- Bachelorthesis -

„Penetrationtesting von Webapplikationen“

zur Erlangung des akademischen Grades eines
Bachelor of Science (B.Sc.)

erarbeitet von
Herrn Paul Böhme (Matr.-Nr. 22336)

vorgelegt **der Hochschule Merseburg (FH)**
Fachbereich Ingenieur- und Naturwissenschaften

Begutachtet von **Herrn Prof. Dr. rer. nat. Uwe Heuert**
Professor für Rechnernetze und Virtuelle Instrumentierung

Herrn Prof. Dr. Andreas Spillner
Professor für Mathematik/Diskrete und Computergestützte Mathematik

Studienrichtung Ingenieur- und Naturwissenschaften
Studiengang Angewandte Informatik
Matrikel BAIN15

Leipzig, 10.09.2019

Inhaltsverzeichnis

1. Einleitung 6
1.1 Motivation 6
1.2 Gliederung der Bachelorarbeit 6
2. Theoretische Grundlagen des „Penetrationtesting“ 7
2.1 Begriffsklärung „Hacking” 7
2.1.1 Arten von Hackern 8
2.2 Was ist „Penetrationtesting“ 9
2.2.1 Ablauf eines Penetrationtests 10
2.2.2 Hacking-Tools 16
2.3 Sicherheitsüberprüfung 19
2.3.1 Arten der Sicherheitsüberprüfung 19
2.3.2 Typen der Testdurchführung 23
3. Web-Applikationen: Angriffsvektoren 25
3.1 Clickjacking 25
3.2 (Cookie)-Replay-Angriff 26
3.3 Cross Side Request Forgery 28
3.4 Cross-Site Scripting (XSS) 30
3.5 File Inclusion 32
3.6 Man-in-the-Middle 34
3.7 Nullbyte-Injection 36
3.8 Session Fixation 37
3.9 Session Hijacking 38
3.10 SQL-Injection 40
4. Simulation von Angriffsszenarien 46
4.1 Szenario #1: SQL-Injection 46
4.1.1 SQL-Injection – Extrahierung von Benutzerdaten 46
4.1.2 SQL-Injection – Authentication Bypass 50
4.1.3 SQL-Injection – Timing-Attack 53
4.2 Szenario #1: Cross-Site-Scripting 56
4.2.1 Cross-Site-Scripting – DOM-based 56
4.2.2 Cross-Site-Scripting – Reflektiert 58
4.2.3 Cross-Sites-Scripting – Stored 60

5. Schlusswort/Fazit 63
Anhang 64
Quellenverzeichnis 64

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit “Penetrationtesting von Webapplikationen” selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Leipzig, den 10.09.2019

.....
Paul Böhme

1. Einleitung

1.1 Motivation

Unsere moderne Industrie ist durch den dauerhaften technischen Wandel geprägt und den sich ständig ändernden Umständen ausgesetzt. Dies gilt letzten Endes auch für den Bereich der IT-Sicherheit. Mit der immer weiter voranschreitenden Vernetzung der Welt steigt auch die anfallende Datenmenge, die vor den Händen der falschen Personen zu schützen ist. Nicht zuletzt wurden im Januar 2019 durch einen Hackerangriff private Daten von 994 deutschen Politikern und Prominenten veröffentlicht. Dies löste hitzige Diskussionen aus, die heutige Maßnahmen zur Sicherung von sensiblen Datenmengen vor fremden Einwirken stark kritisierten.

Damit kritische Daten und IT-Systeme von äußeren Angriffen geschützt sind, können Penetrationstests durchgeführt werden. Mit deren Hilfe kann gezielt nach Schwachstellen in den eingesetzten Systemen gesucht werden, um auf proaktive Weise zukünftige Angriffe abzuwehren oder deren Wirkung abzuschwächen.

Diese Arbeit soll daher als Einführung für die Arbeit in dem Bereich “Penetrationtests” darstellen. Neben der allgemeinen Durchführung sollen auch die rechtlichen Grundlagen aufgezeigt werden, mit der sich die Tester in der Planungsphase auseinandersetzen müssen. Zudem werden die von Hackern häufig verwendeten Angriffe aufgelistet und erklärt. Um einen praktischen Einblick in diese Tests zu erhalten, werden mehrere Szenarios präsentiert, bei denen es sich jedoch nicht um eine Anleitung für “Hacker” handeln soll.

1.2 Gliederung der Bachelorarbeit

Kapitel 2 soll eine Grundlage für die Diskussion mit den Themengebieten “Penetrationtesting” und “Hacking” schaffen. Neben dem Ablauf eines allgemeinen Penetrationtests wird auch die rechtliche Grundlage behandelt, die in die Planung solcher Tests mit einbezogen werden muss.
Kapitel 3 befasst sich mit den verschiedenen Angriffsvektoren. Es wird neben der Erklärung der Funktionsweise der einzelnen Angriffe, auch auf mögliche Gegenmaßnahmen Bezug genommen. Wenn möglich, wurde die Wirkungsweise des Angriffsvektors durch ein Beispiel verdeutlicht.
Im Kapitel 4, und damit auch dem letzten Kapitel, wurden mehrere Angriffsszenarien dokumentiert, um die praktische Durchführung eines Angriffes aufzuzeigen.

2. Theoretische Grundlagen des „Penetrationtesting“

In diesem Kapitel erfolgt eine Einführung in das umfangreiche Gebiet des „Penetrationtesting“, welches sich unter anderem mit dem Thema „Hacking“ befasst.

Im Verlauf dieses Kapitels werden unter anderen folgenden Fragen beantwortet:

- “Was ist Penetrationtesting?”
- Was ist Hacking? Welche Arten von Hackern gibt es?
- Welche Tools werden von Hackern genutzt?
- Welche Gesetze sind für das Thema “Penetrationtesting” relevant?

2.1 Begriffsklärung „Hacking”

Die Definition des Terms „Hacking“ ist sehr vielfältig. Es umfasst eine Vielzahl an unterschiedlichsten Aktivitäten. Generell befasst sich dieses Themengebiet mit dem unbefugten Eindringen in Computer-Systeme, jedoch ist dies nur ein kleiner Bereich dessen, was man darunter verstehen kann.
Im Folgenden sind einige Beispiele aufgelistet, um darüber aufzuklären, wie breit der Definitionsbereich wirklich ist:

- Sogenannte „Life-Hacks“, welche die Funktionsweise alltäglicher Dinge verändern, um imperfekte, unschöne jedoch oftmals praktische Alternativlösungen zu bieten bzw. Produktivität und Effizienz erhöhen.
- Dächer- und Tunnel-Hacking beschreibt die unautorisierte Erkundung von Dächern und Techniktunneln.
- Der Vorgang, durch einfallsreiche Experimentierfreude neue Funktionen aus technischen Geräten zu gewinnen. Beispiel: Bau einer Klingel aus einer Festplatte

Richten wir uns nun nach Wikipedias Definition, handelt es sich beim Hacking um eine Maßnahme, Sicherheitslücken in einem System zu finden und diese zu umgehen. Durch Hacking wird versucht, an Daten auf unvorhergesehenen Weg zu kommen und diese zu verändern, zu manipulieren oder zu zerstören [Wikipedia;1].

Hacking wird in den Medien meist in einem negativen Licht dargestellt. Dies entspricht jedoch nicht der Wahrheit, beim Hacking handelt es sich eher um eine wertneutrale Methode. Wie diese Methode letzten Endes angewandt wird spielt für die Zuweisung moralischer Werte eine wichtige Rolle. Hacking kann entweder zur Erhöhung der Sicherheit von Computer-Systemen beitragen oder für die persönliche bzw. finanzielle Bereicherung verwendet werden [Kofler, et al.;1].

2.1.1 Arten von Hackern

Wird im Volksmund über den Hacker geredet, so wird oftmals ein Krimineller beschrieben, der ohne große Mühe Datensätze von Banken oder Kreditkarten-Unternehmen mithilfe von mysteriösen Werkzeugen an sich reißen kann. Vegh argumentiert, dass die Medien eine erhebliche Rolle in der Schaffung dieses negativen Bildes des Hackers spielen. Durch die verzerrte Darstellung von politischem Aktivismus als Cyberterrorismus wird die Meinung der Öffentlichkeit sowie die zukünftige Gesetzeslage negativ beeinflusst [Vegh;1].

Der Begriff „Hacker" besitzt seinen eigenen historischen Verlauf, dessen Bedeutung von Generation zu Generation unterschiedlich ausfällt [Taylor;1]. In der Regel kann der Hacker als eine technisch versierte Person bezeichnet werden, dessen Ziel es ist, Schwachstellen in IT-Systemen aufzudecken und Angriffe zu entwickeln, die diese Schwachstellen ausnutzen [Eckert;1]. Um einen Hacker jetzt aber in eine der vielen Kategorien einteilen zu können, muss dieser eine moralische Entscheidung treffen: Entweder er nutzt das akkumulierte Wissen um sich persönlich bzw. finanziell zu bereichern, oder er lässt diese Informationen in Vergessenheit geraten. Eine dritte Option wäre den Systemadministrator oder das Unternehmen über die Sicherheitslücken zu informieren.

Natürlich gibt es auch für Hacker Regeln. So verbietet das deutsche Gesetz jegliche unbefugte Datenmanipulation, sowie den Versuch in ein Computer-System einzudringen. Außerdem hat die Hacking-Community eigens Ethik-Regeln aufgestellt. Es gibt jedoch keinen internationalen Standard und hängt eher vom kulturellen und politischen Kontext ab. Berücksichtigt man diese Aspekte, kann man Hacker in drei Gruppen unterteilen, deren Grenzen nicht immer exakt gezogen werden können [Kofler, et al.;2].

2.1.1.1 White-Hat-Hacker

Der „White-Hat-Hacker" wird oft auch als ethischer Hacker oder Computer-Sicherheitsexperte bezeichnet, welcher Penetrationtests und andere Sicherheitsüberprüfungen (Test-Methoden) durchführt und die Sicherheit der Computer-Systeme eines Unternehmens gewährleistet [searchsecurity;1].

White-Hat-Hacker sind oft Experten in ihrem Bereich – sei es Penetrationtesting, System- und Netzwerk-Analyse oder Cyberangriffs-Prävention – und verwenden die gleichen Hacking-Methoden wie Black-Hat-Hacker. Diese Methoden reichen vom Penetrationtesting, über die Arbeit mit Hacking-Tools, bis hin zum „Social Engineering". Jedoch gibt es einen wesentlichen Unterschied – diese testen das System mit der Genehmigung des Besitzers bzw. des Unternehmens, was den Prozess überhaupt erst

legal macht. Das „EC-Council“, sowie weitere Organisationen, entwickelten Zertifikate, Lernsoftware, Kurse und Online-Trainingseinheiten, welches das vielfältige Thema des „Ethical Hackings“ abdeckt [EC-Council;1]

2.1.1.2 Black-Hat-Hacker

Ein "Black-Hat-Hacker” ist ein Hacker, der “die Sicherheit eines Systems verletzt, sei es aus Böswilligkeit oder zur persönlichen Bereicherung" [Moore;1]. Mit ihrem umfangreichen Wissen über Computernetzwerke und Sicherheitsprotokolle nutzen sie gezielt die Schwachstellen eines Systems aus, um sich Zugriff zu diesem zu verschaffen.

Die primäre Motivation dieser Hacker liegt üblicherweise in der persönlichen oder finanziellen Bereicherung. Auch können diese in Cyber-Spionage und Protesten verwickelt sein, oder der Nervenkitzel des Cyberverbrechens stellt einen Reiz dar. Die Expertise eines Black-Hat-Hackers reichen vom einfachen Amateur, welche Malware verbreitet, bis hin zum erfahrenen Hacker, der gezielt Daten finanzieller oder persönlicher Natur stiehlt. Diese Daten werden jedoch nicht nur gestohlen, sondern auch modifiziert und zerstört [Norton;1].

2.1.1.3 Grey-Hat-Hacker

Der „Grey-Hat-Hacker“ bezeichnet einen Hacker, der zwar beim Hacking gegen Gesetze oder ethische Standarte verstößt, dies jedoch nicht aus böswillig Absichten tut. Vielmehr wird die Methode von Hackern genutzt, die etwa einem Unternehmen auf mögliche Schwachstellen in einem System oder einer Software hinweisen möchten – um mögliche zukünftige Black-Hat-Hackerangriffe zu verhindern – jedoch nicht über die vorher eingeholte Erlaubnis verfügen [Harper, et al.;2].

Obwohl ein Grey-Hat-Hacker keine böswilligen Absichten verfolgt, handelt es sich bei dieser Aktion um eine illegale Tätigkeit. Somit kann der Grey-Hat zwischen den White-Hat und dem Black-Hat platziert werden. Eine klare Unterscheidung, ob der Grey-Hat gut oder böse sei, ist, wie der Name bereits andeutet, sehr schwierig bzw. von den eigenen gesellschaftlichen oder politischen Positionen abhängig. Ein Grey-Hat-Hacker würde jedoch niemals eine Schwachstelle im System illegal ausnutzen oder anderen über das „how-to“ berichten [Harper, et al.;3].

2.2 Was ist „Penetrationtesting“

Der Begriff “Penetrationtesting” besitzt eine Vielzahl an Definitionen, je nachdem in welchem Fachbereich man sich bewegt.

Penetration bedeutet laut Wikipedia-Definition „eindringen“ und „durchdringen“ (lat. penetrare). In der Technik wird bei einem Penetrationstest die Umhüllung bei

Akkumulatoren durchstoßen, wodurch ein künstlicher Kurzschluss erzeugt wird, um darauf dessen Reaktion zu beobachten. Die Medizin wiederum überprüft im Rahmen eines (Spermien-)Penetrationstests, ob die Spermien in der Lage sind, Hindernisse auf dem Weg zur Befruchtung der Eizelle zu überwinden, was Aufschlüsse über die Sterilität gibt. Die Informatik wiederum beschreibt Penetrationstests als einen umfassenden Sicherheitstest, von außen in ein System bzw. Netzwerk einzudringen und Schwachstellen zu identifizieren. Da der Begriff „Penetrationstest“ in der Informatik synonym für das Eindringen in ein System / Durchdringen der Sicherheitsvorkehrungen steht, können somit Ähnlichkeiten mit den Definitionen aus den oberen Beispielen klar aufgezeigt werden [Wikipedia;2].

