive Data Exposure
A7	Missing Function Level Access Control
A8	Cross-Site Request Forgery (CSRF)
A9	Using Known Vulnerable Components
A10	Unvalidated Redirects and Forwards

Tabelle 2.1.: Übersicht über OWASP Top 10

A1 - Injection

Bei einer Injection werden Daten die von einer nicht vertrauenswürdigen Quelle stammen (z.B. eine Suchanfrage eines Anwenders) in einen Interpreter geladen. Diese Interpreter können eine SQL-Datenbank oder die Kommandozeile sein. Sucht ein Anwender auf einer Internetseite nach einem Begriff, wird dieser Begriff von der Anwendung in die Datenbank geleitet. Wenn der Angreifer es nun schafft diese Suche mit Hilfe des Suchbegriffs zu manipulieren, kann er selbst Befehle an die Datenbank senden. So könnte eine Suche in PHP folgendermaßen realisiert sein:

Listing 2.1: unsichere Suchfunktion in PHP

```
<?php
$pdo = new PDO('mysql:host=localhost;dbname=demo', 'username', 'password');
echo "<h1>Suchergebnisse für ".$_POST["suchbegriff"]."</h1>";
$sql = "SELECT * FROM pages WHERE content like \"%".$_POST["suchbegriff"]."%\";";
foreach ($pdo->query($sql) as $row) {
   echo make_searchentry($row);
}
?>
```

Sucht nun der Anwender nicht nach einem Begriff, sondern nach: *suchbegriff" union all select * from users where username like "* sucht die Datenbank nicht nur nach relevanten Seiten, sondern gibt auch die User-Tabelle aus (Siehe Abb. 2.4). Ein Angreifer kann sich nun mit Hilfe der Passwörter als Administrator anmelden.

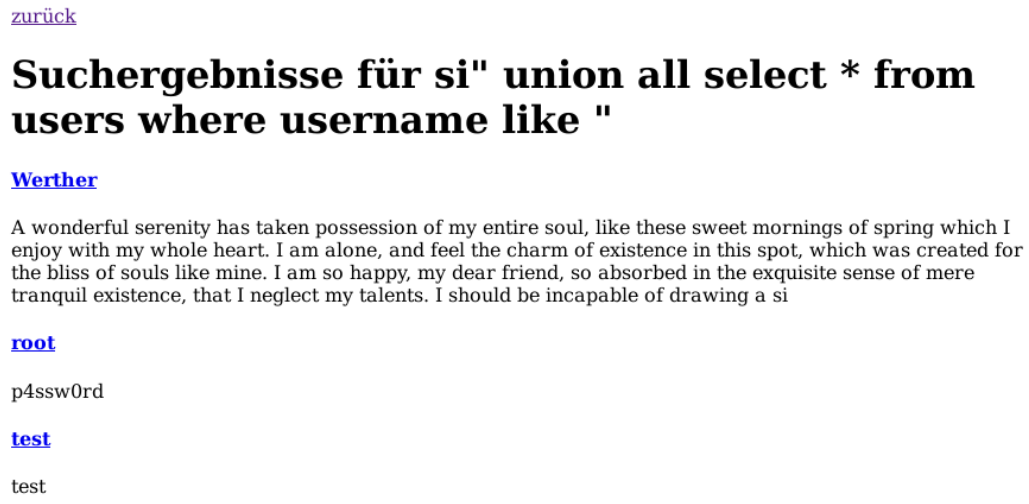


Abbildung 2.4.: Beispielhafte Ausgabe einer SQL-Injection

Diese Injection wird auch SQL-Injection genannt. Denkbar sind diese Schwachstellen für sämtliche Interpreter, etwa die Kommandozeile. Für Injections existieren einige Programme, die diese Schwachstellen finden können. Für einfache HTML-Seiten funktionieren diese Programme verhältnismäßig gut. Sobald allerdings weitere interaktive Elemente hinzukommen, kann es für ein Programm recht schwer sein Eingabefelder und HTTP-Parameter zu erkennen. Für das Ausnutzen dieser Schwachstellen sind Programme teils unerlässlich, da es auch Injections geben kann, die das Ergebnis nicht als HTML zurückliefern, sondern mittels *sleep*-Befehl so ausgeführt werden, sodass man die Antwortzeiten messen kann um darüber ein Ergebnis zu erhalten, sogenannte *Blind-Injections*. Als Beispiel für ein solches Tool sei *sqlmap* genannt [SQL16].

A2 - Broken Authentication and Session Management

Webanwendungen bei denen sich Anwender anmelden müssen, erzeugen nach erfolgreicher Anmeldung eine Session. Der Anwender muss sich dadurch nicht für jede Anfrage erneut authentifizieren. Diese Session wird meist mittels Session-ID referenziert. So schickt der Anwender nun bei jeder Anfrage diese ID mit und der Server kann anhand dieser ID den User identifizieren. Wenn ein Angreifer an diese Session-ID gelangt, kann er womöglich diese Session mitnutzen und im Kontext des Anwenders mit der Anwendung interagieren. Diese Session-IDs sind daher besonders zu schützen. Ein Angreifer hat zahlreiche Möglichkeiten an diese Session-ID zu gelangen, er könnte versuchen diese zu erraten sofern die ID nicht lang genug ist oder vorhersehbar ist. Er könnte auch versuchen den Netzwerkverkehr abzufangen, um dort an die nötigen Informationen zu gelangen.

Neben dem Session Management sollte eine Anwendung auch darauf achten, die Authentifizierung abzusichern. So sollte ein Schutz gegen Brute-Force, also das Durchprobieren von Passwörtern, implementiert werden. Auch das Speichern von Passwörtern sollte sicher umgesetzt werden. In der Vergangenheit gab es Fälle bei denen Anwenderpasswörter unverschlüsselt gespeichert wurden. Sollte ein Angreifer also Zugang zu diesem Speicherort erlangen, kann er alle Passwörter einsehen und sich mit Hilfe dieser an einem anderen System anmelden. Daher sollten Passwörter nur sicher gespeichert werden, zum Beispiel gehasht und salted.

A3 - Cross-Site Scripting (XSS)

Oftmals werden Anwendereingaben vom System auch anderen Anwender angezeigt. Zum Beispiel ein Gästebucheintrag einer Hotel-Webseite. Der Webbrowser kann dann nicht zwischen Webseite und Gästebucheintrag unterscheiden und interpretiert alles als Webseite und führt auch den Gästebucheintrag aus. Dies kann zum Teil gewünscht sein, um Formatierungen im Eintrag zulassen zu können. Wenn ein Angreifer auf solch einer Webseite einen Eintrag hinterlegt und darin einen Schadcode einbaut wird dieser Schadcode von nun an jedem Besucher der Webseite angezeigt. So könnten über diesen Schadcode beispielsweise Besucher mit Schadsoftware infiziert werden oder Cookies bzw. Session-IDs gestohlen werden. Ist dieses Cross-Site Scripting bei jedem Aufruf, auch bei Aufrufen anderer Nutzer, sichtbar ist es persistent und man spricht von **persistent XSS**.

Auch flüchtige Eingaben wie die Eingabe eines Suchparameters der von der Webseite wiederum angezeigt wird kann Schadcode enthalten und wird dann dem Anwender angezeigt. Allerdings nur einem einzigen Anwender. Kann ein Angreifer einem Angegriffenen einen Link mit diesem Schadcode senden und öffnet der Angegriffene diesen, wird ihm der Schadcode angezeigt. Dieser Angriff ist allerdings nicht persistent, er wird nur beim Folgen des infizierten Links angezeigt. Daher spricht man hier von **reflected XSS**. Gibt man beispielsweise bei der in Listing 2.1 beschriebenen Suche *<img src="http://.../pownie.png">XSS* ein, wird das *img*-Tag als solches interpretiert und die Seite wird dadurch verändert. (Siehe Abb: 2.5) Neben HTML kann ebenso JavaScript-Code eingeschleust werden, um damit andere Angriffe durchzuführen.

Abbildung 2.5.: Beispielhafte Ausgabe eines reflected XSS

A4 - Insecure Direct Object References

In Kundenportalen ist es oftmals möglich Dokumente bspw. Rechnungen herunterzuladen. Diese werden meist aus einem anderen System, zum Beispiel einem CRM[1], geladen. Die Dokumente besitzen dann einen Iden-

[1] Customer Relationship Management

tifier (ID), über welchen der Download referenziert wird. Eine URL zum Download einer Rechnung könnte folgendermaßen lauten: *https://kundenportal.firma.de/downloads/20160531009*. Wenn nun ein Angreifer diese ID, *20160531009* in- oder dekrementiert, könnte es ihm gelingen, eine andere Rechnung herunterzuladen, sofern keine Rechteprüfung erfolgt. Der Zugriff auf Dokumente oder Objekte muss somit einer Rechteprüfung unterliegen.

A5 - Security Misconfiguration

Bei manchen Anwendungen kann man bestimmte Eigenschaften oder Verhalten konfigurieren. Es ist denkbar, dass die Administrationsoberfläche einer Anwendung nur aus dem Firmennetzwerk heraus erfolgen darf. Man könnte also den Zugriff auf die Administrationsoberfläche so konfigurieren, dass nur bestimmte IP-Adressen erlaubt sind. Dies würde es einem Angreifer erschweren diese Oberfläche überhaupt anzugreifen.

Auch sollte eine Anwendung so konfiguriert werden, dass ein Angreifer möglichst wenig Informationen erhält. Die Informationen aus technischen Fehlermeldungen können wichtige Informationen für einen Angreifer enthalten. Interne IP-Adressen des Datenbankservers oder gar interne Benutzernamen können in Fehlermeldungen enthalten sein. Ebenso Informationen, die der Anwender benötigt, um zu wissen, was als nächstes tun ist. Anwendungen sollten so konfiguriert werden, dass zum Beispiel Java-Stacktraces oder PHP-Fehler nicht angezeigt werden, sondern nur für Administratoren geloggt werden. Zu umfangreiche Fehlermeldungen haben für den Anwender keinen Nutzen und beinhalten wertvolle Informationen für einen Angreifer (Siehe Abb: 2.6).

Fatal error: Uncaught exception 'PDOException' with message 'SQLSTATE[28000] [1045] Access denied for user 'username'@'localhost' (using password: YES)' in /var/www/html/demo.php:18 Stack trace: #0 /var/www/html/demo.php(18): PDO->__construct('mysql:host=loca...', 'username', 'password') #1 {main} thrown in **/var/www/html/demo.php** on line **18**

Abbildung 2.6.: Besipielhafte Ausgabe eines PHP/SQL Fehlers

A6 - Sensitive Data Exposure

Wie bereits bei Broken Authentication and Session Management beschrieben, muss die Session-ID geschützt sein, um Missbrauch zu verhindern. Neben der Session-ID gibt es je nach Anwendung weitere sensible Daten die geschützt werden müssen. Meist setzt man bei Webanwendungen dabei auf TLS/SSL[2]. Werden hierbei unsichere Algorithmen verwendet, kann ein Angreifer eventuell Teile oder alle sensiblen Daten entschlüsseln. So wurde zum Beispiel im Mai 2015 bekannt, dass zahlreiche Browser und Server schwache Diffie-Hellmann-Schlüssel unterstützen. Ein Angreifer in einer Man-in-the-middle Position könnte somit beim TLS/SSL Verbindungsaufbau die Verbindung so manipulieren, dass diese schwachen Schlüssel genutzt werden und eine Entschlüsselung auch ohne Kenntnis des Schlüssels möglich ist [Adr15].

A7 - Missing Function Level Access Control

Oftmals haben Anwendungen gleiche oder ähnliche Oberflächen für unterschiedlich privilegierte Anwender. Ein Unterschied besteht eventuell nur in den Möglichkeiten, die der jeweilige Anwender nutzen kann. Meist wird darauf verzichtet weniger privilegierten Anwendern die privilegierten Möglichkeiten anzuzeigen. Höher privilegierte Anwender haben jedoch diese Optionen und diese werden auch angezeigt. Manchmal überprüfen Anwendungen diese Privilegien beim Aufruf einer Funktion nicht, sondern prüfen die Privilegien nur bei der Darstellung. Weiß ein Anwender von solch einer privilegierten Funktion und Möglichkeit, kann er diese nutzen, in dem er die Aufrufe manuell durchführt. Er kann somit Aktionen durchführen, die nicht seinen Rechten oder Privilegien entsprechen.

[2]Transport Layer Security / Secure Socket Layer

A8 - Cross-Site Request Forgery (CSRF)

Ermöglicht eine Anwendung es einem Anwender mit einem einzigen Aufruf etwas zu ändern, bspw. eine Profiländerung, so sollte sichergestellt sein, dass der Request auch aus der Intention des Anwenders kommt. So wurde im Juli 2013 bekannt, dass T-Online Nutzer mit einem einzigen Aufruf einen Mail-Alias löschen konnten. Es war für Angreifer möglich, ein Opfer via Spam-Mail auf eine Webseite zu locken, die mittels JavaScript diesen Request losschickte. Der Browser schickt jedoch nicht nur den Request der Webseite, sondern auch Cookies mit, die üblicherweise die Session-IDs beinhalten. Der Webserver von T-Online konnte somit nicht erkennen, ob die Anfrage legitim war oder nicht. Dies ermöglichte den Angreifern das Übernehmen von fremden T-Online Adressen [Eik13]. Dies Schwachstelle konnte man mittels sogenanntem CSRF-Token lösen, der bei jedem Aufruf mitgesendet wird. Der Angreifer müsste also zunächst in den Besitz dieses Tokens kommen und dann die Request-URL aufbauen.

A9 - Using Known Vulnerable Components

Häufig wird bei der Entwicklung einer Anwendung auf bestehende Komponenten zurückgegriffen. So setzen zum Beispiel viele Anwendungen auf den Apache Webserver auf, oder es werden Module und Pakete eingebunden, die vieles erleichtern sollen. Dabei werden diese Komponenten oftmals nicht auf ihre Sicherheit hin geprüft. Selbst wenn eine solche Komponente zunächst sicher scheint kann sich das in der Zukunft ändern. Sollte eine Schwachstelle für eine Komponente bekannt werden, so sollte ein Patch dafür auch bei bereits laufenden Anwendungen angewendet werden. Passiert dies nicht, kann eine ansonsten sichere Anwendung über solche Komponenten angegriffen werden.

A10 - Unvalidated Redirects and Forwards

Manchmal ist es für eine Anwendung notwendig einen Anwender weiterzuleiten. Wenn ein Anwender auf einen sogenannten Deep-Link klickt und er für diesen nicht angemeldet ist, leitet die Seite den Anwender auf die Login-Seite weiter. Um nach erfolgreicher Anmeldung auf die eigentlich gewünschte Seite zu kommen, wird dabei dann ein Parameter mit dem eigentlichen Ziel gesetzt. Die Login-Seite weiß nun, nach erfolgreicher Anmeldung, wohin der User möchte. Ist dieser Parameter jedoch nicht geschützt, wäre es für einen Angreifer möglich, einem Angegriffenen einen gefälschten Link zuzusenden. Der Angegriffene öffnet den Link, kommt auf die Originalseite, meldet sich an und gelangt dann auf eine angreiferkontrollierte Seite. Für den Nutzer könnte dies allerdings so aussehen, als ob er sich weiterhin auf der Originalseite aufhält.

2.4.2. CVSS

Nachdem nun eine Übersicht über mögliche Schwachstellen gegeben wurde, soll nun auf ein Verfahren eingegangen werden, welches zur Bewertung von Schwachstellen dient. Eine Bewertung kann schnell Aufschluss geben, wie groß das resultierende Risiko ist und ermöglicht somit die Behebung von Schwachstellen zu priorisieren.

Zur Bewertung des aus einer Schwachstelle resultierenden Risikos ist es notwendig die Eintrittswahrscheinlichkeit und das Schadenpotential zu ermitteln. Eine weit verbreitete Metrik ist das Common Vulnerability Scoring System (CVSS). Für dieses System existieren mehrere Versionen, die aktuellste ist die Version 3. Dieses System soll im folgenden kurz erläutert werden.

CVSS v3 besteht aus drei verschiedenen Untermetriken, dem basis Score, dem temporären Score und dem Umwelt Score.

Der **basis Score** besteht wiederum aus der Exploitability Metrik die beschreibt wie einfach es ist eine Schwachstelle auszunutzen und soll die Eintrittswahrscheinlichkeit abbilden. Dazu existieren vier Variablen: Attack Vector, Attack Complexity, Privileges Required und User Interaction.

- Der **Attack Vector** beschreibt, wie die Schwachstelle ausgenutzt werden kann. Dazu gibt es vier Werte die die Variable annehmen kann. *Network* für Angriffe über das Netzwerk oder Internet, *Adjacent* für

Angriffe die nur aus dem lokalen Netzwerk erfolgen können, *Local* für Angriffe die nur lokal auf dem System durchgeführt werden können und *Physical* für Angriffe die nur mit physischem Zugang möglich sind.

- Die **Attack Complexity** beschreibt wie schwierig ein Angriff ist und welche Anforderungen an einen Angreifer gestellt werden. Die Attack Complexity kann die Werte *Low* und *High* annehmen.
- **Privileges Required** dient dazu einzustufen ob ein Angriff nur mit bestimmten Rechten möglich ist. Die Werte können *None* sein, wenn der Angreifer sich nicht authentifizieren muss, *Low* falls der Angreifer sich authentifizieren muss, aber keine besonderen Rechte benötigt und *High*, falls der Angreifer hohe Rechte, also zum Beispiel administrative Rechte benötigt.
- Manche Angriffe, beispielsweise Cross-Site-Scripting, benötigen eine Interaktion eines Anwenders. Dies ist über die Variable **User Interaction** mit den Werten *None* und *Required* abgebildet.

Für die Bewertung des Schadenpotentials dienen drei Variablen. Der **Confidentiality Impact**, der **Integrity Impact** und der **Availability Impact**. Diese können jeweils die Werte *High*, *Low* und *None* annehmen.

Neben diesen Variablen gibt es noch die Variable **Scope**. Sie beschreibt, ob es möglich ist den Kontext zu ändern. Also ob man durch einen Angriff als einfacher Anwender Resourcen erreichen kann, die eigentlich nicht für diesen Anwender gedacht sind. (z.B. Privilege Escalation) Die Werte können *Unachanged* und *Changed* sein.

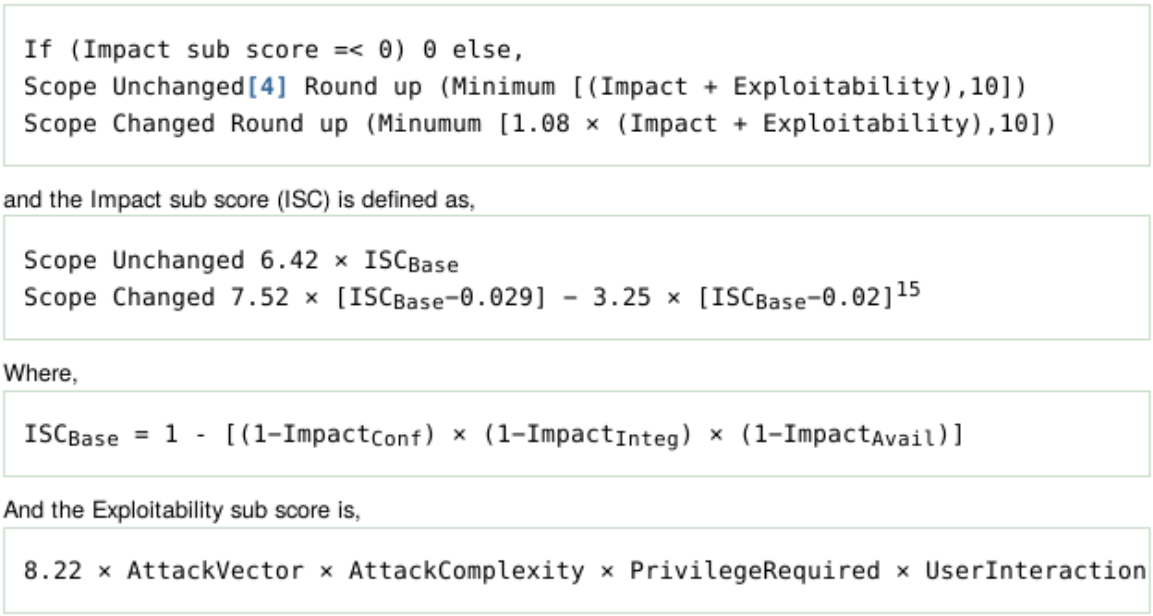


Abbildung 2.7.: Die Formel für den CVSSv3 Basis Score [FIR]

Diese acht Variablen ergeben mittels einer Formel (siehe Abb. 2.7) einen Wert zwischen 1 und 10. Seit der Version 3 des CVSS können über die Werte direkt qualitative Einstufungen erfolgen. (Tabelle 2.2)

Einstufung	CVSS Score
None	0.0
Low	0.1 - 3.9
Medium	4.0 - 6.9
High	7.0 - 8.9
Critical	9.0 - 10.0

Tabelle 2.2.: Qualitative Einstufung anhand des CVSSv3 Scores

Neben dem basis Score gibt es noch den **temporären Score** der beschreibt, ob es bereits funktionierende

Exploits (*Exploit Code Maturity*), also Software die diese Schwachstelle ausnutzen kann gibt und wie gut diese sind. Ob es bereits Lösungen zum Beheben der Schwachstelle gibt (*Remediation Level*) und mit welcher Sicherheit die Schwachstelle tatsächlich existiert (*Report Confidence*). Dieser temporäre Score wird dann mit dem basis Score verrechnet.

Der **Umwelt Score** dient dazu, das Schadenspotential für das Unternehmen besser einschätzen zu können. So kann hier die Kritikalität der Software eingestuft werden. Handelt es sich um einen sehr wichtigen Dienst ist das Schadenspotential ggf. höher. Auch Unterschiede in der Testumgebung können hier berücksichtigt werden. So könnte sich die Konfiguration in der Testumgebung von der Konfiguration im normalen Betrieb unterscheiden und damit Angriffe erst ermöglichen oder das Schadensportential ändern. Um diese Unterschiede zu berücksichtigen, können im Umwelt Score alle Variablen des basis Scores nochmals vergeben werden, um das Risiko im normalen Betrieb besser einschätzen zu können.

Variablenname	mögliche Werte
Base Metrics	
Attack Vector	Network, Adjacent, Local, Physical
Attack Complexity	Low, High
Privileges Required	None, Low, High
User Interaction	None, Required
Scope	Unchanged, Changed
Confidentiality Impact	High, Low, None
Integrity Impact	High, Low, None
Availability Impact	High, Low, None
Temporal Metrics	
Exploit Code Maturity	Not Defined, High, Functional, Proof-of-Concept, Unproven
Remediation Level	Not Defined , Unavailable, Workaround, Temporary Fix, Official Fix
Report Confidence	Not Defined, Confirmed, Reasonable, Unknown
Environmental Metrics	
Security Requirements Confidentiality	Not Defined, High, Medium, Low
Security Requirements Integrity	Not Defined, High, Medium, Low
Security Requirements Availability	Not Defined, High, Medium, Low
Modified Attack Vector	Not Defined, Network, Adjacent, Local, Physical
Modified Attack Complexity	Not Defined, Low, High
Modified Privileges Required	Not Defined, None, Low, High
Modified User Interaction	Not Defined, None, Required
Modified Scope	Not Defined, Unchanged, Changed
Modified Confidentiality Impact	Not Defined, High, Low, None
Modified Integrity Impact	Not Defined, High, Low, None
Modified Availability Impact	Not Defined, High, Low, None

Tabelle 2.3.: Überblick über die CVSS Version 3 Variablen

2.5. Arten von Tests

Um Schwachstellen zu identifizieren, können Anwendungen auf Schwachstellen geprüft werden. Im Folgenden sollen zunächst Kategorien beschrieben werden, anhand derer ein Sicherheitstest kategorisiert werden kann. Im Anschluss wird auf automatisiert und manuell durchgeführte Tests eingegangen.

2.5.1. Klassifizierung von Sicherheitstests

Sicherheitstests sollten auf die zu testende Anwendung ausgerichtet werden. Das fängt bei der Art des Systems an. Handelt es sich zum Beispiel um eine Clientsoftware ohne Netzwerkfunktionalität, können Netzwerkaspekte im Vorhinein vernachlässigt werden. Auch sollte bei produktiven Systemen, also Systeme die während

des Tests von Anwendern genutzt werden, nur soweit getestet werden, dass diese auch weiterhin zur Benutzung zur Verfügung stehen. In Anlehnung an [BSI03] kann man Sicherheitstests nach folgenden Kriterien klassifizieren.

Merkmal	mögliche Ausprägung		
Informationsbasis	White-Box	Grey-Box	Black-Box
Aggressivität	vorsichtig	abwägend	aggressiv
Umfang	fokussiert	begrenzt	vollständig
Vorgehensweise	offensichtlich	verdeckt	
Ausgangspunkt	von innen	von außen	

Tabelle 2.4.: Klassifizierung von Sicherheitstests

Die in Tabelle 6.1 beschriebenen Merkmale werden im Folgenden genauer erläutert.

Informationsbasis

Im Vorhinein sollte festgelegt werden, welche Informationen der Tester erhält. So können einem Tester im Vorhinein schon interne Netzwerkpläne übergeben werden, sodass uninteressante Systeme unberücksichtigt bleiben. Das kann einen Test verkürzen und somit günstiger machen. Generell kann man die Informationsbasis in White-, Grey- und Black-Box unterteilen.

Bei einem **White-Box** Test erhält der Tester alle Informationen zum zu testenden Dienst. Das können IP-Adressen oder Abhängigkeiten zu anderen Diensten sein. Zum anderen können dies auch Quellcode und Konfigurationsdateien sein. Wie bei einem Code Review werden zunächst mögliche Schwachpunkte im Quellcode oder in der Konfiguration gesucht. Anschließend werden diese möglichen Schwachpunkte am Testziel näher untersucht und es wird versucht diese Schwachpunkte auszunutzen.

Beim **Grey-Box** Test erhält der Tester nur wenige Informationen. Dazu können Informationen zu eingesetzter Software und deren Versionen gehören, als auch Informationen zu Schnittstellen. Der Tester kann anhand dieser Informationen dann mögliche Schwachpunkte identifizieren und ausnutzen. Da er keine näheren Informationen hat, muss er versuchen welche anhand anderer Tests zu erhalten.

Bekommt ein Tester keine oder nur sehr wenige Informationen, dann spricht man von einem **Black-Box** Test. Hier erhält der Tester nur Informationen, welche Dienste er angreifen soll. So könnte die Information lauten ”Teste die Webseite http://unternehmen.de”. Welche IP-Adressen oder anderen Dienste sich dahinter verbergen muss der Tester selbst in Erfahrung bringen. Dabei wird versucht ein möglichst echtes Angriffsszenario abzubilden. Es können auch andere mögliche Informationsquellen genutzt werden. So wäre es denkbar, dass ein Tester mittels Social-Engineering versucht an weitere Informationen zu gelangen. Die Quellen der Informationen werden dann, neben den gefundenen Schwachstellen, ebenso protokolliert und in einem Abschlussbericht aufgeführt.

Aggressivität

Auch die Intensität eines Tests sollte im Vorhinein definiert sein. Sollte der Tester sich dazu entschließen einen Brute-Force Angriff durchzuführen, muss klar sein, wie viele Versuche pro Minute durchzuführen sind, da die Performance der zu testenden Anwendung dadurch beeinträchtigt sein kann. Auch sind solche Brute-Force Angriffe sehr leicht in Log-Dateien zu sehen und können zu einem geänderten Verhalten der Anwendung gegenüber dem Tester führen. Soll ein Angriff möglichst unbemerkt durchgeführt werden, muss die Intensität oder Aggressivität sehr gering sein. Tests auf produktiven Systemen sind üblicherweise sehr wenig aggressiv, da das System für die legitimen Benutzer weiter zur Verfügung stehen soll. Bei Test- oder Integrationstestumgebungen kann dagegen meist aggressiver vorgegangen werden.

Umfang

Auch der Umfang eines Tests ist entscheidend. Soll lediglich ein Dienst getestet werden oder das ganze Unternehmen? Wenn nur ein System oder eine Anwendung getestet werden soll, könnte zum Beispiel die Infrastruktur vernachlässigt werden. So wäre ein offener Administrationsport am Webserver kein Gegenstand des Tests, wenn dieser Port nicht zur Anwendung gehört. Tests einzelner Anwendungen sind meist weniger zeitintensiv als Tests eines ganzen Unternehmens. Dennoch sollte das Unternehmen in seiner Gesamtheit regelmäßig geprüft werden. Dabei wird einem Tester meist ein Ziel vorgegeben, etwa ”Erstelle eine Datei auf dem Desktop des Servers XY”. Der Tester muss sich zunächst Zugang zu diesem Server verschaffen und dann einen Weg finden, um auf diesen Server zuzugreifen.

Vorgehensweise

Neben der Sicherheit der Anwendung kann es auch interessant sein Prozesse eines laufenden Betriebs zu testen. Dazu gehören Intrusion Detection Systeme (IDS) und Logfile-Überwachung. Diese sollen möglichst einen Angriff detektieren und ggf. einen Alarm auslösen. Nachdem ein Alarm eines Systems generiert wurde, sollten Prozesse implementiert sein, wie diese Alarme zu bearbeiten sind. Ob diese Prozesse wie geplant funktionieren, kann ebenso Testgegenstand sein. Um die Prozesse zu testen, sollten die Personen und Systeme möglichst nicht informiert sein, dass ein Test durchgeführt wird, um ein reales Szenario zu simulieren. Jedoch kann es durchaus passieren, dass die Prozesse nicht wie geplant funktionieren und der Betrieb aufgrund eines solchen Tests beeinträchtigt wird. Daher sollte man sich vor einem Test die Frage stellen, ob die für den Betrieb verantwortlichen Stellen involviert werden sollen bzw. welche Details zu einem geplanten Test an diese Stellen gegeben werden müssen. Hierbei spielt auch die Aggressivität eine Rolle. Portscans können zum Beispiel sehr leicht bei IDS-Systemen Alarme generieren. Aggressive Tests bedeuten meist auch ein hohes Betriebsrisiko. Wenn die Betriebsgruppen nicht eingebunden sind, kann dies ein Problem für den Betrieb darstellen. Es ist daher ratsam, nur bei wenig aggressiven Tests IDS-Prozesse einzuschließen.

Ausgangspunkt

Meistens kommen Angriffe von außerhalb des Unternehmens. Es ist aber auch denkbar einen Angriff aus dem Unternehmensnetzwerk zu simulieren. Innentäter, also Angreifer innerhalb des Unternehmens, haben oft genaue Kenntnis der Infrastruktur und stellen somit ein enormes Risiko dar. Zudem müssen aus dem internen Netzwerk meist weniger Sicherheitssysteme überwunden werden, was diese Angriffe erleichtern kann. Es sollte somit für einen Test auch überlegt werden, von welchem Ausgangspunkt dieser stattfindet.

2.5.2. Manuelles Testen

Nach [BSI03] besteht ein Penetrationstest aus fünf Phasen: Vorbereitung, Informationsbeschaffung und -auswertung, Bewertung der Informationen / Risikoanalyse, aktive Eindringversuche und Abschlussanalyse.

Phase I: Vorbereitung

Bei der Vorbereitung gilt es die Ziele und den Umfang des Tests zu definieren. Dabei sind die Klassifizierungseigenschaften zu berücksichtigen (vgl. 2.5.1). Neben dem Umfang und den Zielen sollten auch Haftungsfragen geklärt werden, da es trotz großer Sorgfalt zu Problemen im Betrieb kommen kann. Des Weiteren sollten auch rechtliche Bestimmungen insbesondere StGB (§§ 202a[3], 263a[4], 268[5], 303[6]) beachtet werden. Auch die Vorgehensweise im Falle einer Betriebsstörung sollte geklärt werden um die Ausfallrisiken zu minimieren.

[3] Ausspähen von Daten
[4] Computerbetrug
[5] Fälschung technischer Aufzeichnungen
[6] Sachbeschädigung

Phase II: Informationsbeschaffung und -auswertung

Nachdem der Test vorbereitet wurde kann nun damit begonnen werden Informationen über die Ziele zu erlangen. Dabei dienen die vereinbarten Ziele zur Begrenzung und die Informationsbasis als Ausgangspunkt. Oft werden zur Informationsbeschaffung Portscans durchgeführt, um die Infrastruktur kennen zu lernen und ggf. weitere Interessante Ziele auszumachen.

Phase III: Bewertung der Informationen / Risikoanalyse

Die gewonnenen Informationen werden nun Bewertet und interessante Ziele identifiziert. Dazu können bspw. Systeme mit bekannten Schwachstellen zählen. Aufgrund von Zeit- oder Budgetvorgaben kann ein Tester nun die Ziele priorisieren. Die Priorisierung bzw. Vernachlässigung einzelner Systeme sollte allerdings genau dokumentiert werden, da diese entgegen dem Eindruck des Testers dennoch Schwachstellen aufweisen können.