Bei einem Penetration-Test handelt es sich also um einen umfassenden Sicherheitstest, welcher das Computer-System eines Unternehmens auf Schwachstellen untersucht. Dafür werden häufig firmenfremde Personen oder Organisationen beauftragt. Penetration-Tester gehen wie legitime Angreifer vor, nutzen sowohl ähnliche Angriffsmethoden als auch Hacking-Tools. Was Penetration-Tester von Black-Hat-Hackern unterscheidet, ist, dass diese einen expliziten Auftrag für diese Arbeit haben und weder Daten manipulieren noch zerstören. Die gefundenen Mängel werden am Ende des Tests gemeldet, damit diese später behoben werden können [Kofler, et al.;3].

Während des Tests haben Pen-Tester einen erheblich großen Vorteil gegenüber anderen Hackern: Ihre Hacking-Versuche müssen nicht im Verborgenen stattfinden. Ein Pen-Tester kann beispielsweise großflächig Port-Scans durchführen, was für Hacker undenkbar wäre, da dies gut gesicherte Server sofort Alarm schlagen lässt.

2.2.1 Ablauf eines Penetrationtests

Mit Penetrationtests wird versucht, den Angriff auf ein System zu simulieren und die Testergebnisse im Anschluss auszuwerten, um die Sicherheit des Systems zu erhöhen. Dies erfordert ein hohes Maß an Präzision bei der Planung und Durchführung des Tests. Eine Erstellung des exakten Ablaufes und der Rahmenbedingungen kann besonders nützlich sein, um Komplikation und Missverständnisse während und nach dem Test zu verringern oder gar zu verhindern, wie etwa das Übersehen von potentiellen Schwachstellen im System oder das Testen von Funktionen und Software, die für den Test eigentlich nicht relevant sind.

Aus diesem Grund wird in den folgenden Abschnitten auf eine solche Planung und Durchführung eines Penetrationstests eingegangen.

2.2.1.1 Planung des Penetrationstests

Zu Beginn des eigentlichen Tests ist eine Planung dessen zu erstellen, um eine kontrollierte Durchführung des Tests zu gewährleisten. Diese besteht unter anderem aus den Rahmenbedingungen, den Zielen sowie der Vorgehensweise.

Zuallererst müssen klare Ziele definiert und festgelegt werden [Kurtz;1]. Beispiel solcher Ziele können die Komprimittierung des Accounts des Administrators sein, oder Manipulation von sensiblen Daten. Im Anschluss folgt die Festlegung der Rahmenbedingungen.

Für die Planung der Vorgehensweise werden die zu erreichende Ziele festgelegt, um im Anschluss die Schritte zu planen, die zur Erfüllung dieser Ziele erforderlich sind. Neben der Auswahl geeigneter Methoden und Test-Tools bieten sich eigens entwickelte Skripte für den Test einzusetzen, um gewissen Schritte zu automatisieren [Moyer, Schultz;1].

Da durch einen Penetrationstest erheblich Schäden an einem System angerichtet werden können, empfiehlt es sich, ebenfalls eine Risikobewertung durchzuführen. Ist das Risiko, dass sensible Daten durch den Test beschädigt oder sogar vernichtet werden können, sollte der Test in einer simulierten Umgebung stattfinden. Diese Art von simulierter Umgebung könnte zum Beispiel eine “Sandbox”-Umgebung sein, die eine (fast) exakte Nachbildung des Zielsystems darstellt und vom IT-Personal des Kunden eigens für den Penetrationstest erstellt wurde.

Falls auf Seiten des Kunden Zweifel an den Fähigkeiten der Penetrationstester besteht, sollten ihm Referenzen über die Eignungen, Fähigkeiten und Erfahrungen der involvierten Penetrationstester zur Verfügung gestellt werden. Außerdem steht es dem Kunden frei, den Einsatz von Hackern von vorneherein zu verbieten und kann bei Bedarf polizeiliche Führungszeugnisse von den Testern verlangen. Diese Gegebenheiten sollten im Idealfall vertraglich festgelegt werden, was zur Erstellung eines “*Service Level Agreements*” führt.

2.2.1.2 Reconnaissance

Unter dem Wort “Reconnaissance” wird die Durchführung einer vorläufigen Untersuchung zur Beschaffung von Informationen verstanden [Merriam-Webster;1]. Das Ziel des Testers bzw. Angreifer ist es hierbei, Informationen über das Zielsystem zu erhalten, um somit potentielle Schwachstellen aufzudecken.

Um eine realistische Abbildung eines echten Reconnaissance-Versuches zu erzeugen, lohnt sich die “*Zero-Knowledge”*-Variante des Penetrationstests. Die Tester besitzen hier keine Informationen über das Zielsystem und dessen Bestandteile [cybrary;1]. Da ein

Angreifer aber schon bei der Reconnaissance Schwachstellen ausnutzt, ist diese selbst beim Vorhandensein aller, vom Kunden bereitgestellten Systeminformationen sinnvoll, da ansonsten Schwachstellen übersehen werden könnten. Sollte sich herausstellen, dass die vom Kunden bereitgestellten Informationen unvollständig waren, können diese dank der Reconnaissance vervollständigt werden.

Das Schadenspotential durch Reconnaissance ist in den meisten Fällen relativ gering, und Situationen, in denen Schaden am System verursacht werden, sind äußerst selten. Als Beispiel eines solchen Falles ist der Port-Scan zu nennen, der zum Absturz eines schlecht gesicherten Systems führen könnte.

Die Ziele einer Reconnaissance [Cole;1] seien im Folgendem aufgelistet:

- Ermittlung von Hosts, auf die zugegriffen werden kann
- Ermittlung des Standortes von Routern und Firewalls
- Ermittlung der eingesetzten Betriebssysteme
- Ermittlung der offenen Ports
- Ermittlung der laufenden Services
- Ermittlung der Versionen der laufenden Applikationen

2.2.1.3 Scannen nach Schwachstellen

Nach der Beschaffung erster Informationen durch die Reconnaissance-Phase wird nun aktiver nach Schwachstellen gesucht und die bereits gefundenen werden detaillierter unter die Lupe genommen.

Bei einer genaueren Untersuchung bereits gefundener Schwachstellen möchte der Tester sichergehen, dass er seine Zeit nicht in Angriffe steckt, die ein geringes Sicherheitsrisiko darstellen oder gar keine Relevanz zum Zielsystem aufweisen. Hacker mit wenig Erfahrungen, wie etwa die sogenannte Skript Kiddies, greifen so auf Exploits zurück, die auf genau dieses System nicht anwendbar sind [Kurtz;2]. Etwas kann ein Skript Kiddy versuchen, eine Schwachstelle bei einem Microsoft Windows System auszunutzen, die sich nur auf Linux-Systemen befindet.

Die Motivation des Angreifers ist ausschlaggebend dafür, welche Schwachstellen er möglichweise ausnutzen wird. Der Schutz vor einem Angriffsversuch eines Geheimdienstes ist mit erheblich mehr Aufwand verbunden, als wenn etwa ein Black-Hat-Hacker versucht, in ein System einzudringen. Zwar entwickeln und verwenden die als “Sport Intruders” [Moyer;1] bezeichneten Angreifer Angriffs-Tools, die auch von Penetrationtestern verwendet werden, jedoch verfügen sie nicht über das nötige Know-How, gezielt neue Schwachstellen zu finden. Sollte es dennoch dazu kommen, dass sie bisher unbekannte Schwachstellen finden, so werden diese in den meisten Fällen nach

kurzer Zeit in diversen Foren zu finden sein, etwa um “Karma-Punkte” von der Community zu ernten, oder dass die Existenz von anderen Hackern bestätigt werden kann.
Da Sicherheitsexperten oftmals Teil solcher Communities sind, oder sich in den Foren bewegen, wird die Schwachstelle bekannt gemacht und erste Schutzmaßnahmen in die Wege geleitet.

Versuchen Tester die realen Umstände eines Angriffes so genau wie möglich darzustellen, sollten nur die Schwachstellen untersucht werden, die für einen Angreifer von Nutzen sein könnten. Dem Angreifer nutzt die Ausnutzung einer Schwachstelle, die ein Denial-of-Service auslösen würde, recht wenig, wenn er versucht, sensible Daten zu erbeuten.
Jede Ausnutzung einer Schwachstelle ist mit einem gewissen Aufwand verbunden und da der Angreifer versucht, diesen so gering wie möglich zu halten, wird Schwachstellen den Vorrang gewehrt, die einfacher und schneller ausnutzbar sind, als andere. Ist der Angreifer am Ende erfolgreich, werden alle anderen Schwachstellen vom Angreifer ignoriert, ganz gleich wie einfach deren Ausnutzung im Nachhinein auch sei.

Es wird jedoch nicht in allen Fällen unter realen Bedingungen getestet. So wird durch die Tester auf eine Weise gearbeitet, die auffällige Logs erzeugt, die ein normaler Angreifer um jeden Preis versucht zu verhindern. Dies ist dem sehr engen Zeitrahmen geschult, der den Testern zur Verfügung steht [Moyer;1]. Dies ermöglicht aber auch den Einsatz von Vulnerability-Scannern, deren Benutzung unter normalen Umständen zu auffälligen Systemverhalten führen würde. Diese Tools werden auch gern von Skript-Kiddies verwendet, was deren Einsatz auf eine gewissen Weise realistisch macht.

Für den Angreifer und den Tester existieren eine Vielzahl an Quellen, um sich über bekannte Schwachstellen und Exploits zu informieren.
Ein Angreifer wird jedoch ausschließlich öffentlich zugängliche Quellen nutzen, um für zukünftige Angriffe vorbereitet zu sein. Diese Quellen reichen von öffentlichen Foren, IT-Nachrichtenquellen, Fachbücher, bis hin zu diversen Exploit-Datenbanken wie “exploit-db.com” [exploit-db;1] und “nvd.nist.gov” [nvd-nist.gov;1]. Dies ist auch der Grund dafür, wieso nur nach öffentlich bekannten Schwachstellen beim Penetrationstest gesucht werden sollte.

Nachdem nun alle wichtigen Informationen über das System gesammelt wurden, beginnt der eigentliche Teil des Penetrationstest: Die Exploitation [Kurtz;1].

2.2.1.4 Exploitation (Durchführung der Penetration)

Da nun alle wichtigen Informationen gesammelt wurden, ist der Tester bzw. Angreifer bereit, die gefundenen Schwachstellen auszunutzen. Es sind nun zwei Optionen für beide Parteien gegeben: Entweder es sind bereits Tools für die Ausnutzung der Schwachstellen vorhanden oder die benötigten Tools werden in Form von Skripten selber programmiert.

Diese Phase des Penetrationstests erfordert den höchsten Grad an Aufwand unter allen Phasen. Fehler in den Tools oder Skripts müssen ausgemerzt werden und Anpassungen dieser, um auf das Zielsystem zugeschnitten zu sein, müssen ebenfalls vorgenommen werden. Programmierkenntnisse sind für diese Arbeit daher unerlässlich.

Während dieser Phase sollte sich bemüht werden, eine strukturierte Vorgehensweise an den Tag zu legen. Nur so kann garantiert werden, dass es zu einer ordentlichen Analyse der Schwachstellen und deren Auswertung kommt. Das “Open Source Security Testing Methodology Manual” [OSSTMM;1] schlägt zum Beispiel vor, einzelne Aufgaben zu Modulen zusammenzufassen und dann auf verschiedene Objekte anzuwenden.

Da die Exploits oftmals sehr spezifisch für jedes System sind, wird auf die eigentliche Penetration nicht weiter eingegangen. Stattdessen wird in einem separaten Kapitel auf mehrere Szenarios eingegangen, die die Ausnutzung von Schwachstellen in einem System detailliert aufzeigen soll.

Sollte das Schadenspotential durch den Penetrationstests zu hoch sein, so kann der Test auch in einer simulierten Umgebung stattfinden. Das IT-Team des Kunden könnte eine eigens für den Test entworfene Sandbox zu Verfügung stellen, die eine fast (exakte) Abbildung des zu testenden Systems darstellt.

2.2.1.5 Auswertung und Abschluss

Alle Informationen, die während des Penetrationstests gesammelt wurden, sollten nur aufbereitet und dem Kunden in Form eines Berichtes präsentiert werden. [Moyer, Schultz;1] schlagen vor, dass der Bericht für jedes gefundene Problem einen Lösungsvorschlag anbietet.

Der Bericht sollte außerdem die Zielsetzungen, Rahmenbedingungen und verwendeten Techniken zur Suche, Analyse und Ausnutzung der Schwachstellen beinhalten. Durch eine genaue Dokumentation des ganzen Prozesses gewinnt der Kunde mehr Vertrauen in die Ergebnisse und in die Tester und kann sicherstellen, dass die Forderungen erfüllt wurden. Die verwendeten Tools sollten dem Bericht ebenfalls hinzugefügt werden, damit der Kunde die Möglichkeit ausschließen kann, dass ihm nur ein einfacher Scan als Penetrationstest versucht wird zu verkaufen.

Es ist auf jeden Fall darauf zu achten, dass der Bericht von jedem Mitglied des Unternehmens gelesen und verstanden werden kann. Dies betrifft nicht nur Führungspersonal, sondern auch IT-Mitarbeiter und Manager.
[Winkler;1] empfiehlt, dass für das Management selbst, nicht die erfolgten Angriffe aufgelistet werden sollen, sondern auf die gefährdeten Assets eingegangen wird und welche Schäden angerichtet werden kann. Diese Informationen sind kritisch für das Risikomanagement.

Zu beachten ist, dass die Aussagen des Berichtes sich nur auf den Zeitraum beziehen, in dem Der Penetrationstest stattfand. Es kann keine Aussage über mögliche Probleme zu späteren Zeitpunkten getroffen werden, was den Bericht zu einer Momentaufnahme macht. Aus diesem Grund sollte der Penetrationstest regelmäßig durchgeführt werden, muss eine aktuelle Lage über die Sicherheit des Systems treffen zu können.
Zukünftige Tests und Berichte können sich auf hinzugekommene Sicherheitsprobleme und erfolgreich implementierte Maßnahmen fokussieren. Außerdem wird dadurch auch ersichtlich, ob der Kunden die bereits gefunden Schwachstellen behoben und die nötigen Sicherheitsmaßnahmen in Angriff genommen hat.

2.2.1.6 Penetrationtesting – Ein unendlicher Prozess

Die letzten Kapitel hinterlassen beim Leser vielleicht den Eindruck, dass es sich beim Penetrationtesting um einen Test handelt, dessen Phasen sequentiell abgearbeitet werden. Vielmehr ist das Penetrationtesting jedoch ein Kreislauf, ein unendlicher Prozess, wenn man so will.

Gelingt zum Beispiel die Penetration, so kann das System genauer analysiert und nach Schwachstellen untersucht werden, was zur Entdeckung neuer Ausnutzungsmöglichkeiten führen könnte. Eine Wiederholung der Penetration ist dadurch gegeben.

Es wird somit schnell ersichtlich, wieso es sich hier um einen Kreislauf handelt.
Der Penetrationtest kann selbst bei einem nicht geglückten Penetrationsversuch zum Anfang der Reconnaissance-Phase zurückkehren, um durch zusätzliche Informationen zu neuen Erkenntnissen und dadurch hoffentlich zu einer erfolgreichen Penetration kommen.
Durch die dauerhafte Durchführung des Penetrationstest wird der negative Aspekt “Momentaufnahmen” größtenteils beseitigt.

2.2.2 Hacking-Tools

In der Hacker-Community gibt es eine Vielzahl an Tools, die den Hackern und den Penetrationtestern die Arbeit erheblich erleichtern können. Viele dieser Tools sind kostenlos, was die Anschaffung nahezu mühelos gestaltet.
In diesem Kapitel werden einige Hacking-Tools erläutert, deren Aufgabenbereich beleuchtet und die wichtigsten Funktionen aufgelistet. Auf die genaue Nutzung der Funktionen wird jedoch nicht eingegangen. Es ist außerdem zu beachten, dass eine komplette Auflistung aller Hacking-Tools nicht vorgenommen werden kann, da viele der Tools nicht für das Penetrationtesting von Webapplikationen genutzt werden und selbst die Menge der Hacking-Tools für den Bereich der Arbeit zu groß ist.

2.2.2.1 Burp Suite

Burp Suite ist ein grafisches Tool zum Testen der Sicherheit von Webapplikationen und fungiert gleichzeitig als Proxy-Server. Das Tool basiert auf Java und wurde von *PortSwigger Web Security* entwickelt. Burp Suite verfügt neben Funktionen, wie einen Proxy-Server, Scanner und Intruder, auch über fortgeschrittene Funktionen, wie einen Spider-Programm, einen Repeater, einen Decoder, einen Comparer, einen Extender und einen Sequenzer [PortSwigger;1].
Das Tool wurde für das professionelle Penetrationtesting von Webapplikationen entwickelt und gehört heutzutage zur Standardsoftware bei Penetrationstestern im Bereich Websicherheit.

2.2.2.2 OWASP ZED / ZAP

OWASP Zed Attack Proxy (ZAP) ist ein Security-Tool, welches zur automatischen Überprüfung von Schwachstellen in Webapplikationen genutzt wird [OWASP;1].
Das Tool ist komplett kostenlos und wird von hunderten internationalen Freiwilligen gewartet und weiterentwickelt. Ursprünglich wurde die Software von OWASP, einer Non-Profit-Organisation entwickelt, die sich mit der Sicherheit von Webapplikationen beschäftigt.
Bei ZAP handelt es sich, wie bei Burp Suite, um ein HTTP-Proxy-Tool und bietet neben Plug-n-hack-Support auch einen Intercepting Proxy Server, Web Crawler, automatische und passive Scanner, Forced Browsing, einen Fuzzer, WebSocket-Support und Scripting-Sprachen.
Durch die Plugin-basierte Architektur bietet es die Möglichkeit, neue Features über den Marketplace einzubinden.

2.2.2.3 Metasploit

Metasploit ist ein beliebtes Pentesting- oder Hacking-Framework und ist das Hacking-Tool, mit den viele Hacker erste Erfahrungen sammeln. Metasploit stellt als Framework eine Ansammlung von Hacking-Tools dar, welche den Nutzer mit wertvollen Informationen bezüglich bekannter Sicherheitslücken versorgen und hilft bei der Planung von Penetrationtests und Hacking-Strategien für zukünftige Angriffe. Das Tool wurde von Rapid7 entwickelt und ist ein vorinstalliertes Framework im Betriebssystem "*Kali Linux*" [Metsploit;1]

2.2.2.4 Wireshark

Wireshark ist ein Open-Source Paket-Analyser. Es wird für die Analyse und das Troubleshooting von Netzwerken genutzt. Es ist kostenlos und läuft auf den Betriebssystemen Linux, macOS, BSD, Solaris und Microsoft Windows. [Wireshark;1]
Es ermöglicht eine tiefe Inspektion und Analyse der Pakete von hunderten verschiedene Protokolle, vom TCP bis hin zum PPP.
Das Tool ist mit einer Entschlüsselungsfunktionen ausgestattet, die zusammen mit Filterungs- und Anzeigefunktionen dabei hilft, den Netzwerk-Traffic zu analysieren und Angriffe in Echtzeit zu erkennen. Pakete werden dabei in einem Netzwerk in Echtzeit abgefangen und in ein für Menschen lesbares Format konvertiert. Diese Pakete können dann genauer untersucht werden.

2.2.2.5 John the Ripper

„John the Ripper" ist ein Hacking-Tool, mit dem durch Brute-Force-Methoden Passwörter geknackt werden. Seine Hauptaufgabe liegt in der Aufspürung schwacher Unix Passwörter. Es verbindet eine Reihe an beliebten Passwort-Cracker Programmen, erkennt Passwort Hash-Typen automatisch und beinhaltet einen individualisierten Cracker [Openwall;1].
„John the Ripper" nimmt ein Textstring-Muster (von einer Textdatei, auch als „wordlist" bezeichnet, die beliebte und komplexe Wörter enthält, die in einem Wörterbuch vorkommen oder von bereits gecrackten Passwörtern), dieser wird gehasht und dann mit dem Passworthash verglichen.
Dieses ethische Hacking-Tool nutzt Dictionary- und Brute-Force-Technologien um Passwörter zu entschlüsseln, unter anderem:

- DES, MD5, Blowfish
- Kerberos AFS
- Hash LM (Lan Manager), welches in Windows NT / 2000 / XP / 2003 genutzt wird
- MD4, LDAP, MySQL (Wenn man Third-Party-Module nutzt)

2.2.2.6 Xenotix XSS Exploit Framework

Xenotix XSS Exploit Framework ist ein weiteres, kostenloses Tool der OWASP-Community [OWASP;2].

Bei dem Tool handelt es sich genau um ein "*Cross-Site-Scripting Vulnerability Detection and Exploitation Framework*".

Mit diesem Tool können drei verschiedene Browser gleichzeitig ausgeführt und getestet werden. Dies vereinfacht und beschleunigt das Aufspüren und Testen von XSS-Lücken um ein Vielfaches.

Die meisten seiner Funktionen sind auch in HTTPS-Proxy-Tools verfügbar, jedoch besteht der große Vorteil hierbei, dass keine Proxyeinstellungen im Browser vorgenommen werden müssen, sondern direkt in dem Programm verschiedene Browser-Engines gleichzeitig bedient werden können.

2.2.2.7 Brutus

Brutus ist ein Programm, mit dem Passwörter automatisch ausprobiert (gehackt) werden können. Das Tool baut bis zu 60 gleichzeitige Verbindungen zum Ziel auf und testet anhand verschiedener Techniken mögliche Passwörter und Benutzernamen aus. Die genutzten Angriffe sind ein konfigurierbarer Brute-Force-Angriff, ein Wörterbuch-Angriff und ein "Leeren"-Angriff [Wikipedia;3]

2.2.2.8 Clickjacking-Tester/Tool

Webapplikationen sind per Clickjacking und weiteren UI/Redressing/Angriffsvektoren angreifbar. Meist liegt das daran, dass es möglich ist, Iframes oder unsichtbare Seitenelemente einzubinden oder in den Hintergrund zu schieben. X-Frame Options sind ein probates Gegenmittel, um die Einbettung zu unterbinden.

Das Tool Clickjacking-Tester prüft genau diesen http-Header einer betroffenen Webapplikation, um auszugeben, ob Clickjacking denkbar ist oder nicht.

Testet man eine URL mit diesem Tool, kann man in wenigen Momenten feststellen, ob diese für Clickjacking anfällig ist.

2.2.2.9 SQL-ninja

SQL-Ninja ist ein Tool, welches für SQL-Injections innerhalb von Webapplikationen genutzt wird, die mySQL-Server im Backend verwenden. Mit dem Tool wird versucht, Zugriff auf Betriebssystemebene des DB-Servers zu erhalten, um im späteren Verlauf weitere Angriff durchführen zu können [Kali;1].

2.2.2.10 Social Engineering Toolkit (SET)

Das Social Engineering Toolkit (SET) ist eine Open-Source Software, welche vom Gründer der Firma "*TrustedSec*" entwickelt wurde [SET;1].

Das Tool wurde speziell für den Einsatz von Penetrationtests im Bereich Social-Engineering entworfen und biete dazu einige nützliche Funktionen.

2.3 Sicherheitsüberprüfung

Sicherheitsüberprüfungen eines Systems sind immer Momentaufnahmen und sind kein Garant dafür, dass das getestete System frei von Schwachstellen ist [Kofler, et al;1]. Wird eine Schwachstelle gefunden, so wurde deren Existenz bestätigt. Dies bedeutet jedoch nicht, dass Schwachstellen, die nicht gefunden wurden, demnach auch nicht existieren und sollten somit auch nicht ausgeschlossen werden. Je mehr Zeit in die Sicherheitsüberprüfungen gesteckt wird, umso höher ist die Wahrscheinlichkeit unentdeckte Sicherheitslücken ausfindig zu machen. Da jedoch Zeit und Budget in fast allen Fällen limitiert sind, gibt es eine Reihe an verschiedenen Arten von Sicherheitsüberprüfungen, welche passend zur Situation gewählt werden können.

2.3.1 Arten der Sicherheitsüberprüfung

Der Penetrationstest ist nicht die einzige Art der Sicherheitsprüfung, die von modernen Unternehmen zur Erkennung von Schwachstellen im Zielsystem eingesetzt wird.
Aus diesem Grund werden in diesem Kapitel weitere Sicherheitsprüfungen genannt, die dem Penetrationstest ähnlich sind, jedoch ihre eigenen Vorteile und Einsatzgebiete haben.

2.3.1.1 Penetration-Test

Mit dem Penetration-Test versucht man die "*Worst-Case*"-Szenarien zu erreichen, die vor dem Start des Tests definiert wurden. Zum Einsatz kommt dafür neben automatisieren Tools auch ein hohes Maß an manueller Arbeit.
Der Penetration-Test verläuft, anders als das "*Vulnerability Assessment*", nicht in die Breite, sondern konzentriert sich auf die Erreichung des festgesetzten Ziels. Dies könnte beispielweise das Erlangen eines administrativen Zugriffs in einer Webanwendung sein oder das Erlangen von Domänenadministrationsrechten.
Penetration-Tests sind besonders gut dazu geeignet, sicherheitskritische Lücken zu finden, da der Fokus auf diese Lücken gelegt wird und der Hauptteil der Zeit in dieses Ziel fließt [Kofler, et al.;1].

Übliche Tätigkeiten des Penetration-Tests sind:

- Die Durchführung von Port-Scans, um verfügbare Dienste zu erkennen. Die Anzahl der Dienste, die gescannt werden sollen, kann auf die eher interessanten beschränkt werden, um unnötige Scans zu vermeiden
- Die Automatisierung von simplen Tätigkeiten durch den Einsatz von selbst-entwickelten Testtools.

- Die manuelle Suche nach unbekannten Schwachstellen in den Systemen oder der Software
- Die Erstellung eines Berichtes, welcher die Ergebnisse des Tests in einer einfach zu verstehenden Form wiedergibt.

Wann sollte der Penetration-Test eingesetzt werden:

- Maßnahmen zur Sicherung wurden bereits getroffen und sollen im Anschluss überprüft werden, ob diese genügende Sicherheit bieten, um einen möglichen Angreifer abwehren zu können.
- Wenn bewiesen werden soll, dass die Anwendung gravierende Sicherheitslücken aufweist. Dies soll als Grundlage bzw. Berechtigungsgrund für zukünftige, intensivere Tests genutzt werden

2.3.1.2 Red Teaming

"*Red Teaming*" ist eine Sicherheitsprüfung, deren Testumfang sich nicht auf eine Anwendung beschränkt. Stattdessen wird getestet, ob beispielweise Zugriff auf bestimmte Daten erlangt werden kann. Über welche Anwendungen der Tester letzten Endes Zugriff auf die Daten erlangt, ist nicht von großer Wichtigkeit, sondern das gesamte Sicherheitskonzept an sich wird getestet. Die Durchführung des Tests durch das Red Team ist auch ein wichtiger Bestandteil im Training des Blue Teams. Als Blue Team wird das Team beschrieben, welches für den Schutz der Systeme verantwortlich ist. Das Red Team muss aus diesem Grund seine Test-Aktivitäten versuchen zu verstecken, anders als beim Penetration-Test, um ein vorzeitiges Ende des Testes durch das Blue Team zu verhindern [Kofler, et al.;1].

"*Red Teaming*" ermöglicht eine vergleichsweise realistische Sicht auf die möglichen Angriffsflächen, die ein Angreifer bei einem System ausnutzen könnte, jedoch erfolgt keine detaillierte Auswertung dieser Angriffswege.

Übliche Tätigkeiten des Red Teamings sind:

- Das Suchen von möglichst vielen Zugangspunkten, durch die ein Angreifer an die gewünschten Informationen kommen kann.
- Die Evaluierung der gefundenen Zugangspunkte. Hier ist zu bestimmen, welche von den Zugangspunkten die geringste Sicherheit aufweist.
- Das manuelle Suchen von unbekannten Schwachstellen in den relevanten Anwendungen.

Wann sollte das Red Teaming eingesetzt werden:

- Absicherungsmaßnahmen wurden bereits getroffen und es wurde sich ein Überblick darüber verschafft, wie das Netzwerk oder das System auf bestimmte Angriffe reagieren würde.
- Wenn es Unternehmensdaten gibt, die besonders geschützt werden müssen. Vorhandene Sicherheitsmaßnahmen sollen überprüft werden, ob diese die Unternehmensdaten effektiv genug schützen können.
- Wenn das Blue Team auf eine möglichst realistische Art und Weise trainiert werden soll, um gegen zukünftige Angriffe besser gewappnet zu sein.

2.3.1.3 Vulnerability Scan / Schwachstellen-Scans

Das Ziel von "*Vulnerability Scans*" ist es, mit einem möglichst geringen Aufwand eine Sicherheitsprüfung durchzuführen, weswegen diese Art von Test üblicherweise automatisiert wird. Vor allem sollen einfach zu erkennende Schwachstellen auf eine Weise identifiziert werden, wie sie auch von sogenannte Script-Kiddies gefunden werden [Kofler, et al.;1].