Phase IV: Aktive Eindringversuche

In dieser Phase versucht der Tester aktiv die priorisierten Zielsysteme anzugreifen und Schwachstellen zu finden. Findet der Tester eine Schwachstelle, dokumentiert er sie und falls gefordert, versucht er diese auszunutzen. Dabei wird der Test jedoch meist nicht nach der ersten Schwachstelle abgebrochen, sondern fortgeführt und weitere Systeme nach Schwachstellen abgesucht. Es sind auch Szenarien denkbar, bei denen der Test mit Erreichen eines bestimmten Ziels (z.B. Erlangen von Administrationsrechten) beendet wird.

Phase V: Abschlussanalyse

Nach Beendigung der aktiven Tests werden die gefundenen Schwachstellen noch nach Eintrittswahrscheinlichkeit und Schadenspotential bewertet und ein Bericht erstellt. Dabei ist darauf zu achten, dass die Nachvollziehbarkeit des Tests und der Schwachstellen gewährleistet ist. Üblich ist auch eine Präsentation der Testergebnisse gegenüber dem Auftraggeber bzw. dem Anwendungsverantwortlichen.

2.5.3. Automatisches Testen

Häufig stehen die Ressourcen für einen manuellen Test nicht zur Verfügung, daher werden manche Teile eines Tests automatisiert durchgeführt. Ein automatisierter Test bedarf jedoch auch einer Vorbereitung, da die eingesetzten Programme konfiguriert werden und der Test überwacht werden muss. Bei einem Schwachstellenscanner werden nach Beginn automatisch weitere Informationen beschafft und auf eventuell bekannte Schwachstellen geprüft. Eine automatische Auswertung ist nur rudimentär möglich, da individuelle Begebenheiten kaum berücksichtigt werden können.

Auch die Bewertung und Risikoanalyse kann automatisiert nur sehr rudimentär durchgeführt werden, da die Bewertung von Schwachstellen Kontext-Wissen benötigt. Bei einem Blogsystem zum Beispiel, bei dem die Blogeinträge indiziert sind und direkt über diesen Index aufrufbar sind, ist es für ein Programm schwierig zu entscheiden, ob es sich um eine *A4 - Insecure Direct Object References* (vgl. 2.4.1) handelt oder nicht.

Auch die aktiven Eindringversuche sind für ein Programm nur schwer durchzuführen, da bekannte Exploits oftmals angepasst werden müssen. Zudem müsste das Programm alle Exploits kennen bzw. suchen können.

Automatische Tests liefern oftmals viele Warnungen, die anschließend genau bewertet werden müssen. Bei einem manuellen Test werden solche Programme in der Informationsbeschaffungsphase eingesetzt. Allerdings werden die Ergebnisse anschließend manuell bewertet und eingestuft.

Automatisierte Tests stellen daher zwar nur eine Ergänzung zu manuellen Tests dar, und können bei richtiger Anwendung durchaus zu einer Verbesserung der Sicherheit beitragen.

3. Anforderungsanalyse

In diesem Kapitel wird die Notwendigkeit eines Managementsystems für Sicherheitstests und dessen Anforderungen erläutert. Zunächst wird auf die besonderen Herausforderungen an einem Hochschulrechenzentrum eingegangen, um verschiedene Anforderungen besser herzuleiten.

3.1. Herausforderungen in einem Hochschulrechenzentrum

Hochschulrechenzentren sind Dienstleister von Hochschulen. Daher soll zunächst einmal auf die Herausforderungen an Hochschulen und deren Besonderheiten eingegangen werden.

3.1.1. Herausforderungen im Hochschulkontext

Hochschulen und Universitäten haben eine lange Vergangenheit. Die Ludwig-Maximilians-Universität (LMU) wurde bereits im Jahr 1472 gegründet. Da im Laufe der Jahre immer wieder neue Fakultäten hinzugekommen sind oder sich Fakultäten geteilt haben, sind die Strukturen teils sehr verwachsen. So hat die Ludwig-Maximilians-Universität 18 Fakultäten die jeweils aus verschiedenen Lehrstühlen bestehen. Diese Lehrstühle haben verschiedene Arbeitsgruppen und in den einzelnen Arbeitsgruppen wird an verschiedenen Projekten, teilweise sogar übergreifend gearbeitet. Durch die permanente Änderung der Verantwortlichen und der Mitarbeiter ergeben sich diverse Hierarchien.

Neben den verwachsenen Strukturen ist auch die Anzahl an IT-Nutzern sehr hoch. Die LMU hat bspw. über 50.000 Studenten [LMU15]. Dazu kommen noch zahlreiche Verwaltungsangestellte, wissenschaftliche Mitarbeiter und Professoren. Neben der LMU gibt es allein in München und Umland noch die Technische Universität München (TUM) mit circa 40.000 Studenten [TUM], die Hochschule München mit 17.500 Studenten [HM] und die Hochschule Weihenstephan-Triesdorf mit 6.000 Studenten [HSW]. In den 1960er Jahren, als Computer Einzug in die Forschung hielten, wurde von der Bayerischen Akademie der Wissenschaften und dem Freistaat Bayern ein Rechenzentrum errichtet. Dieses Rechenzentrum sollte den beiden Universitäten LMU und TUM gleichermaßen zur Verfügung stehen. 1966 wurde das Rechenzentrum dann zu Ehren von Gottfried Wilhelm Leibniz in Leibniz-Rechenzentrum umbenannt [LRZ].

3.1.2. Herausforderungen am LRZ

Neben dem Rechenzentrumbetriebs, betreibt das LRZ auch das Münchener Wissenschaftsnetz (MWN), welches die Hochschulen, viele Studentenwohnheime und andere Forschungseinrichtungen verbindet. Dabei werden Teilnetze teilweise selbstverwaltet und nur die Anbindung dieser Netze an das MWN und das Internet erfolgt vom LRZ.

Die Kunden des LRZ, Hochschulen aber auch andere wissenschaftliche Einrichtungen, können neben der Anbindung an das MWN auch weitere Dienstleistungen beim LRZ beziehen. Dazu gehören u.a. Mailhosting, virtuelle Firewalls und Webhosting. Die Heterogenität der Kunden, also zum einen Hochschulen, zum anderen private wissenschaftliche Einrichtungen oder Museen und die teilweise vorhandene Selbstverwaltung der Netze und Dienste stellen dabei eine große Herausforderung dar. Insbesondere im Hinblick auf Sicherheitstests stellt dies eine enorme Herausforderung dar, da angebotene Dienste wie z.B. Mailhosting für die Kunden essentiell sind und Ausfälle können große Schäden verursachen. Die selbstverwalteten Netze und Dienste im MWN, deren Betrieb nicht vom LRZ verantwortet werden, stellen ebenfalls eine große Herausforderung dar. Sollten diese Schwachstellen besitzen, können Angreifer dadurch über diese Netze und Dienste weiter ins

MWN eindringen. Da diese Dienste jedoch nicht im Verantwortungsbereich des LRZs liegen, ist es schwierig diese auf Schwachstellen zu untersuchen und die Behebung von Schwachstellen sicherzustellen. Neben dem Betrieb des MWN werden auch Anwendungen am LRZ selbst entwickelt. Teils in Form von studentischen Arbeiten, teils im Rahmen von Projekten.

Die vorgenannt erwähnten Herausforderungen am LRZ sind jedoch nicht LRZ spezifisch, sondern treffen auf fast alle Hochschulrechenzentren zu. Viele Hochschulrechenzentren betreiben ebenso das Hochschulnetz und bieten verschiedene Dienste an.

3.2. Ausgangssituation

Da Hochschulrechenzentren als Dienstanbieter im Sinne des Telekommunikationsgesetz (TKG) gelten [DFN07] und Daten, die unter den Schutz des Bundesdatenschutzgesetzes (BDSG) fallen verarbeiten, sind Hochschulrechenzentren von Gesetzes wegen zur Datensicherheit verpflichtet. Aufgrund dieser Verpflichtung ist ein Informationssicherheits Managementsystem für ein Hochschulrechenzentrum quasi verpflichtend. Dabei ist die Ausgestaltung nicht vorgegeben. Häufig wird sich bei der Ausgestaltung jedoch an der ISO 27001 oder dem IT-Grundschutz orientiert. Diese Standards fordern, regelmäßige Sicherheitsaudits durchzuführen. Um dieser Anforderung nachzukommen und auch um generell den Sicherheitsprozess nach ISMS überprüfen zu können, soll ein Managementsystem für Sicherheitstests konzipiert werden. Dazu sollen im Folgenden die Anforderungen erläutert werden.

3.3. Managementsystem für Sicherheitstests

Sicherheitstests sollen Dienste auf deren Sicherheit prüfen. Dabei simulieren sie echte Angriffe auf Dienste. Solche Angriffe stellen ein Risiko für den Betrieb von Diensten dar. So kommt es bei Sicherheitstests nicht selten zu Ausfällen der Testsysteme. Dies ist teilweise nötig, um Schwachstellen in der Verfügbarkeit eines Dienstes zu finden. Für produktive Dienste allerdings, die während des Tests weiter benutzt werden sollen, kann dies fatal sein. Daher müssen Sicherheitstests detailliert geplant werden, um Störungen bei produktiven Diensten zu verhindern.

Da Sicherheitstests als Kontrolle für den Sicherheitsprozess im ISMS dienen können, sollten sie trotz Risiken für den Betrieb regelmäßig durchgeführt werden. In einem ISMS nach IT-Grundschutz oder ISO 27001 werden Sicherheitstests oder Audits sogar explizit gefordert.

Um Sicherheitstests zu planen und durchzuführen, sollte daher ein Managementsystem implementiert werden, um zum einen das Risiko für Betriebsunterbrechungen zu minimieren und zum anderen eine Zahlenbasis zum Status der Sicherheit zu erhalten.

3.3.1. Geltungsbereich des Managementsystems

Zunächst muss ein Geltungsbereich festgelegt werden um Abgrenzungen zu ermöglichen. Als Geltungsbereich sollen im Folgenden nur die Dienste gelten, die vom Hochschulrechenzentrum erbracht werden. Denkbar wäre auch, das gesamte Hochschulnetz einzubeziehen. Jedoch verkompliziert dies die Abläufe sehr, da das Hochschulrechenzentrum nicht zwingend alle Dienste in einem Hochschulnetz erbringt. Die Verantwortlichen für die anderweitig erbrachten Dienste könnten spezielle Anforderungen haben, die erfüllt werden müssen. Daher sollen zunächst nur Dienste, die vom Hochschulrechenzentrum erbracht werden, berücksichtigt werden. Sollte das Managementsystem für diesen Geltungsbereich fertiggestellt sein und reibungslos funktionieren, kann eine Öffnung für extern betriebene Dienste im Hochschulnetz nach Anpassungen ermöglicht werden.

3.3.2. Rahmenbedingungen

Nachdem der Geltungsbereich festgelegt wurde, werden nun die Rahmenbedingungen erläutert. Dazu gehört bei der Sicherheit von Diensten zum einen die Softwareentwickler die nach dem bekannt werden einer Schwachstelle diese im Dienst beheben müssen. Bei einer Schwachstelle zu A1 - Injection müssten diese bspw. die Eingabevalidierung verbessern. Dazu sollten diese bei den Ergebnissen berücksichtigt werden und Hilfen bei der Behebung erhalten. Das kann bspw. über eine Beschreibung der Schwachstelle und Informationen zu Möglichkeiten der Behebung erfolgen.

Auch Administratoren die für den Betrieb einzelne Server administrieren müssen, müssen ggf. eingebunden werden, da diese Konfigurationsänderungen vornehmen können oder neue Versionen eines Dienstes installieren müssen. Hierzu sollten Schwachstellen die aufgrund von Fehlkonfigurationen entstanden sind ausführlich beschrieben werden und ggf. Hinweise zu den zu ändernden Konfigurationsparametern gegeben werden.

Der Sicherheitsverantwortliche der nach gängigem ISMS definiert worden sein muss, sollte ebenso über Schwachstellen und deren Ursprung informiert werden. Ggf. müssen Richtlinien zum Administrieren von Servern oder Richtlinien zur Softwareentwicklung angepasst werden. Zudem muss der Sicherheitsverantwortliche gegenüber der Unternehmensführung auskunftsfähig zur Sicherheit im Unternehmen sein. Daher muss er bei allen Sicherheitstests involviert werden. Des Weiteren hat er von der Unternehmensführung den Auftrag erhalten für die Sicherheit zu sorgen. Er muss somit auch die Einhaltung des Sicherheitsprozesses kontrollieren.

Ein Hochschulrechenzentrum ist wie bereits beschrieben für den Betrieb des Rechenzentrums sowie für den Betrieb eigener Dienste verantwortlich. Daher muss bei einem Sicherheitstest stets darauf geachtet werden, dass der Betrieb nicht gestört wird. Die Unternehmensführung trägt zum einen die Verantwortung für die Sicherheit zum anderen auch für den Betrieb. Sollte bei einem Sicherheitstest ein Problem im Betrieb auftreten ist die Unternehmensführung letztlich dafür verantwortlich.

Auch zahlreiche Gesetze beeinflussen die Durchführung eines Sicherheitstests. So sind insbesondere die Strafgesetzbuch Paragraphen §§ 202a[1], 263a[2], 268[3], 303[4] von Bedeutung. Daher muss der Tester eine explizite Genehmigung für einen definierten Test erhalten.

3.4. Erläuterungen der Anforderungen

Da nun der Geltungsbereich und die Rahmenbedingungen festgelegt wurden, sollen im Folgenden die einzelnen Anforderungen an ein Managementsystem für Sicherheitstests aufgelistet und erläutert werden. Die Anforderungen wurden in einem Gespräch mit Mitgliedern des Arbeitskreises Informationssicherheit des LRZ herausgearbeitet. Die Anforderungen können jedoch auch für andere Hochschulrechenzentren gelten. Da die Herausforderungen im Hochschulkontext bei nahezu allen Hochschulrechenzentren ähnlich sind und auch andere Hochschulrechenzentren für den Betrieb des Hochschulnetzes verantwortlich sind. Das LRZ hat zwar, gerade in Hinblick auf die verschiedenen Kundentypen besondere Herausforderungen zu meistern, diese können aber abstrahiert auf andere Hochschulrechenzentren übertragen werden.

Die Anforderungen wurden zur besseren Übersicht in Kategorien unterteilt. Die Kategorien ergeben sich aus den Teilbereichen des Managementsystems.

- Allgemeine Anforderungen (**A**)
- Anforderungen an das Dienstmanagement (**D**)
- Anforderungen an das Testmanagement (**T**)
- Anforderungen an das Schwachstellenmanagement (**S**)

[1] Ausspähen von Daten
[2] Computerbetrug
[3] Fälschung technischer Aufzeichnungen
[4] Sachbeschädigung

Üblich ist auch die Unterscheidung zwischen funktionalen und nicht-funktionalen Anforderungen. Auf diese Unterscheidung wird jedoch verzichtet, da eine exakte Unterscheidung nicht immer möglich ist und eine weitere Anforderungskategorie keine Vorteile bzgl. der Übersichtlichkeit bietet.

Für eine bessere Übersichtlichkeit und eine Möglichkeit auf Anforderungen Bezug zu nehmen, sollen die Anforderungen eine Bezeichnung bekommen. Diese setzt sich wie folgt zusammen:

$$< KAT >< NUM >$$

Dabei kann *KAT* die Werte **A** für Allgemeine Anforderungen, **D** für Anforderungen an das Dienstmanagement, **T** für Anforderungen an das Testmanagement und **S** für Anforderungen an das Schwachstellenmanagement annehmen. *NUM* soll anhand einer fortlaufenden Nummer eine Unterscheidung innerhalb der Kategorien ermöglichen und so eine eindeutige Identifizierung gewährleisten.

3.5. Analyse der Anforderungen

Im Folgenden sollen nun die einzelnen Anforderungen aufgelistet und erläutert werden. Zur besseren Übersichtlichkeit wurden sie nach Kategorie getrennt.

3.5.1. Allgemeine Anforderungen

A01 - Flexibilität des Anwendungsbereichs
Zwar wurde als Geltungsbereich das Hochschulrechenzentrum festgelegt, jedoch ist ein Hochschulrechenzentrum auch der Betreiber des Hochschulnetzes. Da an einem Hochschulnetz auch andere Netzwerke, die von Dritten verwaltet und betrieben werden, angebunden sind, stellen diese Netze und die darin betriebenen Dienste auch ein Gefahrenpotential für das Rechenzentrum und das gesamte Netz dar. Es wäre denkbar, das Testmanagement und das Durchführen von Sicherheitstests als zentrale Dienstleistung des Hochschulrechenzentrums dem gesamten Hochschulnetz zur Verfügung zu stellen. Daher sollte die Möglichkeit der Erweiterung des Konzept auf das gesamte Netz und als Dienstleistung berücksichtigt werden.

A02 - Definition der Aktivitäten
Für eine vollständige Beschreibung eines Prozesses ist es notwendig, die darin enthaltenen Aktivitäten zu definieren und zu beschreiben. Daher sollen auch beim Sicherheitstest-Prozess die Aktivitäten soweit wie möglich oder sinnhaft definiert und beschrieben werden.

A03 - Definition der beteiligten Rollen
Neben der Beschreibung der Aktivitäten sollen auch die Rollen, die die jeweilige Aktivität ausführen detailliert beschrieben werden.

A04 - Schnittstellen zu anderen Prozessen
Um Mehraufwand zu verhindern soll geprüft werden, ob bestimmte Aktivitäten bereits von anderen Prozessen abgedeckt werden. Ist dies der Fall, soll eine Schnittstelle zu diesen Prozessen ermöglicht werden. Dies kann zur Steigerung der Effizienz beitragen.

A05 - Automatisierung
Um den Prozess effizient zu gestalten und Rücksicht auf die begrenzten personellen und finanziellen Kapazitäten eines Hochschulrechenzentrums zu nehmen, sollten Aktivitäten auf ihre Automatisierbarkeit hin geprüft werden.

A06 - Unterstützung manueller Aktivitäten
Aktivitäten die nicht automatisierbar sind, sollen dennoch soweit wie möglich technisch unterstützt werden, um den personellen Aufwand so gering wie möglich zu halten.

A07 - Messbar
Um den Erfolg des Sicherheitstestmanagements zu verifizieren und kontrollieren zu können, sollen die Prozesse so gestaltet werden, dass messbare Ergebnisse produziert werden können, um eine Zahlenbasis für Key Performance Indicators (KPI) zu erhalten.

A08 - Berichte an die Unternehmensführung
Da das Managementsystem für Sicherheitstests als Überprüfung des Sicherheitskonzepts dienen kann, sollte die Unternehmensführung, als verantwortliche Stelle der Informationssicherheit, über die Ergebnisse informiert werden, um den Erfolg oder Misserfolg des ISMS messen zu können.

3.5.2. Anforderungen an das Dienstmanagement

D01 - Existenz einer Dienst-Dokumentation
Um Ausfälle während eines Sicherheitstests im Betrieb zu verhindern, sollten diese bei der Planung eines Tests berücksichtigt werden. Da es für jemanden der mit einem Dienst nicht vertraut ist schwierig sein kann das Risiko eines Betriebsausfalls zu erkennen, sollte eine Dokumentation über den Dienst vorhanden sein.

Auch zur Weitergabe an die Tester bei einem White-Box Test sollte eine Dokumentation zur Verfügung stehen.

D02 - Register von Diensten
Da alle Dienste regelmäßig auf ihre Sicherheit hin geprüft werden sollen, ist es notwendig, dass ein Register aller Dienste vorhanden ist. Nur so kann kontrolliert werden, dass alle Dienste bei der Planung von Tests berücksichtigt werden.

D03 - Zentralisierung des Registers von Diensten
Um ein einheitliches Datenformat und eine einheitliche Datenbasis zu haben, sollte das Register von Diensten zentral existieren.

D04 - Kritikalität von Diensten
Um eine Priorisierung der Dienste vornehmen zu können ist ein geeignetes Kriterium notwendig. Die Kritikalität, also das Risiko eines Dienstes, bietet sich hier an. Dazu muss das Risiko für jeden Dienst bekannt und dokumentiert sein.

3.5.3. Anforderungen an das Testmanagement

T01 - Nicht betriebsgefährdend
Da neben der Vertraulichkeit und der Integrität auch die Verfügbarkeit ein Schutzziel der Informationssicherheit ist, soll die Verfügbarkeit eines Dienstes während eines Tests erhalten bleiben. Ein Dienst der getestet wird, wird oftmals weiterhin produktiv genutzt. Um dies zu ermöglichen, sollen alle Sicherheitstests im Hinblick auf die Wahrung der Verfügbarkeit geplant und durchgeführt werden.

T02 - Wiederholbar
Um Testergebnisse zu überprüfen oder die Behebung einer gefundenen Schwachstelle zu verifizieren, sollen alle Sicherheitstests reproduzierbar durchgeführt und dokumentiert werden.

T03 - Zentrales Register von Sicherheitstests
Da alle Dienste möglichst regelmäßig getestet werden sollen, ist es notwendig, einen Überblick über alle bereits erfolgten Sicherheitstest zu haben. Des Weiteren soll der ganze Prozess auch messbar sein (vgl. *FA07 - Messbar*). Um dies zu ermöglichen, ist ein Register über alle erfolgten Sicherheitstests notwendig. Um eine einheitliche Datenbasis zu gewährleisten, um eine vollständige Übersicht und um eine korrekte Übersicht zu ermöglichen, sollte dies zentral erfolgen.

T04 - Dokumentation des Tests
Damit Sicherheitstests nachvollziehbar und wiederholbar sind, müssen sie dokumentiert werden. Dazu sollen zum einen die Rahmenbedingungen des Tests, also welcher Dienst getestet wurde, welche Art von Test durchgeführt wurde und welche Informationen dem Tester übergeben wurden, dokumentiert werden. Die Ergebnisse des Tests müssen dokumentiert werden.

T05 - Regelmäßiges Testen aller Dienste
Um einen Überblick über die Sicherheit aller Dienste zu erhalten und einen möglichst aktuellen Status zu haben, sollten die Dienste regelmäßig getestet werden. Durch regelmäßige Sicherheitstests kann

auf neue Angriffsmethoden reagiert werden und bei der Weiterentwicklung eines Dienstes können diese Weiterentwicklungen ebenfalls regelmäßig auf Sicherheit geprüft werden. Da der Sicherheitsprozess kontinuierlich verbessert werden sollte, können die Verbesserungen somit schnell messbar gemacht werden und evaluiert werden.

T06 - Risikobasierte Festlegung des Testintervalls

Um Ressourcen zu schonen ist es notwendig die Testintervalle eines Dienstes nicht zu gering zu gestalten. Auch das Risiko für Betriebsausfälle wäre bei sehr kurzen Testintervallen sehr hoch. Daher sollten die Testintervalle wohl überlegt sein. Auch die begrenzten Ressourcen für einen Sicherheitstest schränken das Testintervall ein. Für die Festlegung ist es daher ratsam das Risiko eines Dienstes bei der Festlegung der Testintervalle zu berücksichtigen. So kann ein sehr kritischer Dienst öfter und ein unkritischer seltener getestet werden.

T07 - Prozess zur Durchführung eines Tests

Da Sicherheitstests ein hohes Risiko für Betriebsstörungen bergen, müssen diese genau geplant werden. Da sie regelmäßig durchgeführt werden sollen und dabei nachvollziehbar und wiederholbar sein sollen, ist es ratsam einen Prozess zu definieren.

T08 - Genehmigungen einholen

Im Falle einer Betriebsstörung stellt sich schnell die Frage nach der Verantwortlichkeit. Daher sollte der Verantwortliche des zu testenden Dienstes bestätigen, dass ein Dienst getestet werden kann und die Rahmenbedingungen des Tests etwa die Aggressivität bestätigen. Sollten bei einem Dienst zum Zeitpunkt des Tests anderweitige Störungen vorhanden sein, könnte auch ein sehr vorsichtiger Test einen kompletten Ausfall des Systems verursachen. Um die Bereitschaft für einen Test zu bestätigen, sollte daher ein Verantwortlicher für den Dienst dem Test zustimmen. Um dies zu formalisieren, sollte seine Genehmigung eingeholt werden. Auch andere Beteiligte könnten Test verhindernde Informationen haben und sollten daher vor dem Test dazu befragt werden. Der Sicherheitsverantwortliche könnte zum Beispiel von anderen Tests wissen, diese Information ist für die Testplanung sehr wichtig und es sollte sichergestellt werden, dass diese Informationen vorliegen.

T09 - Testbenachrichtigungen verteilen

Neben den Genehmigungen könnten andere Beteiligte existieren, deren Tätigkeiten vom Sicherheitstest beeinträchtigt werden könnten. Ein Serveradministrator der seine Log-Dateien prüft und dabei die simulierten Angriffe bemerkt, sollte wissen, dass es sich um simulierte und keine echten Angriffe handelt. Um Fehlalarme und unnötige Mehraufwände zu verhindern, sollten auch Benachrichtigungen an andere Beteiligte verteilt werden.

T10 - Festlegung eines Testbereichs

Um spezifisch einen Dienst zu testen und somit die Aufwände für einen Test möglichst gering zu halten, muss ein Testbereich für einen Sicherheitstest festgelegt werden. Dazu zählt der Dienst selbst kann aber auch angebundene Dienste umfassen. Soll z.B. eine Webanwendung getestet werden die die Authentifizierung über einen zentralen Identity Provider ermöglicht, wäre es ratsam, in Teilen auch die Authentifizierung zu testen. Sollte dies nicht erwünscht sein, so muss dies auch dokumentiert werden.

T11 - Bereitstellung von Informationen über den Dienst

Damit der Tester die Möglichkeit hat den Dienst direkt zu testen, sollte er Informationen über das Testziel erhalten. Die Bereitstellung der Informationen und deren Dokumentation soll daher ebenfalls als Prozessschritt definiert werden.

T12 - Austausch von Kontakten im Falle von Betriebsstörungen

Obwohl ein Test in Hinblick auf die Verfügbarkeit geplant und durchgeführt wird, kann es trotzdem zu Betriebsstörungen oder gar Ausfällen kommen. Um in einem solchen Fall den Sicherheitstest als Ursache auszuschließen oder zu bestätigen, sollten daher Kontaktdaten zwischen dem Tester und den für den Betrieb Zuständigen ausgetauscht werden. So kann bei einer Störung ohne Umwege Kontakt aufgenommen werden.

3.5.4. Anforderungen an das Schwachstellenmanagement

S01 - Register von Schwachstellen
Um die Ergebnisse des Sicherheitstestprozesses nach ISMS ermitteln zu können und somit Aussagen über die Sicherheit im Unternehmen zu ermöglichen ist es wichtig, die gefundenen Schwachstellen zu sammeln. Dies ermöglicht es einfach messbare Ergebnisse zu liefern. Gefundene Schwachstellen sollen auch behoben werden. Um eine Nachverfolgung, also eine Überwachung des Schwachstellenbehebungsprozesses zu ermöglichen, ist es wichtig, dass die Schwachstellen registriert werden.

S02 - Dokumentation einer Schwachstelle
Um eine schnelle Behebung zu ermöglichen, sollten Schwachstellen dokumentiert werden. Durch diese Dokumentation soll es möglich sein, die Schwachstelle schnell zu identifizieren und zu beheben. Dazu sollte eine Schwachstelle folgende Informationen beinhalten:

- **Beschreibung der Ursache**, um die Schwachstelle nachvollziehen zu können und Möglichkeiten der Behebung zu erkennen.
- **Lösungsvorschläge**, um den Behebern einer Schwachstelle eine Hilfestellung zur schnellen Behebung zu ermöglichen.
- **Bewertung**, um das daraus resultierende Risiko besser einschätzen zu können und die Behebung von Schwachstellen zu priorisieren.
- **Kategorisierung**, um häufig auftretende Schwachstellen zu erkennen und geeignete Gegenmaßnahmen ergreifen zu können, z.B. Schulungen.

S03 - Schwachstellennachverfolgung
Um sicherzustellen, dass gefundene Schwachstellen behoben wurden muss die Behebung verfolgt werden. Sollte die Behebung nicht schnell genug erfolgen, können geeignete Schritte unternommen werden, um dies zu beschleunigen bspw. eine Eskalation zur Unternehmensführung.

S04 - Behebung einer Schwachstelle
Um eine Schwachstelle zu schließen soll die Schwachstelle behoben werden. Nach der Behebung durch bspw. eine neue Version des Dienstes, sollte die Behebung noch verifiziert werden.

S05 - Schnittstelle ins Risikomanagement
Falls die Behebung einer Schwachstelle nicht möglich ist oder z.B. aus Kostengründen nicht gewünscht ist, sollte die Schwachstelle bzw. das daraus resultierende Risiko im Risikomanagement behandelt werden. Dazu muss eine Schnittstelle zum Risikomanagement geschaffen werden.

3.6. Prozessunterstützende Anwendung

Um die Prozessschritte zu etablieren und zu manifestieren sowie die Anforderungen *A05 - Automatisierung*, *A06 - Unterstützung manueller Aktivitäten* und *A07 - Messbar* zu erfüllen, soll eine Anwendung konzipiert werden, die den Prozess unterstützt und möglichst viele Aktivitäten automatisiert.

Dabei soll die Anwendung es auch ermöglichen die eigentlichen Tests möglichst automatisch ablaufen zu lassen, um Sicherheitstests kostengünstig realisieren zu können.

3.7. Tabellarische Auflistung der Anforderungen

Im Folgenden sollen die einzelnen Anforderungen nochmals zur Übersichtlichkeit aufgelistet werden. Zusätzlich wurden die Anforderungen priorisiert. Die Priorisierungen sind *notwendig* für die Anforderungen die absolut erforderlich für ein Managementsystem für Sicherheitstests sind und *empfohlen* für Anforderungen die wünschenswert sind, ohne die jedoch trotzdem ein rudimentäres Managementsystem möglich wäre.

ID	Beschreibung	Gewichtung
Allgemeine Anforderungen		
A01	Flexibilität des Anwendungsbereichs	empfohlen
A02	Definition der Aktivitäten	notwendig
A03	Definition der beteiligten Rollen	notwendig
A04	Schnittstellen zu anderen Prozessen	empfohlen
A05	Automatisierung	empfohlen
A06	Unterstützung manueller Aktivitäten	empfohlen
A07	Messbar	empfohlen
A08	Berichte an die Unternehmensführung	notwendig
Anforderungen an das Dienstmanagement		
D01	Existenz einer Dienst-Dokumentation	empfohlen
D02	Register von Diensten	notwendig
D03	Zentralisierung des Registers von Diensten	empfohlen
D04	Kritikalität von Diensten	empfohlen
Anforderungen an das Testmanagement		
T01	Nicht betriebsgefährdend	notwendig
T02	Wiederholbar	empfohlen
T03	Zentrales Register von Sicherheitstests	notwendig
T04	Dokumentation des Tests	empfohlen
T05	Regelmäßiges Testen aller Dienste	notwendig
T06	Risikobasierte Festlegung des Testintervalls	empfohlen
T07	Durchführung eines Tests	notwendig
T08	Genehmigungen einholen	notwendig
T09	Testbenachrichtigungen verteilen	empfohlen
T10	Festlegung eines Testbereichs	notwendig
T11	Bereitstellung von Informationen über den Dienst	notwendig
T12	Austausch von Kontakten im Falle von Betriebsstörungen	notwendig
Anforderungen an das Schwachstellenmanagement		
S01	Register von Schwachstellen	empfohlen
S02	Dokumentation einer Schwachstelle	notwendig
S03	Schwachstellennachverfolgung	notwendig
S04	Behebung einer Schwachstelle	notwendig
S05	Schnittstelle ins Risikomanagement	empfohlen

Tabelle 3.1.: Tabellarische Auflistung der Anforderungen

4. Related Work

Nachdem die Anforderungen an ein Managementsystems für Sicherheitstests für Hochschulrechenzentren erläutert wurden, sollen nun bereits existierende Managementsysteme für Sicherheitstests erläutert und mit den Anforderungen verglichen werden. Da die Anforderungen für regelmäßige Sicherheitstests aus dem ISMS nach 27001 bzw. IT-Grundschutz abgeleitet werden, soll zunächst auf diese beiden Richtlinien eingegangen werden.

4.1. ISO 27000

Neben dem ISO Standard 27001 *Information security management systems* existieren noch die Richtlinien ISO 27002 *Code of practice for information security controls* und die ISO 27007 *Guidelines for information security management systems auditing*. Auf diese Richtlinien wird nachfolgend eingegangen.

4.1.1. ISO 27002

Die Richtlinie ISO 27002 *code of practice for information security controls* beschreibt einen Standard um Kontrollen des gesamten Sicherheitsprozesses auszuwählen und zu implementieren. Dabei listet der Standard selbst 114 Kontrollen auf, die durchgeführt werden sollen. Diese Kontrollen sind sehr abstrakt und reichen von Kontrollen der Sicherheitsrichtlinie und welche Themen diese adressieren soll, bis zu Kontrollen was Logeinträge beinhalten sollen [ISO13].