Tools, wie "*nmap*", "*Nessus*", "*Nexpose*" und "*OpenVas*" kommen hierbei zum Einsatz. Die eigentliche Liste der genutzten Schwachstellen-Scanner ist jedoch etwas länger.

Die Kosten für einen Vulnerability Scan sind im Vergleich zu gezielten Tests wie Penetrationtests wesentlich geringer, da der Anteil manueller Arbeit auf ein Minimum reduziert wird. Der Auftraggeber erhält, je nach Anbieter, entweder eine Ausgabe welche direkt nach dem Ende des Tests von den jeweiligen Tools ausgegeben wird, oder einen eigenen Abschlussbericht, welcher von falschen Tool-Ergebnissen bereinigt und für den Kunden verständlich aufbereitet wurde.

Übliche Tätigkeiten des "*Vulnerability Scans*" sind:

- Die Erkennung von verfügbaren Diensten durch das automatische Scannen von Ports.
- Die Durchführung von Schwachstellen-Scans mit einem oder mehreren Schwachstellen-Scannern.
- Die Entfernung von falschen Tool-Ergebnissen und die Aufbereitung der Ergebnisse in Form eines Berichts.

Wann sollte ein "*Vulnerability-Scan*" eingesetzt werden:

- Wenn eine erste Einschätzung der Sicherheit der Komponenten eines Systems erfolgen soll und zuvor keine Sicherheitsüberprüfung stattgefunden hat

- Wenn regelmäßige Scans der Systeme durchgeführt werden sollen. Dies dient zur Früherkennung von möglichen sicherheitsrelevanten Veränderungen im System. Diese Scans können auch zur Performance-Messung des Sicherheitsmanagements genutzt werden.
- Compliance-Vorgaben, die automatisierbar getestet werden können und kostengünstig nachgeprüft werden sollen.

2.3.1.4 Vulnerability Assessment

Das "*Vulnerability Assessment*" stellt die gebräuchlichste Form der Sicherheitsüberprüfung dar. Bei dieser Sicherheitsprüfung werden auch schwer zu identifizierenden Schwachstellen, durch ein hohes Maß an manueller Arbeit gefunden. Um auszuschließen, dass einfach zu identifizierbare Schwachstellen übersehen werden, können Schwachstellen-Scanner eingesetzt werden [Kofler, et al.;1].

Der Unterschied zum Penetrationtest liegt in dem viel höheren Abdeckungsgrad der Sicherheitsprüfung. Es wird versucht, so viele unterschiedliche Schwachstellen in so vielen verschieden Bereichen wie möglich zu finden und dennoch einen recht tiefen Einblick in das Zielsystem zu erhalten.

"*Vulnerability Assessments*" haben gegenüber "*Vulnerability Scans*" einen großen Vorteil: es können weitaus mehr und zuverlässiger unbekannte Schwachstellen im System gefunden werden. Vor allem werden Schwachstellen erkannt, die oftmals nicht sehr verbreitet sind und von automatische Scannern nicht erkannt werden.

Übliche Tätigkeiten von "*Vulnerability Assessments*" sind:

- Die Durchführung von Port-Scans zur Erkennung von verfügbaren Diensten.
- Die Durchführung eines Schwachstellen-Scans.
- Die Automatisierung einfacher Tätigkeiten durch den Einsatz von selbst programmierten Tools.
- Die Überprüfung der identifizierten Schwachstellen auf "*False Positives*".
- Die Suche nach unbekannten Schwachstellen.
- Die Aufbereitung der Ergebnisse in Form eines eigenen Berichts.

Wann sollte das "*Vulnerability Assessment*" eingesetzt werden:

- Wenn die Software einer Sicherheitsprüfung unterzogen werden soll, die von einer Partnerfirma oder selbst entwickelt wurde.
- Wenn ein möglichst vollständiges Bild über die Sicherheit der Software gewünscht wird.
- Wenn die Sicherheit eines internen Netzwerks evaluiert werden soll.

2.3.1.5 Open-Source-Intelligence-Analyse

Bei dieser Methode werden Informationen aus öffentlich zugänglichen Quellen extrahiert und Zusammenhänge interpretiert. Aus einzelnen, zum Teil zusammenhangslosen Datenpunkten entstehen wertvolle Informationen, die für das Suchen von Schwachstellen in einem Unternehmen oder einer Schlüsselperson des Unternehmens genutzt werden [Kofler, et al.;1].

Üblichen Tätigkeiten der "*Open-Source-Intelligence-Analyse*" sind:

- Die Nutzung von gängigen Suchmaschinen für die Suche nach interessanten Informationen.
- Die Extrahierung von Metadaten. Dies könnten Benutzernamen, Informationen über das Betriebssystem oder Softwareversionen sein.
- Beschaffung von Informationen von ausgewählten Mitarbeitern. Dies können Hobbies, Familie, Bildungsstand, etc. sein. Mithilfe dieser Informationen können geeignete Opfer für Angriffe ausfindig gemacht werden.

Wann sollte eine "*OSINT-Analyse*" eingesetzt werden:

- Wenn überprüft werden soll, ob die internen Sicherheitsrichtlinien, die einen bestimmten Umgang mit Informationen vorschreiben, eingehalten werden.
- Zur Schulung des Bewusstseins einzelner Personen für den Umgang mit privaten sowie Firmendaten.
- Wenn Zielpersonen von Spear-Phishing-Angriffen identifiziert und auf diese Angriffe vorbereitet werden sollen.
- Wenn nach Angriffspunkten der Infrastruktur gesucht werden soll, ohne das Zielunternehmen direkt anzugreifen.

2.3.2 Typen der Testdurchführung

Hat man sich nun für einen der Assessment-Typen entschieden, steht man vor der Entscheidung, welche Durchführungsart für diesen Typ ausgwählt werden soll. Im Folgenden ist eine Beschreibung der drei grundlegenden Einteilungen und deren Vor- und Nachteile:

2.3.2.1 White-Box-Test

Ein White-Box-Test bildet das Gegenstück zum Black-Box-Test. Das White-Box-Testing beschreibt eine Methode, bei der dem Tester alle nötigen Informationen über ein System vor Beginn der Planungsphase gegeben werden [Kofler, et al.;1]. Diese Informationen kommen in Form von Sourcecode, administrative Accounts und eine komplette Flusskontrolle des Systems oder der Software.

Dem Tester wird somit eine sehr detaillierte und, im Vergleich zu den anderen beiden Typen, vollständigere Sicherheitsübersicht des Systems oder der Software geliefert. Aufgrund der höheren Informationsdichte und der notwendigen Einarbeitungszeit, die

für den vorhandenen Sourcecode benötigt wird, kann der Aufwand auch höher als bei Grey-Box-Tests ausfallen.
Es muss außerdem beim White-Box-Testing darauf geachtet werden, dass nicht “um Fehler herum” getestet wird.

2.3.2.2 Black-Box-Test

Der Black-Box-Test bildet das Gegenstück zum White-Box-Test. Das Black-Box-Testing beschreibt eine Methode, bei der den Testern vor dem Beginn der Planungsphase so wenig Informationen wie möglich über das Zielsystem gegeben werden [Kofler, et al.;1]. Dies soll den Tester in eine Lage versetzen, die der eines externen Angreifers so ähnlich wie möglich ist, der über minimales oder sogar kein Vorwissen über das System verfügt. Wird dem Tester eine entsprechende Vorbereitungszeit gewährt, so lässt sich eine vergleichsweise realistische Testumgebung schaffen.

Die Zeitspanne für die Tests ist jedoch begrenzt, da in den meisten Fällen das Budget limitiert ist. Dies ist auch der größte Nachteil des Black-Box-Tests. Dem Angreifer werden keine Informationen vor Beginn der Tests zur Verfügung gestellt. Er muss sich diese selber erarbeiten. Zum Beispiel muss der Tester durch Durchprobieren herausfinden, ob mehrere Anwendungen unter unterschiedlichen Verzeichnissen am Webserver verfügbar sind.

Der Tester kann jedoch nicht unendlich viele Optionen durchprobieren, da dieser nur über ein eingeschränktes Zeitfenster verfügt. Dies führt dazu, dass der Tester möglichweise nicht alle Schwachstellen und Fehler finden kann. Selbst eine andere Sortierung der verwendeten Wörterliste könnte zu drastisch anderen Ergebnissen führen, wenn nach Verzeichnissen gesucht wird.

Demnach sollte dem Tester mehr Zeit zur Verfügung gestellt werden. Ist dies keine Option, muss der Auftraggeber akzeptieren, dass der Test nur einen ersten Eindruck über die Sicherheit eines Systems geben kann.

2.3.2.3 Grey-Box-Test

Die Grey-Box-Tests stellen eine Art Mittelweg zwischen Black- und White-Box-Test dar.
Der Tester soll möglichst frei, kreativ und ohne zu viel Einfluss durch vorhandenen Informationen die Sicherheitsüberprüfung durchführen, um eine möglichst realitätsnahe Sicht eines unbekannten Angreifers widerzuspiegeln [Kofler, et al.;1].
Dem Tester können Informationen gegeben werden, die er in einer vertretbaren Zeit sowieso selbst herausgefunden hätte. Dies gestaltet die Durchführung des Tests möglichst effizient und lässt mehr Zeit für die Suche nach Schwachstellen im System.

Zu den Informationen, die dem Tester ausgehändigt werden können, gehören beispielweise versteckte Ordner, zusätzliche Domains oder auch der Zugriff auf einen zusätzlichen Administrator-Account zur Einsicht von vorhandenen Admin-Funktionen.

3. Web-Applikationen: Angriffsvektoren

Es gibt zahlreiche Gründe für den Angriff auf Webanwendungen. Die Motivation der Hacker reicht von politischen Gründen über den Diebstahl von Kreditkarteninformationen und Kundendaten bis hin zur Nutzung der gehackten Systeme für die Verbreitung von Malware und den Missbrauch im Rahmen von Phishing-Angriffen. Auch Denial-of-Service-Angriffe und die Erpressung von Lösegeldern fallen in diese Kategorie.

Die Angriffe können dabei auf den Webbrowser, auf die Kommunikation zwischen den beteiligten Systemen, den Webserver oder das Backend-Datenbanksystem abzielen. Ein Großteil der Angriffe wird heute nicht mehr manuell, sondern automatisiert ausgeführt. Crawler suchen nach der Veröffentlichung von Sicherheitslücken im Internet nach verwundbaren Systemen und führen dann Attacken automatisch aus.

In den folgenden Abschnitten wird auf einige typische Angriffsvektoren aus dem Bereich “Webapplikation-Pentesting” eingegangen.

3.1 Clickjacking

Clickjacking, auch bekannt als >UI-Redressing-Angriff<, beschreibt ein Vorgehen, bei dem schädliche Funktionen hinter scheinbar harmlosen Buttons versteckt werden.
Auch Bezeichnungen wie >Likejacking< und >Sharejacking< sind heutzutage weit verbreitet, welche das Vorgehen mit den, in sozialen Netzwerken vorkommenden, Funktionen >Teilen< und >Gefällt mir!< beschreiben [Schäfer;1].

Diese Funktion wurde erstmals 2008 von Jeremiah Hansen und Robert Grossman in dem gleichnamigen Paper als >Clickjacking< bezeichnet, jedoch ist diese schon etwas länger bekannt und in der Diskussion [Hanse, Grossman;1]. In diesem Paper geht es konkret um die Möglichkeit, dass Elemente durch CSS-Styling unsichtbar bzw. transparent zu machen, daher auch der Name >UI-Redressing<.
Elemente können somit durch diese Methode transparent gestylt und sogar übereinandergelegt werden. Dadurch ist es möglich, dem Benutzer auf etwas anderes klicken zu lassen, als der Benutzer wahrnimmt, wodurch der Benutzer vertrauliche Informationen preisgibt oder im Hintergrund einen ungewollten Link öffnet.

Clickjacking selbst gehört zu den Schwachstellen, die man am einfachsten ausnutzen kann, solange keine Abwehrmechanismen implementiert wurden. Die einzige Schwierigkeit besteht in der Platzierung der verschiedenen Elemente, dem korrekten Übereinanderlegen der Aktionen, sodass dem Opfer ein bestimmtes Funktionsverhalten vorgetäuscht wird, obwohl im Hintergrund eine völlig andere Aktion stattfindet.

Gegenmaßnahmen

Mittlerweile wurden eine Vielzahl an Maßnahmen eingeführt, um gegen Clickjacking vorzugehen. Da deren Anzahl jedoch recht groß ist, seien hier nur wenige Möglichkeiten aufgezählt [Microsoft;1].

X-Frame-Options

Dieser Header, welcher 2008 eingeführt wurde, bietet einen teilweisen Schutz vor Clickjacking und muss vom Besitzer der Webseite gesetzt werden. Die Funktionen >DENY<, >SAMEORIGIN< und >ALLOW-FROM origin< verhindern Framing, Framing von externen Seiten und erlauben das Framing von spezifischen Seiten. Clickjacking-Attacken, bei denen keine Frames genutzt werden, können von >X-Frame-Options< nicht abgefangen werden.

Content Security Policy

Die >frame-ancestors<-Direktive der Content Security Policy kann das Einbinden von Content erlauben oder verbieten. Sogenannte iFrames und Objekte werden genutzt, um bestimmte Seiten an der Einbindung zu hindern. Durch die Content Security Policy werden die X-Frame-Options abgelöst und somit obsolet [Mozilla;1].

NoScript

Dieser Clickjacking-Schutz ist ein exklusives Add-On des Browsers Mozilla Firefox. Es verhindert, dass der Benutzer auf unsichtbare oder “redressed” Webseitenelemente klickt. Dies ist durch die ClearClick-Funktion möglich [noscript;1].

3.2 (Cookie-)Replay-Angriff

Der (Cookie-)Replay-Angriff ist ein Angriff, bei dem die Datenübertragung durch den Angreifer wiederholt (Replay) oder verzögert wird, um anschließend zu einem späteren Zeitpunkt erneut an die Web-Applikation oder Server zu senden [Kaspersky;1]. Dies kann der Angreifer bewerkstelligen, indem er den Datenverkehr der Session des Benutzers abfängt und die Session-ID stiehlt. Die Session-ID wird unter anderem als Cookie gespeichert, daher auch der Name “Cookie-Replay”. Der Angriff auf die

Authentifizierungs- und Zugangsdaten kann über den Man-in-the-Middle-Angriff erfolgen. Hat der Angreifer nun die Session-ID erlangt, kann dieser sich als autorisierter Benutzer ausgeben und alle Funktionen ausführen, die der autorisierte Benutzer ursprünglich auch nutzen konnte.