So gibt es unter anderem eine definierte Kontrolle für die Existenz eines Asset-Inventars (Kapitel 8), wobei im Rahmen der ISO 27000 Familie alles, was einen Wert für ein Unternehmen hat, ein Asset ist. Dienste fallen klar in diese Definition. Dabei muss für jedes Asset ein Besitzer (Owner) definiert sein, also jemand der Verantwortung für das Asset trägt. Auch muss ein Asset klassifiziert sein, die Kriterien sollen dabei u.a. rechtliche Anforderungen, Wert und Kritikalität des Assets sein.

Auch Kontrollen zum Schwachstellenmanagement finden sich in der ISO 27002 (Kapitel 12.6). Dort wird gefordert, dass Schwachstellen priorisiert und nachverfolgt werden.

Unter Kapitel 14.2.8 beschreibt die Richtlinie auch Sicherheitstests. Dabei besagt die Richtlinie lediglich, dass bereits während der Entwicklung bzw. bei Einführung eines Assets dieses getestet werden soll. Sie verbietet jedoch nicht, dass produktive Assets oder Dienste getestet werden. Vielmehr beschreibt die Richtlinie bereits in der Einführung, dass Sicherheit ein Prozess ist und eine kontinuierliche Verbesserung einschließlich Tests notwendig sind.

Trotz vieler Überschneidungen mit den Anforderungen aus Abschnitt 3 Anforderungsanalyse, eignet sich die ISO 27002 nicht als Hilfe für eine Implementierung, da diese nur die Existenz von Kontrollen fordert aber nicht beschreibt, wie diese umgesetzt werden können.

4.1.2. ISO 27007

Die ISO 27007 *Guidelines for information security management systems auditing* beschreibt einen Standard um Informationssicherheits Managementsysteme zu prüfen. Dabei setzt die ISO 27007 auf die ISO 19011 auf, die beschreibt wie Managementsysteme generell prüft werden sollen.

In diesem Standard wird das gesamte ISMS auf den ISO 27001 Standard getestet. Er dient dazu einem Prüfer einen Leitfaden für die Prüfung eines Unternehmens an die Hand zu geben und somit die Prüfungen zu vereinheitlichen. Bei der Prüfung werden außerdem nur organisatorische Prozesse und Richtlinien geprüft. Die technische Umsetzung wird dabei eher vernachlässigt.

4.2. IT-Grundschutz

Nachdem die ISO 27000 Familie keine möglichen Umsetzungen beschreibt, soll nun der IT-Grundschutz daraufhin untersucht werden. Im IT Grundschutz des BSI sind, neben den formalen Anforderungen an ein ISMS, auch empfohlene Maßnahmen beschrieben [BSI16].

Die Maßnahme M 5.150 des Grundschutzkatalogs empfiehlt kritische Dienste regelmäßig einem Penetrationstest zu unterziehen. Penetrationsstest sind dabei manuelle Sicherheitstests. Neben den manuellen Tests werden auch automatisierte Tests als Ergänzung empfohlen.

Als Vorgehensweise wird zunächst eine Vorbereitungsphase erläutert. Der Testrahmen, also welcher Dienst und welche Schnittstellen, sowie wie getestet werden soll, muss mit dem Tester besprochen werden. Auch Genehmigungen werden erwähnt, bspw. die des Datenschutzbeauftragten. Jedoch wird meist von Testern als externe Dienstleister gesprochen. Dabei ist der einzige Unterschied zu internen Tests, dass neben den Genehmigungen auch Verschwiegenheitserklärungen zwischen dem Dienstleister und dem beauftragenden Unternehmen ausgehandelt werden müssen.

Nach der Vorbereitungsphase wird anschließend direkt zur Testdurchführung übergegangen. Dabei wird kurz die Vorgehensweise bei einem Test erläutert. (vgl. Abschnitt 2.5.2) Nach der Vorbereitung wird in eine Auswertungs- und Berichtsphase übergegangen bei der der Tester einen detaillierten Bericht erstellen muss. Dieser wird dann an den IT-Sicherheitsverantwortlichen sowie verantwortliche Führungskräfte verteilt.

Die Schwachstellenbehebung sowie die Inventarisierung der Dienste wird zwar mehrfach erwähnt, jedoch werden Prozesse dazu nicht weitergehend beschrieben.

4.3. Dienstmanagement

Da sich zu einem Managementsystem für Sicherheitstests keine fertigen Lösungen fanden, die alle Aspekte der Anforderungen erfüllten, sollen nun die einzelnen Teile eines Managementsystems für Sicherheitstests, das **Dienstmanagement**, das **Testmanagement** und das **Schwachstellenmanagement** auf bestehende Arbeiten und Konzepte untersucht werden.

Zunächst soll das Dienstmanagement näher betrachtet werden. Neben der Notwendigkeit ein Inventar für Sicherheitstests zu haben, haben auch andere Unternehmensprozesse etwa die Vermögensverwaltung oder das IT-Servicemanagement einen Bedarf für ein Dienstmanagement. Die IT Infrastructure Libary (ITIL) hat dazu den Prozess *Service Asset und Configuration Management* (SACM) definiert. Dieser soll nun kurz vorgestellt werden.

Nach ITIL dient ein SACM zur Identifizierung, Kontrolle, Aufzeichnung, Auditierung und Verifizierung von Service-Assets [Lac07]. Dabei sind Assets alles was es zur Erbringung eines Dienstes benötigt, z.B. Mitarbeiter, Richtlinien, Hardware, Software, Unterdienste, etc.) Das SACM soll die anderen ITIL-Prozesse, etwa das Problem- und Incidentmanagement, unterstützen.

Um das managen von größeren Infrastrukturen zu ermöglichen, wird ein unterstützendes zentrales System, ein Configuration Management System (CMS), benötigt. Dieses soll alle Informationen über ein Configuration Item beinhalten. Dabei kann ein Configuration Item (CI) ein Asset, eine Service Komponente oder ein anderer Gegenstand sein.

Im Service Asset und Configuration Management gibt es fünf Aktivitäten. Die erste, das **Management and Planning** beschäftigt sich mit der Planung des CMS. Dabei gilt es den Anwendungsbereich und die Detailgenauigkeit der Configuration Items festzulegen.

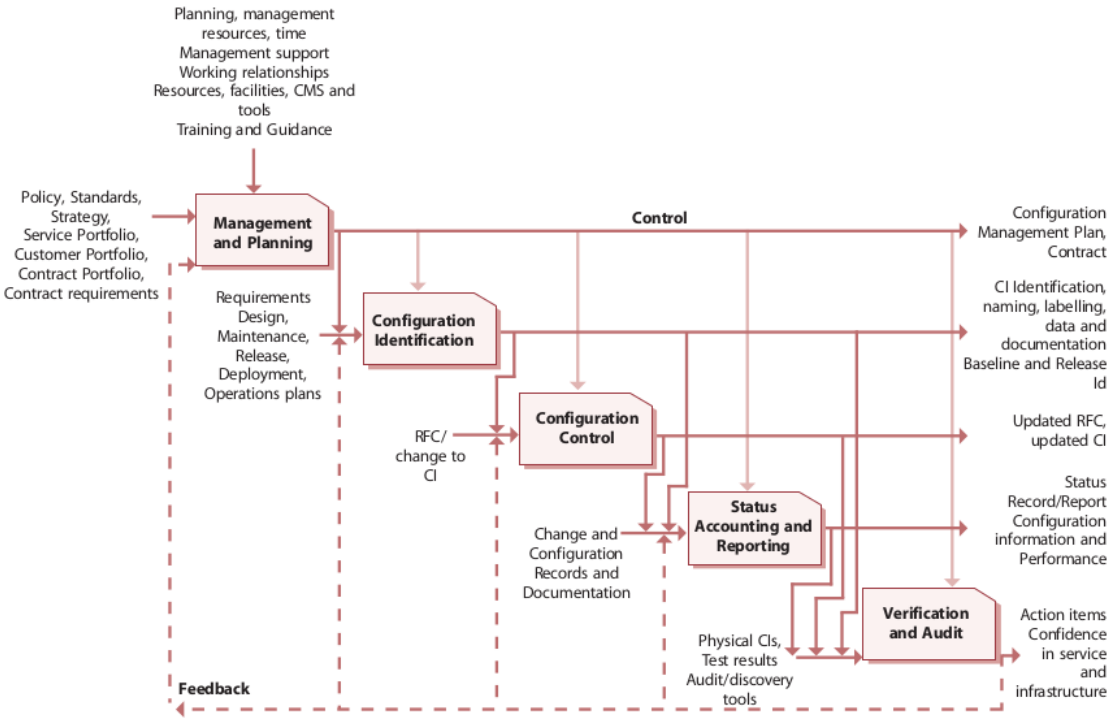


Abbildung 4.1.: Die Aktivitäten des Service Asset und Configuration Management (entnommen aus [Lac07])

Die zweite Aktivität ist die **Configuration identification**. Dort wird festgelegt wie eine Kategorisierung durchgeführt werden kann und wie die Hierarchien der Assets abgebildet werden. Auch die Informationen über ein Asset oder CI kann nach Typ variieren. Die Informationen, die zu einem Asset-Typen gespeichert werden, sollen ebenfalls in dieser Phase definiert werden.

Bei der **Configuration control** Aktivität geht es um den Betrieb des CMS. Das heißt sollten Änderungen an Assets vorgenommen werden, müssen diese Änderungen auch im CMS übernommen werden. Dazu gilt es andere Prozesse z.B. das Changemanagement so anzupassen, dass möglichst automatisiert eine Änderung auch an das CMS übertragen wird.

Die **Status accounting and reporting** Aktivität soll den Status eines Assets aktuell halten. Dabei geht es um den Asset-Lebenszyklus der auch in einem CMS abgebildet sein soll. Der Status des Lebenszyklus spielt bei anderen Prozessen etwa dem Monitoring eine wichtige Rolle. Sollte ein Asset nicht mehr vorhanden sein, da es aus dem Betrieb genommen wurde, sollte das Monitoring davon wissen und keine Alarme wegen der plötzlichen Nichtverfügbarkeit erzeugen.

Die **Verification and audit** Aktivität dient der Qualitätssicherung des CMS. Dabei soll der Inhalt des Configuration Management Systems mit der Realität abgeglichen werden. So können Assets untersucht werden und mit den Informationen im CMS abgeglichen werden oder Assets aus dem CMS ausgewählt werden und das reale Asset dazu gesucht und verglichen werden. Diese Qualitätssicherungstests sollen vor und nach Änderungen der Infrastruktur als auch regelmäßig erfolgen.

4.4. Testmanagement

Zum Testen von Software gibt es sehr viele Arbeiten u.a. [Rät02], [Spi14]. Als Beispiel für Testmanagement soll abermals ITIL herangezogen werden, da dort auch Prozesse beschrieben werden, die als best-practice gelten.

4.4.1. "Service Validation and Testing" nach ITIL

Bei der IT Infrastructure Libary (ITIL) steht das IT Servicemanagement im Vordergrund. Dabei spielt die Sicherheit von Diensten eine wichtige Rolle. Generell berücksichtigt ITIL hauptsächlich Anforderungen der Kunden an Dienste, die mit Hilfe von Service Level Agreements (SLA) festgelegt wurden. Diese SLAs sind meist so ausgelegt, dass die Anforderungen messbar sind und es somit eindeutig festgelegt ist, ab wann eine Anforderung nicht mehr erfüllt ist. Dies ist bei der Sicherheit von Anwendungen naturgemäß schwierig.

Um die Anforderungen einzuhalten wird dabei der Prozess *Service Validation and Testing* verwendet. Dieser soll verifizieren dass alle Anforderungen eingehalten werden. Dabei setzt der Prozess stark auf ein Releasemanagement auf. Es sollen somit neue Versionen, die eingeführt werden sollen, bereits vor der Einführung getestet werden. Dabei erwähnt ITIL auch die IT-Sicherheit. Auch wenn dies ein wünschenswerter Zustand wäre, dass nur getestete Versionen produktiv eingesetzt werden, entspricht dies leider nicht immer der Realität. Des weiteren wäre dies gerade bei der IT-Sicherheit sehr teuer, da man Spezialisten benötigt um die Sicherheit ausreichend zu testen. Auch im Kontext eines Hochschulrechenzentrums mit sehr vielen Diensten würde dies sehr teuer werden.

Da der Prozess *Service Validation and Testing* hauptsächlich auf neue Versionen zielt eignet er sich nicht für Sicherheitstests in Hochschulrechenzentren im Hinblick auf die Anforderungen, da diese fordern, dass auch bereits im Betrieb befindliche Dienste getestet werden sollen. Dennoch wäre das Testen von Diensten bevor diese produktiv werden wünschenswert, sofern die nötigen Ressourcen vorhanden sind.

4.4.2. BSI Leitfaden Penetrationstests

Das Bundesamt für Sicherheit in der Informationstechnologie (BSI) als Herausgeber des IT-Grundschutzes hat für die geforderten Sicherheitstests einen Leitfaden für die Durchführung von Penetrationstests erstellt. Dabei zielt der Leitfaden hauptsächlich auf manuelle Sicherheitstests ab, die als Penetrationstests bezeichnet werden [BSI14].

Das BSI unterteilt dazu die Durchführung eines Sicherheitstests in drei Phasen. Die **Einarbeitung des Testers**, die **Durchführung des Tests** und die **Berichtsphase**. Zuvor müssen jedoch noch die Voraussetzungen herausgearbeitet werden. Dies wird zwar vom BSI nicht direkt als Phase bezeichnet kann aber als eine **Vorbereitungsphase** interpretiert werden.

Vorbereitungsphase

Das BSI empfiehlt die Beauftragung eines externen Anbieters für die Durchführung eines Sicherheitstests. Die Gründe dazu sind hauptsächlich, dass die Unabhängigkeit des Testers gewahrt bleiben soll und Betriebsblindheit verhindert werden soll. Dabei wird erwähnt, dass das Kosten-Nutzen Verhältnis gewahrt bleiben soll. Wegen der Empfehlung externe Dienstleister einzusetzen, wird jedoch explizit ein Vertrag zwischen dem Dienstleister und dem Unternehmen gefordert. Auch wenn bei internen Tests solch ein Vertrag nicht notwendig ist, so ist es trotzdem ratsam, die Inhalte schriftlich festzuhalten. Das BSI spricht hierzu einige Inhalte an:

- Die Festlegung des Prüfobjekts, also welcher Dienst oder welche Dienste getestet werden sollen.
- Die Prüftiefe, also der Umfang (siehe Klassifizierung von Sicherheitstests 2.5.1).
- Der Prüfort, wo der Tester sich zum Test befindet und wie er Zugang zu den Systemen erhält, bspw. in den eigenen Räumlichkeiten und Zugriff mittels Virtual Private Network (VPN).
- Die Prüfbedingungen beschreiben, ob es sich um ein produktives System handelt bei dem Daten nicht verändert oder gelöscht werden dürfen oder um ein Testsystem, bei dem auch das Löschen und Editieren von Daten möglich ist.
- Der Prüfzeitraum, zum einen der Beginn und das Ende des Tests zum anderen die täglichen Testzeiten. Wenn es sich um Tests auf produktiven Diensten handelt, könnte z.B. vereinbart werden, dass die Tests

zu Randzeiten durchgeführt werden, sodass die legitimen Benutzer keine Störungen durch den Test erfahren.

Auch weitere Informationen zum Prüfobjekt werden Übergeben. Dabei empfiehlt das BSI ausdrücklich White-Box Tests durchzuführen, da bei Black-Box Tests Schwachstellen leichter übersehen werden können. Für White-Box Tests sollten alle Informationen über einen Dienst an den Tester übergeben werden, damit dieser sie analysieren kann und mit Hilfe dieser Informationen den Test planen kann.

Verantwortlichkeiten sind ebenfalls festzuhalten. Es wird daher empfohlen Ansprechpartner auf Seite des Unternehmens als auch auf Seiten des Anbieters zu definieren und die Kontaktdetails auszutauschen. Auch eine Benachrichtigung von betroffenen Personen wird empfohlen, damit diese Personen, die aufgrund des Tests in ihrer Arbeit beeinträchtigt werden, vom Test wissen und somit die Beeinträchtigungen richtig einordnen können. Auch die Wartung eines Dienstes während eines Tests sollte verhindert werden, um den Test nicht zu verfälschen.

Neben den spezifischen Details zu einem Test, sollte gerade bei einem externen Anbieter zusätzlich eine Verschwiegenheitserklärung eingeholt werden, die es verhindern soll, dass die Ergebnisse an unbefugte weitergegeben werden. Sollte der Tester Zugang zu Daten erhalten die unter den Schutz des Bundesdatenschutzgesetzes fallen, muss der Umgang mit diesen Daten auch definiert werden. Dazu sollte vereinbart werden, dass die Daten nur anonymisiert in den Bericht aufgenommen werden.

Einarbeitung des Testers

Nachdem der Test vorbereitet wurde und die Rahmenbedingungen geklärt wurden, kann der Tester sich mit Hilfe der übergebenen Informationen auf den Test vorbereiten. Bei exotischen Diensten kann es z.B. nötig sein sich mit den Standards vertraut zu machen und geeignete Tools auszuwählen. Auch eine Recherche zu aktuellen Schwachstellen in der eingesetzten Software kann dazugehören.

Durchführung des Tests

Die eigentliche Durchführung setzt sich dann aus einem **Anfangsgespräch**, indem die Rahmenbedingungen des Tests nochmals durchgesprochen werden um Missverständnisse zu beseitigen. Anschließend werden die **Prüfbedingungen** nochmals abgeklärt und es wird geprüft ob die zu testenden Systeme erreichbar sind.

Nachdem aller Voraussetzungen erfüllt sind, beginnt die **Praktische Prüfung**. Hierbei untersucht der Tester die übermittelten und selbst gewonnene Informationen, z.B. aus einer Recherche, auf konzeptionelle Schwachstellen. Auch werden die Systeme selbst mit Hilfe eines Portscanners überprüft. Anhand dieser Informationen kann der Tester wieder neue Schwachstellen entdecken. Wenn ein Tester herausgefunden hat welche Software und welche Versionen dieser Software vom Dienst genutzt wird, kann er nach bereits bekannten Schwachstellen für diese Versionen suchen. Auch kann ein Tester selbstständig nach Schwachstellen suchen und versuchen diese Auszunutzen. Dabei muss er stets darauf achten, dass produktive Dienste erreichbar bleiben und auch weiterhin funktionieren.

Nachdem der Test abgeschlossen ist findet noch ein **Abschlussgespräch** statt. In diesem werden direkt die besonders kritischen Schwachstellen dem Auftraggeber präsentiert. Die übrigen Schwachstellen werden dann vom Tester in einem Bericht dokumentiert der anschließend dem Auftraggeber übergeben wird.

Berichtsphase

Nachdem der Bericht dem Auftraggeber übergeben wurde, muss dieser innerhalb des Unternehmens so verteilt werden, dass zum einen die Behebung der Schwachstellen möglich ist und zum anderen, dass nur berechtigte Beteiligte den Bericht lesen können, da dort ggf. sehr kritische Schwachstellen beschrieben sind.

Im Bericht sollen für Techniker eine genaue Beschreibung der Schwachstelle sowie Behebungsmöglichkeiten vermerkt sein. Auch eine Einstufung der Kritikalität soll enthalten sein, sodass die Behebung der einzelnen Schwachstellen priorisiert werden kann.

4.5. Schwachstellenmanagement

Nachdem Schwachstellen in einem Sicherheitstest gefunden wurden, muss mit diesen Umgegangen werden. Zum Thema Schwachstellenmanagement existieren wenige Arbeiten, diese legen jedoch meist den Fokus auf automatisierte Schwachstellenscans und die Behebung dieser Ergebnisse. Interpretiert man einen Sicherheitstest als Schwachstellenscan, können auch Konzepte aus dem Schwachstellenmanagement übernommen werden.

Das *SysAdmins, Networking and Security* (SANS) Institute ist eine Organisation mit dem Ziel kooperative Forschung und Bildung zu ermöglichen. Dazu bieten sie zum einen Schulungen und Zertifizierungen an und zum anderen veröffentlichen sie Arbeiten zu Themen der IT-Security. Unter anderem wurde eine Arbeit mit dem Titel *Implementing a Vulnerability Management Process* veröffentlicht. Diese soll nun kurz erläutert werden [Pal13].

In dem beschriebenen Schwachstellenmanagement Prozess sind vier Rollen definiert.

- Der **Security Officer** als Gestalter und verantwortlicher für die Umsetzung des Prozesses.
- Ein **Vulnerability Engineer** als Verantwortlicher für die Konfiguration des Schwachstellenscans und der Planung der Scans.
- Ein **Asset Owner** als Verantwortlicher für einen Dienst.
- Ein **IT System Engineer** der die Behebung einer Schwachstelle durchführen muss.

Der Prozess gliedert sich in fünf Phasen. Die Vorbereitung, die Durchführung des Schwachstellenscans, die Definition der Behebungsmaßnahmen, die Durchführung der Behebungsmaßnahmen und einem Prüfscan.

1. In der **Vorbereitungsphase** wird zunächst das Ziel des Schwachstellenscans definiert und anschließend wird der Asset Owner sowie die IT informiert. Die Informationen für die IT sollten im speziellen eine Information an die Betriebseinheiten für Firewalls, Intrusion Detection und Prevention Systeme sowie für Logfile Analysen sein, sodass diese bei einem Alarm, der aus dem Scan resultiert, die Ursache schnell erkennen und den Alarm dementsprechend behandeln können. Auch der Scan-Zeitraum wird in der Vorbereitungsphase definiert.

2. Nachdem die Vorbereitung abgeschlossen wurde, kann die **Durchführung des Schwachstellenscans** beginnen. Dabei ist es wichtig die Performance und Verfügbarkeit der gescannten Dienste genau zu überwachen und aufzuzeichnen um in Zukunft die Scans so anzupassen, dass die Dienste keine Beeinträchtigung mehr erfahren.

3. Für die **Definition der Behebungsmaßnahmen** bekommen der Security Officer als auch der IT System Engineer das Scanergebnis um anschließend Möglichkeiten der Behebung auszuarbeiten. Die Empfehlungen der beiden Rollen werden dann dem Asset Owner übergeben, der sich dann entweder für eine Lösung entscheiden muss oder aber das Risiko akzeptieren muss. Wenn es sich dazu entschließt das Risiko zu tragen, wird zu einem Risikomanagement Prozess übergegangen. Soll die Schwachstelle behoben werden wird ein Behebungsplan ausgearbeitet.

4. Die **Durchführung der Behebungsmaßnahmen** werden vom IT System Engineer umgesetzt. Sofern die Umsetzung erfolgreich war wird zu nächsten Phase übergegangen. Falls nicht, müssen neue Behebungsmaßnahmen ausgearbeitet werden, oder es muss sich doch dazu entschlossen werden das Risiko zu tragen. Dies stellt einen Rücksprung in die zweite Phase dar.

5. Sind die Maßnahmen erfolgreich umgesetzt worden wird ein **Rescan** vom Vulnerability Engineer durchgeführt. Falls die Schwachstelle auch im Scan nicht mehr auftritt war die Behebung erfolgreich und die Schwachstelle gilt als geschlossen. Ergibt der Rescan, dass die Schwachstelle immer noch existiert, muss wieder zur zweiten Phase übergegangen werden und neue Maßnahmen ausgearbeitet werden.

4.6. Unterstützende Anwendungen

Unterstützende Anwendungen die alle Bereiche des Sicherheitstestprozesses abdecken, konnten nicht gefunden werden. Daraufhin wurden unterstützende Anwendungen zu den einzelnen Teilbereichen gesucht. Für das Dienstmanagement, insbesondere im Hinblick auf ITIL, wurden zahlreiche Anwendungen gefunden. Für das Testmanagement und das Schwachstellenmanagement konnten ebenfalls Anwendungen gefunden werden, allerdings erheblich weniger. Die Anwendungen im Hinblick auf das Testmanagement und Schwachstellenmanagement legen jedoch einen deutlichen Schwerpunkt auf automatisierte Tests und automatisierte Schwachstellenscans. Neben der Tatsache, dass die Anwendungen nur spezielle Bereiche des Sicherheitstestprozesses abdecken, haben die Anwendungen sehr wenige Schnittstellen, um die verschiedenen Anwendungen so zu verbinden, dass der ganze Prozess abgebildet werden kann.

4.7. Zusammenfassung der erfüllten Anforderungen

Im Folgenden sollen nun die beschriebenen und existierenden Prozesse mit den Anforderungen aus Abschnitt 3 verglichen werden. Da die beschriebenen ISO-Standards keinerlei Prozesse enthalten und nur sehr abstrakte Anforderungen an ein Managementsystem für Sicherheitstests stellen, werden diese nicht weiter berücksichtigt. Auch der IT-Grundschutz und die darin enthaltene Maßnahme zur Durchführung von Sicherheitstests ist recht abstrakt gehalten und stellt keine Lösung vor, weshalb dieser auch nicht weiter berücksichtigt wird.

Daher soll im Folgenden der Prozess nach ITIL für das Dienstmanagement **Service Asset und Configuration Management (SACM)**, das Vorgehen für die Durchführung eines Sicherheitstests nach dem BSI Leitfaden **Ein Praxis-Leitfaden für IS-Penetrationstests (BSI)** und das Schwachstellenmanagement aus der Arbeit **Implementing a Vulnerability Management Process (SANS)** verglichen werden.

Zur besseren Übersichtlichkeit wird hierzu die tabellarische Auflistung der Anforderungen aus Abschnitt 3.7 verwendet. Erweitert um drei Spalten, jeweils für **ITIL**, **BSI** und **SANS**. Die Erfüllung der jeweiligen Anforderung ist abgestuft dargestellt. *oo* bedeutet, die Anforderung wird gar nicht erfüllt, *o*, die Anforderung wird teilweise erfüllt, +, die Anforderung wird erfüllt und ++ bedeutet, die Anforderung wird überfüllt, also bietet Möglichkeiten über die Anforderungen hinaus.

ID	Beschreibung	Gewichtung	**ITIL**	**BSI**	**SANS**
Allgemeine Anforderungen					
A01	Flexibilität des Anwendungsbereichs	empfohlen	++	o	++
A02	Definition der Aktivitäten	notwendig	++	+	+
A03	Definition der beteiligten Rollen	notwendig	++	+	+
A04	Schnittstellen zu anderen Prozessen	empfohlen	++	o	+
A05	Automatisierung	empfohlen	o	oo	++
A06	Unterstützung manueller Aktivitäten	empfohlen	+	o	o
A07	Messbar	empfohlen	++	o	++
A08	Berichte an die Unternehmensführung	notwendig	o	+	o
Anforderungen an das Dienstmanagement					
D01	Existenz einer Dienst-Dokumentation	empfohlen	++	+	o
D02	Register von Diensten	empfohlen	++	o	o
D03	Zentralisierung des Registers von Diensten	empfohlen	++	oo	oo
D04	Kritikalität von Diensten	empfohlen	o	o	oo
Anforderungen an das Testmanagement					
T01	Nicht betriebsgefährdend	notwendig	oo	++	o
T02	Wiederholbar	empfohlen	oo	oo	+
T03	Zentrales Register von Sicherheitstests	empfohlen	oo	oo	o
T04	Dokumentation des Tests	empfohlen	oo	+	o
T05	Regelmäßiges Testen aller Dienste	notwendig	oo	o	+

T06	Risikobasierte Festlegung des Testintervalls	empfohlen	oo	o	o
T07	Durchführung eines Tests	notwendig	oo	+	+
T08	Genehmigungen einholen	notwendig	oo	+	o
T09	Testbenachrichtigungen verteilen	empfohlen	oo	+	+
T10	Festlegung eines Testbereichs	notwendig	oo	+	+
T11	Bereitstellung von Informationen über den Dienst	notwendig	o	+	oo
T12	Austausch von Kontakten im Falle von Betriebsstörungen	notwendig	oo	+	+
Anforderungen an das Schwachstellenmanagement					
S01	Register von Schwachstellen	empfohlen	oo	oo	+
S02	Dokumentation einer Schwachstelle	notwendig	oo	+	+
S03	Schwachstellennachverfolgung	notwendig	oo	oo	+
S04	Behebung einer Schwachstelle	notwendig	oo	o	+
S05	Schnittstelle ins Risikomanagement	empfohlen	oo	o	+

Tabelle 4.1.: Tabellarische Auflistung der Anforderungen in Gegenüberstellung mit existierenden Prozessen.

Die allgemeinen Anforderungen erfüllt **ITIL** am Besten, bis auf *A05 - Automatisierung* werden alle Anforderungen wenigstens teilweise erfüllt. Bei den Anforderungen an das Dienstmanagement erfüllt **ITIL** ebenso die meisten Anforderungen. Die Anforderung *D04 - Kritikalität von Diensten* wird zwar nicht direkt gefordert, ist aber in einem CMS abbildbar. Die Anforderungen an das Testmanagement werden von **ITIL** gar nicht erfüllt. **BSI** legt bei Sicherheitstests einen Schwerpunkt auf manuelle Sicherheitstests, eine Wiederholbarkeit und ein regelmäßiges Testen sind daher nicht vorgesehen. Im Gegensatz zu **SANS**, dort wird sehr viel Wert auf Automatisierung und Wiederholbarkeit gelegt, leider zu Lasten von manuellen Tests und der Dokumentation dieser. Bei den Anforderungen an das Schwachstellenmanagement erfüllt **SANS** alle Anforderungen. **BSI** fordert zwar die Behebung von Schwachstellen, wie eine solche Behebung aussehen könnte erwähnen sie nicht.

Keiner der drei Prozesse deckt alle Anforderungen alleine ab, unterteilt man die Anforderungen jedoch in die Kategorien, so deckt **ITIL** das Dienstmanagement, **BSI** das Testmanagement und **SANS** das Schwachstellenmanagement fast vollständig ab.

5. Konzepterstellung

Die soeben beschriebenen Prozesse decken jeder für sich genommen nicht alle der beschriebenen Anforderungen ab. Eine Kombination aus dem in ITIL beschriebenen Service Asset und Configuration Management Prozess, dem Prozess des BSI Leitfadens zur Durchführung eines Penetrationstests sowie der Schwachstellenmanagement Prozess von SANS deckt jedoch eine Vielzahl der Anforderungen ab. Daher soll nun in Anlehnung an diese drei Prozesse ein Prozess zum Management für Sicherheitstests entwickelt werden.

5.1. Prozess zum Management für Sicherheitstests

Im Folgenden soll nun der Prozess zum Management für Sicherheitstests beschrieben werden. Dazu sollen zunächst Rollen definiert werden, die Aktivitäten in diesem Prozess wahrnehmen sollen.

5.1.1. Rollen

In dem Prozess nach *SANS* wurden diverse Rollen beschrieben. Im folgenden sollen diese nun auf ein Hochschulrechenzentrum übertragen werden. Die Rollen **Sicherheitsverantwortlicher**, **Dienstverantwortlicher**, **Entwickler**, **Administrator** und **CSIRT** können teilweise auch einer einzigen Person zugeordnet sein. Nur der **Tester** sollte möglichst nichts mit dem zu testenden Dienst zu tun haben, um Betriebsblindheit und somit das Übersehen von Schwachstellen zu vermeiden.

Sicherheitsverantwortlicher

Der Sicherheitsverantwortliche entspricht dem **Security Officer** nach *SANS*. Er ist für die komplette Sicherheit eines Unternehmens verantwortlich. Das heißt, er hat dafür Sorge zu tragen, dass die Infrastruktur und die darauf laufenden Dienste sicher sind. Hierzu dienen Arbeitsanweisungen und Richtlinien für Mitarbeiter, die der Sicherheitsverantwortliche entwirft. Des Weiteren muss er das Top-Management über Risiken der Informationssicherheit informieren. Es liegt somit hauptsächlich in seinem Interesse, Dienste und Prozesse auf deren Sicherheit zu prüfen.

Ein Sicherheitsverantwortlicher muss nicht in der Lage sein, eine Schwachstelle selbstständig zu finden oder schließen zu können. Er muss aber eine Schwachstelle und deren Hintergründe kennen. So sollte er unter anderem wissen, dass Passwörter gehasht und salted sein müssen und das Benutzereingaben stets zu validieren und zu filtern sind. Auch Schadenspotential und Eintrittswahrscheinlichkeit muss er einschätzen können, um so das daraus resultierende Risiko ermitteln zu können. Sollte es unwirtschaftlich sein, eine Schwachstelle zu schließen, muss er das Risiko an das Management berichten und Möglichkeiten zur Risikobeherrschung kennen.

Dienstverantwortlicher

Der Dienstverantwortliche enspricht der Rolle **Asset Owner** nach *SANS*. Er ist nur für einen oder mehrere Dienste verantwortlich, er muss sich um das Management des Dienstes kümmern. Er dient als zentraler Ansprechpartner für alle Belange. Er ist für den Betrieb des Dienstes, dessen Weiterentwicklung und auch die Sicherheit verantwortlich.

Ein Dienstverantwortlicher kann keinerlei Wissen in Bezug auf Sicherheit haben. Er kann sich bei Fragen zur Sicherheit an den Sicherheitsverantwortlichen wenden und muss dessen Aussagen vertrauen. Er muss bei der

Risikobewertung dem Sicherheitsverantwortlichen bei der Bestimmung des Schadenspotentials helfen, da er am Besten beurteilen kann, welche Daten im Dienst verarbeitet werden.

Tester

Der Tester ist angelegt an die Rolle **Vulnerability Engineer** nach *SANS*, im Gegensatz zu *SANS* ist er nur eingeschränkt für die Planung eines Sicherheitstests verantwortlich. Die Planung wird vom Dienstverantwortlichen übernommen, der Tester muss allerdings zum geplanten Zeitpunkt zur Verfügung stehen. Wird ein Sicherheitstest durchgeführt, wird dies von einem Tester oder einem Testteam getan. Dieser Tester muss selbstständig Schwachstellen identifizieren und auch ausnutzen können. Dabei muss er auch ein Verständnis vom Betrieb haben, da er abschätzen muss, in wieweit seine Tests den Betrieb stören könnten. Neben dem bloßen Testen muss der Tester auch in der Lage sein diese nachvollziehbar zu dokumentieren.

Auch bei einem automatisierten Test muss ein Tester dabei sein, damit dieser eingreifen kann, sollten die Tests Probleme im Betrieb verursachen. Auch die Parameter eines Tests sollten von ihm überprüft und ggf. angepasst werden, damit potentielle Schwachstellen auch gründlich Untersucht werden können.

Entwickler

Der Entwickler ist ein Teil des **IT System Engineer** nach *SANS*. Um eine möglichst effiziente Behebung einer Schwachstelle zu gewährleisten, wurden administrativen Tätigkeiten des IT System Engineer in eine Rolle Administrator ausgelagert. Die Tätigkeiten bzgl. der Entwicklung von Software unterscheiden sich bei der Behebung von Schwachstellen von den Tätigkeiten eines Administrators. Teilweise sind Dienste auch nicht direkt vom Unternehmen entwickelt worden und es müssen externe Entwickler über Schwachstellen informiert werden, sodass diese behoben werden können. Die Softwareentwickler müssen somit nicht zum Unternehmen gehören. Sie benötigen Informationen über eine Schwachstelle, wenn diese aus einem Fehler im Programm resultiert. Diese Informationen sollten detailliert beschreiben, wie es zu dem Fehlverhalten des Dienstes kommt und was das Problem ist.

Der Softwareentwickler sollte ein Grundverständnis von Sicherheit haben und wissen, wie man sicher programmiert. Jedoch wird er nicht alle Schwachstellen kennen, daher sollten die Informationen zu den Schwachstellen auch Hinweise zur Schwachstelle und deren Behebung beinhalten.

Administrator

Neben dem Entwickler ist der Administrator ebenfalls eine Rolle die aus der Rolle **IT System Engineer** nach *SANS* entstand. Nachdem der Dienst Fehlerkorrekturen seitens der Softwareentwickler erhalten hat, typischerweise in Form eines Patches oder einer neuen Version, muss ein Administrator diese einspielen. Auch bei Schwachstellen, die aus einer Konfiguration stammen, muss der Administrator diese ändern oder erklären, warum diese so bleiben müssen. Er benötigt daher ebenso detaillierte Informationen welche Konfiguration zu ändern ist.

Sollte es nicht möglich sein, eine Konfiguration zu ändern oder einen Patch einzuspielen, zum Beispiel aufgrund einer Inkompatibilität mit einem anderen Dienst, hat der Administrator noch die Möglichkeit, andere Maßnahmen zu ergreifen. Dazu können Firewalländerungen, Änderungen im Logging, etc. zählen.

Ein Administrator sollte ein Verständnis von Sicherheit haben. Dazu gehören jedoch hauptsächlich Konfigurationsmöglichkeiten und deren Auswirkungen. Er muss dazu nicht alle Angriffe kennen, jedoch muss er in der Lage sein, diese in Grundzügen zu verstehen.

CSIRT

Für operative Sicherheitsaufgaben und im Speziellen zur Behandlung von Sicherheitsvorfällen gibt es meist ein spezielles Team, das sogenannte Computer Security Incident Response Team (CSIRT). Bei anderen Organisationen kann dies andere Namen haben, gebräuchliche Bezeichnungen sind etwa Security Operation Center

(SOC) oder Computer Emergency Response Team (CERT), auch wenn diese teils unterschiedliche Teilaufgaben haben. Im Falle einer Betriebsstörung mit Sicherheitszusammenhang wird diese Einheit eingebunden und soll zur Lösung beitragen. In *SANS* wird dieses Team zwar nicht als Rolle definiert, jedoch wird pauschal von der IT gesprochen, insbesondere die Einheiten für die Überwachsung von Logs. Diese zu informierenden Stellen und Personen wurden als Rolle CSIRT uminterpretiert.

Das CSIRT muss fundiertes Wissen zu Sicherheit besitzen. Es sollte Schwachstellen kennen und diese nachvollziehen können. Es kennt sich auch mit dem Betrieb aus und kann somit anderen Beteiligten bei der Findung von Gegenmaßnahmen helfen.

5.1.2. Sicherheitstestprozess

Nachdem nun die Rollen beschrieben wurden, soll im Folgenden auf die Prozesse eingegangen werden. Dabei wurden Teilprozesse definiert, die zur besseren Übersichtlichkeit separat beschrieben werden. In Abbildung 5.1 ist der Prozess als Diagramm abgebildet. Die Diagramme sind in Anlehnung an die Business Process Model and Notation (BPMN) erstellt worden.

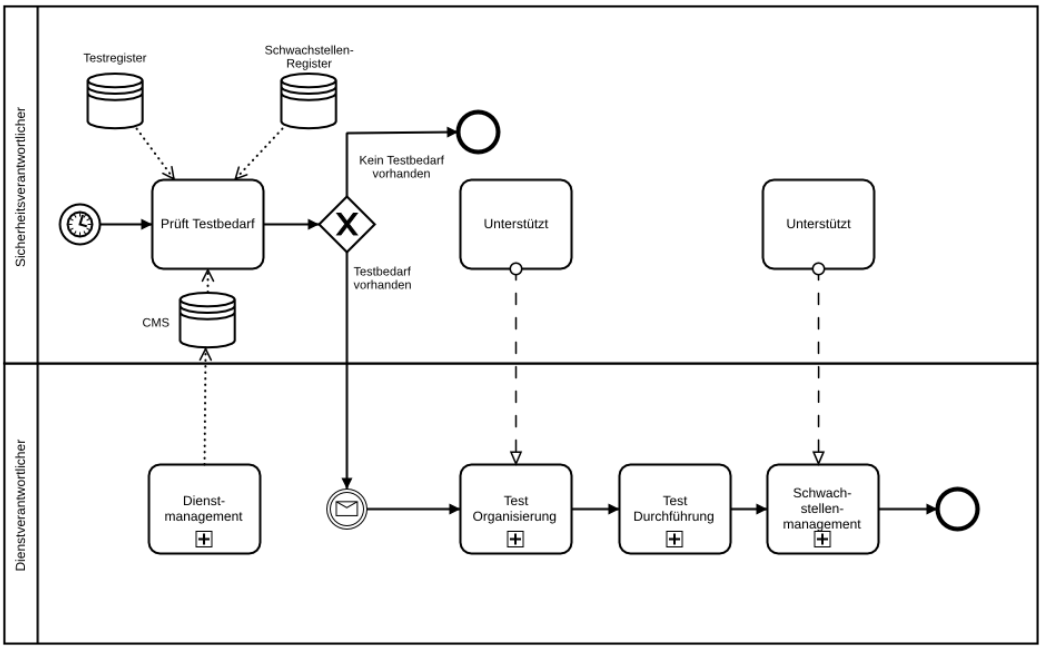


Abbildung 5.1.: Der Sicherheitstestprozess

Der Start des Prozesses ist wiederkehrend und kann bspw. wöchentlich, monatlich oder quartalsweise erfolgen. Der Sicherheitsverantwortliche erstellt dabei eine Auswertung mit Diensten, die überprüft werden sollen. Die Kriterien dazu können die Dauer seit dem letzten Test in Kombination mit dem Risiko des Dienstes sein, auch die Anzahl der Schwachstellen in vorherigen Tests kann eine Rolle spielen. Letztlich sind jedoch die Ressourcen, die für einen Test benötigt werden, der begrenzende Faktor. Ein intern durchgeführter Test benötigt bspw. Arbeitszeit, die nur begrenzt vorhanden ist. Ein extern durchgeführter Test benötigt zwar auch interne Ressourcen, jedoch weniger. Er benötigt allerdings mehr Geld, das auch nur begrenzt zur Verfügung steht. Die Aufgabe des Sicherheitsverantwortlichen besteht darin, die einzelnen Dienste so zu priorisieren, dass möglichst alle Dienste regelmäßig getestet werden. Die risikoreichen Dienste sollten nach Möglichkeit öfter getestet werden.

Erkennt der Sicherheitsverantwortliche einen Testbedarf bei einem oder mehreren Diensten und stehen ihm genügend Ressourcen zur Verfügung, diese Dienste zu testen, so informiert er die jeweiligen Dienstverantwortlichen, sodass diese einen Test planen und durchführen können. Während der Planung und der Durchführung soll der Sicherheitsverantwortliche beratend unterstützen, da er die nötige Fachkompetenz besitzt. Nachdem

der Test durchgeführt wurde, werden die gefundenen Schwachstellen im Schwachstellenmanagement behandelt.

5.1.3. Dienstmanagementprozess

Um es dem Sicherheitsverantwortlichen zu ermöglichen, den Testbedarf der Dienste zu erkennen, benötigt er ein Register aller Dienste. Um einen einheitlichen Datenbestand zu haben, sollte dieses Register auch zentral existieren. Dieses zentrale Register mit Diensten ist allerdings bei fast allen IT-Servicemanagement (ITSM) Prozessen gefordert. Da Hochschulrechenzentren Dienstleister für IT-Dienstleistungen sind, haben sie meist ein zumindest rudimentäres ITSM. Des Weiteren ist die Umsetzung eines ITSM in Universitäten nicht einfach und eine Beschreibung des Konzepts würde den Rahmen dieser Arbeit deutlich übersteigen [Kni12]. Daher soll im Folgenden ein existierendes Configuration Management System angenommen werden und dieses über eine Schnittstelle angebunden werden.

5.1.4. Sicherheitstest Planungsprozess

Nachdem der Sicherheitsverantwortliche den Dienstverantwortlichen auf den Testbedarf hingewiesen hat, oder der Dienstverantwortliche selbst einen Testbedarf festgestellt hat, muss dieser den Test organisieren (Abb 5.2).

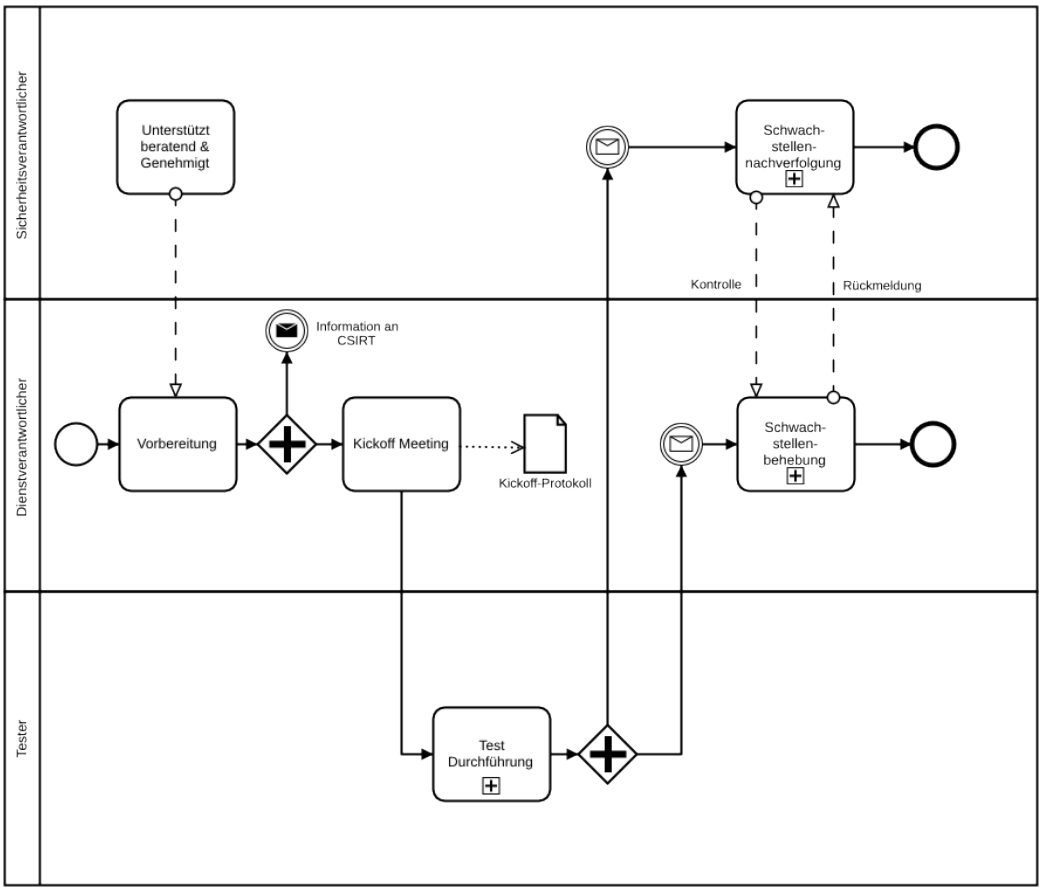


Abbildung 5.2.: Der Sicherheitstest Planungsprozess

Der Dienstverantwortliche muss dabei zunächst den Test und die Rahmenbedingungen planen. Dazu gehört es, den Dienst sowie die Schnittstellen, die getestet werden sollen, auszuwählen. Auch wie getestet werden soll (*vorsichtig* oder *aggresiv*) und welche Informationen der Tester erhalten soll, muss geplant werden (*White-Box* oder *Black-Box*). Neben den Rahmenbedingungen muss auch ein Zeitraum für den Test definiert werden. Bei

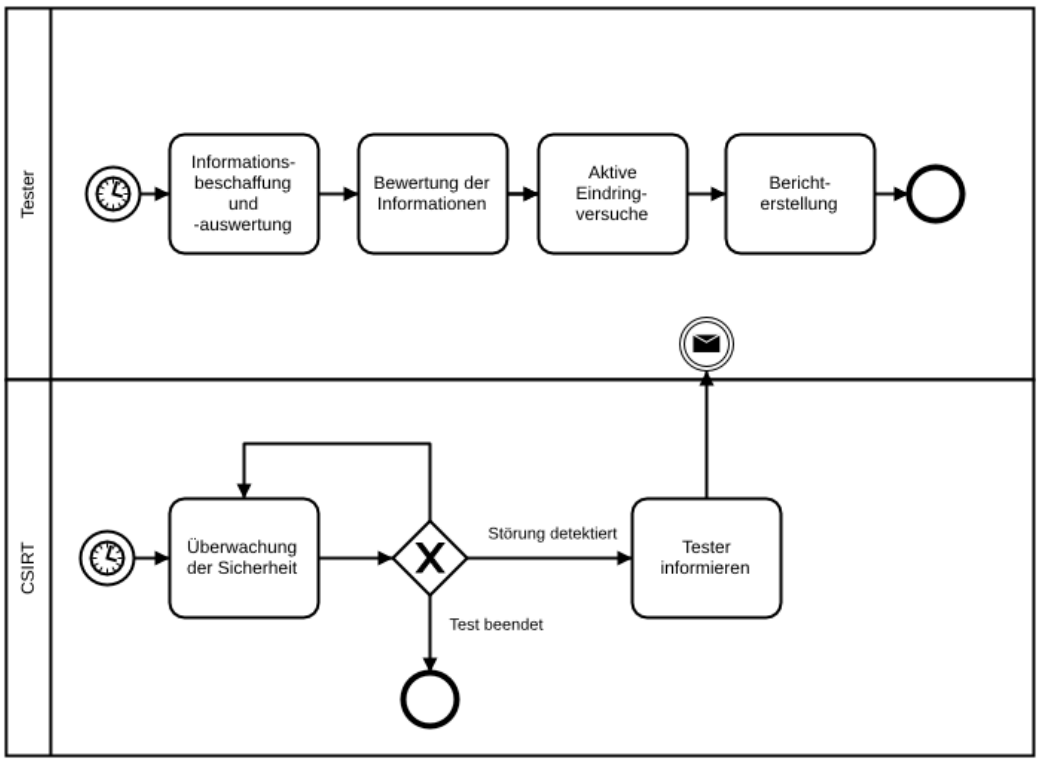


Abbildung 5.3.: Der Sicherheitstest Durchführungsprozess

der Planung soll der Sicherheitsverantwortliche unterstützend beraten, da dieser die nötige Fachkompetenz hat. Neben der Planung müssen auch die Genehmigungen für einen Test eingeholt werden. Dazu soll eine Genehmigung des Sicherheitsverantwortlichen eingeholt werden. Der Sicherheitsverantwortliche kann dann prüfen ob der Test sinnvoll geplant wurde und ob andere Tests im gleichen Zeitraum stattfinden, sodass diese Tests sich ggf. wechselseitig stören könnten. Auch das Risiko einer Betriebsstörung muss behandelt und ein Risikoträger gefunden werden, das kann ggf. die Unternehmensführung sein oder bei geringen Risiken der Dienstverantwortliche.

Nachdem die Planung durchgeführt wurde, werden in einem Kickoff Meeting die Testdetails mit dem Tester besprochen. Dabei sollen Informationen über die zu testenden Dienste besprochen und dokumentiert werden. Auch die Ansprechpartner während des Tests sollen definiert und festgehalten werden. Die IP-Adressen der Tester können ebenfalls dokumentiert werden, da diese bei Betriebsstörungen ein wichtiges Indiz sein können, ob der Test die Störung verursacht. Bei einem extern durchgeführten Sicherheitstest kann das Kickoff-Protokoll bspw. als Vertragsbestandteil dienen. Generell dient es als Dokumentation des Tests und anhand dieses soll man den Test auch reproduzieren können.

Nach dem Kickoff-Meeting kann mit dem Test begonnen werden. Die Testdurchführung ist in Abschnitt 5.1.5 näher beschrieben. Sobald der Test beendet ist, wird vom Tester ein Bericht erstellt, der dem Sicherheitsverantwortlichen und dem Dienstverantwortlichen übermittelt wird. Der Dienstverantwortliche muss nun versuchen, die Schwachstellen zu beheben und der Sicherheitsverantwortliche muss die Behebung überwachen. (Siehe Abschnitt 5.1.6)

5.1.5. Sicherheitstest Durchführungsprozess

Nachdem der Test geplant wurde und die Rahmenbedingungen im Kickoff-Meeting geklärt wurden, beginnt der Durchführungsprozess (Abb. 5.3) zum vereinbarten Zeitpunkt.

Zu Beginn prüft der Tester zum einen die übergebenen Informationen über einen Dienst und zum anderen

versucht er selbst Informationen über das Testziel zu erhalten, bspw. durch Scans. Im Anschluss wertet der Tester diese Informationen aus und untersucht die Informationen auf Schwachstellen. Nachdem er potentielle Schwachstellen gefunden hat, versucht er diese auszunutzen und zu bestätigen. Wenn der Test beendet wird, entweder da alles getestet wurde, oder da der Test das zeitliche Ende erreicht hat, erstellt der Tester einen Bericht mit den Ergebnissen des Tests. Der Bericht hat üblicherweise zuerst eine Zusammenfassung um die Testergebnisse kurz zusammenzufassen, nach der Zusammenfassung wird jede gefundene Schwachstelle näher beschrieben und Lösungsvorschläge werden genannt. Dieser Bericht soll dem Dienstverantwortlichen und dem Sicherheitsverantwortlichen zugehen.

Während des Tests überwacht das CSIRT wie im normalen Betrieb die Dienste. Sollte ein Problem auftreten, kann es mit Hilfe des Kickoff-Protokolls die Störung ggf. auf den Test zurückführen. Sollte dies der Fall sein, wird der im Kickoff definierte Ansprechpartner kontaktiert und die Testmethoden werden angepasst, sodass der Betrieb störungsfrei weitergeführt werden kann. Auch bei Störungen, die nicht vom Test verursacht wurden, sollte der Tester auf diese aufmerksam gemacht werden, da diese Störungen die Testergebnisse verfälschen könnten, bspw. durch längere Antwortzeiten.

5.1.6. Schwachstellenmanagementprozess

Nachdem der Sicherheitstest abgeschlossen ist und die Berichte übergeben wurden, muss mit den gefundenen Schwachstellen umgegangen werden. (Abb 5.4)

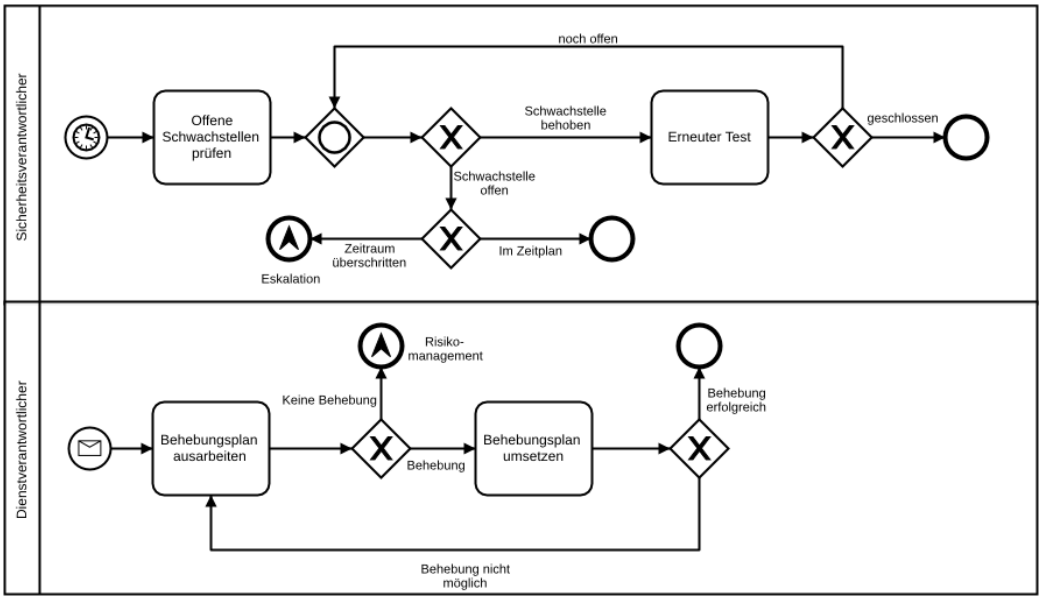


Abbildung 5.4.: Der Schwachstellenprozess

Dazu arbeitet der Dienstverantwortliche mit den Softwareentwicklern und Administratoren einen Behebungsplan aus. Sollten keine Behebungsmöglichkeiten existieren, z.B. da es keine Updates zu einer Software gibt und keine alternativen Produkte schnell eingesetzt werden können, kann sich dazu entschlossen werden, die Schwachstelle nicht zu beheben. Wenn dies geschieht, muss die Schwachstelle im Risikomanagement behandelt werden und die Unternehmensführung muss das Risiko tragen. Soll die Schwachstelle jedoch geschlossen werden, wird der Behebungsplan von den Entwicklern und Administratoren umgesetzt. War die Behebungsmaßnahme nicht erfolgreich, muss erneut ein Behebungsplan erstellt werden. Sofern die Schwachtstelle behoben wurde, wird der Sicherheitsverantwortliche darüber informiert.

Der Sicherheitsverantwortliche bekommt jeden Sicherheitstestbericht und hat somit einen Überblick über alle Schwachstellen. Er muss regelmäßig den Stand der offenen Schwachstellen prüfen. Der Dienstverantwortliche erstellt zu jeder Schwachstelle einen Behebungsplan, der auch einen Termin für die Behebung beinhaltet. Ist

die Behebung einer Schwachstelle noch in dieser Frist und entspricht die Frist den Vorgaben des Sicherheitsverantwortlichen, muss die Schwachstelle bei der nächsten Prüfung erneut geprüft werden. Wenn die Frist zur Umsetzung einer Behebungsmaßnahme nicht eingehalten wurde, muss dies zur Unternehmensführung eskaliert werden, da eventuell nicht genügend Ressourcen zur Behebung zur Verfügung stehen.

Hat der Dienstverantwortliche die Behebung einer Schwachstelle gemeldet, kann ein erneuter Test veranlasst werden, um dies zu prüfen. Sollte diese Prüfung erfolgreich sein, kann die Schwachstelle als behoben markiert werden. Sollte die Prüfung nicht erfolgreich sein, so muss der Dienstverantwortliche darüber informiert werden, um einen neuen Behebungsplan zu erstellen und der Schwachstellenstatus muss weiter regelmäßig geprüft werden.

5.2. Konzipierung einer Prozessunterstützenden Anwendung

Da die Prozesse jetzt geklärt sind, kann nun eine Anwendung geplant werden, die diese Prozesse unterstützen kann, um zum einen teilweise Tätigkeiten zu automatisieren und zum anderen manuelle Tätigkeiten zu unterstützen.

5.2.1. Anwendungsbereich

Wie bereits erwähnt, ist das Dienstmanagement in einem Hochschulrechenzentrum ein derartig wichtiges Instrument, sodass die Anwendung das Dienstmanagement nicht abbilden soll. Vielmehr soll das Dienstmanagement und Configuration Management System (CMS) über eine Schnittstelle angebunden werden. Somit ist es in der Anwendung möglich, die Dienste zu filtern und Tests für diese zu priorisieren. Die Pflege der Dienste und die Prozesse, um das CMS aktuell zu halten, müssen nicht abgebildet werden, was ggf. zu einer Dezentralisierung führen könnte.

Die Testplanung soll jedoch soweit wie sinnvoll abgebildet werden. Die Testdurchführung soll bei intern durchgeführten Sicherheitstests unterstützt werden, externe Dienstleister sollen keinen Zugriff auf die Anwendung erhalten. Gefundene Schwachstellen sollen jedoch auch bei externen Tests importierbar sein, um ein zentrales Schwachstellenregister zu ermöglichen. Auch die Berichtserstellung soll nach Möglichkeit von der Anwendung automatisiert durchgeführt werden, somit kann ein einheitliches Format für alle Tests sichergestellt werden und auf ein Rechenzentrum abgestimmte Texte verwendet werden.

Die Behebung der Schwachstellen soll ebenfalls nicht mit der Anwendung realisiert werden, da die Behebung sehr unstrukturiert sein kann. Handelt es sich um eine extern entwickelte Anwendung, müssen Entwickler des Herstellers eine Behebung realisieren. Bei Konfigurationsproblemen muss eventuell ein Administrator eine Konfiguration anpassen, etc. Aus diesem Grund soll die Behebung nur nachverfolgt werden.

5.2.2. Konzept einer prozessunterstützende Anwendung

Im Folgenden wird eine prozessunterstützende Anwendung konzipiert. Dabei werden die einzelnen Prozesse und die darin enthaltenen Aktivitäten auf Automatisierbarkeit überprüft und eine Automatisierung beschrieben.

Auf Testbedarf prüfen

Zu Beginn des Sicherheitstestprozesses prüft der Sicherheitsverantwortliche unter Benutzung eines Testregisters, des Configuration Management Systems und des Schwachstellenregisters, ob ein Dienst Bedarf für einen Sicherheitstest hat. (vgl. Abb. 5.1) Um diese Aktivität zu unterstüzen, soll es möglich sein, alle Dienste des Hochschulrechenzentrums in der Anwendung einsehen zu können und mit den Informationen aus einem Testregister und dem Schwachstellenregister zu verknüpfen. Es soll somit in einer Übersicht sofort einsehbar sein, welche Dienste existieren, wann jeder zuletzt getestet wurde und wieviele offene Schwachstellen dieser Dienst hat. Aus diesen Informationen kann der Sicherheitsverantwortliche dann Testkanidaten auswählen.

Da das CMS bzw. das Dienstmanagement ein zentraler Bestandteil anderer Prozesse und Managementsysteme ist, wird davon ausgegangen, dass ein CMS bereits existiert. Das muss keine vollwertige CMS-Anwendung sein, sonder kann auch eine tabellarische Auflistung in Form eines Textdokuments oder Excel-Dokuments sein. Dieses Dienstregister soll in die Anwendung importiert werden können. Dazu soll das Dateiformat Comma-Seperated-Values (CSV) genutzt werden. CSV-Dateien können sehr einfach erzeugt werden und sind sehr einfach maschinell auslesbar. Exportfunktionen sind für viele Programme (u.a. Microsoft Excel) bereits vorhanden, oder können sehr leicht implementiert werden.

Die geplanten und bereits durchgeführten Sicherheitstests sollen in einem Testregister hinterlegt werden. Da die Anwendung die Durchführung von solchen Tests unterstützen soll, ist es sinnvoll, das Testregister in die Anwendung zu integrieren. Sollte ein Dienst nicht mehr benötigt und somit abgeschaltet werden, ist dieser ggf. nicht mehr im CMS vorhanden. Für ein Testregister, welches auch Tests aus der Vergangenheit enthalten soll, ist es jedoch wichtig, auch abgeschaltete Dienste zu speichern. Daher sollen Dienste aus dem Dienstregister der Anwendung nicht gelöscht werden, sondern nur als abgeschaltet markiert werden. Abgeschaltete Dienste sollen nicht mehr in der Übersicht der Dienste angezeigt werden, da diese keine weiteren Tests benötigen. In einer Testübersicht hingegen sollen die Tests in Verbindung mit den Diensten, auch den abgeschalteten, angezeigt werden.

Auch das Schwachstellenregister soll direkt in die Anwendung integriert werden. Die Sicherheitstests dürften viele Schwachstellen aufdecken und durch die Integration des Schwachstellenregisters in die Anwendung ist es somit sehr leicht, die Schwachstellen direkt in das Schwachstellenregister zu übernehmen. Der Aufwand für die Pflege des Schwachstellenregisters ist somit sehr gering.

Hat der Sicherheitsverantwortliche einen Testbedarf festgestellt, so geht er auf den Dienstverantwortlichen zu und bittet diesen, einen Test zu planen und durchzuführen. Dieser Kontakt soll von der Anwendung nicht automatisiert werden. Der Kontaktweg soll vielfältig bleiben. Eine automatisch generierte E-Mail kann sehr leicht einmal vergessen werden, der Sicherheitsverantwortliche, der den Dienstverantwortlichen im besten Fall sogar kennt, kann somit einen auf den Dienstverantwortlichen zugeschnittenen Kontaktweg nutzen. Ein Dienstverantwortlicher, der Sicherheitstests sehr häufig durchführen lässt, kann anders aufgefordert werden, als jemand, der dies zum ersten mal tun soll.

Planung eines Sicherheitstests

Nachdem der Dienstverantwortliche auf den Testbedarf aufmerksam gemacht wurde oder diesen selbst erkannt hat, muss dieser einen Test planen. Bei der Planung und der Festlegung der Rahmenbedingungen des Tests soll die Anwendung unterstützen. So soll der Dienstverantwortliche den Testzeitraum und die Art des Tests (Informationsbasis, Aggressivität, etc.) sowie die mitzutestenden Dienste festlegen. Bei der Planung kann er den Sicherheitsverantwortlichen um Hilfe bitten, da dieser mehr Kenntnisse zu Sicherheitstests aufweist.

Nachdem ein Test geplant wurde, müssen noch Genehmigungen zur Durchführung eingeholt werden. Des Weiteren muss ein Risikoträger für eventuell auftretende Betriebsstörungen definiert werden. Dieser Teilprozess soll von der Anwendung nur soweit abgebildet werden, dass die Genehmiger festgehalten werden. Wer in einem Unternehmen solche Genehmigungen erteilen darf und wer Risikoträger sein kann, variert von Unternehmen zu Unternehmen. Auch die Prozesse zur Genehmigung können sehr varieren. So kann in einem Unternehmen bereits eine E-Mail als Genehmigung ausreichen, in einem anderen Unternehmen muss eine Genehmigung schriftlich erteilt werden. Daher soll nur festgehalten werden, wer den Test genehmigt hat, um zum einen sicherzustellen das nicht vergessen wird eine Genehmigung einzuholen und zum anderen den Genehmiger schneller ausfindig zu machen. Diese Genehmigung soll neben weiteren Absprachen mit dem Tester und anderen beteiligten Rollen (etwa CSIRT) in einem Kickoff-Protokoll festgehalten werden. Da das Kickoff-Protokoll als Dokumentation der Planung dienen soll, sollen dort auch die Rahmenbedingungen des Tests nochmals festgehalten werden. Dieses Kickoff-Protokoll kann somit auch als Vertragszusatz bei einem extern durchgeführten Sicherheitstest dienen. Um die Erstellung des Kickoff-Protokolls zu vereinfachen und ein einheitliches Format sicherzustellen, soll dieses Protokoll von der Anwendung vorausgefüllt und dem Dienstverantwortlichen zur Verfügung gestellt werden. Nachdem das Protokoll vervollständigt wurde, soll es in der Anwendung wieder hinterlegt werden, um einen schnellen Zugriff darauf zu ermöglichen.