Im Folgendem ist ein Beispiel gegeben, welches den Cookie-Replay-Angriff verdeutlichen soll [thesecuritybuddy;1].

1. Alice und Bob kommunizieren über ein Netzwerk miteinander. Bob möchte sich nun bei Alice authentifizieren. Hierbei übermittelt Bob den Password Hash an Alice.
2. Der Angreifer Charles schneidet während dieses Vorganges den Datenverkehr mit und versucht so, an den Password Hash von Bob zu gelangen, welcher an Alice gesendet wurde.
3. Charles öffnet nun eine Verbindung zu Alice und authentifiziert sich mit dem Password Hash von Bob, welches er im vorherigen Schritt erlangt hat.
4. Das System von Alice wird die Täuschung nicht durchschauen und authentifiziert Charles. Somit hat Charles nun Zugriff auf das System von Alice und kann diese Verbindung für böswillige Zwecke mitbrauchen, etwa um sensible Daten zu stehlen oder weitere Angriffe auszuführen.

Gegenmaßnahmen

Jeder Sitzung sollte ein vordefiniertes Zeitlimit für dessen Leerlauf gegeben werden, welches bei Überschreitung den Benutzer automatisch abmeldet. Dieses Zeitlimit sollte am besten so kurz wie möglich und so lang wie nötig eingestellt werden.

Beide Kommunikationspartner sollten eine völlig zufällige Session-ID erstellen, welche nur für die Dauer der gegenwärtigen Session gilt und nicht wiederverwendet werden kann.

Jede Transaktion kann mit einem Passwort versehen werden, welches nur für diese eine Transaktion gilt und danach ungültig wird. Somit ist sichergestellt, dass selbst wenn der Angreifer die Nachricht aufnimmt und wieder versendet, der Code abgelaufen und damit unbrauchbar ist.

Nachrichten können mit Zeitstempeln versehen werden. Dies verhindert, dass sein Angreifer Nachrichten nach einem späteren Zeitpunkt erneut versenden kann, was das “*Window of Opportunity*” für einen Angriff einschränkt.

Die “*Barracuda Web Application Firewall*” bietet die Möglichkeit, Cookie-Protection-Richtlinien zu konfigurieren, um Cookie-Wiedergabeangriffe zu

verhindern. Diese kann außerdem erkennen, wenn ein Cookie von mehreren IP-Adressen aus gesehen wird, und ermöglicht in diesem Fall eine Abschwächung der Kontrollen [Barracuda;1].

3.3 Cross Side Request Forgery

Ein Cross Side Request Forgery ist ein Angriff, bei der der Benutzer – ohne sein Wissen – eine Aktion innerhalb einer Web-Applikation ausführt. Dem Browser des Benutzers wird eine bösartige HTTP-Anfrage untergeschoben, die dazu genutzt wird, die vom Angreifer gewünschte Aktion auszuführen [Schäfer;1].

Diese Sicherheitslücke tritt auf, weil das HTTP-Protokoll zustandslos ist, also mehrere Anfragen als voneinander unabhängige Transaktionen behandelt, auch wenn diese vom selben Auftraggeber kommen [w3;1]. Anfragen werden ohne Bezug zu früheren Anfragen behandelt und Sitzungsinformationen, wie etwa die Session-ID, werden weder ausgetauscht noch verwaltet, was ein Angreifer ausnutzen kann.

Vor dem Beginn des eigentlichen Angriffes muss der Benutzer an dem System authentifiziert sein. Durch die Authentifizierung wird ein Cookie erstellt, der alle relevanten Information der Sitzung enthält, wie etwa die IP-Adresse, Session-ID und diverse Benutzerdaten. Dieses Cookie wird mit jeder gesendeten Anfrage angehängt. Der Angreifer schickt dem Benutzer eine E-Mail, welche einen Link enthält, die bei dessen Aufruf eine bestimmte Aktion ausführt (etwa die Überweisung einer Summe X auf das Konto Y). Dies gelingt dem Angreifer, indem er durch diverse Social Engineer-Techniken den Benutzer der Web-Applikation dazu animiert, den Link auszuführen und dadurch verschiedene Aktionen auszuführen. Die Art der Aktion kommt auf die Berechtigungen des Benutzers an. Ein normaler Benutzer kann dazu angeleitet werden, Geldsummen an ein bestimmtes Konto zu überweisen oder seine E-Mail-Adresse zu ändern. Hat der Benutzer jedoch Administrationsrechte, so könnte die ganze Web-Applikation kompromittiert werden.

In manchen Fällen ist es sogar möglich, CSRF-Angriffe in eine Webseite einzubetten, der sogenannte “*Stored CSRF Flaw*” [OWASP;3]. Dies wird zum Beispiel erreicht, indem ein IMG- oder IFRAME-Tag in ein Feld gespeichert wird, welches HTML akzeptiert. Die erhöht die Effektivität des Angriffes, da der Benutzer üblicherweise auf der Seite authentifiziert sein muss, um die manipulierte Seite aufrufen zu können. Diese Technik ist allerdings nicht ganz ohne Risiko für den Angreifer, da diese Manipulation eine offensichtlichere Spur legt und zu ihm zurückführen könnte.

CSRF zielt nicht darauf ab, wie manch andere Angriffe, die Daten des Benutzers zu stehlen. Vielmehr interessiert sich der Angreifer für die Status-ändernde Abfragen, da er die Antworten der manipulierten Anfragen sowieso nicht sehen kann. Dies kann nur der Benutzer. CSRF wird deshalb auch als "State-Changing"-Angriff bezeichnet.

Im Folgendem ist ein Beispiel beschrieben, welches den Cross-Site-Request-Forgery-Angriff verdeutlichen soll [thesecuritybuddy;2]

1. Zuallererst wird die Webseite vom Angreifer manipuliert, die von potentiellen Opfern zu einem späteren Zeitpunkt besucht wird. Der Besuch der Seite kann durch Phishing provoziert bzw. beschleunigt werden.
2. Das Opfer besucht nun die Webseite und fordert die HTML-Seite an, die vom Angreifer manipuliert wurde.
3. Der manipulierte Server sendet eine Antwort, welche mit dem Schadcode versehen ist, mit dem später die CSRF-Schwachstelle ausgenutzt wird.
4. Der Browser des Benutzers rendert die manipulierte Seite, und sendet automatisch eine Anfrage an den eigentlichen Server, ohne dass der Benutzer etwas davon merkt.

Gegenmaßnahmen

Die Technik "*Synchronizer Token Pattern*" (STP) bettet ein Token, welche geheim und einzigartig für jede Anfrage ist, in alle HTML-Formen ein. Dieser wird anschließend auf Serverseite verifiziert. Dieser Token kann auf jede nur erdenkliche Art generiert werden, die Unvorhersehbarkeit und Einzigartigkeit garantiert. Ohne den exakten Token ist es dem Angreifer unmöglich, seine Anfrage authentifizieren zu lassen [OWASP;4].

Sollte die Arbeit mit CSRF-Token zu problematisch sein, steht als Alternative die "*Double Submit Cookie*"-Technik bereit. Besucht ein Benutzer eine Webseite, so wird eine Session erstellt und ein dazugehörige Session-ID als Cookie im Browser angelegt. Zur gleichen Zeit wird außerdem ein pseudo-zufälliger Zahlenwert bestimmt, der als Cookie auf dem Rechner des Benutzers angelegt wird. Sendet der Benutzer nun ein gesichertes HTML-Formular, so wird der CSRF-Token aus dem Cookie extrahiert und in ein verstecktes Input-Feld des HTML-Formulars eingefügt. Jede Anfrage muss mit diesem CSRF-Token ausgestattet werden, um vom Server verarbeitet werden zu können. Der Server vergleicht den Wert des Cookies mit dem Wert des CSRF-Tokens und akzeptiert die Anfrage, wenn beide Werte identisch sind bzw. verwirft diese, wenn die Werte unterschiedlich sind. Der große Vorteil bei dieser Technik ist es, dass der Token nicht auf dem Server gespeichert wird [OWASP;4].

Die "*Cookie-to-Header*"-Technik wäre eine Option für Web-Applikationen, die JavaScript für den Großteil ihrer Funktionen nutzen. Diese Technik basiert auf der "*Same-Origin-Policy*". Besucht ein Benutzer eine Webseite ohne zugehörige Server-Session, so setzt die Web-Applikation einen Cookie, die einen zufälligen Token enthält und für den Rest der Web-Session bestehen bleibt. Das client-seitige JavaScript liest diesen Wert aus, kopiert diesen in einen besonderen HTTP-Header und sendet diesen mit jeder Transaktions-Anfrage. Der Server validiert im Anschluss das Vorhandensein und die Integrität des Tokens. Zu beachten ist, dass diese Methode nicht durchführbar ist, wenn die Webseite ihre "*Same-Origin-Policy*" deaktiviert hat.

3.4 Cross-Site Scripting (XSS)

Cross-Site Scripting (XSS) ist ein Angriffsvektor, der normalerweise in Webanwendungen auftritt. Mithilfe von XSS können clientseitige Skripte in Webseiten eingefügt werden, die von anderen Benutzern angeschaut werden [OWASP;5]. Dieser Angriffsvektor kann dazu genutzt werden, um die Zugriffskontrollen wie die "Same Origin Policy" auszuhebeln. Ziel des Cross-Site Scriptings ist es, an sensible Daten des Benutzers zu gelangen, beispielweise an die Daten eines Benutzerkontos (Identitätsdiebstahl).

Der Term "*Cross-Site*" bezieht sich darauf, dass der Angriff zwischen verschiedenen Aufrufen einer Seite stattfindet. Für diese Angriffsart werden normalerweise Skript-Sprachen wie JavaScript verwendet, da diese weit verbreitet sind, daher der Name "*Scripting*".
Trotz der Namensähnlichkeit sind das "*Cross-Site-Cooking*" und die "*Cross-Site-Request-Forgery*", sowie deren Anwendungen wie etwa die "Cross-Site-Authentication-Attacke" keine Formen von Cross-Site-Scripting.

Cross-Site-Scripting gehört zu den HTML-Injections. Cross-Site-Scripting tritt dann auf, wenn eine Webanwendung Daten annimmt, die von einem Nutzer stammen, und diese Daten dann an einen Browser weitersendet, ohne den Inhalt zu überprüfen. So kann es einem Angreifer gelingen, Skripte indirekt an den Browser des Opfers zu senden und damit Schadcode auf der Seite des Clients auszuführen.

Ein klassisches Beispiel für Cross-Site-Scripting ist die Übergabe von Parametern an ein serverseitiges Skript, das eine dynamische Webseite erzeugt. Dies kann etwa das Eingabeformular einer Webseite sein, wie in Webshops, Foren, Blogs und Wikis üblich. Die eingegebenen Daten werden auf der Webseite wieder als Seiteninhalt ausgegeben, wenn die Seite von Benutzern aufgerufen wird. So ist es möglich, manipulierte Daten an

alle Benutzer zu senden, sofern das Serverskript dies nicht verhindert. Diese Daten sind oft Code einer clientseitigen Skriptsprache (meist JavaScript).

In Verbindung mit CSS können unterschiedlichen Angriffe durchgeführt werden. Einige der Beispiele wären die Entführung von Benutzer-Sessions, Phishing-Angriffe und das Entwenden der Kontrolle des Benutzerbrowsers.

Die Effektivität von XSS reicht von geringfügiger Belästigung bis hin zu einem erheblichen Sicherheitsrisiko, abhängig von der Empfindlichkeit der Daten, die von der anfälligen Webseite verarbeitet werden, und der Art der Sicherheitsvorkehrungen, die vom Netzwerk des Webseiten-Eigentümers implementiert wird.
Im Jahre 2007 war XSS für 84% aller Sicherheitslücken bei Webseiten verantwortlich und somit eine der größten Gefahren für Benutzer und Servicebetreiber [symantec;1]. Nach Angaben der Bug Bounty Plattform "*HackerOne*" gehört XSS zu der am häufigsten festgestellten Schwachstelle, welche von Hackern auf ihrer Plattform gefunden wurde [zdnet;1].

Es gibt drei grundlegende Arten von XSS: reflektiert, persistent und DOM-basiert. Diese drei Arten werde ich im Folgenden erläutern.

XSS – reflektiert
Das reflektierte/nicht-persistente Cross-Site Scripting ist ein serverseitiger Angriffsvektor, bei dem das Opfer einen präparierten Link anklicken muss. Schadhafter Code wird in den Variablen und Parametern dieser URL eingefügt, den die Webapplikation über den Server in die Webseite einbettet und an den Benutzer reflektiert [Hope, Walther;1]
Die vom Angreifer veränderte URL wird durch ein Opfer angeklickt und somit an den betroffenen Server als Anfrage verschickt. Ist die Web-Applikation nun nicht gegen diesen Angriff geschützt, so wird es dem Angreifer ermöglicht, den dynamisch generierten HTML-Code zu verändern. Der Benutzer sieht dann die manipulierte Seite in seinem Browser.

XSS – persistent
Das persistente XSS unterscheidet sich vom reflektiertem XSS lediglich darin, dass der Schadcode auf dem Webserver gespeichert wird, was zu einer Auslieferung dieses Codes bei jeder neuen Anfrage führt [project.webappsec;1].
Der Angreifer ruft eine manipulierte URL auf, wodurch eine Anfrage an den Server gesendet wird. Ist die Web-Applikation nun nicht gegen persistentes XSS geschützt, so wird die Anfrage verarbeitet der injizierte Code von der Applikation in der Datenbank gespeichert. Von nun an wird jedem neuen Besucher die Webseite mit dem veränderten HTML-Code präsentiert.

XSS – Dom-basiert

Das DOM-basierte XSS ist ein clientseitiger Angriffsvektor, d.h. der Angriff findet auf dem Rechner des Opfers statt. Der Schadcode wird hier zur Ausführung direkt an ein clientseitiges Skript übergeben [OWASP; 6].

Der Benutzer muss hierbei auf einen manipulierten Link klicken und schickt damit eine Anfrage an die Web-Applikation. Diese übergibt dann den Skriptcode an den Browser, um dort die Ausführung des Skripts zu starten. Die manipulierten Parameter aus der URL werden nun als Teil des Skripts im Browser des Benutzers interpretiert und ausgeführt. Über DOM-Zugriffe wird die im Browser dargestellte Webseite verändert, wodurch der Benutzer die manipulierte Seite sieht.

Gegenmaßnahmen

Alle eingehenden Eingabewerte sollten als unsicher betrachtet werden und vor weiteren Verarbeitungen auf der Serverseite geprüft werden. "*Gute*" Ergebnisse sollten exakt definiert werden (White Listing), da das Verbot von "*Bösen*" Ergebnissen (Black Listing) durch die unbekannte Menge von Angriffen schier unmöglich ist.

Die Content Security Policy kann verwendet werden, um erlaubte Quellen für Webseiten-Bestandteile zu definieren, und somit XSS effektiv verhindern[Mozilla;1].

Mit "NoScript" können gezielt XSS-Angriffe erkannt und verhindert werden. Diese Erweiterung ist jedoch nicht für jeden Browser erhältlich [noscript;1]

Eine Maskierung und Ersetzung von problematischen Zeichen ist ebenfalls eine Option und von Programmiersprache zu Programmiersprache unterschiedlich [selfhtml;1].

3.5 File Inclusion

"*File Inclusion*" beschreibt im allgemeinem eine Sicherheitslücke, bei der sicher der Angreifer unautorisierten Zugriff zu sensiblen Dateien auf einem Webserver verschaffen kann. Hierbei wird die "*include*"-Funktion des Webservers genutzt, um bösartige Dateien auf diesem ausführen zu können [webmaster;1].

Schuld an dieser Stelle ist oftmals ein fehlerhafter Validierungsmechanismus, der die Benutzereingaben ohne ausreichende Kontrolle an den "include"-Befehl weitergibt.

Es existieren zwei grundlegende Arten von "*File Inclusion*"-Angriffen. Im Folgendem sind diese Angriffe aufgeführt:

Local File Inclusion (LFI)

Die >Local File Inclusion< ist eine Sicherheitslücke, mit der Angreifer ausschließlich lokale Dateien in einem Skript einbinden können [webmaster;1].
Durch diese Sicherheitslücke ist es beispielweise möglich, Inhalte aus Dateien auszulesen, die sich auf dem Zielserver befinden. Beispiele solcher Dateien sind Passwortdateien, Konfigurationsdateien und jede andere Datei, welche sensible oder für den Angreifer interessante Informationen enthält.
Falls die eingebundenen Dateien PHP-Code oder clientseitigen Code enthalten, welcher der Webserver oder Browser interpretieren kann, so kann dieser ausgeführt werden. Auf diese Weise lassen sich PHP-Dateien einbinden und ausführen.
Durch diverse Techniken ist es versierten Angreifer möglich, eigenen Code in vorhandene Dateien einzuschleusen, der mithilfe einer LFI-Sicherheitslücke ausgeführt werden kann.

Remote File Inclusion (RFI)

Die "*Remote File Inclusion*" ist eine Sicherheitslücke, bei der schadhafte Dateien über die "*include*"-Funktion auf einen Webserver eingebunden werden können. Anders als beim LFI können entfernte Dateien eingebunden werden, die sich auf anderen Servern befinden. Somit ist es einem Angreifer ein leichtes, Schadcode in das System einzuschleusen.

Unter PHP ist eine RFI möglich, wenn die Funktionen "*allow_url_open*" und "*allow_url_include*" in der PHP-Konfigurationsdatei auf >ON< geschaltet sind. Wird nun die "*include*"-Funktion genutzt, um eine Datei einzubinden, kann der Angreifer einfach auf eine Datei mit Schadcode verweisen, die an dessen Stelle geladen und ausgeführt wird.

Gegenmaßnahmen

Als guter Schutz vor "*File Inclusion*"-Angriffen erweist sich das sogenannte White-Listing. Das Erteilen einer Erlaubnis für einzelne Vorgänge ist oft einfacher, als das Verbot einer unbekannten Menge [webmaster;1].

Local File Inclusion

- Ähnlich wie beim "*Path Traversal*" können zur Prävention von LFI White-Listen oder Switch-Cases erstellt werden, die festlegen, welche Dateien eingebunden werden dürfen und welche nicht.

- Darüber hinaus sollte der Webserver bzw. die Webapplikation so eingestellt sein, dass der Aufruf von internen Konfigurationsdateien eine Art Alarm auslöst, damit ein Administrator über die Situation informiert wird und sie im Anschluss auswerten kann.

Remote File Inclusion

- PHP bietet die Konfigurationsoption "*allow_url_open*" an, mit der das Öffnen von URLs verboten werden kann. Dies schränkt jedoch gleichzeitig andere Funktionen ein. Dieser Mangel wurde jedoch in der PHP-Version 5.2 behoben, in dem durch diese Konfigurationsoption nur noch das Einbinden und Ausführen von entfernten Dateien mit den genannten PHP-Anweisungen verboten werden kann.
- Auch die Verwendung von Switch-Cases stellt eine Möglichkeit dar, sich vor RFI zu schützen.

3.6 Man-in-the-Middle

Der "*Man-in-the-Middle*"-Angriff, auch Janusangriff genannt, ist ein Angriff, bei dem der Angreifer sich logisch oder auch physisch zwischen zwei Kommunikationspartner stellt. Das System des Angreifers hat dabei die volle Kontrolle über den Datenverkehr zwischen den Kommunikationspartnern und kann diesen nach Belieben einsehen, mitschneiden und sogar manipulieren. Der Angriff ist nur dann erfolgreich, wenn er es schafft, den Kommunikationspartner vorzutäuschen, der jeweilige Gegenüber zu sein. Ende-zu-Ende-Verschlüsselung spielt hierbei eine große Rolle, da dadurch diese Form des Angriffes weitestgehend verhindern werden kann [Imperva;1].
Das Ziel von "*Man-in-the-Middle*"-Angriffen liegt in dem Erhalt von Informationen, mit der der Angreifer Aktionen, wie Identitätsdiebstahl, Transaktionsfälschungen und das Stehlen von geistigem Eigentum durchführen kann.
"*Man-in-the-Middle*"-Attacken ermöglichen das Auslesen der Session-ID, was für Angriffe, wie "*Cookie-Replay*" und "*Session-Hijacking*" verwendet werden kann. Das Vorgehen ist jedoch mit einem gewissen Aufwand verbunden, da der Angreifer Zugang zum Netzwerk benötigt und dann mithilfe von sogenannten "*Packet-Sniffern*" die entsprechenden Informationen mitschneiden zu können [Schäfer;1].

Gegenmaßnahmen

Grundsätzlich sollte bei Webanwendungen immer eine Verschlüsselung (SSL/TLS) im Einsatz sein, um die Möglichkeit eines Man-in-the-Middle-Angriffs auszuschließen bzw. auf ein Minimum zu senken.

Der effektivste Weg, dem Man-in-the-Middle-Angriff entgegenzuwirken, ist die Verschlüsselung der Datenpakete. Hier ist jedoch zu beachten, dass die Schlüssel der Kommunikationspartner über ein zuverlässiges Medium verifiziert werden. Erst durch die gegenseitige Verifikation der Schlüssel, etwa durch eine Zertifizierungsstelle, kann ein solcher Angriff abgewendet werden, da ansonsten den beiden Opfer falsche Schlüssel vorgetäuscht werden können.

Secure Shell (SSH) überprüft beim erstmaligen Login durch den Einsatz von Fingerabdrücken, ob es sich tatsächlich um den Zielrechner handelt, mit dem man die Kommunikation aufnehmen will [Symantec;2].
Das Verschlüsselungsprotokoll "Transport-Layer-Security" kann und wird zur sicheren Datenübertragung im Internet genutzt. Diese Methode beruht auf der Nutzung eines Zertifikates, welches aus einem Schlüsselpaar (öffentlicher und privater) und den zu beschreibenden Informationen besteht. Dieses Zertifikat wird von einer Vertrauensquelle, der Zertifizierungsstelle, unterschrieben, nachdem diese die Identität des Antragstellers verifiziert hat. Es ist jedoch zu beachten, dass bei TLS lediglich die Datenübertragung gesichert und die Identitäten der involvierten Personen verifiziert werden kann. Die Feststellung oder gar der Ausschluss von Datenmanipulation im System der betroffenen Personen ist nicht möglich [Wikipedia;4].

Die Veränderung von Daten kann durch die sogenannte "*Integrity Protection*" verhindert werden. Jede zu übertragende Nachricht wird dem "*Message Authentication Code*" (MAC), einem Identitätsstempel, versehen. Dieser Identitätsstempel wird mithilfe eines Codes erzeugt, der zwischen Netz und Benutzer ausgehandelt wurde. Entspricht der mit der Nachricht empfangene MAC mit dem vom Empfänger erwarteten, wird die Nachricht als gültig anerkannt und verarbeitet [Mobarhan;1].

Eine Mobile TAN (mTAN) kommt als Schutz ebenfalls in Frage. Dem Anwender wird über einen zweiten Kanal per SMS eine TAN zugesendet, die nur für eine einzelne Transaktion genutzt werden kann. Zusätzlich zur TAN werden auch Empfängerdaten übermittelt, so dass der Benutzer überprüfen kann, welche Transaktion er gerade bestätigt. Es ist jedoch zu beachten, dass Trojaner die Zugangsdaten und PIN ausspähen können und somit für "*Man-in-the-Middle-*" Angriffe verwundbar machen [securepedia;1].

3.7 Nullbyte-Injection

Die "*Nullbyte-Injection*" ist ein Angriffsvektor, um URL-codierte Nullbyte-Zeichen an einen anfälligen Parameter anzuhängen. Dieser Vorgang kann die Funktionen einer Web-Applikation so ändern, dass der Angreifer unerlaubten Zugriff zu den Systemdateien erhält [OWASP;7].

Das Nullbyte signalisiert normalerweise das Ende eines Strings. In Sprachen, wie C und C++, wird eine implizierte Zeichenlängencodierung verwendet. Anhand des Nullbytes wird dann verdeutlicht, wann eine Bytesequenz zu Ende ist. Das Nullbyte wird also häufig als Ende einer Bytesequenz interpretiert, was von Angreifern genutzt werden kann [Schäfer;1].

Als Beispiel sei ein kleines Skript gegeben:

```
<?php
   If(isset($_GET[‘datei]))
      Include($_GET[‘datei’] . ‘.jpg’);
?>
```

Auf dem ersten Blick scheint es, als wäre es implementiert worden, so dass nur JPEGs abgerufen werden können. Wird jedoch folgender Aufruf getätigt, so kann die Validierung des Skripts ausgetrickst werden:

```
?datei=geheim.pdf%00
```

Die JPEG-Validierung wird ausgeblendet und stattdessen wird eine PDF-Datei aufgerufen. Das Nullbyte schließt den String, sodass der abschließende Teil ignoriert wird und die PDF-Datei aufgerufen werden kann.
Diese Aktion ist jedoch nicht auf PDF-Dateien beschränkt und kann auf alle anderen Dateiarten angewendet werden und ist äußert effektiv, wenn systeminterne Dateien aufgerufen werden sollen, die überhaupt keine Endung besitzen, wie z.B. die Datei *"/etc/passwd*" [Schäfer;1].

Gegenmaßnahmen

Nullbytes werden selten bei Benutzereingaben für Web-Applikationen genutzt. Somit kann gezielt nach Nullbytes in Benutzereingaben gesucht werden, um diese ohne Ausnahme zu verwerfen [OWASP;7].

PHP bietet die Funktion "magic_quotes_gpc". Diese fügt bei Anfragen mit einem einfachen (‘) und doppelten Anführungszeichen (") einen Backslash (\) ein und

bei einem Nullbyte einen zusätzlichen Backslash (\). Dies verhindert weitestgehend ungewollte Anfragen [php;1].
Da diese Funktion jedoch veraltet ist, sollten eher die Funktionen "*mysql_escape_string*" und "*mysql_real_escape_string*" verwendet werden [php;2].

Moderne APIs sind bereits mit Funktionen ausgestattet, die im günstigsten Fall Nullbyte erkennen und automatisch entfernen oder zum Fehlschlagen der Benutzeranfragen führen.

3.8 Session Fixation

Session Fixation ist ein Angriff, bei dem der Angreifer versucht, die Session-ID eines Benutzers zu fixieren, also zu finden oder zu setzen. Der Angreifer nutzt eine Schwachstelle in der Limitierung bei der Verwaltung von Session-IDs durch Web-Applikationen aus. Wenn ein Benutzer authentifiziert wird, wird diesen keine neue Session-ID zugeteilt, sondern eine, die ihm für eine vergangene Session zugeteilt wurde. Dies ermöglicht dem Angreifer, diese bereits existierende Session-ID zu nutzen [OWASP;8].

Viele Web-Applikationen sehen es vor, dass HTTP-Anfragen einem Benutzer für einen längeren Zeitraum zugeordnet werden können. Aus diesem Grund werden logische Verbindungen, auch Sessions genannt, aufgebaut, die mit einer eindeutigen, möglichst schwer zu erratenden Session-ID ausgestattet werden [w3;1].
Damit der Angreifer Zugriff auf das System erlangen kann, lässt er sich vom anzugreifenden System ein gültige Session-ID ausstellen. Schafft der Angreifer es nun, dass der Benutzer sich auf Basis der Session-ID des Angreifers am System authentifiziert, etwas durch das Unterschieben dieser ID, so erlangt er Zugriff auf das System, solange die von ihm festgelegte Session-ID gilt. Der Angreifer kann sich nun als der legitime Benutzer ausgeben und die Session komplett übernehmen bzw. entführen.

Im Folgendem ist ein Beispiel beschrieben, welches die Session-Fixation-Attacke verdeutlichen soll [thesecuritybuddy;3].

1. Der Angreifer meldet sich am Zielsystem an und notiert den Session-Key (Session-ID) der Session.
2. Der Angreifer sendet nun einen Link, welche seine Session-ID enthält, an das Opfer, indem er eine Reihe an Social-Engineering Tricks anwendet.
3. Das Opfer klickt nun auf den Link und meldet sich am System an.

4. Die Session-ID des Opfers wurde auf dem Wert der Session-ID des Angreifers gesetzt.
5. Der Angreifer kann sich nun an das System anmelden und sich als das Opfer, den legitimen Benutzer, auszugeben.

Gegenmaßnahmen

Jede Session sollte mit zwei Timeouts versehen werden. Ein weiches Timeout sollte bei Nicht-Benutzung der Session und ein hartes Timeout als Maximalzeit einer Session genutzt werden. Die Session-ID sollte außerdem eine begrenzte Gültigkeit bekommen, damit die Session nach Ablauf dieser Zeit eine neue Session-ID erhält.