Durchführung eines Sicherheitstests

Nachdem der Test geplant wurde, alle Genehmigungen eingholt wurden und alles in einem Kickoff-Protokoll festgehalten wurde, kann der Test zum vereinbarten Zeitpunkt starten. Bei einem intern durchgeführten Test soll der Tester bei der Durchführung eines Sicherheitstests unterstützt werden.

Dazu gehört eine Anbindung an häufig genutzte Sicherheitstest-Programme und Schwachstellenscanner. Dabei soll die Anwendung Informationen über die Scanner bereitstellen und die Konfiguration des Scanners soweit wie möglich vorbereiten. Das können Aggressivitätsparameter, die sich aus den Rahmenbedingungen ableiten, sein oder auch IP-Adressen der Testziele. Der Tester kann diese aus der Anwendung kopieren und weiter anpassen. Die tatsächlich verwendete Konfiguration sowie die Ausgabe des Scanners soll wieder in der Anwendung hinterlegt werden, um den Test wiederholbar zu machen und um die Testdurchführung zu dokumentieren. Die Ausgabe der Scanner soll auch von der Anwendung überprüft werden um ggf. automatisiert Schwachstellendokumentationen erzeugen zu können.

Die vom Scanner gefundenen Schwachstellen sowie manuell gefundene Schwachstellen sollen ebenfalls in der Anwendung dokumentiert werden. Wenn der Test beendet wurde, soll der Tester noch eine Zusammenfassung der Testergebnisse verfassen, die ebenfalls in der Anwendung gespeichert werden soll. Die Anwendung soll nach der Beendigung des Tests automatisch Berichte für die verschiedenen Rollen generieren.

Für die Generierung der Berichte sollen für die verschiedenen Schwachstellenkategorien Hinweise für die Rollen hinterlegt werden. Zum einen soll eine Beschreibung der Schwachstellenkategorie hinterlegt werden, um dem Leser erste Informationen zu liefern und zum anderen sollen Behebungshinweise für die Kategorien gespeichert werden, um den Lesern erste Tipps bzgl. der Behebung zu geben. Die Berichte sollen sich wie folgt zusammensetzen:

- **Zusammenfassung der Testergebnisse** für einen ersten Überblick über den Test und dessen Ergebnisse.
- **Auflistung der gefundenen Schwachstellen** um einen Überblick über die gefundenen Schwachstellen und deren Schwere zu geben.
- **Beschreibung der einzelnen Schwachstellen** um einzelne Schwachstellen detailiert zu beschreiben und dem Leser Details über die Schwachstelle zu geben, sodass dieser Behebungsmaßnahmen entwickeln kann.

Sollte ein Sicherheitstest extern durchgeführt werden, sollen die Berichte trotzdem einheitlich bleiben. Da der externe Tester keinen Zugang zu der Anwendung haben soll, soll daher zu Beginn des Tests vereinbart werden, dass die gefundenen Schwachstellen in einem maschinenlesbaren Format übermittelt werden müssen, sodass die Anwendung diese Schwachstellen ohne großen manuellen Aufwand importieren kann. Als Format soll abermals CSV verwendet werden, da dieses sehr leicht zu erzeugen ist und das Format sehr bekannt ist.

Das CSIRT soll bei einer Betriebsstörung schnell einen Sicherheitstest als Ursache bestätigen oder ausschließen. Daher soll das CSIRT zum einen über die Durchführung benachrichtigt werden, zum anderen soll eine Anzeige der aktuell in Durchführung befindlichen Tests sowie der geplanten Tests zur Verfügung gestellt werden. Das CSIRT kann somit bei einer Betriebsstörung alle laufenden Tests schnell überprüfen und mittels Informationen aus dem Kickoff-Protokoll den Tester informieren.

Schwachstellenmanagement

Nachdem ein Sicherheitstest durchgeführt wurde und Schwachstellen bekannt geworden sind, sollten diese behoben werden. Dazu sollen zum einen die Berichte an den Sicherheitsverantwortlichen, den Dienstverantwortlichen, die Entwickler und Administratoren verteilt werden. Der Sicherheitsverantwortliche und der Dienstverantwortliche sollen die Möglichkeit haben, diese Berichte direkt aus der Anwendung zu beziehen. Die Entwickler und Administratoren haben jedoch keinen Zugang zur Anwendung. Der Versand an diese ist jedoch nicht einfach automatisierbar, da diese ggf. nicht im CMS hinterlegt sind. Daher sollen diese Berichte manuell vom Dienstverantwortlichen verteilt werden.

Neben den Berichten soll es auch möglich sein, Ticketing Systeme zur Nachverfolgung zu nutzen. Im Rahmen eines IT Servicemanagements existiert häufig ein Problem- und Incident-Management. Dabei wird zur Nach-

verfolgung von Problemen oder Änderungswünschen ein Ticketing System verwendet. Sollte so ein Ticketing System vorhanden sein, soll die Anwendung automatisch Tickets für die Schwachstellen erzeugen und die Referenznummer speichern. Somit kann für eine Schwachstelle schnell der Behebungsstand eingesehen werden, da über die Referenznummer das Ticket im Ticketing System aufzufinden ist. Die Planung und Umsetzung von Behebungsplänen soll weiter nicht abgebildet werden, da diese sehr unstrukturiert sein können.

Zur Schwachstellennachverfolgung soll dem Sicherheitsverantwortlichen eine Übersicht über alle Schwachstellen ermöglicht werden. Neben den Details zu einer Schwachstelle soll auch ein Status hinterlegt werden. So soll eine Schwachstelle als behoben oder gemanaget markiert werden können, um nur nach offenen Schwachstellen suchen zu können.

6. Implementierung

Das soeben beschriebene Konzept für eine prozessunterstützende Anwendung wurde prototypisch implementiert. Die Implementierung wird im Folgenden näher beschrieben. Dabei wird zunächst auf die Architektur und die verwendeten Frameworks eingegangen und anschließend für einige Prozessschritte die Implementierung beispielhaft beschrieben.

6.1. Architektur

Aufgrund der vielen Nutzer, insbesondere der Dienstverantwortlichen, sollte die Anwendung möglichst wenig Anforderungen an die Nutzer stellen. Die Anwendung sollte daher nicht lokal bei den Dienstverantwortlichen installiert werden müssen. Da eine gemeinsame Datenbasis notwendig ist, wurde entschieden, eine Webanwendung zu entwickeln.

Als Design-Framework wurde sich für Bootstrap entschieden. Bootstrap wurde von Twitter entwickelt und dient als Designframework für Webanwendungen. Es ist sehr weit verbreitet, bietet viele Möglichkeiten und ist zudem auch responsive, also auch für die Anzeige auf Tablets und Smartphones ausgelegt.

Als Programmiersprache wurde sich für Python in der aktuellen Version 3 entschieden. Python ist eine einfach zu erlernende Sprache mit einem relativ leichtverständlichem Quellcode. Es ist eine sogenannte interpretierte Sprache, muss also von einem Interpreter zur Laufzeit kompiliert werden. Es wird daher auch oft als Skriptsprache bezeichnet. Python ist aufgrund dieser Eigenschaften sehr verbreitet, gerade im Sicherheitsumfeld. Zahlreiche Projekte verwenden Python auch wegen der Möglichkeit, C und C++ Programme anzubinden. Es ermöglicht daher sehr einfach eine Integration in andere existierende Programmlandschaften.

Um Python-Programme als Webanwendung zu integrieren existieren neben einer CGI[1]-Schnittstelle auch einige Frameworks, die es erlauben mit Python Webanwendungen zu schreiben, Django und Flask sind zwei Beispiele. Zu Beginn wurden beide Frameworks etwas näher betrachtet. Flask ist ein sehr leichtgewichtiges Framework und ermöglicht somit eine schnelle Einarbeitung in den Quellcode. Django ist zwar sehr mächtig, benötigt allerdings mehr Einarbeitungszeit als Flask. Da Flask zur Umsetzung völlig ausreichend war, wurde sich daher für Flask entschieden.

6.1.1. Flask

Flask beschreibt sich selbst als Microframework [FLA]. Es ist einfach über den Python Paketmanager (PIP) zu installieren. Mittels eingebautem minimalem Webserver kann man sehr einfach die Webanwendung während der Entwicklung testen.

Flask enthält auch eine HTML-Template Engine, die es ermöglicht, HTML-Dateien mit eine Art Platzhaltern zu verwenden. Flask lädt diese Templates und befüllt die Platzhalter anschließend mit Daten. Eine Trennung von Logik und Darstellung ist somit fast durchgehend gegeben und die Darstellung könnte leicht getauscht werden ohne große Änderungen an der Logik durchführen zu müssen.

Um Mehrsprachigkeit zu ermöglichen wurde noch eine Erweiterung von Flask verwendet: flask-babel. Damit ist es möglich, Texte die dem Benutzer ausgegeben werden im Nachhinein zu sammeln und Übersetzungen für diese Texte zu erstellen. Nachdem die Übersetzungen erstellt wurden, können diese in ein Babel-Format kompiliert werden und anschließend wird je nach Sprache des Nutzers die dementsprechende Sprache von

[1] Common Gateway Interface

Flask verwendet. Sollte für die Benutzersprache keine Übersetzung vorliegen, wird auf den Standard-Text, der im Quellcode verwendet wurde, zurückgegriffen.

6.1.2. SQLAlchemy

Für die Datenhaltung wurde sich für eine Datenbank entschieden, diese ermöglicht ein schnelles Lesen und Verarbeiten von Daten. Für die Anbindung von Python zu einer Datenbank wurde sich für SQLAlchemy entschieden. SQLAlchemy ist ein Object Relations (OR) Mapper der es zum einen ermöglicht, aus Klassendefinitionen direkt SQL-Schemas abzuleiten und zum anderen Datensätze aus der Datenbank direkt als Klasse wiederzugeben. SQLAlchemy ist wohl der am weitesten verbreitete OR-Mapper für Python und besitzt zahlreiche Schnittstellen zu unterschiedlichen Datenbanksystemen. Somit ist es einfach möglich, bei der Entwicklung eine Datei-Datenbank etwa SQlite zu nutzen, den Datenbanktyp aber für den produktiven Einsatz mit Hilfe einer Konfigurationsänderung bspw. zu einer mySQL Datenbank zu ändern.

6.1.3. odfpy

Für die Generierung des Kickoff-Protokolls und der Berichte wurde *odfpy* verwendet. Dieses Python-Modul ermöglicht es, Dokumente im Open Document Format (ODF) zu lesen, zu bearbeiten und zu generieren. ODF ist ein international genormter Standard, der zudem quelloffen ist. Er wird seit langem von OpenOffice und Derivaten verwendet. ODF ist vergleichbar mit den Formaten von Microsoft Office. Die Spezifikationen der Office-Formate war im Gegensatz zu ODF zunächst nicht öffentlich zugänglich. Aufgrund der Quelloffenheit ist ODF in einigen Ländern (teilweise auch in Deutschland [Rai08]) als Dateiformat in Behörden verpflichtend.

ODF basiert auf einer XML-Struktur und odfpy kann diese lesen und interpretieren. Wie sich jedoch herausgestellt hat, ist die Dokumentation dieses Moduls äußerst knapp. Jedoch war es vom Funktionsumfang den Modulen zum Lesen und Schreiben von Microsoft-Office Dokumenten weit überlegen.

6.2. Testbedarf prüfen

Nachdem nun die Architektur und die verwendeten Frameworks kurz erläutert wurden, wird im Folgenden die Implementierung der einzelnen Aktivitäten bzw. Teilprozesse erläutert.

6.2.1. Import der Dienste aus einem CMS

Um möglichst generisch in der Lage zu sein, Daten aus einem Configuration Management Systems (CMS) zu importieren, sollen die Daten zunächst vom CMS als CSV exportiert werden. Für den Import wurde ein kleines Programm geschrieben, um diese CSV-Dateien zu importieren. Dieses Programm kann an andere CSV-Formate angepasst werden. Im Folgenden wird das implementierte Format näher beschrieben. Dazu soll zunächst auf die verwendeten CSV-Dateien bzw. deren Struktur eingegangen werden. Dabei ist die Struktur der CSV-Datei fiktiv.

Format des CMS-CSV-Exports

Für den Import aus dem CMS wurden drei verschiedene CSV-Dateien verwendet. Eine **cmdb.csv** mit der Auflistung aller Dienste und Details zu diesen, eine **cmdb_ips.csv** mit IP-Adressen und der Zuordnung zu den Diensten und eine **cmdb_service_relationships.csv** für die Abhängigkeiten zwischen den Diensten.

Die **cmdb.csv** besteht aus den folgenden Feldern:

1. **CMS-ID**, der eindeutige Identifikator des CMS, um bei erneuten Imports die Dienste eindeutig zuordnen zu können.
2. **Dienstname** für den Namen bzw. die Bezeichnung eines Dienstes.

3. **E-Mail des Dienstverantwortlicher** für ein eindeutiges Zuordnungsmerkmal für den Dienstverantwortlichen.
4. **Name des Dienstverantwortlichen**.
5. **Vorname des Dienstverantwortlichen**.
6. **Abteilung des Dienstverantwortlichen**, um im Falle einer Abwesenheit Vertretungen ermitteln zu können.
7. **Beschreibung des Dienstes**, falls die Bezeichnung des Dienstes nicht eindeutig ist und weitere Informationen nötig sind.
8. **Risikoscore** als Repräsentierung des Risikos für einen Dienst.
9. **Ticketing-Adresse** als Adresse für Schwachstellen-Tickets

Die Informationen über die Dienstverantwortlichen können redundant auftreten, eine Auslagerung in eine weitere CSV-Datei wäre zwar möglich, würde allerdings zu mehr Komplexität führen. Daher wurde darauf verzichtet. Durch die Verwendung eines dedizierten Programms zum Import dieser Informationen wäre es mit wenig Aufwand möglich, diese Informationen auszulagern und separat zu importieren.

Die CSV-Datei **cmdb_ips.csv** besteht nur aus zwei Feldern, zum einen die **CMS-ID** für die Zuordnung zum Dienst und zum anderen aus der **IP-Adresse**.

Die CSV-Datei **cmdb_service_relationships.csv** besteht ebenfalls aus nur zwei Feldern, der **CMS-ID** des Dienstes mit der **CMS-ID** des Dienstes der über eine Schnittstelle angebunden ist. Dabei werden nur Hierarchien über eine Ebene abgebildet.

Programm zum importieren des CMS-CSV-Exports

Für das Importieren der drei CSV-Dateien wurden drei Programme geschrieben, die jede Datei einzeln importieren. Diese sind gleich aufgebaut und nehmen jeweils einen Kommandozeilenparameter an, der die zu importierende CSV-Datei ist. Die Programme iterieren dabei über die Zeilen der CSV-Dateien und rufen pro Zeile die Funktion *import_csv_row* auf.

Die Funktion *import_csv_row* erhält dabei zwei Parameter, zum einen die Datenbank-Session und zum anderen die Zeile als Liste. Die Funktion liest die Datenfelder der Zeile ein und behandelt diese, indem sie z.B. ein Attribut setzt. Bei der Ausführung der Programme ist es wichtig, dass zunächst die *cmdb.csv* importiert wird, da diese alle Dienste enthält und die anderen Dateien Verweise auf neue Dienste enthalten könnten, die noch nicht in der Datenbank sind.

Für einen regelmäßigen Import aus einem CMS sollten die Export-CSV-Dateien der Anwendung zugänglich gemacht werden, bspw. mittels Dateitransfer. Die Import-Programme können dann entweder mittels regelmäßiger Ausführung (cronjob) oder manuell (ssh) ausgeführt werden.

6.2.2. Ansicht der Dienste

Nachdem die Dienste erfolgreich importiert wurden, hat der Sicherheitsverantwortliche die Möglichkeit, eine Übersicht der Dienste einzusehen (Abb. 6.1).

Dabei sieht er eine tabellarische Auflistung der Dienste mit den Informationen zum Risiko, dem letzten Test auf den Dienst und die Anzahl der offenen Schwachstellen. Der Name des Dienstes verlinkt dabei auf eine Seite mit den Details zu einem Dienst, das Datum des letzten Tests auf den entsprechenden Test.

Die Detailansicht eines Dienstes besteht aus vier Reitern. Die **Dienstdetails** zeigen die Informationen aus dem CMS an und bieten die Möglichkeit an einen neuen Test für diesen Dienst zu planen. Der Reiter **Schnittstellen** zeigt die Schnittstellen zu anderen Diensten auf, um eine Übersicht über die Komplexität zu erhalten. Der Reiter **Sicherheitstests** listet alle Tests auf, bei denen der Dienst getestet wurde. Sollte der Dienst im Rahmen eines Tests auf einen anderen Dienst mit getestet worden sein, ist dieser Test auch dort aufgelistet.

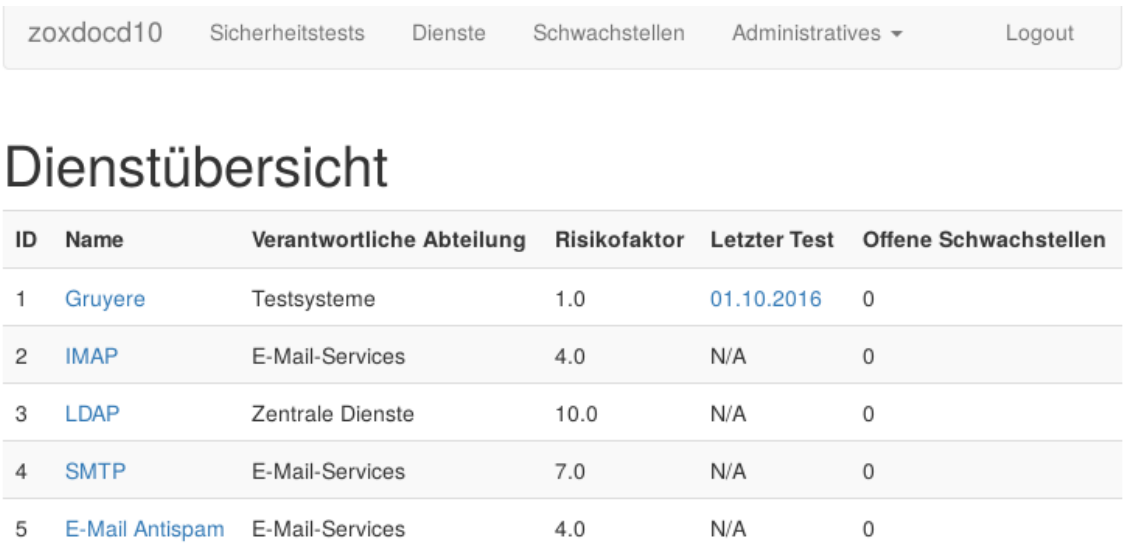


Abbildung 6.1.: Screenshot der Übersicht über die Dienste

Die Darstellung erfolgt mittels Tabelle. (Abb. 6.2) Die Testbezeichnung wird aus den Rahmenbedingungen des Tests generiert und soll eine schnelle Übersicht über diese Rahmenbedingungen geben. Der Reiter **Offene Schwachstellen** zeigt eine tabellarische Ansicht der offenen und gemanagten Schwachstellen. Über den Titel der Schwachstelle ist die Detailansicht der Schwachstelle verlinkt.

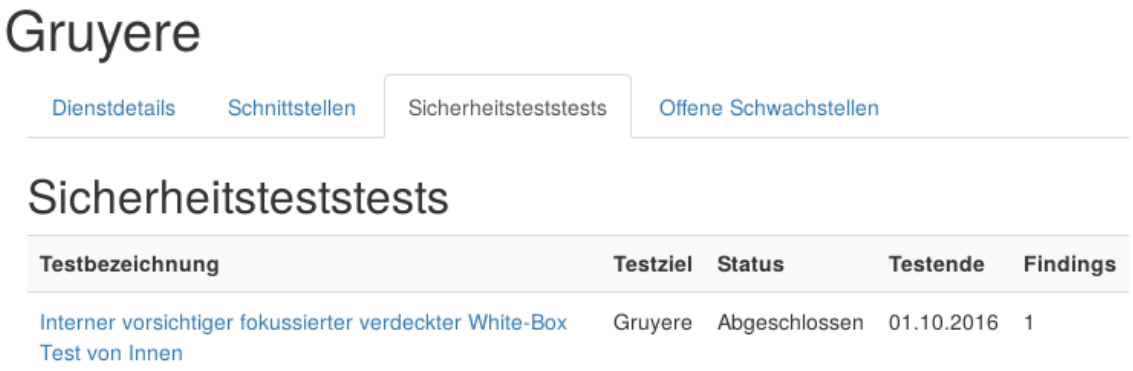


Abbildung 6.2.: Screenshot der Detailansicht eines Dienstes

6.3. Planung eines Sicherheitstests

Sollte der Sicherheitsverantwortliche Testbedarf bei einem Dienst feststellen, so muss der Dienstverantwortliche einen Test planen. Der Dienstverantwortliche hat dabei ebenfalls eine Übersicht über die Dienste, jedoch sieht er nur die Dienste, für die er selbst verantwortlich ist. Er kann dann aus den Dienstdetails über einen Button einen neuen Test für diesen Dienst planen.

6.3.1. Planungsansicht

In der Planungsansicht existieren abermals vier Reiter. Der Reiter **Dienstdetails** listet nochmals einige Informationen zu dem Dienst, der getestet werden soll, auf, um dem Dienstverantwortlichen zu zeigen, für welchen

Dienst er gerade einen Test plant. Über einen Button lässt sich auch ein Assistent starten.

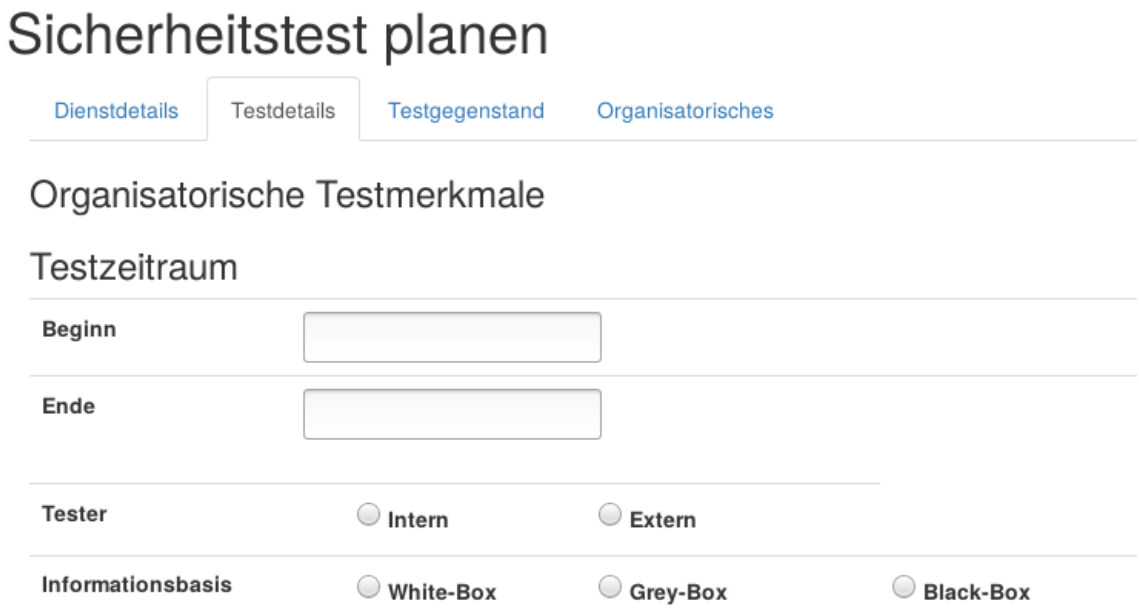


Abbildung 6.3.: Screenshot der Planungsansicht eines Sicherheitstests (Reiter Testdetails)

Der Reiter **Testdetails** (Abb. 6.3) zeigt die zu definierenden Rahmenbedingungen des Tests an. So kann zum einen der Testzeitraum definiert werden und zum anderen können die Testmerkmale ausgewählt werden. (vgl. Abschnitt 2.5.1)

Merkmal	mögliche Ausprägung		
Informationsbasis	White-Box	Grey-Box	Black-Box
Aggressivität	vorsichtig	abwägend	aggressiv
Umfang	fokussiert	begrenzt	vollständig
Vorgehensweise	offensichtlich	verdeckt	
Ausgangspunkt	von innen	von außen	

Tabelle 6.1.: Testmerkmale und deren mögliche Werte

Im Reiter **Testgegenstand** können die mit zu testenden Dienste und die dahinterstehenden IP-Adressen ausgewählt werden. Dabei werden nur die IP-Adressen der ausgewählten Dienste angezeigt um die Ansicht übersichtlich zu halten (Abb. 6.4). Die mit zu testenden Dienste sind die Schnittstellen, die aus dem CMS-Import stammen. Dabei werden die Schnittstellen rekursiv aufgelistet.

Im Reiter **Organisatorisches** kann die Planung gelöscht werden, falls der Testbedarf nicht mehr besteht, die Planung zwischengespeichert werden, falls der Dienstverantwortliche z.B. Fragen an den Sicherheitsverantwortlichen hat und diese Fragen noch klären möchte und es kann die Planung abgeschlossen werden. Bei Abschluss der Planung wird in die Kickoff-Phase übergegangen und ein nachträgliches Editieren des Tests ist nicht mehr möglich. Dies ermöglicht es dem CSIRT und dem Sicherheitsverantwortlichen sich auf den Test einzustellen, ohne regelmäßig die Details überprüfen zu müssen, da diese sich ständig ändern können.

6.3.2. Kickoff-Phase

In der Kickoff-Phase muss das Kickoff-Protokoll in einem Kickoff-Meeting ausgefüllt werden und die Informationen an den Tester übergeben werden. Dazu bietet die Anwendung unter einem neuen Reiter **Kickoff** das Herunterladen der Vorlage des Kickoff-Protokolls an.

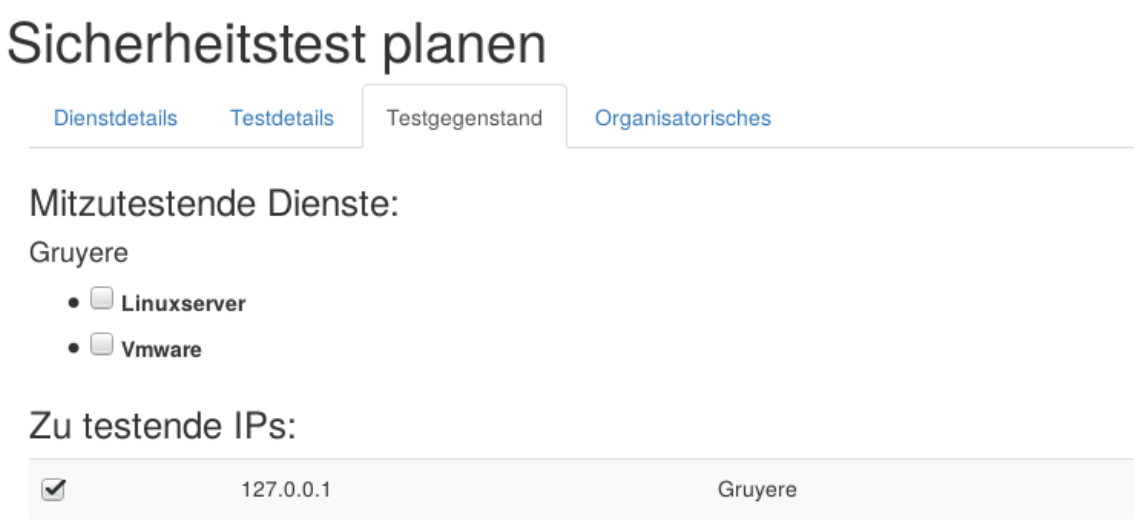


Abbildung 6.4.: Screenshot der Planungsansicht eines Sicherheitstests (Reiter Testgegenstand)

Das Kickoff-Protokoll

Für das Generieren des Protokolls wird eine Vorlage genutzt. Die Vorlage kann bspw. mit LibreOffice erstellt und so an das Design im Unternehmen angepasst werden. In der Vorlage besteht die Möglichkeit, sogenannte *TextInputs* zu verwenden. Dies sind Formularfelder. Beim Herunterladen der Vorlage für das Kickoff-Protokoll werden diese Formularfelder dann von der Anwendung vorgefüllt. Dabei wird eine Funktion *create_template* verwendet. Sollten neue Formularfelder hinzukommen oder bestehende wegfallen, so muss dies in dieser Funktion implementiert werden. (Listing 6.1)

Listing 6.1: Erstellung des Kickoff-Protokolls

```
from odf.opendocument import load
from odf.element import Text
from odf.text import TextInput

def create_template(securitytest):
    """create␣the␣kickoff␣template,␣return␣file␣as␣bytes"""
    # the buffer for the result
    result_bytes = BytesIO()
    # get the path to the template
    config = configparser.ConfigParser()
    config.read('zoxdocd10.conf')
    # load template
    kickoff_doc = load(
        config.get("kickoff", "template")
    )
    # iterate over all the TextInputs
    for input_field in kickoff_doc.getElementsByType(TextInput):
        # this is imho the best way to get the name of the field
        input_field_name = str(input_field)
        try:
            # get the value for this TextInput
            field_text = FUNCTIONMAPPER[input_field_name](securitytest)
            input_field.childNodes = [Text(field_text)]
        except KeyError:
            # a TextInput we can't fill, doesn't matter, perhaps a human workflow
            pass
    # write the document in the buffer
    kickoff_doc.write(result_bytes)
```

```
    #return as bytes
    return result_bytes.getvalue()
```

Dabei wird zunächst aus der Konfigurationsdatei der Pfad zum Template ausgelesen und von odfpy eingelesen. Dann wird über alle TextInputs iteriert und der Name ausgelesen. Durch Zufall wurde entdeckt, dass die *str* Funktion den Feldnamen wiedergibt. Eine genauere Untersuchung der TextInput-Objekte ergab zwar, dass der Feldnamen auch in einem Attribut des Objekts zu finden war, allerdings war es sehr aufwändig, dieses auszulesen, daher wurde die *str* Funktion genutzt.

Die Konstante *FUNCTIONMAPPER* ist ein Dictonary, bei dem der Key der Feldname und der Wert ein Verweis auf eine Funktion ist. Die Input-Felder werden nun mit dem Rückgabewert der entsprechenden Funktion gefüllt, wobei den Funktionen jeweils das Objekt *securitytest* übergeben wird. Dort sind alle Informationen zum Test enthalten und können ausgelesen und aufbereitet werden. Das Konstrukt des *FUNCTIONMAPPER* macht es sehr einfach, neue Felder im Template zu hinterlegen und bestehende oder neue Funktionen darauf zu schlüsseln.

Nachdem die Vorlage ausgefüllt wurde, kann es in der Anwendung hochgeladen werden. Erst nach dem Hochladen ist ein Übergang in die Durchführungsphase möglich.

Das Schwachstellen-Sheet

Sollte der Sicherheitstest von einem externen Dienstleister durchgeführt werden, sollte dieser die gefundenen Schwachstellen in einer CSV-Datei übermitteln, um einen Import in die Anwendung zu ermöglichen. Da bei einer Schwachstelle jedoch auch die Schwachstellenkategorie sowie der betroffene Dienst hinterlegt werden soll, wäre es bei CSV-Dateien schwierig sicherzustellen, dass der Dienstleister exakt die Kategorien und Dienstbezeichnungen verwendet wie die Anwendung. Daher wurde beschlossen, analog zum Kickoff-Protokoll eine Vorlage für ein Tabellenkalkulationsprogramm (Microsoft Excel oder LibreOffice Calc) bereitzustellen.

Dabei wird eine Vorlage die analog zum Kickoff-Protokoll erstellt werden kann, in der Anwendung hinterlegt. Dabei sind für die Spalten *Kategorie* und *Dienst* Wertebereiche festgelegt. Diese Wertebereiche sind ein Verweis auf ein weiteres Tabellenblatt. Diese Tabellenblätter werden dann beim Herunterladen der Vorlage mit den aktuell hinterlegten Kategorien und den zu testenden Diensten hinterlegt. Für das Füllen der Vorlage wurde wieder odfpy verwendet. Der Dienstleister kann diese Vorlage in einem Tabellenkalkulationsprogramm seiner Wahl bearbeiten, ein versehentliches Verschreiben bei der Kategorie oder dem betroffenen Dienst ist somit nicht mehr möglich.

6.4. Die Testdurchführung

Wurde das Kickoff-Protokoll ausgefüllt und wieder in die Anwendung hochgeladen, kann zur Testdurchführung übergegangen werden. Die Testdurchführung erfolgt nun vom Tester. Die Ansicht wird dabei von der Planung übernommen und es werden zwei Reiter ergänzt.

6.4.1. Schwachstellen Dokumentation

Der Reiter **Schwachstellen** ermöglicht es dem Tester, neue gefundene Schwachstellen dem Test hinzuzufügen und diese näher zu dokumentieren. (Abb. 6.5)

Zur Dokumentation einer Schwachstelle sind die folgenden Felder vorgesehen:

- **Titel** für eine kurze Bezeichnung der Schwachstelle.
- **Dienst** für den von der Schwachstelle betroffenen Dienst.
- **Beschreibung** für die Beschreibung der Schwachstelle.
- **Beweis** für einen Screenshot und Text, die beweisen sollen, dass die Schwachstelle existiert.