Bei der Erstellung einer Session sollte diese an eine IP-Adresse, einen Fingerabdruck oder ähnliche einzigartige Informationen gebunden sein. Sollte also nun versucht werden, die Session von einem Computer aufzurufen, dessen Informationen von den originalen abweichen, so wird diesem eine neue Session zugewiesen [acros;1].

Mit jedem neuen Login sollte dem Benutzer immer eine neue Session zugewiesen werden und frühere Informationen über die Identität des Benutzers sollten nicht übernommen werden [thesecuritybuddy;3].

3.9 Session Hijacking

Das "*Session Hijacking*" beschreibt einen Angriff auf eine verbindungsbehaftete Datenübertragung, bei dem der Angreifer versucht, Informationen über die Session-ID anderer Nutzer zu erhalten [OWASP;9].
HTTP ist ein verbindungsloses Protokoll und jede HTTP-Anfrage wird nach dessen Abarbeitung vom Webserver geschlossen [w3;1]. Die meisten Web-Applikationen sind jedoch darauf angewiesen, solche Anfragen bestimmen Benutzer über einen gewissen Zeitraum zuzuordnen. Um dies zu bewerkstelligen, wird eine eigene Sitzungsverwaltung implementiert. Zu Beginn einer neuen Session wird eine eindeutige Session-ID generiert, die dem Benutzer vom Browser zugeteilt wird, um sich damit beim Server identifizieren zu können. Die Session-ID wird über ein GET-, ein POST-Argument, oder über ein Cookie den Benutzer übermittelt. Ist es dem Angreifer nun möglich, diese Session-ID mitzulesen oder zu erraten, kann er sich durch das Mitsenden der Session-ID als authentifizierter Benutzer ausgeben und die Sitzung übernehmen bzw. "entführen". Sollte beim Ändern des Passwortes nicht das alte Passwort erforderlich sein, so begünstigt dies das Aussperren des legitimen Benutzers aus seinem eigenen Account (Account Lockout), falls sich der Angreifer dazu entschließt die Daten des Benutzerkontos zu ändern [Schäfer;1].

Es existieren drei Methoden, um eine Session-Hijacking-Attacke durchzuführen:

Session Fixation. Bei diesem Angriff ändert der Angreifer die Session ID eines Users zu seiner eigenen. Dies wird bewerkstelligt, indem dem Benutzer eine E-Mail mit einem Link zugesendet wird, die eine ganz bestimmte Session ID enthält. Nun muss der Angreifer nur noch warten, bis der betroffene Benutzer sich einloggt.

Session Side Jacking. Bei diesem Angriff nutzt der Angreifer Packet-Sniffer, um den Datenverkehr zwischen zwei Parteien zu lesen und den Session-Cookie zu stehlen. Viele Webseiten nutzen SSL-Verschlüsselung, um die Login-Seite vor Angreifern zu schützen, jedoch kommt keine Verschlüsselung zum Einsatz, nachdem die Authentifizierung stattfand. Somit kann der Angreifer die Datenübertragung mitlesen und die nötigen Daten abfangen, die zum Server gesendet werden. Unter diesen Daten befindet sich der Session-Cookie, mit dem sich der Angreifer als das Opfer ausgeben kann, ohne dass dem Angreifer das Passwort selbst bekannt ist.

Cross-Site-Scripting. Bei diesem Angriff täuscht der Angreifer den Computer des Benutzers, indem diesem weißgemacht wird, dass es sich beim Skript des Angreifers, um vertrauenswürdigen Code handelt, welcher zum Server gehört, um diesen letztendlich auszuführen. Dadurch ist es dem Angreifer möglich, an den Session-Cookie zu kommen und weitere Aktionen auszuführen.

Die oben genannten Methoden "*Session Fixation*" und "*Cross-Site-Scripting*" werden in den gleichnamigen Kapiteln näher erläutert.

Gegenmaßnahmen

Die Verschlüsselung der Datenübertragung durch SSL/TLS stellt eine effektive Form der Abwehr dar. Hierbei muss lediglich der Session-Key verschlüsselt werden, damit der Angriff abgewendet werden kann. Im Idealfall sollte dennoch der komplette Datenverkehr verschlüsselt werden. Durch die Verschlüsselung werden Angriffe, die auf "*Packet-Sniffing*" angewiesen sind, vollständig unterbunden [Nielsen;1].

Webapplikationen sollten einen String bestehend aus einer langen Kette von zufälligen Zahlen nutzen, um das Risiko zu verringern, dass der Angreifer den richtigen Session-Key einfach erraten oder "Brute-Forcen" kann.

Die Webapplikation sollte außerdem zusätzlichen Checks, wie den Vergleich der IP-Adresse mit älteren Sessions, vornehmen, um die Sicherheit zu erhöhen.

Es sollte sichergestellt werden, dass Webapplikationen nicht anfällig für "*Cross-Site-Scripting*" sind. Dies stellt nämlich die Hauptmethode von Angreifern da, um per JavaScript das Dokument "*document.cookie*" auszulesen und somit die Session zu entführen [OWASP;10].

3.10 SQL-Injection

Die SQL-Injection ist ein Angriff, bei dem der Angreifer versucht, eigene Datenbankbefehle über die SQL-Anfrage einer Web-Applikation einzuschleusen. Durch erfolgreiche SQL-Injection-Angriffe können sensitive Daten der Datenbank ausgelesen und modifziert werden, Administrations-Aktionen auf Datenbank-Ebene ausgeführt und in manchen Fällen sogar Befehle an das Betriebssystem gesendet werden. Ziel dieses Angriffes ist es, Daten auszuspähen oder zu verändern, Kontrolle über den Server zu erhalten oder um die Identität eines anderen Benutzers zu übernehmen [Schäfer;1].

SQL-Anfragen, auch "*SQL-Queries*" gennant, sind die Methode, mit der eine Web-Applikation mit der Datenbank eines Servers kommuniziert. SQL-Injection-Angriffe sind immer dann möglich, wenn die SQL-Queries nicht richtig gepflegt und damit die Benutzereingaben nicht ordentlich abgefangen und kontrolliert werden und somit in den SQL-Interpreter gelangen. Die Benutzereingabe könnte dann so modifiziert werden, dass der SQL-Interpreter Sonderfunktionen ausführt und dem Angreifer Einfluss auf die ausgeführten SQL-Befehle ermöglicht. In SQL werden die Sonderfunktionen durch Metazeichen repräsentiert und ausgeführt, darunter Zeichen wie Backslash, das Anführungszeichen, der Apostroph und das Semikolon.

Es gibt viele verschiedene Varianten des "*SQL-Injection*"-Angriffes, welche für die unterschiedlichsten Situationen genutzt werden. Im Folgenden sind zwei dieser Varianten vorgestellt:

SQL-Injection – Log-In Bypass

Bei dieser Variante wird auf das Umgehen des Log-In mittels einer SQL-Injection abgezielt [Schäfer;1].
Die Datenbank besitzt üblicherweise eine Tabelle, die alle Benutzernamen und Passwörter enthält und in etwa so aussieht:

benutzername	passwort
admin	geheimespassword
testnutzer	sicherespassword

Wenn der Benutzer nun im Log-in-Bildschirm seinen Benutzernamen (admin) und sein Passwort (geheimespassword) eingibt, so wird ein SQL-Befehl erstellt, welcher an die Datenbank weitergeleitet wird. Der SQL-Befehl kann folgendermaßen aussehen:

SELECT * FROM benutzer WHERE benutzername='admin' and password='geheimespassword'

Dieser SQL-Befehl wird anschließend an die Datenbank weitergeleitet, die dann die eingegebenen Daten mit denen in der Datenbank vergleicht. Zuallererst wird nach dem Benuternamen "*admin"* gesucht und, falls dieser existiert, das eingegebene Password mit dem aus der Datenbank verglichen. Stimmen diese Daten überein, wird ein positiver Wert zurückgegeben und der Log-in erfolgreich ausgeführt. Somit ist der Benutzer am System angemeldet und zur Aufrechterhaltung wird eine Session erstellt.
Sollte der Benutzername in der Datenbank nicht gefunden werden, oder stimmen die Passwörter nicht überein, wird die Anfrage verworfen und ein Nullwert zurückgegeben.

Die Herausforderung bei dem Login Bypass besteht nun darin, SQL-Code einzuschleusen, damit ein positiver Wert beim Log-in zurückgegeben wird. Die Eingabefelder des Log-in-Formulars eignen sich als hervorragender Injektionspunkt für die "*SQL-Injection*".
Sollte es zu Syntaxfehlern oder Fehlermeldungen bei der Injektion von SQL-Code kommen, so führt dies zwar zu einem erfolglosen Log-in-Versuch, jedoch gibt diese Ausschluss darüber, ob und welche unbereinigten Anfragen die Applikation erlaubt [Schäfer;1].

In diesem Versuch nehmen wir an, dass uns einer der Benutzernamen bekannt ist, beispielweise "*admin"*.
Tragen wir nun "*admin"* in das Feld 'Benutzernamen' ein und den Wert >or 1='1'< in das Feld 'Password', so wird folgender SQL-Befehl erstellt:

SELECT * FROM benutzer WHERE benutzername='admin' and password='or 1='1'

Wird dieser Wert nun an die Datenbank weitergeleitet, wird ein positiver Wert zurückgegeben und der Log-in war erfolgreich. Dies ist möglich, da der Benutzername "*admin*" existiert und der letzte Teil der Anfrage immer positive sein wird, denn 1=1 ist immer zutreffend. Durch die eingefügte "*or*"-Verknüpfung muss nur eine der beiden letzten Bedingungen erfüllt sein.

Eine weitere Möglichkeit wäre außerdem, dass der doppelte Bindestrich '- -' genutzt werden kann:

SELECT * FROM benutzer WHERE benutzername='admin' - -' and password=' '

Der doppelte Bindestrich '—' wird in SQL als Kommentarbeginn genutzt, es wird also nur der erste Teil des SQL-Befehls ausgeführt. Da der Benutzername "*admin*" sich in der Datenbank befindet, wird auch hier ein positiver Wert zurückgegeben und der Log-in erfolgreich ausgeführt.

Für die oben aufgelisteten Beispiele wurde immer ein Benutzername benötigt, der in der Datenbank hinterlegt ist. Dieses Wissen steht einem Angreifer jedoch in den wenigsten Fällen zur Verfügung, wodurch andere Schritte von Nöten sind. Entweder wir erraten eines der Benutzernamen mithilfe des >Brute-Force-Angriffes< oder wir nutzen weiterhin "*SQL-Injection*", um das gewollte Ergebnis zu erzielen.

Statt nur das Feld "*Password*" mit der Eingabe >or '1'='1'< zu versehen, kann das gleiche für das Feld 'Benutzername' getan werden:

SELECT * FROM benutzer WHERE benutzername=' ' or '1'='1' and password=' ' or '1'='1'

Diese Anfrage wird für beide Teile der Anfrage einen positiven Wert zurückgeben und der Log-in ist wieder einmal erfolgreich.
Es kann jedoch auch dazu kommen, dass der Log-in komplett fehlschlägt, da kein Benutzername angegeben wurde.

Blind-SQL-Injection
Die Blind-SQL-Injection stellt einen weiteren Typ der SQL-Injection dar. Sie unterscheidet sich vom normalen SQL-Injection-Angriff in der Hinsicht, dass hier der Angreifer die Datenbank nicht direkt updated oder löscht, sondern lediglich Informationen über die Datenbank per SQL-Queries sammelt. Dies ermöglicht dem Angreifer, vorhandene Schwachstellen für weitere Angriffe ausfindig zu machen.

Für die Durchführung von Blind-SQL-Injections gibt es zwei unterschiedlichen Methoden. Beide basieren jedoch darauf, dass der Angreifer einen Wahrheitswert als Rückgabewert erhält und somit Informationen aus der Datenbank auslesen kann. Diese Art der SQL-Injection benötigt jedoch weitaus mehr Aufwand, als die bereits vorgestellte Methode.

Boolean-based Blind-SQL-Injection
Diese Technik beschreibt das Vorgehen, einzelne Zeichen aus Strings zu extrahieren und im Anschluss zu verarbeiten. Durch die Umwandlung versuch man einen Wahrheitswert zu erhalten. Dadurch, dass es keine Ausgabe gibt, müssen wir uns mit den Zuständen "*wahr*" und "*falsch*" begnügen. Um an die benötigten Informationen zu kommen, reichen die Angaben jedoch aus [Schäfer;1].

Nehmen wir einmal an, dass wir uns in einer Web-Applikation "*booksearch.com*" befinden, in der der Benutzer nach einem bestimmten Buch mit dem Namen des Autors, dem Buchtitel oder der Buchbeschreibung suchen kann [thesecuritybuddy;1]. Gibt der Benutzer den Namen 'John' in die Suchleiste ein, so wird die folgende URL geladen:

booksearch.com/buecher.php?author=john

Die Anfrage der Benutzer nimmt folgende Form als SQL-Code an:

SELECT * FROM bookinfo WHERE author = 'john';

Nach dessen Ausführung werden die Daten der Datenbank entnommen und dem Benutzer formattiert präsentiert.
Nun könnte ein Angreifer die URL modifizieren:

booksearch.com/books.php?author=john OR 1=1

Sollte die Web-Applikation den Input des Benutzers direkt in die SQL-Query einbetten, so wird folgender SQL-Befehl erstellt:

SELECT * FROM bookinfo WHERE author = 'john' OR 1=1;

Tritt dieser Fall ein, so werden dem Benutzer alle Bücher, die sich in der Datenbank befinden, auf der Webseite angezeigt.
Dies kann nun mit einer leichten Modifizierung an der URL wiederholt werden:

booksearch.com/books.php?author='john' AND 1=1

Besteht die vorhin ausgenutzte Schwachstelle weiterhin, so der folgende SQL-Befehl erstellt:

SELECT * FROM bookinfo WHERE author='john' AND 1=2;

Die Datenbank wird natürlich den Boolean-Wert "*false*" an die Web-Applikation zurückgeben, was eine generische Fehlermeldung verursacht oder eine Fehlermeldung der Datenbank anzeigt. Dies liefert dem Angreifer die Information, dass die Web-Applikation gegen "*Boolean-based Blind-SQL-Injection*" nicht geschützt ist. Der Angreifer kann daraufhin weitere Queries vorbereiten, die ihm zusätzliche Informationen liefern.