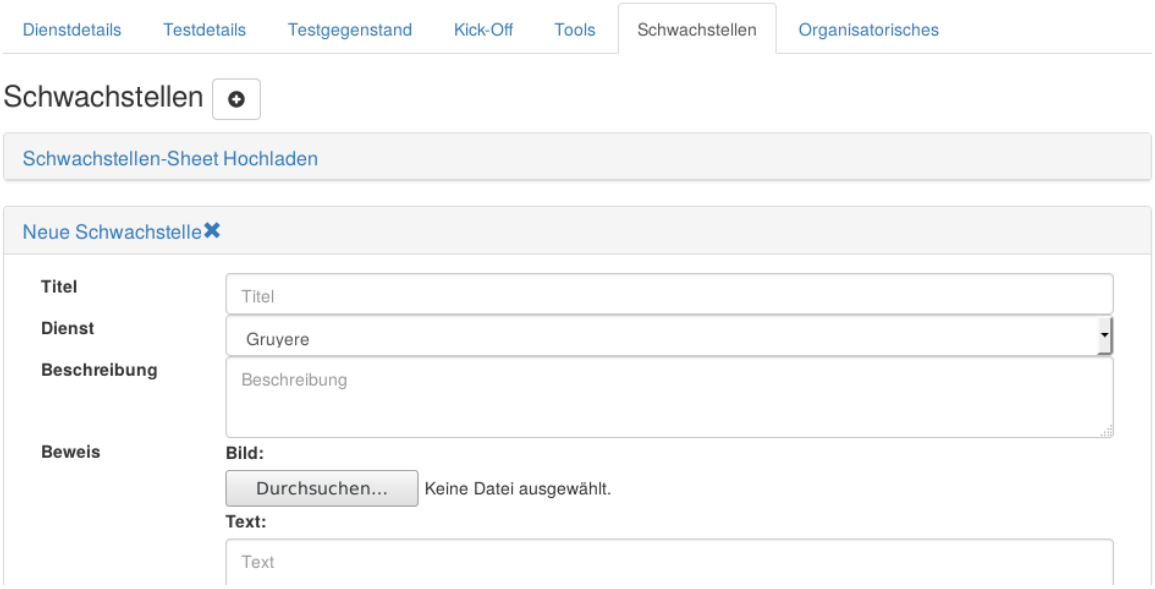


Abbildung 6.5.: Screenshot der Durchführung eines Sicherheitstests (Reiter Schwachstellen)

- **Kategorie** für die Kategorie der Schwachstelle.
- **Empfehlung** für die Empfehlung des Testers zur Behebung der Schwachstelle.
- **CVSS** für den CVSS3-Basis Score als Kritikalität der Schwachstelle.

Falls der Tester die Dokumentation der Schwachstellen nicht in der Anwendung vornehmen will oder es sich um einen extern durchgeführten Test handelt, so kann auch das Schwachstellen-Sheet hier hochgeladen werden. Die Schwachstellen werden dann importiert und angezeigt. Eine Bearbeitung der Schwachstellen ist nach dem Import weiter möglich.

6.4.2. Anbindung automatisierter Scanner

Ein weiterer Reiter ist der Reiter **Tools**. Dabei sind häufig verwendete Sicherheitsprogramme hinterlegt, die bei einem Sicherheitstest hilfreich sein können.

Um es möglich zu machen, schnell neue Scanner hinzuzufügen, wurde darauf verzichtet, die Scanner in der Datenbank abzubilden. Dennoch war es notwendig, verschiedene Informationen, z.B: Name und Beschreibung eines Scanners, zu speichern und auslesen zu können. Dazu wurde ein eigenes Untermodul konzipiert mit dem Namen *security_scanner*. Dort wurde eine Klasse *Scanner* angelegt, von der in Zukunft alle Scanner erben sollen. (Siehe Listing 6.2)

Listing 6.2: Die Scanner Klasse

```
class Scanner(object):
    """the scanner class"""
    name = "Name"
    description = "Beschreibung"
    documentation_url = "URL"
    # eg generate via python -c "import uuid; print(uuid.uuid4())"
    uuid = ""

    def get_prio(self, securitytest):
        """return a int, how important this scanner is in the test szenario"""
        raise NotImplementedError

    def generate_config(self, securitytest):
```

```
        """generate␣the␣config␣or␣command␣line␣attributes␣of␣the␣scanner"""
        raise NotImplementedError

    def usage(self):
        """return␣how␣to␣call␣the␣scanner␣with␣the␣config"""
        # NO HTML-ENTITY FILTERING!!!
        raise NotImplementedError

    def import_results(self, securitytest, results):
        """import␣the␣results␣of␣the␣scanner␣and␣return␣a␣list␣of␣findings"""
        raise NotImplementedError
```

Dabei soll jeder angebundene Scanner folgende Attribute und Methoden haben:

- **name** für den Namen des Scanners.
- **description** für eine kurze Beschreibung des Tools.
- **documentation_url** für einen weiterführenden Link zur Dokumentation des Scanners.
- **uuid** als eindeutiges Identifizierungsmerkmal. Der Universally Unique Identifier (UUID) wurde notwendig, da Testergebnisse und Konfigurationen hochgeladen werden sollen und eine eindeutige Zuordnung zum Scanner notwenig war. Eine UUID, die wie im Kommentar beschrieben ist, erzeugt wurde, hat 32 Hexadezimalstellen und wird zufällig generiert. Die Chance auf eine Kollision ist daher sehr unwahrscheinlich (2^128 Möglichkeiten).
- Die Methode **get_prio** soll anhand des Parameters *securitytest*, der die Informationen zum Test beinhaltet, errechnen, wie nützlich der Scanner in diesem Test sein wird, um die verschiedenen Scanner zu priorisieren und zu sortieren.
- Die Methode **generate_config** soll die Kommandozeilenparameter oder Konfigurationsdateien erstellen und zurückliefern, um diese dem Tester anbieten zu können.
- Die Methode **usage** war ursprünglich als Attribut geplant. Wie sich herausstellte, waren die Texte allerdings teils lange und zur besseren Lesbarkeit des Quellcodes wurde entschieden, daraus eine Methode zu machen.
- Die Methode *import_results* erhält neben den Informationen über den Test auch die Scanner-Ergebnisse. Aus diesen soll diese Methode dann Findings generieren und diese als Liste zurückgeben. Diese sollen dann als Funde aufgenommen werden.

In der Oberfläche werden die Scanner dann wie in Abb. 6.6 angezeigt. Dabei kann die Konfiguration vom Tester angepasst werden. Die verwendete Konfiguration und das Ergebnis des Scanners sollen wieder in die Anwendung geladen werden. Dabei wird zum einen die Konfiguration und die Ausgabe gespeichert, zum anderen wird die Ausgabe von der entsprechenden Scanner-Klasse ausgelesen und sofern Schwachstellen direkt ersichtlich sind, diese angelegt und dokumentiert.

6.4.3. Beendigung des Tests

Wenn der Sicherheitstest beendet werden soll, muss der Tester noch eine Zusammenfassung des Tests verfassen und unter dem Reiter **Organisatorisches** hinterlegen. Anschließend kann er den Test abschließen. Beim Abschluss des Berichts wird den Schwachstellen, die in dem Test gefunden wurden, der Status *offen* zugewiesen. Von nun an ist er in der Schwachstellenübersicht zu finden.

Export in ein Ticketing System

Neben der Status-Zuweisung wird auch eine Funktion namens **export_to_ticketing_system** aufgerufen. Diese bekommt als Parameter ein Schwachstellenobjekt übergeben. Diese Funktion kann nun über eine vom Ticketing System definierte Schnittstelle ein Ticket im Ticketing System erzeugen. Um diese Erzeugung generisch

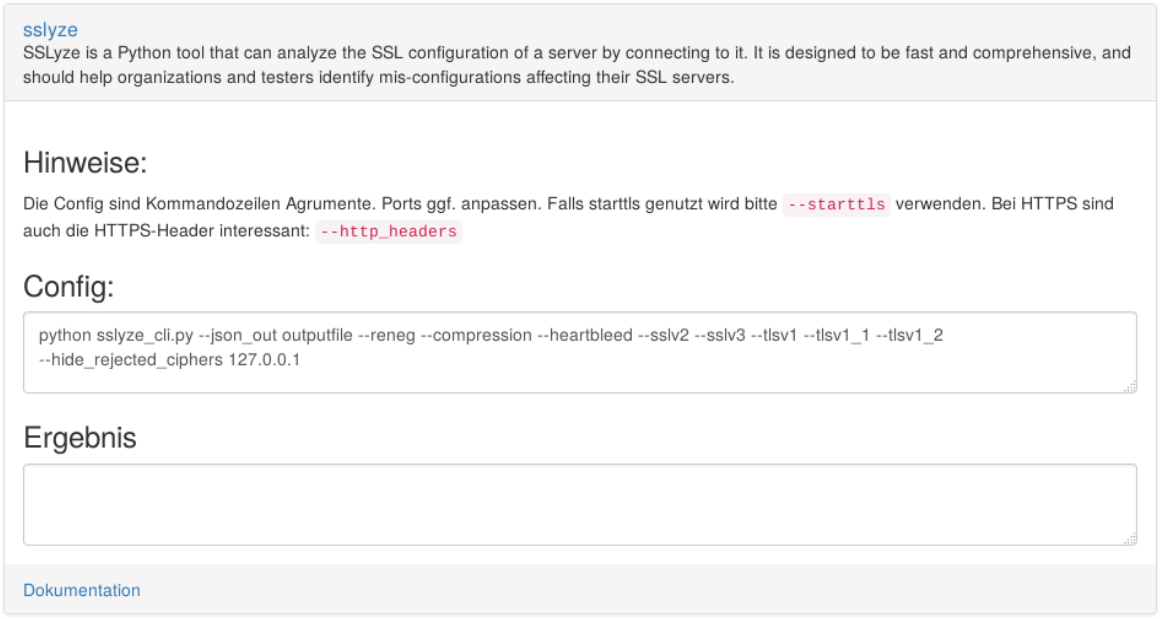


Abbildung 6.6.: Screenshot der Durchführung eines Sicherheitstests (Reiter Tools)

zu halten und eine Anbindung unterschiedlicher Ticketing Systeme zu ermöglichen, kann diese Funktion beliebig verändert werden. Dadurch wäre es auch möglich, zum Beispiel eine E-Mail an eine entsprechende E-Mail-Adresse zu versenden. In Listing 6.3 ist beispielhaft eine REST-Schnittstelle zu einem fiktiven Ticketing System implementiert.

Listing 6.3: Die Methode zum Export der Findings in ein Ticketing System

```
def export_to_ticketing_system(finding):
    """export a finding to the ticketing system"""
    # This is an example, using a rest-api

    # calculate the severity
    if finding.cvss is None:
        # default
        severity = 1
    else:
        # our target-system has only five severity-classes
        severity = round(finding.cvss.calculate_basescore()/2)
        if severity == 0:
            # since 0 is not allowed
            severity = 1

    #init the data
    data = {
        #auth
        "user": APIUSER,
        "token": APITOKEN,
        #data
        "recipient": finding.asset.ticketing_address,
        "subject": "[Schwachstelle] {}".format(finding.title),
        "description": finding.description_text,
        "severity": severity
    }

    try:
        # send the request
```

```
        response = requests.post(APIURL, data=data)
    except Exception as e:
        # there might be a timeout error, or a dns-error, etc.
        # so log and ignore
        logging.getLogger("zoxdocd10").warning(
            "Could not create ticket for finding id=%s",
            finding.finding_id
        )
        return None
    # return the ticketnumber, so it can be stored in the db
    return response.text
```

Das fiktive Ticketing System akzeptiert dabei einen **user** und einen **token** zur Authentifizierung. Parameter:

- **recipient** ist die Gruppe, der das Ticket zugewiesen werden soll. Diese Information zum Dienst stammt aus dem CMS.
- **subject** ist der Betreff und soll eine Übersicht über Tickets ermöglichen. Der Schwachstellentitel eignet sich z.B. dazu.
- **description** ist die Beschreibung des Problems. Im Beispiel wird dabei immer die Beschreibung der Schwachstelle übergeben.
- **severity** beschreibt die Kritikalität, also welche Priorität das Ticket für die Bearbeitung hat. Das fiktive Ziel-System hat dabei nur fünf Stufen, der CVSS3-Wert wird daher umgerechnet.

Nachdem die Daten für das Ticketing System zusammengestellt wurden, werden sie mittels HTTP-POST an das Ticketing System versendet. Dabei liefert das Ticketing System im Erfolgsfall eine Ticket-ID zurück. Falls ein Fehler auftritt, wird dieser zur Analyse geloggt und keine Ticket-ID gespeichert. Ein erneutes Senden im Fehlerfall wäre denkbar, wurde jedoch nicht implementiert, da es sich um ein fiktives Ticketing System handelt. Für ein reales System sollten die möglichen Fehlerfälle im Vorhinein abgeklärt werden, sodass auf diese speziell reagiert werden kann.

Generierung der Berichte

Neben den Tickets, die eher der Nachverfolgung dienen, sollen auch Berichte generiert werden. Von einer Generierung zum Abschluss eines Tests wurde abgesehen, da die Generierung zum einen performant ist und somit auch bei Bedarf durchgeführt werden kann und zum anderen ergibt sich bei einer Generierung nach Bedarf die Möglichkeit, neue Vorlagen zu verwenden. Sollte die Vorlage für Berichte geändert werden, können so auch für ältere Tests neuartige Berichte generiert werden. Die Berichte bleiben so nach der Änderung der Vorlage weiterhin einheitlich.

Für die Generierung wurden drei Funktionen erstellt, jeweils eine für eine Empfänger-Rolle (Manager, Entwickler und Administrator). Die Berichte für die Manager sollten dabei nur aus der Zusammenfassung und einer Auflistung der gefundenen Schwachstellen bestehen, um einen kurzen Überblick über den Test und die Ergebnisse zu ermöglichen.

Für die Rollen Entwickler und Administrator sind neben der Zusammenfassung und der Auflistung der Schwachstellen auch Details zu den Schwachstellen wichtig. Daher wird nach der Auflistung aller Schwachstellen jede Schwachstelle detailliert beschrieben. Dabei setzen sich die Details wie folgt zusammen:

- **Titel**: Der Titel der Schwachstelle.
- **Beschreibung**: Die Beschreibung der Schwachstelle vom Tester.
- **Beweis**: Ein kurzer Beweis, dass die Schwachstelle existiert.
- **Tester-Empfehlung**: Die Behebungsempfehlung des Testers.
- **Empfehlung für die entsprechende Kategorie**: Die generelle Behebungsempfehlung für die entsprechende Kategorie der Schwachstelle.

- **Kategorie**: Die Kategorie der Schwachstelle.
- **CVSS**: Der CVSS3 Score der Schwachstelle, um eine Priorisierung zu ermöglichen.

Analog zum Kickoff-Protokoll und zum Schwachstellen-Sheet wird auch hier eine Vorlage eingelesen und dann befüllt. Da allerdings das ganze Dokument generiert werden muss und nicht nur Felder gefüllt werden müssen, darf in der Vorlage kein Text enthalten sein. Die Vorlage dient daher nur als Designvorlage, um etwa Fußzeilen und Schriftarten zu definieren. Somit kann auch dieser Bericht an das Design des Unternehmens angepasst werden.

Die Optionen zum Herunterladen der Berichte sind in der Ansicht des Tests möglich. Die Ansicht wurde durch den Reiter **Berichte** ergänzt.

6.5. Schwachstellenmanagement

Nachdem der Test abgeschlossen wurde, die Berichte durch den Dienstverantwortlichen an die Entiwckler, Administratoren und das Management versendet wurden, erfolgt die Behebung der Schwachstellen. Um diese Behebung zu überwachen und die Umsetzung zu kontrollieren, gibt es eine Schwachstellenübersicht, in der alle Schwachstellen aufgeführt sind. (Abb. 6.7)

Abbildung 6.7.: Screenshot der Schwachstellenübersicht

In dieser Übersicht kann der Sicherheitsverantwortliche alle Schwachstellen einsehen und kann über die Verlinkung des Titels einer Schwachstelle sich die Details dazu anzeigen lassen. In der Detailansicht (Abb. 6.8) kann er dann die Details (Beschreibung, den Beweis, die Empfehlung des Testers, die Ticket-ID, etc.) ansehen. Der Status einer Schwachstelle kann dort auch geändert werden. Diese Änderung sollte erfolgen, nachdem die Behebung verifiziert wurde.

Schwachstelle: XSS in der Suche

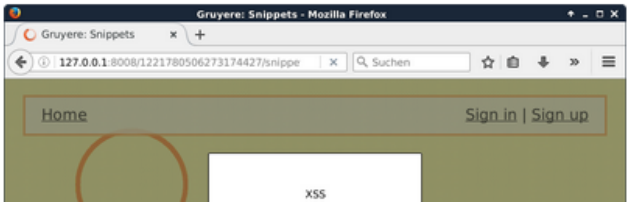
Titel	XSS in der Suche
Funddatum	17.10.2016
Dienst	E-Mail
Test	Interner vorsichtiger fokussierter verdeckter White-Box Test von Innen
Beschreibung	Der uid-Parameter der Unterseite snippets.gtl ist anfällig für Cross-Site-Scripting. snippets.gtl?uid=cheddar<script>alert("XSS")</script> Ein Angreifer könnte einem Anwender einen "infizierten" Link, z.B. via E-Mail, zusenden, wenn der Anwender den Link öffnet, kann die Session vom Angreifer übernommen werden und Inhalte ausgelesen werden. Auch die Inhalte können für den Anwender manipuliert werden.
Beweis	

Abbildung 6.8.: Screenshot der Detailansicht einer Schwachstellen

7. Prototypische Anwendung

Nachdem nun ein Konzept für ein Managementsystem für Sicherheitstests entwickelt und eine prozessunterstützende Anwendung entwickelt wurde, wurde der Prozess einmal beispielhaft am LRZ durchlaufen. Dieser Durchlauf wird im Folgenden näher beschrieben.

7.1. Planung des Sicherheitstests

Zunächst musste ein Dienst gefunden werden, der getestet werden soll. In die engere Auswahl kamen dabei Splunk und LISC. Splunk ist ein Log- und Monitoring-Tool, das Administratoren dabei helfen soll, Störungen im Betrieb frühzeitig zu erkennen und zu beheben. LISC steht für *Learn Interactive Server Configuration* und stellt eine Plattform da, mit der es Administratoren möglich gemacht werden soll, zu lernen, wie man Server sicher konfiguriert.

Da der Test im Folgenden näher beschrieben werden sollte, war es wichtig, einen Dienst so zu testen, dass der Test nicht zu umfangreich würde. Bei Splunk hätten einige Aspekte getestet werden müssten, u.a. die Schnittstellen zur Überwachung von Systemen, die Übergabeschnittstelle der Logs und die Weboberfläche. Da es sich bei LISC um eine reine Webanwendung handelt, musste für einen vollständigen Test des Dienstes nur die Weboberfläche getestet werden. Daher wurde sich für LISC als Testobjekt entschieden. Das Testen von Splunk wäre ebenso möglich gewesen, würde aber den Rahmen dieser Arbeit überschreiten.

7.1.1. Testplanung

Nachdem der Testbedarf für den Dienst LISC festgestellt wurde, wurde von der Dienstverantwortlichen ein Test mit Hilfe der prozessunterstützenden Anwendung für diesen Dienst geplant.

Die Testmerkmale des Sicherheitstests waren (Abb. 7.1):

- **Informationsbasis**: *Black-Box*. Um den Test besser beschreiben zu können, wurde darauf verzichtet, existierende Dokumentationen an den Tester zu übergeben. Die Auflistung und die Auswertung solcher Informationen wäre sehr umfangreich gewesen, daher wurde entschieden, einen Black-Box-Test durchzuführen.
- **Aggressivität**: *vorsichtig*. Obwohl LISC nur von wenigen Anwendern genutzt wird, könnte ein aggressiver Sicherheitstest trotzdem zu Betriebsstörungen führen. So wird LISC in einem virtuellen Rechner betrieben. Der Host, auf dem der virtuelle Rechner wiederum betrieben wird, könnte jedoch noch andere virtuelle Rechner betreiben, die beeinträchtigt werden könnten. Auch wurde der Zugang zu LISC über VPN realisiert. Zu der Bandbreite des VPN und der dahinterstehenden Infrastruktur lagen keine Informationen vor. Diese zunächst einzuholen hätte den zeitlichen Rahmen dieser Arbeit überschritten. Ein aggressiver Test über VPN könnte somit andere Anwender, die ebenfalls VPN nutzen, beeinträchtigen. Um diese Störungen zu vermeiden, wurde beschlossen, einen vorsichtigen Test durchzuführen.
- **Umfang**: *fokussiert*. Um die Beschreibung des Sicherheitstests kurz zu halten, sollte nur der Dienst LISC getestet werden.
- **Vorgehensweise**: *offensichtlich*. Da es sich um einen kleinen Dienst handelt, wird dieser nicht streng überwacht. Um die Prozesse für Netzwerküberwachung (IDS/IPS) mit zu prüfen, eignet sich dieser Dienst daher nicht.

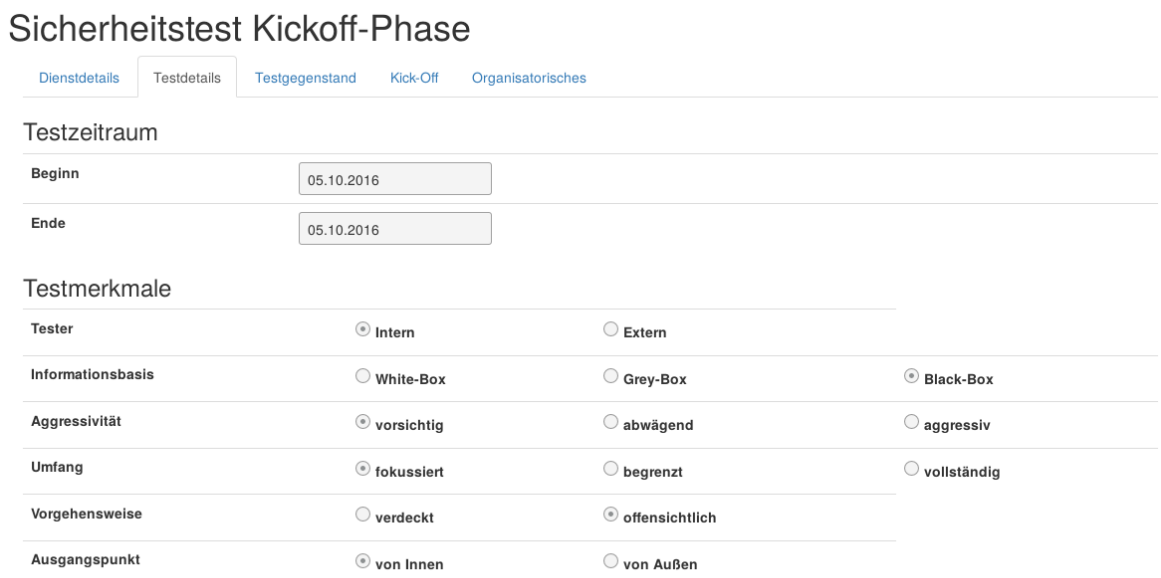


Abbildung 7.1.: Screenshot der Testplanung für LISC

- **Ausgangspunkt**: *von innen*. Der Dienst ist nur aus dem MWN erreichbar, es gibt keine Funktionalitäten, die nur aus bestimmten Netzbereichen zugänglich sind, daher wurde als Ausgangspunkt das MWN genutzt.

Schnittstellen sollten, aufgrund des Umfangs, keine mit getestet werden. Als zu testende IP-Adressen wurde die einzige IP-Adresse ausgewählt. Das CSIRT wurde via E-Mail informiert. Die E-Mail beinhaltete den Zeitraum, den Dienst und die IP-Adresse des Dienstes sowie die Kontaktinformationen des Testers und der Dienstverantwortlichen. Auch das Kickoff Protokoll wurde mitgesendet. Da binnen zwei Tagen kein Einspruch des CSIRT folgte, wurde auch vom Sicherheitsverantwortlichen noch die Genehmigung eingeholt.

7.1.2. Kickoff

Nachdem die Rahmenbedingungen geklärt waren, fand ein Kickoff Meeting statt. Die Inhalte wurden in einem Kickoff Protokoll (Siehe A.1) festgehalten. Dabei wurden die Rahmenbedingungen nochmals durchgesprochen, die Kontaktdetails ausgetauscht und Account-Informationen übermittelt, mit denen sich der Tester an dem Dienst anmelden kann. Auch die Genehmigung des Sicherheitsverantwortlichen wurde bereits eingeholt und die Einholung im Protokoll bestätigt.

7.2. Testdurchführung

Nachdem der Test geplant wurde und das Kickoff-Meeting statt gefunden hat, wurde der Sicherheitstest durchgeführt. Die Druchführung dieses Sicherheitstests wird im Folgenden näher beschrieben. Dazu werden zunächst zwei verwendete Programme erläutert, die bei dem Test eingesetzt wurden. Anschließend wird der Verlauf des Tests beschrieben.

7.2.1. Verwendete Tools

Um Teile eines Sicherheitstests zu automatisieren oder den manuellen Aufwand zu verringern, werden häufig Programme eingesetzt, die den Tester unterstützen sollen. In diesem Sicherheitstest wurden hauptsächlich zwei

Programme verwendet, zum einen die Burp Suite und zum anderen SSLyze. Diese beiden Programme werden nun kurz beschrieben.

Burp Suite

Die Burp Suite ist ein Programm zur Unterstützung von Sicherheitstests von Webanwendungen. Dabei besteht die Burp Suite aus mehreren Komponenten.

- Der **Proxy** ermöglicht es, die Kommunikation zwischen dem Browser und der Webanwendung zum einen anzusehen und zum anderen zu manipulieren. Dabei ist es auch möglich, verschiedene Manipulationen zu automatisieren, um bspw. HTML-Header oder übergebene Parameter dauerhaft zu manipulieren.
- Ein sogenannter **Crawler** ermöglicht es, alle Webseiten einer Anwendung zu katalogisieren, um sicherzustellen, dass alle Webseiten getestet werden und ggf. versteckte Seiten zu finden.
- Der **Intruder** ermöglicht es, automatisiert Parameter zu manipulieren und so Angriffe durchzuführen. Dabei erkennt der Intruder auch, ob ein Angriff erfolgreich war und meldet dies dem Tester.
- Mit dem **Sequenzer** können verschiedene Anfragen automatisiert wiederholt werden. Bspw. die Initialisierung einer Session, um die Zufälligkeit der Sessiontokens zu prüfen.

Neben diesen Komponenten existieren noch zahlreiche weitere Erweiterungen. Die Burp Suite ist ein kommerzielles Produkt, es existiert jedoch auch eine *Free Edition* mit eingeschränktem Funktionsumfang als Gratisversion. Diese Gratisversion wurde im Folgenden verwendet.[BUR]

SSLyze

Um die TLS[1] Serverkonfiguration zu testen, gibt es u.a. SSLyze. SSLyze ist in Python 2.7 geschrieben und simuliert zum einen den TLS-Handshake mit dem Server unter Verwendung verschiedener Verschlüsselungsalgorithmen. Zum anderen können auch bekannte TLS Sicherheitslücken getestet werden, z.B. Heartbleed. Oftmals wird die Verbindung zwischen einem Client und dem Server zwar mittels TLS verschlüsselt, jedoch kann es unter Umständen sein, dass diese Verschlüsselung nicht sicher ist. Angriffe auf die Verschlüsselung sind zwar meist unrentabel, da man häufig Voraussetzungen für einen Angriff benötigt, die nicht häufig vorkommen, lassen sich aber oft sehr einfach lösen, indem zum Beispiel unsichere Algorithmen oder Protokollversionen deaktiviert werden. Für SSLyze wurde bereits eine Anbindung an die prozessunterstützende Anwendung implementiert.

7.2.2. Testverlauf

Nachdem der Test geplant wurde und auch genehmigt wurde, konnte dieser durchgeführt werden. Im folgenden wird die Testdurchführung beschrieben. Die Testdurchführung geschah größtenteils manuell.

Informationsbeschaffung und Bewertung

Da es sich um einen Black-Box-Test handelte, mussten zunächst Informationen gesammelt werden. Zunächst wurde der HTML[2] Quelltext der Login-Seite untersucht. Dieser erbrachte bis auf die jQuery, ein JavaScript Framework (Version 1.12), keine Versionsinformationen. Jedoch wurden statische Dateien aus einem Unterordner */static/* eingebunden. Solche Ordner für statische Dateien haben oftmals ein sogenanntes "Directory Listing" aktiv, das es ermöglicht, alle dort gespeicherten Dateien einzusehen. Das muss keine Schwachstelle sein, kann aber bei der Informationsbeschaffung hilfreich sein. Auch hier war das Directory Listing aktiv. (Abb. 7.2)

[1] Transport Layer Security
[2] Hypertext Markup Language

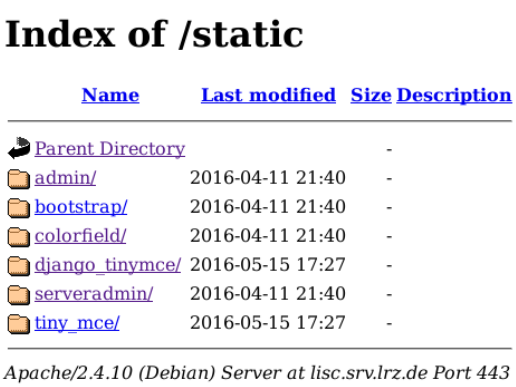


Abbildung 7.2.: Screenshot des Directory Listing im /static-Ordner

Im Footer der angezeigten Seite ist erkennbar, dass es sich um einen Apache Webserver in der Version 2.4.10 und einen Debian Server handelt. Die Dateien in diesem */static/*-Ordner waren hauptsächlich Stylesheets, Bilder und JavaScript Dateien. Die gefundenen JavaScript Bibliotheken sowie die Information über die eingesetzte Apache Version lieferten bei einer Internetrecherche keine bekannten ausnutzbaren Schwachstellen.

Neben dem */static/*-Ordner wurden noch weitere Verzeichnisse vermutet, die nicht öffentlich bekannt sein sollten. Daher wurden beliebte Verzeichnisnamen durchprobiert. Dabei wurde ein Ordner */admin/* gefunden. (Abb. 7.3) Für diese Seite funktionierte die vereinbarte Benutzerkennung nicht, was ein weiteres Testen nicht möglich machte. Auf Bruteforce-Angriffe, also das Durchprobieren von Benutzer- und Passwortkombinationen, wurde aufgrund der niedrigen Aggressivität verzichtet.

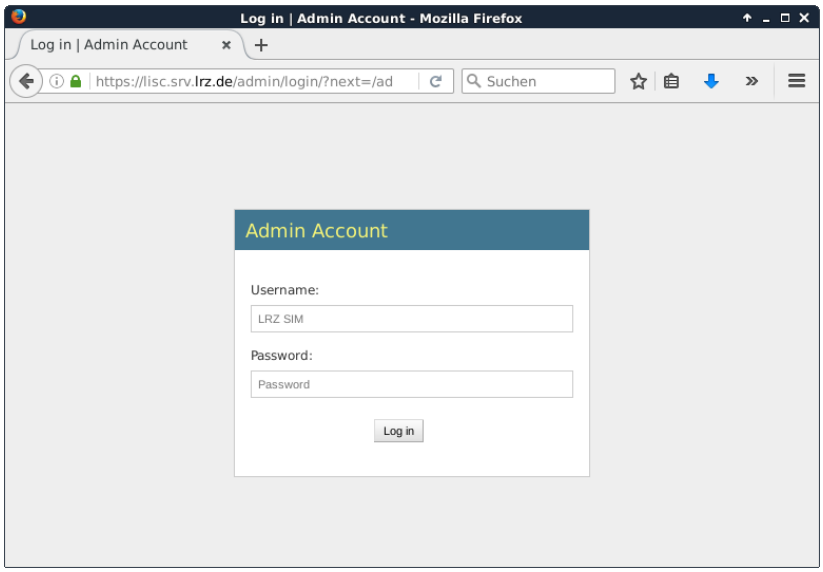


Abbildung 7.3.: Screenshot des Log ins für Administratoren

Aktive Eindringversuche

Nachdem diese Möglichkeiten ausgeschöpft wurden, wurde die Anwendung näher betrachtet und einige Abläufe wurden getestet. Um übertragene Parameter kontrollieren und manipulieren zu können, wurde das Programm BurpSuite aktiviert und im Browser als HTTP-Proxy eingetragen. Von nun an konnte jeder HTTP-Request zum Server und jede Response vom Server manipuliert werden, um das Verhalten der Anwendung bei unerwarteten Eingaben zu testen. Dabei wurden sehr viele Parameter auf Injection-Schwachstellen getestet, jedoch ohne Erfolg.

Als Gegenmaßnahme zu Cross-Site-Request-Forgery (CSRF) werden in LISC CSRF-Token verwendet. Dabei wird ein solcher Token in den übertragenen Formularen mitübertragen, um zu prüfen, ob der Anwender selbst das Formular abgesendet hat. Bei gefälschten Anfragen sendet der Webbrowser zwar ein Session-Cookie mit, dieses enthält jedoch nicht den CSRF-Token, daher kann der Server anhand dieses CSRF-Tokens prüfen, ob die Anfrage von dem Anwender kam oder nicht. In LISC gibt es seltsamerweise zwei Speicherorte für diese Tokens. Zum einen wird der Token im Formular übertragen, zum anderen existiert auch ein Cookie, welches den Token überträgt. Mittels Manipulation in Burp Suite wurde herausgefunden, dass der Server beide Parameter prüft. Sollte einer abweichen, gibt die Anwendung einen Fehler aus. Interessant dabei war allerdings, dass wenn der CSRF-Token bei einem Get-Request manipuliert wurde, der CSRF-Token im Formular auch manipuliert war. Der User kann somit diesen Token selbst setzen. (Abb. 7.4) Dies ist zwar nicht direkt ausnutzbar im Sinne eines Angriffs, könnte aber in Verbindung mit anderen Schwachstellen oder in einer anderen Umgebung ausnutzbar werden.

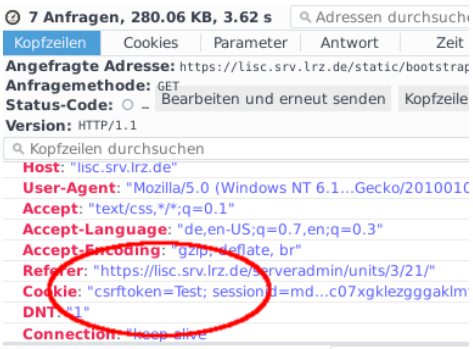


Abbildung 7.4.: Anfrageheader mit manipuliertem CSRF-Token

Bei näherer Betrachtung der Cookies fiel weiterhin auf, dass die Cookies eine sehr lange Haltbartkeit aufweisen. Die Haltbarkeit der Session entspricht vermutlich auch der Haltbarkeit des Cookies. Sollte es einem Angreifer gelingen, einen Session-Token zu stehlen, kann er die Session übernehmen und benötigt keine Anmeldung an der Anwendung. Da die Token sehr lange (14 Tage) haltbar sind, ist ein solcher Angriff auch lange möglich. Daher sollten Sessions eine möglichst kurze Haltbarkeit haben.

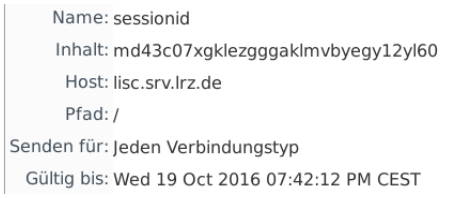


Abbildung 7.5.: Eigenschaften des Session-Cookies

Neben der langen Haltbarkeit sind auch die Cookie-Flags *http-only* und *secure* nicht gesetzt (Abb. 7.5). Der Cookie-Flag *http-only* verhindert, dass Cookies mittles Javascript gelesen und verändert werden können. Im Falle einer Cross-Site-Scripting Lücke wäre es somit möglich, das Cookie auszulesen und den Inhalt an einen Angreifer zu senden, sodass dieser die Session übernehmen kann. Da der Dienst von der Möglichkeit, diesen Cookie mittels Javascript zu manipulieren, keinen gebrauch macht, kann dieses Flag gesetzt werden, um das Schadenpotential einer Cross-Site-Scripting Schwachstelle zu veringern. Das Cookie-Flag *secure* bewirkt, dass das Cookie nur bei TLS gesicherten Verbindungen übertragen wird. Sollte ein Angreifer in eine Position zwischen dem Browser und dem Server gelangen, ein sogenannter Man-in-the-middle Angriff, kann er ggf. die Verschlüsselung der Verbindung verhindern und anschließend die Cookies bei der Übertragung mitlesen und somit die Session übernehmen. Da der Dienst die Verbindungen verschlüsselt, kann dieses Flag auch gesetzt werden, um ggf. andere Angriffe zu verhindern.

Um die TLS Konfiguration zu testen, wurde SSLyze genutzt. Für SSLyze wurde dazu eine Anbindung an die prozessunterstützende Anwendung implementiert. Die Konfiguration wurde daher bereits von der Anwendung

vorgefüllt. Da es sich um eine Webanwendung handelt, wurde noch der Parameter *–http-headers* ergänzt. (Abb. 7.6)

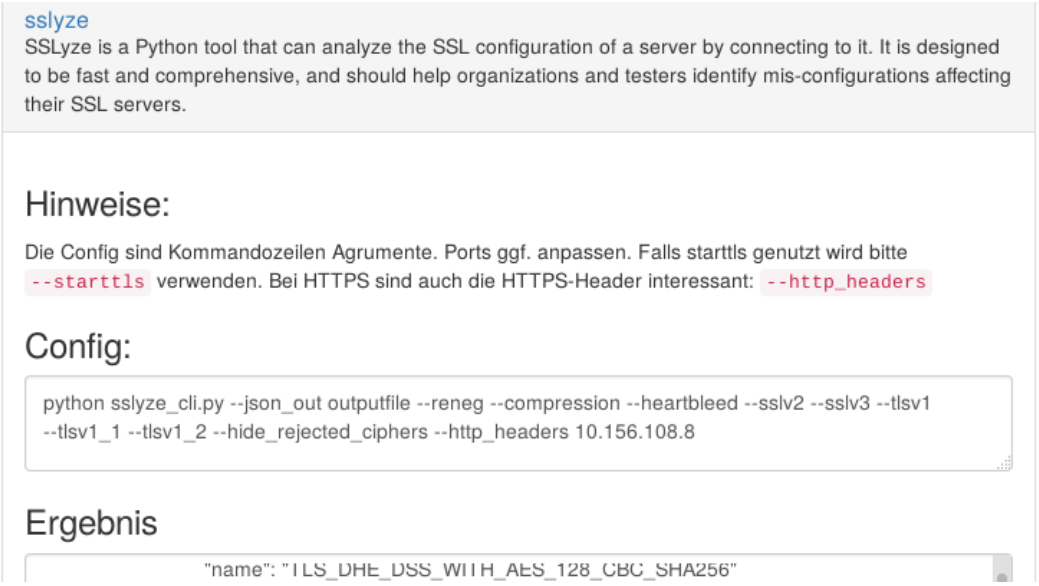


Abbildung 7.6.: Screenshot der Tool-Anbindung von SSLyze

Dabei war die einzig entdeckte Schwachstelle das Fehlen der HSTS-Header. HSTS steht für HTTP Strict Transport Security und bewirkt, dass der Webserver dem Browser mitteilen kann, in Zukunft nur noch verschlüsselt mit dem Server zu kommunizieren. Spätere Man-in-the-middle Angriffe werden so deutlich erschwert.

Berichtserstellung

Nachdem der Dienst auf Schwachstellen überprüft wurde, wurden die Schwachstellen in der prozessunterstützenden Anwendung mittles Weboberfläche hinterlegt. Anschließend wurde noch eine Zusammenfassung erstellt und der Test beendet. Der Tester hat daraufhin die Dienstverantwortliche über die Beendigung via E-Mail informiert. Daraufhin konnten die Berichte (siehe A.2f) aus der Anwendung heruntergeladen werden. Die Dienstverantwortliche hat daraufhin die Berichte an die Entwickler gesendet. Ein automatisierter Export in das Ticketing System hat mangels Anbindung nicht stattgefunden. Für die Anbindung an das Ticketing System hätte zuvor ein Benutzerkonto am Ticketing System eingerichtet werden müssen, was zeitlich nicht mehr umzusetzen war.

7.3. Schwachstellennachverfolgung

Nachdem der Test beendet und die Berichte verteilt wurden, sind die gefundenen Schwachstellen in der Schwachstellenübersicht zu sehen. Der Sicherheitsverantwortliche kann diese nun einsehen und Fristen zur Behebung setzen. Die gefundenen Schwachstellen sind allerdings nicht so kritisch, dass diese behoben werden müssen. Der Entwickler und Administrator müssen sich nun überlegen, ob die Schwachstellen behoben werden sollen, oder ob mit einer Behebung ggf. andere Funktionalitäten beeinträchtigt werden würden. So nutzt die Anwendung LISC zwar nur das HTTPS-Protokoll, es könnte aber noch andere Anwendungen geben, die ebenfalls auf dem selben Server betrieben werden, die HTTPS nicht unterstützen. Sollten nun die HSTS-Header konfiguriert werden, könnten diese Anwendungen nicht mehr genutzt werden. Da diese Diskussion Zeit benötigt, war es leider nicht möglich, die Behebungsmaßnahmen zu beschreiben.

8. Zusammenfassung und Ausblick

Nachdem der Prozess für Sicherheitstests einmal prototypisch angewendet wurde, wird im Folgenden die Anwendung aus Abschnitt 7 in Hinblick auf die Anforderungen aus Abschnitt 3 kritisch geprüft.

8.1. Erfüllung der Anforderungen

Zunächst soll zu jeder Anforderung der Erfüllungsgrad beschrieben werden. Sollte eine Anforderung nicht komplett erfüllt werden, sollen Möglichkeiten aufgezeigt werden, wie diese zu erfüllen sind.

A01 - Flexibilität des Anwendungsbereichs

Da die Informationen über die Dienste aus einem Configuration Management System stammen, ist es möglich, nur bestimmte Dienste, bspw. nach bestimmten Abteilungen gefiltert, zu berücksichtigen. Der Anwendungsbereich kann somit verkleinert werden. Für eine Vergrößerung des Anwendungsbereichs auf z.B. das gesamte Hochschulnetz samt angeschlossener selbstverwalteter Netze müssten nur wenige Anpassungen vorgenommen werden.

- Es müsste ein einheitliches CMS im gesamten Netzwerk etabliert werden. Sollte die prozessunterstützende Anwendung genutzt werden, wäre ein einheitliches Export-Format bereits ausreichend, um alle Dienste des Netzwerks zu importieren.
- Neben dem Sicherheitsverantwortlichen des Rechenzentrums müsste eine Person benannt werden, die Sicherheitsverantwortlicher des gesamten Netzes wäre. Neben der Ernennung einer Person wäre es auch möglich, diese Rolle einem Gremium, bspw. der Sicherheitsverantwortlichen der einzelnen Netze, zuzuordnen. Dieses Gremium müsste dann über die Genehmigung eines Sicherheitstests entscheiden.

A02 - Definition der Aktivitäten

Die Aktivitäten in den Prozessen sind soweit definiert, dass durchaus etwas Interpretationsspielraum besteht. Jedes Hochschulrechenzentrum kann die Aktivitäten beliebig genau definieren und für die eigenen Bedürfnisse anpassen. Die Grundidee einer Aktivität ist aber soweit beschrieben, dass das Konzept direkt anwendbar ist.

A03 - Definition der beteiligten Rollen

Die beteiligten Rollen wurden definiert. Dabei soll diese Definition nur ein Hinweis sein. Da es durchaus möglich ist, mehrere Rollen einer Person zuzuordnen, kann das Konzept auch in sehr kleinen Rechenzentren eingesetzt werden. Durch die Möglichkeit, ganzen Teams eine Rolle zuzuordnen, ist es auch möglich, in sehr großen Rechenzentren oder sogar ganzen Hochschulnetzen das Konzept umzusetzen. In der prototypischen Anwendung wurden darüber hinaus keine weiteren Rollen entdeckt, die noch nicht ausreichend adressiert waren. Die beschriebenen Fähigkeiten dieser Rollen kann sich von Rechenzentrum zu Rechenzentrum unterscheiden, sollten jedoch das Mindestmaß an Fachkompetenz darstellen.

A04 - Schnittstellen zu anderen Prozessen

Auch die Anbindung an andere Prozesse ist vorhanden und wurde berücksichtigt. So kann bspw. ein bereits bestehendes Dienstmanagement einfach angebunden werden. Auch eine Schnittstelle in das Risikomanagement ist vorgesehen. Mittels der prozessunterstützenden Anwendung ist es ebenfalls sehr leicht, ein Ticketing System anzubinden, um ein bestehendes Change- und Incidentmanagement zu integrieren.

A05 - Automatisierung

Die prozessunterstützende Anwendung ermöglicht einige Aktivitäten zu automatisieren. So werden die Berichte automatisch generiert und Tickets können automatisch erzeugt werden. Des Weiteren sind alle Sicherheitstests und Schwachstellen maschinenlesbar gespeichert, es können somit noch weitere Prozesse automatisiert werden, bspw. Berichte ins Management.

A06 - Unterstützung manueller Aktivitäten
Neben der Automatisierung werden auch einige Aktivitäten, die nicht vollständig zu automatisieren sind, von der konzipierten Anwendung unterstützt. So werden die Kickoff Protokolle bereits vor ausgefüllt, Konfigurationen für Schwachstellenscanner können vorbereitet werden und das Ergebnis auf Schwachstellen untersucht werden.

A07 - Messbar
Durch die Nutzung eines Testregisters und eines Schwachstellenregisters, bspw. in der konzipierten Anwendung, sind alle Sicherheitstests und Schwachstellen mit Details hinterlegt. Dies stellt eine solide Basis für Berichte ins Management da, bspw. für Key Performance Indicators (KPIs). Denkbare KPIs wären die Anzahl an offenen Schwachstellen, die Summe der CVSS-Scores der offenen Schwachstellen, Anzahl der durchgeführten Sicherheitstests, etc.

A08 - Berichte an die Unternehmensführung
Berichte an die Unternehmensführung sind nur in Form der Managementberichte zu den Sicherheitstests vorgesehen. Allgemeine Berichte zum Stand der Sicherheit sind nicht definiert. Der Bedarf für Managementberichte unterscheidet sich von Unternehmen zu Unternehmen. Durch die Messbarkeit sind allerdings Grundlagen für KPIs gelegt. Aus den verschiedenen Registern können einfach Berichte erzeugt und an den jeweiligen Bedarf angepasst werden.

8.1.1. Anforderungen an das Dienstmanagement

D01 - Existenz einer Dienst-Dokumentation
Im Rahmen des Dienstmanagements sind Dienst-Dokumentationen vorgesehen. Zum einen können dies Informationen aus einem CMS sein, zum anderen können dies auch Betriebs- oder Sicherheitskonzepte sein. In der prozessunterstützenden Anwendung werden nur Informationen aus einem CMS importiert. Sollte es anderweitig weitere Dokumentationen geben, werden diese derzeit nicht berücksichtigt. Die Anwendung könnte jedoch mit wenig Aufwand so angepasst werden, dass auch dies möglich ist. Das Problem ist erfahrungsgemäß allerdings, dass es solche Dienstdokumentationen entweder gar nicht gibt, oder dass diese nicht zentral abgelegt sind. Eine Integration in die Anwendung wäre trotzdem hilfreich.

D02 - Register von Diensten
Das Dienstregister ist der Ausgangspunkt des Sicherheitstestprozesses. Die Erstellung und Pflege eines Dienstregisters ist zwar nicht explizit beschrieben, kann aber nach ITIL eingeführt werden. Auch Textdateien und Tabellenkalkulationen sind denkbar und können genutzt werden.

D03 - Zentralisierung des Registers von Diensten
Um eine Zentralisierung des Dienstregisters zu ermöglichen, wurde in der Anwendung darauf verzichtet, ein selbstständiges Dienstmanagement zu implementieren. Durch die Möglichkeit, ein bestehendes Dienstmanagement, etwa ein CMS, zu importieren, kann ein zentrales Dienstregister genutzt werden. Auch auf die Möglichkeit, Details zu Diensten zu ändern, wurde verzichtet, um einen Zwang zur Nutzung eines zentralen Dienstmanagements aufzuerlegen.

D04 - Kritikalität von Diensten
Auch die Kritikalität kann über das CMS dargestellt werden. Im fiktiven CMS (vgl. Abschnitt 6.2.1) wurde dazu ein Feld *Risikofaktor* genutzt. Dieser Risikofaktor ist zwar etwas undifferenziert, kann aber zur Sortierung und Priorisierung genutzt werden.

8.1.2. Anforderungen an das Testmanagement

T01 - Nicht betriebsgefährdend

Durch die Einholung einer Genehmigung des Sicherheitsverantwortlichen und die Planung durch den Dienstverantwortlichen sowie die Information an das CSIRT können Sicherheitstests bereits bei der Planung auf Betriebsstörungsgefahren hin geprüft werden. Durch die Information des CSIRT und den Austausch von Kontaktinformationen des Testers werden auch die Reaktionswege bei einer Betriebsstörung deutlich verkürzt, was das Schadenpotential und die Dauer einer Betriebsstörung minimiert.

T02 - Wiederholbar
Durch die Definition der Rahmenbedingungen und der Dokumentierung der Testplanung im Kickoff Protokoll kann die Testplanung nachvollzogen werden. Die Testdurchführung ist durch die Speicherung von Schwachstellenscannerkonfigurationen und die Dokumentation der gefundenen Schwachstellen nachvollziehbar. Ob diese wiederholbar sind, ist allerdings nicht gewährleistet, da gerade bei manuellen Sicherheitstests der Weg zur Identifizierung einer Schwachstelle nicht ausreichend dokumentiert ist. Um die Behebung einer Schwachstelle zu verifizieren sollte die Dokumentation der Schwachstelle jedoch ausreichen.

T03 - Zentrales Register von Sicherheitstests
Ein zentrales Register von Sicherheitstests ist zum einen durch die prozessunterstützende Anwendung möglich, zum anderen muss der Sicherheitsverantwortliche alle Sicherheitstests genehmigen. Der Sicherheitsverantwortliche kann somit ohne großen Aufwand ein solches Testregister auch manuell pflegen.

T04 - Dokumentation des Tests
Die Rahmenbedingungen eines Tests werden im Kickoff Protokoll festgehalten. Neben den Eigenschaften des Tests können auch weitere Informationen, etwa IP-Adressen des Tester, übergebene Anmeldeinformationen für den Test, etc., dokumentiert werden. Durch die Dokumentation von Schwachstellen und dem Erstellen von Berichten ist auch das Ergebnis eines Tests dokumentiert.

T05 - Regelmäßiges Testen aller Dienste
Durch das zentrale Dienst- und Testregister ist es möglich, die Zeit seit dem letzten Test auf einen Dienst zu ermitteln. Das Testintervall sollte jedoch auf Basis des Risikos eines Dienstes und der verfügbaren Ressourcen definiert werden. Diese Parameter können von Unternehmen zu Unternehmen deutlich variieren. Daher ist die Festlegung eines Testintervalls nicht genau definiert, aber vorgesehen.

T06 - Risikobasierte Festlegung des Testintervalls
Das Testintervall muss neben dem Risiko auch auf Basis der verfügbaren Ressourcen festgelegt werden. Dieses Intervall muss der Sicherheitsverantwortliche festlegen. Diese Festlegung ist in den Prozessen auch so vorgesehen. Die prozessunterstützende Anwendung unterstützt den Sicherheitsverantwortlichen dabei nicht. Es wäre allerdings wünschenswert, eine Formel hinterlegen zu können, um aus dem Risikofaktor ein Testintervall zu errechnen und dann die Dienste hervorzuheben, die nach dieser Formel einen Sicherheitstest benötigen.

T07 - Prozess zur Durchführung eines Tests
Die Durchführung eines Sicherheitstests ist meist eine sehr individuelle Tätigkeit. Gerade bei manuellen Tests ist es schwierig, einen Prozess genau zu definieren. Daher wurde der Prozess zur Durchführung eines Sicherheitstests nur sehr abstrakt definiert und beschrieben.

T08 - Genehmigungen einholen
Um das Risiko für Betriebsstörungen zu verringern, muss die Genehmigung des Sicherheitsverantwortlichen vor Beginn eines Tests eingeholt werden. Auch das CSIRT wird über einen Test informiert und kann bei Bedenken diese anbringen. Da die Genehmigungsprozesse und auch die Genehmiger von Unternehmen zu Unternehmen sehr unterschiedlich sein können, wurden diese nicht genau definiert, ist aber als Aktivität vorgesehen und muss auch im Kickoff Protokoll festgehalten werden.

T09 - Testbenachrichtigungen verteilen
Um auf eventuell eintretende Betriebsstörungen während eines Tests reagieren zu können, muss das CSIRT über die Durchführung eines Sicherheitstests informiert werden. Es ist auch denkbar, andere Personen und Rollen über einen solchen Test zu informieren. Daher sollte überlegt werden, ggf. einen Verteiler für diese Personen einzurichten. Die Benachrichtigung des CSIRT ist als Aktivität genannt. Da die Ausgestaltung dieser Information allerdings sehr variieren kann, wurde darauf verzichtet, dies genau

zu beschreiben. Auch eine Automatisierung einer solchen Benachrichtigung ist denkbar und wünschenswert, wurde allerdings auf Grund mangelnder Zeit nicht umgesetzt.

T10 - Festlegung eines Testbereichs
Die Festlegung des Testbereichs ist im Prozess in der Planung eines Tests vorgesehen. In der prozessunterstützenden Anwendung werden die Abhängigkeiten aus dem CMS-Import übernommen und die Schnittstellen können individuell ausgewählt werden.

T11 - Bereitstellung von Informationen über den Dienst
Eine Bereitstellung von Informationen über einen Dienst an den Tester ist im Prozess so vorgesehen. Die Informationen aus dem CMS reichen hierfür allerdings nur für Black-Box Tests, weitere Informationen, etwa in Form einer Dienstdokumentation, müssen manuell bereitgestellt werden. Die Informationen, die übergeben wurden werden im Kickoff Protokoll festgehalten.

T12 - Austausch von Kontakten im Falle von Betriebsstörungen
Auch der Austausch von Kontakten für den Fall einer Betriebsstörung wurde im Prozess verankert. Die Kontaktinformationen des Testers werden neben der Information an das CSIRT ebenfalls im Kickoff Protokoll festgehalten. Das Kickoff Protokoll dient somit als zentrale Dokumentation einer Testplanung.

8.1.3. Anforderungen an das Schwachstellenmanagement

S01 - Register von Schwachstellen
Da alle Berichte der Sicherheitstests auch dem Sicherheitsverantwortlichen übermittelt werden müssen, hat der Sicherheitsverantwortliche ohne viel Aufwand die Möglichkeit, ein Schwachstellenregister zu erstellen. In der prozessunterstützenden Anwendung werden alle Schwachstellen in einer Ansicht dargestellt. Derzeit ist leider kein Filtern und Sortieren in dieser Ansicht möglich, kann aber mit überschaubarem Aufwand implementiert werden. Auch die Möglichkeit, eine Anomalie, also ein Verhalten, das keine ausnutzbare Schwachstelle ist, aber verbessert werden könnte, als solche zu kennzeichnen, fehlt leider. Dies war zwar keine Anforderung, ist aber im Rahmen der prototypischen Anwendung aufgefallen und sollte noch implementiert werden.

S02 - Dokumentation einer Schwachstelle
Die Schwachstellen müssen vom Tester dokumentiert werden. Dabei ist die Bewertung der Schwachstellen mittels CVSS nicht ausreichend. Ein Tester hat derzeit nicht die Möglichkeit, eine Anomalie, dessen CVSS Wert sich nicht optimal berechnen lässt, als solche zu kennzeichnen. Sollten bei einem Test mehrere Server getestet werden und nur ein Teil dieser anfällig für eine Schwachstelle sein, lässt sich dies nur über einen Freitext beschreiben. Ein Filtern nach Schwachstellen bzgl. einzelner Server ist nicht möglich. Das Schlüsseln von Schwachstellen auf einzelne Server war ebenfalls keine Anforderung, könnte aber sehr hilfreich sein. Ob der Aufwand dieser Zuordnung für den Tester allerdings im Verhältnis zum Nutzen steht, muss separat geprüft werden.

S03 - Schwachstellennachverfolgung
Auch die Schwachstellennachverfolgung ist als Prozess verankert. Der Tester hat die Möglichkeit, offene Schwachstellen einzusehen und über die Anbindung eines Ticketing Systems hat er auch die Möglichkeit, die Behebung zu verfolgen. Verbesserungen bzgl. der prozessunterstützenden Anwendung wären z.B. die Möglichkeit, ein Datum zu hinterlegen, bis zu welchem die Schwachstelle behoben sein muss. Der Sicherheitsverantwortliche würde ein solches Datum definieren oder es würde sich aus einer Formel errechnen. Sobald dieses Datum überschritten ist, muss der Sicherheitsverantwortliche dann die Behebung ins Management eskalieren.

S04 - Behebung einer Schwachstelle
Die Behebung einer Schwachstelle ist ein sehr individueller Prozess und lässt sich daher nur schwer formalisieren. Daher ist der Prozess dazu nur sehr abstrakt gehalten. Die Verifizierung der Behebung kann mittels Testdokumentation durchgeführt werden. Sollte ein Schwachstellenscanner die Schwachstelle gefunden haben, ist die Konfiguration dieses Scanners ebenfalls gespeichert und der Scan kann wiederholt werden.

S05 - Schnittstelle ins Risikomanagement

Sollte eine Schwachstelle nicht behoben werden, sei es, dass die Behebung in keinem Verhältnis zum Nutzen steht oder eine Behebung nicht möglich ist, besteht eine Schnittstelle in ein Risikomanagement.

8.2. Tabellarische Auflistung der Anforderungen und deren Erfüllungsgrad

Zur besseren Übersicht sind die Anforderungen im Folgenden nochmals tabellarisch aufgelistet. Neben der Auflistung ist auch der Erfüllungsgrad des Konzepts und der Anwendung aufgeführt. Sollte eine Anforderung nicht erfüllt werden, wird dies mit einem **o** gekennzeichnet. Sollte eine Anforderung erfüllt sein, wird dies durch ein **+** gekennzeichnet.

ID	Beschreibung	Gewichtung	Konzept	Anwendung
Allgemeine Anforderungen				
A01	Flexibilität des Anwendungsbereichs	empfohlen	+	+
A02	Definition der Aktivitäten	notwendig	+	o
A03	Definition der beteiligten Rollen	notwendig	+	o
A04	Schnittstellen zu anderen Prozessen	empfohlen	+	+
A05	Automatisierung	empfohlen	o	+
A06	Unterstützung manueller Aktivitäten	empfohlen	o	+
A07	Messbar	empfohlen	+	+
A08	Berichte an die Unternehmensführung	notwendig	+	+
Anforderungen an das Dienstmanagement				
D01	Existenz einer Dienst-Dokumentation	empfohlen	+	+
D02	Register von Diensten	notwendig	+	+
D03	Zentralisierung des Registers von Diensten	empfohlen	+	+
D04	Kritikalität von Diensten	empfohlen	+	+
Anforderungen an das Testmanagement				
T01	Nicht betriebsgefährdend	notwendig	+	+
T02	Wiederholbar	empfohlen	+	+
T03	Zentrales Register von Sicherheitstests	notwendig	+	+
T04	Dokumentation des Tests	empfohlen	+	+
T05	Regelmäßiges Testen aller Dienste	notwendig	+	+
T06	Risikobasierte Festlegung des Testintervalls	empfohlen	+	o
T07	Durchführung eines Tests	notwendig	+	+
T08	Genehmigungen einholen	notwendig	+	o
T09	Testbenachrichtigungen verteilen	empfohlen	+	o
T10	Festlegung eines Testbereichs	notwendig	+	+
T11	Bereitstellung von Informationen über den Dienst	notwendig	+	+
T12	Austausch von Kontakten im Falle von Betriebsstörungen	notwendig	+	+
Anforderungen an das Schwachstellenmanagement				
S01	Register von Schwachstellen	empfohlen	+	+
S02	Dokumentation einer Schwachstelle	notwendig	+	+
S03	Schwachstellennachverfolgung	notwendig	+	+
S04	Behebung einer Schwachstelle	notwendig	+	o
S05	Schnittstelle ins Risikomanagement	empfohlen	+	o

Tabelle 8.1.: Tabellarische Auflistung der Anforderungen und deren Erfüllungsgrad

Die in Abschnitt 5 definierten Prozesse erfüllen fast alle Anforderungen. Nur die Anforderungen *A05 - Automatisierung* und *A06 - Unterstützung manueller Aktivitäten* werden von den Prozessen naturgemäß nicht erfüllt. Die Anwendung erfüllt ebenfalls fast alle Anforderungen, lediglich Aktivitäten, die gezielt nicht abgebildet werden sollten, etwa *T08 - Genehmigungen einholen*, werden nicht unterstützt.

8.3. Zusammenfassung der Arbeit

Zur Ermittlung der Anforderungen wurden zunächst die zahlreichen Herausforderungen an Hochschulrechenzentren am Beispiel des LRZs beschrieben. Anschließend wurden die Anforderungen an ein Managementsystem für Sicherheitstests erläutert.

Nachdem die Anforderungen gestellt wurden, wurde nach bereits bestehenden Konzepten und Arbeiten zu einem solchen Managementsystem gesucht. Dabei wurden die Standards ISO 27002 und IT-Grundschutz näher betrachtet. Diese Standards beinhalteten allerdings keine Vorgehensweisen oder Anhaltspunkte für eine Umsetzung. Da keine bestehenden Konzepte zu einem Managementsystem für Sicherheitstests existierten, wurden Konzepte für die Teilbereiche (Dienstmanagement, Testmanagement und Schwachstellenmanagement) eines solchen Managementsystems gesucht. Dabei wurden für die einzelnen Teilbereiche Konzepte, die diese Teile für sich genommen gut umsetzen, gefunden und erläutert.

Anschließend wurden drei dieser Teilkonzepte vereinheitlicht und erweitert, sodass ein einheitliches Konzept für ein Managementsystem für Sicherheitstests entstand. Dieses Konzept kann nun von Hochschulrechenzentren genutzt werden, um die betriebenen Dienste regelmäßig auf ihre Sicherheit hin zu testen. Neben den Prozessen wurde des Weiteren eine Anwendung konzipiert, die diese Prozesse unterstützen soll und die Einführung eines solchen Managementsystems ressourcenschonend ermöglicht.

Nachdem das Konzept für die Planung und Durchführung eines Sicherheitstests, sowie die Schwachstellenbehebung und -nachverfolgung konzipiert wurden, wurde die Anwendung als Webanwendung implementiert.

Anschließend wurde das Konzept einmal prototypisch angewendet und ein Dienst im LRZ auf die Sicherheit überprüft. Dabei wurde das Konzept auf seine Durchführbarkeit getestet.

8.4. Ausblick

Das in dieser Arbeit entwickelte Konzept hat bei der ersten Anwendung gut funktioniert. Es sollte jedoch häufiger eingesetzt werden, um Verbesserungsmöglichkeiten zu finden. Die Anwendung kann noch weiterentwickelt werden. So fehlt momentan die Möglichkeit, Ansichten zu filtern oder zu sortieren. Zudem könnten weitere Informationen aus einem Configuration Management System importiert werden, um die Informationen für den Tester bereitzustellen und um die Anbindung von Schwachstellenscannern zu verbessern.

Das Konzept zielt derzeit mehr auf manuelle Sicherheitstests. Eine bessere Anbindung von Schwachstellenscannern wäre sehr wichtig, um gerade im Hochschulumfeld mit den beschränkten Ressourcen eine bessere Testabdeckung zu erzielen. Auch die Möglichkeit, Schwachstellenscans ohne Tester durchzuführen, wäre denkbar und bedarf weiterer Analyse.

In dieser Arbeit lag der Fokus auf technischen Schwachstellen. Prinzipiell sollten auch Sicherheitstests mit einem anderen Fokus mit diesem Konzept realisieren lassen. So lässt sich ein Test bzgl. Social-Engineering, mit Hilfe des beschriebenen Konzepts, planen und durchführen. Die Behandlung der organisatorischen Schwachstellen würde sich jedoch von den technischen Schwachstellen teilweise unterscheiden. Welche Anpassungen hierzu nötig wären, könnte ebenfalls weiter analysiert und ggf. implementiert werden.

A. Anhang

A.1. Kickoff Protokoll des Sicherheitstests zu LISC

Im Folgenden ist das Kickoff-Protokoll für den Sicherheitstest zum Dienst LISC zu sehen, welches im Rahmen des Abschnitts 7 entstanden ist.

Kickoff Protokoll

Allgemeines

Allgemeine Informationen

Dienstname:	Lisc
Ort:	Garching
Datum/Zeit:	28.09.16

Teilnehmer

Teilnehmer	Abteilung/Firma	Rolle
Frau Hanauer	LRZ	Dienstverantwortliche
Herr Wirtz	LMU	Tester

Testdetails

Allgemeines

Dienstname:	Lisc
Dienstverantwortlicher:	Hanauer, Tanja XXXXXX@lrz.de
Verantwortliche Abteilung:	LRZ
Tester:	Maximilian Wirtz XXXXXX@campus.lmu.de
Test Beginn:	05.10.2016
Test Ende:	05.10.2016

Testkategorisierung

Tester	intern
Informationsbasis	Black-Box
Aggressivität	vorsichtig
Umfang	fokussiert
Vorgehensweise	offensichtlich
Ausgangspunkt	von Innen

Testgegenstand

Dienste:	Lisc
IP-Adressen:	10.XXX.XXX.XXX

Weitere Informationen

Es wurden keine weiteren Informationen übermittelt.

Erlaubnis

Die Erlaubnis, die Systeme zu testen, wurde durch
Herrn Metzger
erteilt.

Testerinformationen

Tester IP-Adressen	Dynamische Adressvergabe über VPN VPN-Benutzername: XXXXXX@campus.lmu.de
Tester Benutzerkonten	XXXXXXX

A.2. Management Bericht über den Sicherheitstest zu LISC

Im Folgenden ist der Management Bericht des Sicherheitstest zum Dienst LISC zu sehen, welcher im Rahmen des Abschnitts 7 entstanden ist.

Sicherheitstestsreport

Anwendung: Lisc

Managementversion

1 Zusammenfassung

Die Anwendung machte einen sehr guten Eindruck. Es wurden keine ausnutzbaren Schwachstellen gefunden. Dennoch könnten an der Anwendung bzw. dem Webserver noch Konfigurationen geändert werden, um die Anwendung sicherer zu machen.

2 Übersicht

#	Titel	Kategorie	CVSS
1	Directory Listing aktiv	A5 - Security Misconfiguration	3.9
2	Cookies unzureichend abgesichert	A5 - Security Misconfiguration	1.6
3	Keine HSTS-Header	A5 - Security Misconfiguration	1.2
4	CSRF-Token manipulierbar	A5 - Security Misconfiguration	1.2

A.3. Entwickler Bericht über den Sicherheitstest zu LISC

Im Folgenden ist der Entwickler Bericht des Sicherheitstest zum Dienst LISC zu sehen, welcher im Rahmen des Abschnitts 7 entstanden ist.

Sicherheitstestsreport

Anwendung: Lisc

Entwicklerversion

1 Zusammenfassung

Die Anwendung machte einen sehr guten Eindruck. Es wurden keine ausnutzbaren Schwachstellen gefunden. Dennoch könnten an der Anwendung bzw. dem Webserver noch Konfigurationen geändert werden, um die Anwendung sicherer zu machen.

2 Übersicht

#	Titel	Kategorie	CVSS
1	Directory Listing aktiv	A5 - Security Misconfiguration	3.9
2	Cookies unzureichend abgesichert	A5 - Security Misconfiguration	1.6
3	Keine HSTS-Header	A5 - Security Misconfiguration	1.2
4	CSRF-Token manipulierbar	A5 - Security Misconfiguration	1.2

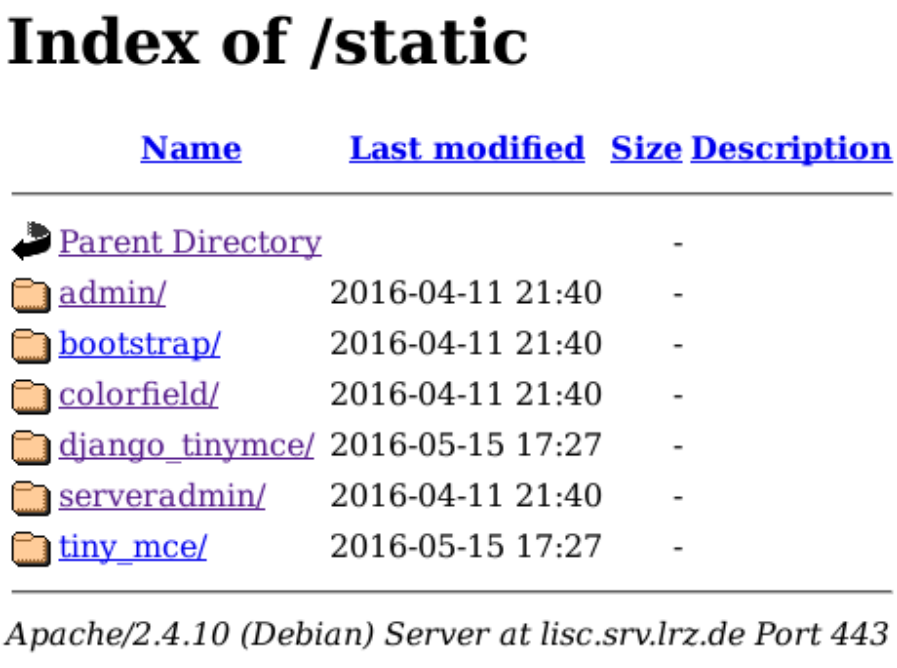
3 Schwachstellen

3.1 Directory Listing aktiv

3.1.1 Beschreibung

Durch Directory Listing kann ein Angreifer Informationen über die Anwendung erhalten, die es ihm vereinfacht, eine Schwachstelle zu finden. Des Weiteren können versehentlich sensible Informationen, etwa Konfigurationsdateien, dort gespeichert werden, die ein Angreifer dann ggf. auslesen kann.

3.1.2 Beweis



Index of /static

Name	Last modified	Size	Description
Parent Directory		-	
admin/	2016-04-11 21:40	-	
bootstrap/	2016-04-11 21:40	-	
colorfield/	2016-04-11 21:40	-	
django_tinymce/	2016-05-15 17:27	-	
serveradmin/	2016-04-11 21:40	-	
tiny_mce/	2016-05-15 17:27	-	

Apache/2.4.10 (Debian) Server at lisc.srv.lrz.de Port 443

https://lisc.srv.lrz.de/static/

3.1.3 Tester-Empfehlung

Deaktivieren des Directory Listings, z.B. folgendermaßen:
Options -Indexes FollowSymLinks MultiViews

3.1.4 Empfehlung für A5 - Security Misconfiguration

Alle folgenden Empfehlungen sollten berücksichtigt werden:
1. Ein wiederholbarer Härtungsprozess, der eine schnelle und einfache Verteilung einer neuen, abgesicherten Umgebung erlaubt. Entwicklungs-, QA-, und Produktionsumgebungen sollten identisch konfiguriert sein (mit unterschiedlichen Passwörtern pro Umgebung). Der Prozess sollte automatisiert sein, um den nötigen Aufwand bei Erstellung einer neuen, sicheren Umgebung zu minimieren.
2. Ein Prozess, der zeitnah neuentwickelte Softwareupdates auf allen ausgerollten Umgebungen ermöglicht. Davon sind auch alle Bibliotheken und Komponenten betroffen.
3. Eine robuste Anwendungsarchitektur, mit effektiver und sicherer Trennung und Absicherung einzelner Komponenten.
4. Periodisch durchgeführte Tests und Audits helfen zukünftige Fehlkonfigurationen oder fehlende Patches zu erkennen und zu vermeiden.

3.1.5 Sonstiges

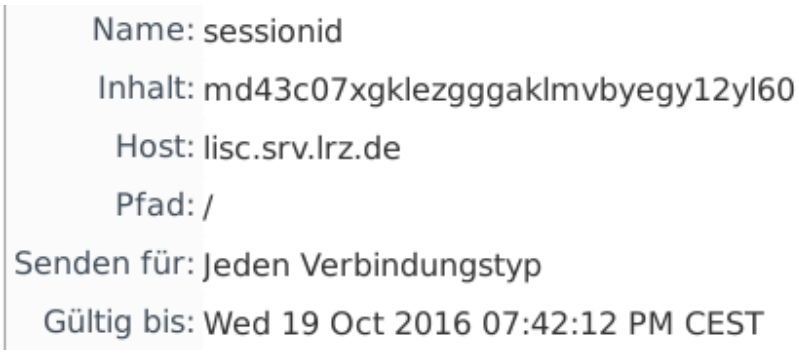
Kategorie	A5 - Security Misconfiguration
CVSS3	3.9/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

3.2 Cookies unzureichend abgesichert

3.2.1 Beschreibung

Die Cookies haben eine zu lange Lebensdauer. Der Session-Token ist 14 Tage gültig und der CSRF-Token ein Jahr. Des Weiteren sind die Cookie-Flags secure und http-only nicht gesetzt, was einem Angreifer Cookie-Diebstahl vereinfachen kann.

3.2.2 Beweis



3.2.3 Tester-Empfehlung

Es sollte geprüft werden, wie lange eine Session dauern soll. Für die Cookie-Haltbarkeit wird eine negative Haltbarkeit empfohlen, der Cookie würde somit bei einem Browserneustart gelöscht werden. Auch sollten die Cookies mit den Flags secure und http-only versehen werden, da dies andere Angriffe erschweren kann.

3.2.4 Empfehlung für A5 - Security Misconfiguration

Alle folgenden Empfehlungen sollten berücksichtigt werden:
1. Ein wiederholbarer Härtungsprozess, der eine schnelle und einfache Verteilung einer neuen, abgesicherten Umgebung erlaubt. Entwicklungs-, QA-, und Produktionsumgebungen sollten identisch konfiguriert sein (mit unterschiedlichen Passwörtern pro Umgebung). Der Prozess sollte automatisiert sein, um den nötigen Aufwand bei Erstellung einer neuen, sicheren Umgebung zu minimieren.
2. Ein Prozess, der zeitnah neuentwickelte Softwareupdates auf allen ausgerollten Umgebungen ermöglicht. Davon sind auch alle Bibliotheken und Komponenten betroffen.
3. Eine robuste Anwendungsarchitektur, mit effektiver und sicherer Trennung und Absicherung einzelner Komponenten.
4. Periodisch durchgeführte Tests und Audits helfen zukünftige Fehlkonfigurationen oder fehlende Patches zu erkennen und zu vermeiden.

3.2.5 Sonstiges

Kategorie	A5 - Security Misconfiguration
CVSS3	1.6/AV:N/AC:H/PR:N/UI:R/S:U/C:N/I:N/A:N

3.3 Keine HSTS-Header

3.3.1 Beschreibung

Bei den Servern: 10.156.108.8
sind die HSTS-Header nicht gesetzt.

3.3.2 Beweis

SSLyze Scan:
"http_headers": {
"hpkp_header": null,
"hsts_header": null,
"is_backup_pin_configured": null,
"is_valid_pin_configured": null,
"plugin_options": {},
"verified_certificate_chain": null
}

3.3.3 Tester-Empfehlung

Auf den Servern HSTS aktivieren, bspw. durch mod_headers.

3.3.4 Empfehlung für A5 - Security Misconfiguration

Alle folgenden Empfehlungen sollten berücksichtigt werden:
1. Ein wiederholbarer Härtungsprozess, der eine schnelle und einfache Verteilung einer neuen, abgesicherten Umgebung erlaubt. Entwicklungs-, QA-, und Produktionsumgebungen sollten identisch konfiguriert sein (mit unterschiedlichen Passwörtern pro Umgebung). Der Prozess sollte automatisiert sein, um den nötigen Aufwand bei Erstellung einer neuen, sicheren Umgebung zu minimieren.
2. Ein Prozess, der zeitnah neuentwickelte Softwareupdates auf allen ausgerollten Umgebungen ermöglicht. Davon sind auch alle Bibliotheken und Komponenten betroffen.
3. Eine robuste Anwendungsarchitektur, mit effektiver und sicherer Trennung und Absicherung einzelner Komponenten.
4. Periodisch durchgeführte Tests und Audits helfen zukünftige Fehlkonfigurationen oder fehlende Patches zu erkennen und zu vermeiden.

3.3.5 Sonstiges

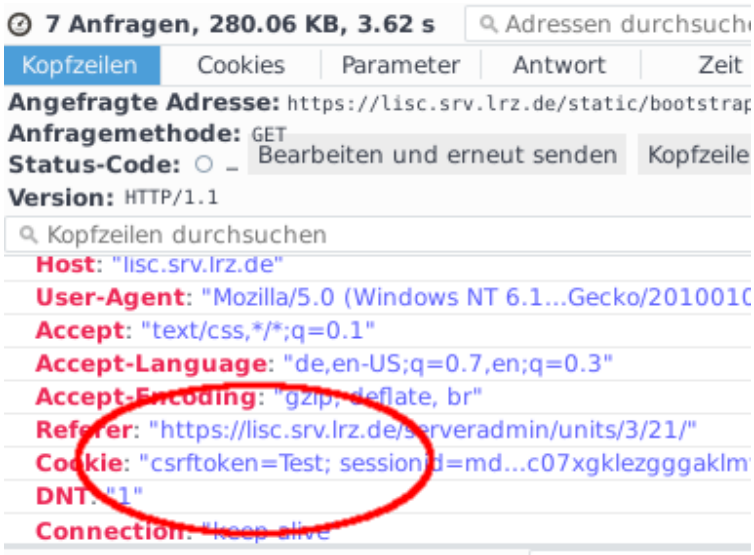
Kategorie	A5 - Security Misconfiguration
CVSS3	1.2/AV:A/AC:H/PR:N/UI:R/S:U/C:N/I:N/A:N

3.4 CSRF-Token manipulierbar

3.4.1 Beschreibung

Es ist möglich, den CSRF-Token im Cookie selbst zu setzen. Ein Angreifer könnte den Token bspw. mittels anderer XSS-Schwachstelle auf einen bestimmten Wert ändern, sodass anschließend CSRF-Angriffe möglich sind.

3.4.2 Beweis



3.4.3 Tester-Empfehlung

Der CSRF-Token sollte vom Server generiert werden und nicht vom User veränderbar sein.

3.4.4 Empfehlung für A5 - Security Misconfiguration

Alle folgenden Empfehlungen sollten berücksichtigt werden:
1. Ein wiederholbarer Härtungsprozess, der eine schnelle und einfache Verteilung einer neuen, abgesicherten Umgebung erlaubt. Entwicklungs-, QA-, und Produktionsumgebungen sollten identisch konfiguriert sein (mit unterschiedlichen Passwörtern pro Umgebung). Der Prozess sollte automatisiert sein, um den nötigen Aufwand bei Erstellung einer neuen, sicheren Umgebung zu minimieren.
2. Ein Prozess, der zeitnah neuentwickelte Softwareupdates auf allen ausgerollten Umgebungen ermöglicht. Davon sind auch alle Bibliotheken und Komponenten betroffen.
3. Eine robuste Anwendungsarchitektur, mit effektiver und sicherer Trennung und Absicherung einzelner Komponenten.
4. Periodisch durchgeführte Tests und Audits helfen zukünftige Fehlkonfigurationen oder fehlende Patches zu erkennen und zu vermeiden.

3.4.5 Sonstiges

Kategorie	A5 - Security Misconfiguration
CVSS3	1.2/AV:N/AC:H/PR:L/UI:R/S:U/C:N/I:N/A:N

A.4. Administratoren Bericht über den Sicherheitstest zu LISC

Im Folgenden ist der Administratoren Bericht des Sicherheitstest zum Dienst LISC zu sehen, welcher im Rahmen des Abschnitts 7 entstanden ist.

Sicherheitstestsreport

Anwendung: Lisc

Administratorenversion

1 Zusammenfassung

Die Anwendung machte einen sehr guten Eindruck. Es wurden keine ausnutzbaren Schwachstellen gefunden. Dennoch könnten an der Anwendung bzw. dem Webserver noch Konfigurationen geändert werden, um die Anwendung sicherer zu machen.

2 Übersicht

#	Titel	Kategorie	CVSS
1	Directory Listing aktiv	A5 - Security Misconfiguration	3.9
2	Cookies unzureichend abgesichert	A5 - Security Misconfiguration	1.6
3	Keine HSTS-Header	A5 - Security Misconfiguration	1.2
4	CSRF-Token manipulierbar	A5 - Security Misconfiguration	1.2

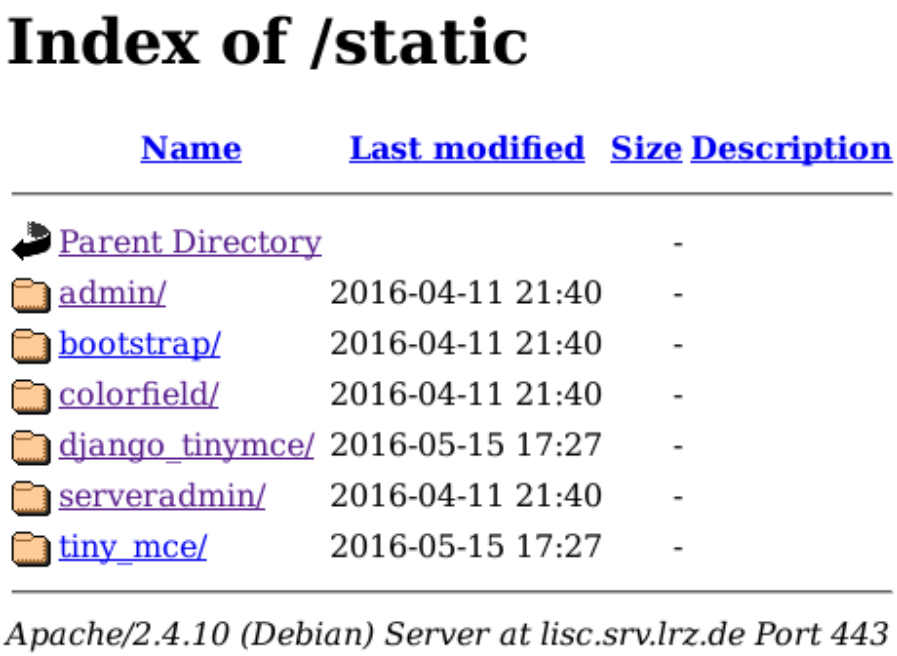
3 Schwachstellen

3.1 Directory Listing aktiv

3.1.1 Beschreibung

Durch Directory Listing kann ein Angreifer Informationen über die Anwendung erhalten, die es ihm vereinfacht, eine Schwachstelle zu finden. Des Weiteren können versehentlich sensible Informationen, etwa Konfigurationsdateien, dort gespeichert werden, die ein Angreifer dann ggf. auslesen kann.

3.1.2 Beweis



Index of /static

Name	Last modified	Size	Description
Parent Directory		-	
admin/	2016-04-11 21:40	-	
bootstrap/	2016-04-11 21:40	-	
colorfield/	2016-04-11 21:40	-	
django_tinymce/	2016-05-15 17:27	-	
serveradmin/	2016-04-11 21:40	-	
tiny_mce/	2016-05-15 17:27	-	

Apache/2.4.10 (Debian) Server at lisc.srv.lrz.de Port 443

https://lisc.srv.lrz.de/static/

3.1.3 Tester-Empfehlung

Deaktivieren des Directory Listings, z.B. folgendermaßen:
Options -Indexes FollowSymLinks MultiViews

3.1.4 Empfehlung für A5 - Security Misconfiguration

Alle folgenden Empfehlungen sollten berücksichtigt werden:
1. Stellen Sie sicher, dass alle Einstellungen für jede Funktionalität auf die sicherst mögliche Option eingestellt ist.
2. Ändern Sie Konfigurationen nicht für eventuell kommende Neuerungen.
3. Regelmäßiger Review der Konfigurationen.

3.1.5 Sonstiges

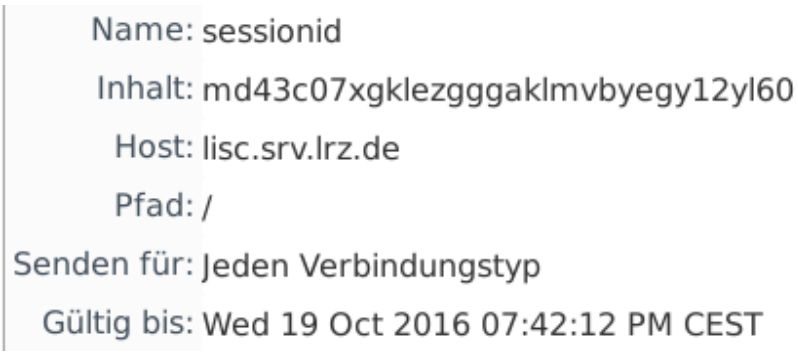
Kategorie	A5 - Security Misconfiguration
CVSS3	3.9/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

3.2 Cookies unzureichend abgesichert

3.2.1 Beschreibung

Die Cookies haben eine zu lange Lebensdauer. Der Session-Token ist 14 Tage gültig und der CSRF-Token ein Jahr. Des Weiteren sind die Cookie-Flags secure und http-only nicht gesetzt, was einem Angreifer Cookie-Diebstahl vereinfachen kann.

3.2.2 Beweis



3.2.3 Tester-Empfehlung

Es sollte geprüft werden, wie lange eine Session dauern soll. Für die Cookie-Haltbarkeit wird eine negative Haltbarkeit empfohlen, der Cookie würde somit bei einem Browserneustart gelöscht werden. Auch sollten die Cookies mit den Flags secure und http-only versehen werden, da dies andere Angriffe erschweren kann.

3.2.4 Empfehlung für A5 - Security Misconfiguration

Alle folgenden Empfehlungen sollten berücksichtigt werden:
1. Stellen Sie sicher, dass alle Einstellungen für jede Funktionalität auf die sicherst mögliche Option eingestellt ist.
2. Ändern Sie Konfigurationen nicht für eventuell kommende Neuerungen.
3. Regelmäßiger Review der Konfigurationen.

3.2.5 Sonstiges

Kategorie	A5 - Security Misconfiguration
CVSS3	1.6/AV:N/AC:H/PR:N/UI:R/S:U/C:N/I:N/A:N

3.3 Keine HSTS-Header

3.3.1 Beschreibung

Bei den Servern: 10.156.108.8
sind die HSTS-Header nicht gesetzt.

3.3.2 Beweis

SSLyze Scan:
"http_headers": {
"hpkp_header": null,
"hsts_header": null,
"is_backup_pin_configured": null,
"is_valid_pin_configured": null,
"plugin_options": {},
"verified_certificate_chain": null
}

3.3.3 Tester-Empfehlung

Auf den Servern HSTS aktivieren, bspw. durch mod_headers.

3.3.4 Empfehlung für A5 - Security Misconfiguration

Alle folgenden Empfehlungen sollten berücksichtigt werden:
1. Stellen Sie sicher, dass alle Einstellungen für jede Funktionalität auf die sicherst mögliche Option eingestellt ist.
2. Ändern Sie Konfigurationen nicht für eventuell kommende Neuerungen.
3. Regelmäßiger Review der Konfigurationen.

3.3.5 Sonstiges

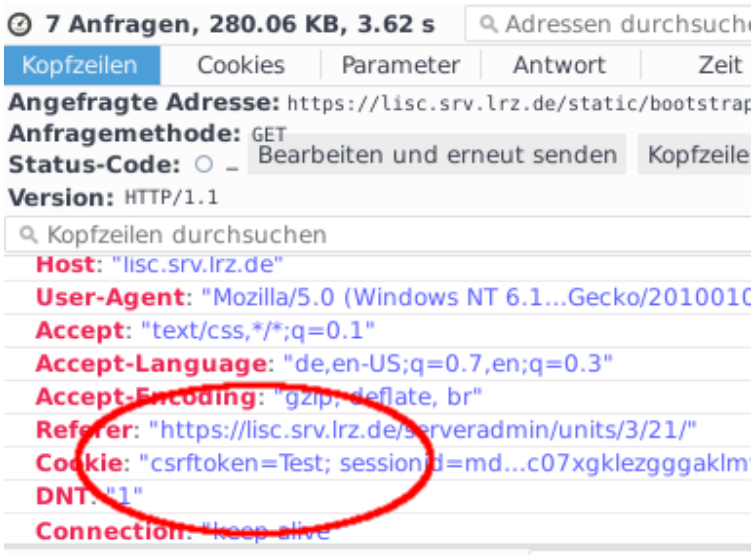
Kategorie	A5 - Security Misconfiguration
CVSS3	1.2/AV:A/AC:H/PR:N/UI:R/S:U/C:N/I:N/A:N

3.4 CSRF-Token manipulierbar

3.4.1 Beschreibung

Es ist möglich, den CSRF-Token im Cookie selbst zu setzen. Ein Angreifer könnte den Token bspw. mittels anderer XSS-Schwachstelle auf einen bestimmten Wert ändern, sodass anschließend CSRF-Angriffe möglich sind.

3.4.2 Beweis



3.4.3 Tester-Empfehlung

Der CSRF-Token sollte vom Server generiert werden und nicht vom User veränderbar sein.

3.4.4 Empfehlung für A5 - Security Misconfiguration

Alle folgenden Empfehlungen sollten berücksichtigt werden:
1. Stellen Sie sicher, dass alle Einstellungen für jede Funktionalität auf die sicherst mögliche Option eingestellt ist.

2. Ändern Sie Konfigurationen nicht für eventuell kommende Neuerungen.
3. Regelmäßiger Review der Konfigurationen.

3.4.5 Sonstiges

Kategorie	A5 - Security Misconfiguration
CVSS3	1.2/AV:N/AC:H/PR:L/UI:R/S:U/C:N/I:N/A:N

A.5. Inhalt der CD

Auf der beiliegenden CD sind drei Verzeichnisse zu finden:

- **Dokumentation**: Mit dem Latex-Quelltext dieser Arbeit.
- **Folien**: Mit den Präsentationen des Antritts- und Abschlussvortrags.
- **Sourcen**: Mit den Quelltexten der entwickelten Anwendung.

Um die Anwendung zu installieren, bzw. diese auszuführen sind mehrere Abhängigkeiten zu erfüllen. Im Folgenden werden die Installationsschritte erläutert um die Anwendung auszuführen. Als Umgebung wurde ein Ubuntu Server in der Version 16.04 LTS verwendet.

Die meisten Abhängigkeiten können mittels Python-Packetmanager (PIP) installiert werden, dieser lässt sich mittels *sudo apt-get install python3-pip* installieren. Im Ordner Sourcen befindet sich eine *requirements.txt* in der aller benötigten Pakete aufgelistet sind. Am schnellsten geht die Installation mittels PIP: *sudo pip3 install -r requirements.txt*.

Nachdem alle Abhängigkeiten installiert wurden muss noch die Konfigurationsdatei angepasst werden. Dazu liegt eine Vorlage *zoxdocd10.conf.dist* bereit. Diese muss als *zoxdocd10.conf* umbenannt werden und die Eintäge die mit *changeme* markiert sind müssen angepasst werden. Dazu bitte die Pfade anpassen, sodass diese absolut sind.

Um die Datenbank zu erzeugen und mit Daten zu befüllen, muss das Shell-Skript *make_test_db.sh* ausgeführt werden. Anschließend kann die *run.py* ausgeführt werden und ein lokaler Webserver wird auf Port 5000 gestartet. Nun kann ein beliebiger Browser verwendet werden um die URL **http://localhost:5000** aufzurufen. Um sich anzumelden muss als Benutzername **admin@foo.de** und ein beliebiges Passwort verwendet werden (das Passwort wird nicht geprüft). Sollte die Anwendung von einem anderen Rechner geöffnet werden wollen, kann in der *run.py* der Parameter *host* auf *0.0.0.0* geändert werden.

Literaturverzeichnis

[Adr15] ADRIAN, DAVID, KARTHIKEYAN BHARGAVAN, ZAKIR DURUMERIC, PIERRICK GAUDRY, MATTHEW GREEN, J. ALEX HALDERMAN, NADIA HENINGER, DREW SPRINGALL, EMMANUEL THOMÉ, LUKE VALENTA, BENJAMIN VANDERSLOOT, ERIC WUSTROW, SANTIAGO ZANELLA-BÉGUELIN und PAUL ZIMMERMANN: *Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice*. In: *22nd ACM Conference on Computer and Communications Security*, Oktober 2015.

[All15] CYBER-SICHERHEIT, ALLIANZ FÜR: *Cyber-Sicherheits-Umfrage 2015*, 2015. https://www.allianz-fuer-cybersicherheit.de/ACS/DE/_/downloads/cybersicherheitslage/umfrage2015_ergebnisse.pdf Abruf: 18.03.16.

[BSI03] BSI, HRSG.: *Studie Durchführungskonzept für Penetrationstests*, 2003. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Studien/Penetrationstest/penetrationstest_pdf.pdf?__blob=publicationFile Abruf: 11.03.16.

[BSI08] *BSI-Standard 100-1*. Technischer Bericht, Bonn, Germany, 2008.

[BSI14] *Ein Praxis-Leitfaden für IS-Penetrationstests*, 2014.

[BSI16] *IT-Grundschutz-Kataloge*. Technischer Bericht, Bonn, Germany, 2016.

[BUR] LTD., PORTSWIGGER: *Burp Suite*. https://portswigger.net/burp/ Abruf: 11.10.16.

[DFN07] *DFN-Infobrief Recht 07*. Verein zur Förderung eines Deutschen Forschungsnetzes e. V., 2016.

[Eik13] EIKENBERG, R.: *Bundestags-Hack: Angreifer sollen gigabyteweise E-Mails kopiert haben*, Juli 2013. http://heise.de/-1914004 Abruf: 03.07.16.

[FIR] *Common Vulnerability Scoring System v3.0: Specification Document*. https://www.first.org/cvss/cvss-v30-specification-v1.7.pdf Abruf: 13.09.16.

[FLA] *Welcome*. http://flask.pocoo.org/ Abruf: 12.09.16.

[HM] *Porträt Hochschule München*. https://www.hm.edu/allgemein/hochschule_muenchen/portraet/index.de.html Abruf: 17.09.16.

[HSW] *Hochschulprofil*. https://www.hswt.de/hochschule/hochschulprofil.html Abruf: 17.09.16.

[ISO13] *ISO/IEC 27002:2005 - Information technology – Security techniques – Code of practice for information security management*. Technischer Bericht, 2013. http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=54533.

[Kni12] KNITTL, SILVIA, ACHIM GRINDLER und KARMELA VELLGUTH: *IT-Servicemanagement Rahmenwerke-wie sie sinnvoll in Universitäten einsetzbar sind-*. In: *DFN-Forum Kommunikationstechnologien*, Seiten 85–94, 2012.

[Lac07] MACFARLANE, SHIRLEY LACY; IVOR: *Service Transition Book*. The Stationery Office, 2007.

[LMU15] MÜNCHEN, LUDWIG-MAXIMILIANS-UNIVERSITÄT: *Zahlen und Fakten*, 2015. http://www.uni-muenchen.de/ueber_die_lmu/zahlen_fakten/index.html Abruf: 17.08.16.

[LRZ] *Geschichte des Leibniz-Rechenzentrums*. http://www.lrz.de/wir/geschichte/ Abruf: 17.09.16.

[Men15] MENZ, N, ET. AL.: *SAFETY UND SECURITY AUS DEM BLICKWINKEL DER ÖFFENTLICHEN IT*, 2015. https://www.oeffentliche-it.de/documents/10181/14412/Safety+und+Security+aus+dem+Blickwinkel+der+%C3%B6ffentlichen+IT Abruf: 05.06.16.

[MIT] CORP., MITRE: *Corporate Overview*. https://www.mitre.org/about/our-history Abruf: 16.08.16.

[OWA13] FOUNDATION, HRSG. THE OWASP: *OWASP Top 10 - 2013*, 2013. http://owasptop10.googlecode.com/files/OWASP%20Top%2010%20-%202013.pdf Abruf: 18.04.16.

[OWA16] FOUNDATION, THE OWASP: *About The Open Web Application Security Project*, 2016. https://www.owasp.org/index.php/About_OWASP Abruf: 16.08.16.

[Pal13] PALMAERS, TOM: *Implementing a vulnerability management process*, 3 2013. https://www.sans.org/reading-room/whitepapers/threats/implementing-vulnerability-management-process-34180.

[Rai08] RAISON, A. VON: *Barrierefreie Dokumente beim BSI*, 2008. https://heise.de/-210310 Abruf: 03.10.16.

[Rät02] RÄTZMANN, MANFRED: *Software-Testing - Rapid Application Testing, Softwaretest, Agiles Qualitätsmanagement (Galileo Computing)*. Galileo Computing, 2002.

[Sch15] SCHERSCHEL, F.: *Bundestags-Hack: Angreifer sollen gigabyteweise E-Mails kopiert haben*, 2015. http://heise.de/-2715881 Abruf: 18.03.16.

[Sok16] SOKOLOV, D.: *Milliarden-Coup in NY: Zentralbank-Konto per Überweisung geleert*, 2016. http://heise.de/-3131832 Abruf: 18.03.16.

[Spi14] SPILLNER, ANDREAS UND LINZ, TILO: *Praxiswissen Softwaretest - Testmanagement*. Dpunkt.Verlag GmbH, 2014.

[SQL16] SQLMAP: *Automatic SQL injection and database takeover tool*. http://sqlmap.org/ Abruf: 16.08.16.

[Sto02] STONEBURNER, GARY, ALICE Y. GOGUEN und ALEXIS FERINGA: *SP 800-30. Risk Management Guide for Information Technology Systems*. Technischer Bericht, Gaithersburg, MD, United States, 2002.

[SYM15] CORPORATION, SYMANTEC: *INTERNET SECURITY THREAT REPORT*, 2015. https://www4.symantec.com/mktginfo/whitepaper/ISTR/21347932_GA-internet-security-threat-report-volume-20-2015-social_v2.pdf Abruf: 18.03.16.

[TUM] *Die TUM in Zahlen*. http://www.tum.de/die-tum/die-universitaet/die-tum-in-zahlen/ Abruf: 17.09.16.