Time-based Blind-SQL-Injection

Bei dieser Methode nutzt man ebenfalls boolesche Werte, jedoch ohne, dass der Angreifer eine Fehlerausgabe erhält. Die Datenbank wird dazu angewiesen, eine Ausgabe nach einer gewissen Zeit vorzunehmen. Mithilfe der Zeitabstände kann man die Wahrheitswerte ermitteln und die Textwerte auslesen [Schäfer;1].

Wird eine Anfrage verschickt, die nach fünf Sekunden antworten soll, und dann auch nach etwa fünf Sekunden antwortet, so weiß der Angreifer, dass die Anfrage 'TRUE' war. Sollte die Antwort länger oder kürzer als erwartet sein, so ist die Anfrage 'FALSE'.

Eine Zeitverzögerung kann bei MySQL mit folgendem Befehl bewerkstelligt werden:

```
SLEEP(5)
```

Wird die Zeitverzögerung nun mit anderen SQL-Befehlen verknüpft, können dadurch mächtige Befehle entstehen. Falls zum Beispiel die folgende Anfrage nach zehn Sekunden erfolgreich durchgeführt wird, wurde der Wert 'TRUE' zurückgegeben. Das bedeutet, dass das MySQL-System die Version 5.X.X hat.

```
SELECT MID(VERSION( ),1,1) = '5', SLEEP(10),0)
```

Diese Art von Injektion ist äußerst selten, weswegen es für Angreifer die Möglichkeit gibt, "*Heavy Queries*" zu benutzen. Diese führt bewusst zu einer kurzfristigen Überlastung der Datenbank, um zu kontrollieren, wie schnell geantwortet wird.
Bei MySQL-Datenbanken kann zur Überlastung der Befehl "*benchmark()*" genutzt werden. Dieser führt einen Befehl beliebig oft aus und erwartet zwei Parameter: die Anzahl der Durchläufe und die auszuführende Aktion.

```
SELECT BENCHMARK (1000000, MD5(65)));
```

Dieser Befehl kann die Datenbank für einige Sekunden überlasten. In diesem Beispiel wird der MD5-Hash des Buchstaben 'A' 1000000-mal berechnet. Diese Aktion dauert zwar nur einige Sekunden, kann aber durch Modifikation des Befehls auf mehrere Stunden ausgeweitet werden.
Die Überlastung der Datenbank macht sich im Arbeitsspeicher oder auf der CPU bemerkbar, was nicht im Interesse des Angreifers liegt. Wird die Überlastung von einem Admin frühzeitig bemerkt, so läuft der Angreifer Gefahr, ertappt zu werden.

Gegenmaßnahmen

Vom Benutzer eingegeben Daten dürfen nicht ohne Kontrolle in einen SQL-Befehl eingebaut werden. Durch das 'Escapen' der Eingaben werden z.B. Anführungszeichen maskiert und somit vom SQL-Interpreter ignoriert.

Mit ‘Prepared Statements’ kann auf das Kappen der Eingaben verzichtet werden. Die eingegebenen Daten werden als Parameter an den bereits kompilierten SQL-Befehl übergeben. Das verhindert die SQL-Injection, da die Daten nicht interpretiert werden [thesecuritybuddy;5].
Web Application Firewalls (WAFs) können SQL-Injections teilweise verhindern, falls der Betreiber der Webserver keine Kontrolle über die Anwendungen hat.

4. Simulation von Angriffsszenarien

Um mit den verschiedenen Szenarios beginnen zu können, muss eine geeignete Testumgebung gewählt werden. An dieser Stelle entschied ich mich für eine Kombination aus den Betriebssystemen "*Kali Linux*" und "*Metasploitable*".

"*Metasploitable*" dient als Testumgebung für Szenarios. Hierbei handelt es sich um ein Betriebssystem, welchem absichtlich Schwachstellen eingebaut wurden, um eine Testumgebung für Schulungszwecke zu schaffen. Ausgestattet ist das Betriebssystem mit mehreren Anwendungen und werde ich mich hier auf die Arbeit mit der Web-Applikation "*Mutillidae*" konzentrieren.

"Kali Linux" erfüllt den Zweck des Test-Frameworks, mit dem die eigentlichen Szenarios durchgeführt werden. Da es über zahlreiche Tools für das Auffinden und Ausnutzen von Schwachstellen verfügt, eignet es sich hervorragend für diese Aufgabe.

Auf die Installation und die Konfiguration der Testumgebung wird in dieser Arbeit nicht weiter eingegangen. Zum einen existieren dazu genügend Tutorials online. Zum anderen bietet es keine relevanten Informationen zum eigentlichen Penetrationtesting.

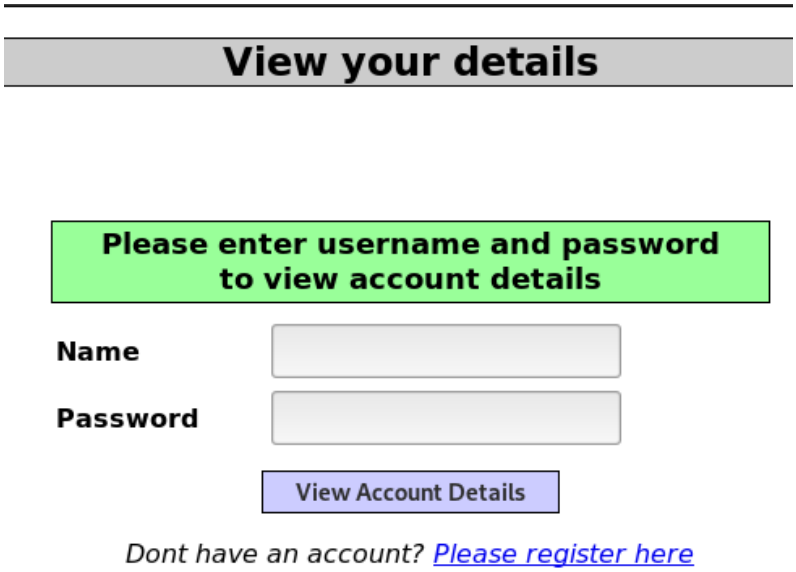
4.1 Szenario #1: SQL-Injection

In diesem Szenario gehe ich auf drei verschiedene Formen der "*SQL-Injection*" ein, welche sich in der Durchführung leicht voneinander unterscheiden.

4.1.1 SQL-Injection – Extrahierung von Benutzerdaten

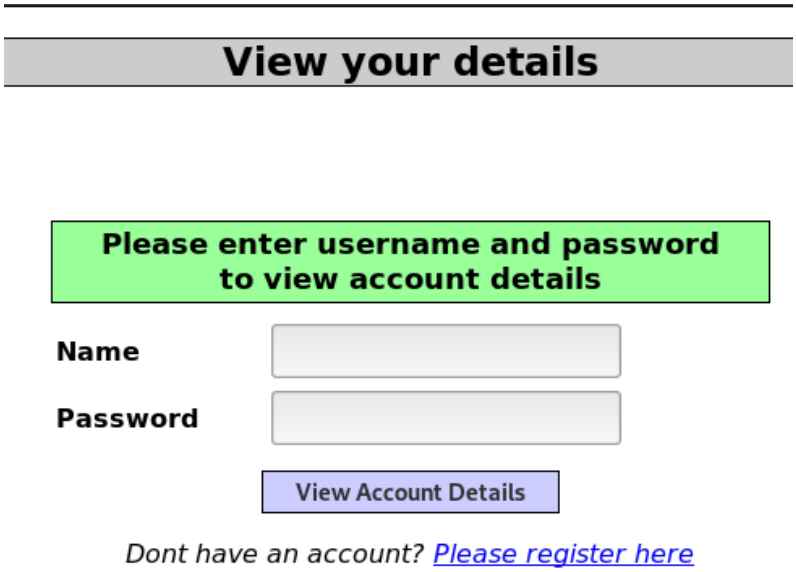
In der Web-Applikation "Mutillidae" muss man nun folgendem Pfad folgen:

Dies öffnet ein Web-Formular, mit dessen Hilfe man mit der Datenbank kommuniziert.



Da wir weder über einen Account verfügen, noch den Namen eines Benutzers des Systems kennen, wird dementsprechend ein Account angelegt, um zwei Ziele zu verfolgen. Das erste Ziel liegt darin, einen Benutzeraccount im System angelegt zu haben, um diesen für zukünftige Angriffe nutzen zu können. Das zweite Ziel ist die Analyse der Funktionsweise der Applikation, denn das Anlegen eines Benutzers kann schon erste Informationen liefern.

Um einen Account anlegen zu können, muss das Feld "*Please register here*" ausgewählt werden und die nötigen Felder ausgewählt werden.



Account created for pentest. 1 rows inserted.

Nach der Bestätigung der Benutzerdaten werden uns erste Informationen über die Applikation preisgegeben. Zum einem werden Benutzerdaten nach dem Ausfüllen des Web-Formulars ausgegeben, zum anderen kann anhand der Ausgabe "*1 rows inserted*" gemutmaßt werden, dass im Hintergrund ein SQL-Befehl abgearbeitet wird. Dies deutet darauf hin, dass die Datenbank durch unsere Aktion manipuliert wurde.

Nun gehen wir zurück auf die Webseite, auf der sich das eigentliche Web-Formular befand. Durch die Eingabe der Benutzerdaten des gerade eben angelegten Benutzers, wird uns bei erfolgreichem Vergleich der Eingaben mit dem Datenbestand eine Ausgabe zurückgegeben.

Results for . 1 records found.

Username=pentest
Password=pentest
Signature=sql-injection

Wie erwartet werden alle Informationen des Benutzers ausgegeben, da der Log-in erfolgreich war. Dies beweist außerdem, dass wir mit der Datenbank direkt interagieren können.

Nehmen wir nun einmal die URL genauer unter die Lupe.

192.168.0.177/mutillidae/index.php?page=user-info.php&username=pentest&password=pentest&user-info-php-submit-button=View+Account+Details

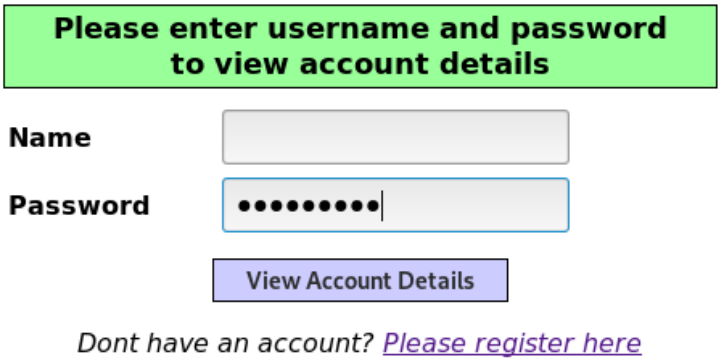
Anhand der URL ist zu erkennen, dass die Antwort der Datenbank, eine weitere Webseite aufruft, der die Benutzerdaten in Klartext (unverschlüsselt) übergeben werden.
Dadurch ist ein direktes Editieren der URL möglich und die Eingaben müssen nicht mehr über das Web-Formular erfolgen. Dies ermöglicht die Erstellung eines Skripts, welches durch Brute-Forcing versucht, den Benutzernamen und das Passwort eines Benutzers zu erraten.

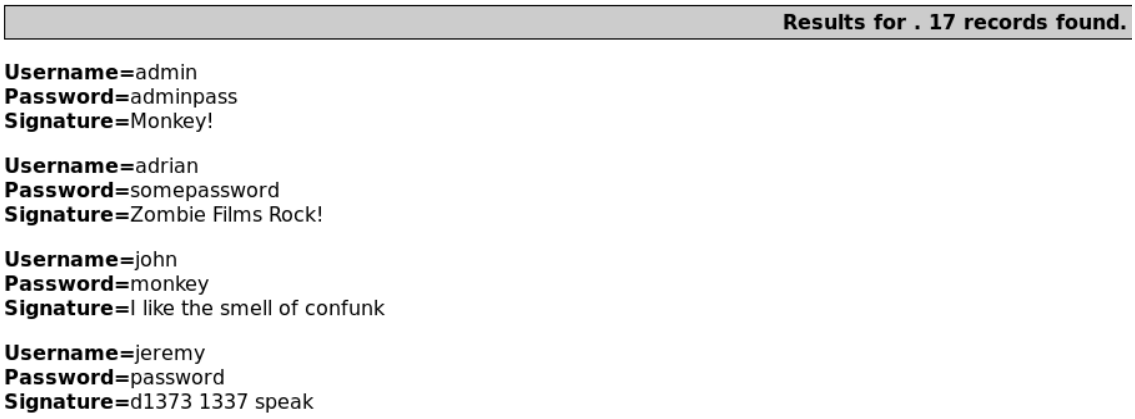
Um herauszufinden, wie gut die Web-Applikation geschrieben ist, fügen wir dem Wert von "Password" ein Apostroph hinzu.

Error: Failure is always an option and this situation proves it	
Line	126
Code	0
File	/var/www/mutillidae/user-info.php
Message	Error executing query: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near ''pentest1'' at line 1
Trace	#0 /var/www/mutillidae/index.php(469): include() #1 {main}
Diagnotic Information	SELECT * FROM accounts WHERE username='pentest' AND password='pentest1''
Did you setup/reset the DB?	

Diese Eingabe führte zur Erstellung einer Fehlermeldung, welche nun weiter untersucht werden kann.
Die Fehlermeldung besitzt zwei interessante Einträge: Die betroffene Datei und die Query, die nicht ausgeführt werden konnte. Wenn wir nun die Query manipulieren können, wäre die Ausgabe weiterer Daten der Datenbank möglich. Dafür werden jedoch Kenntnisse in der Programmiersprache “SQL” benötigt.

Wir manipulieren nun die Query, indem man den String ‘ OR 1=’1 in das Password-Feld des Web-Formulars eingibt. Durch die nun erstellte Query muss nur eines der Statements wahr sein und da 1=1 immer wahr ist, wird der Query ausgeführt.

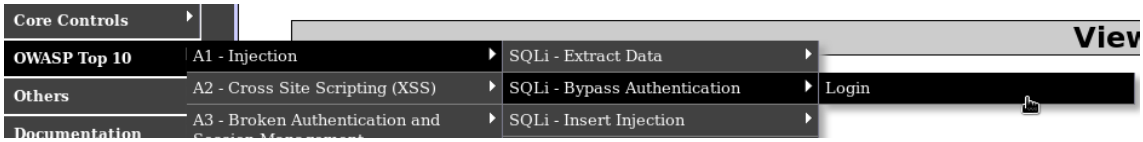




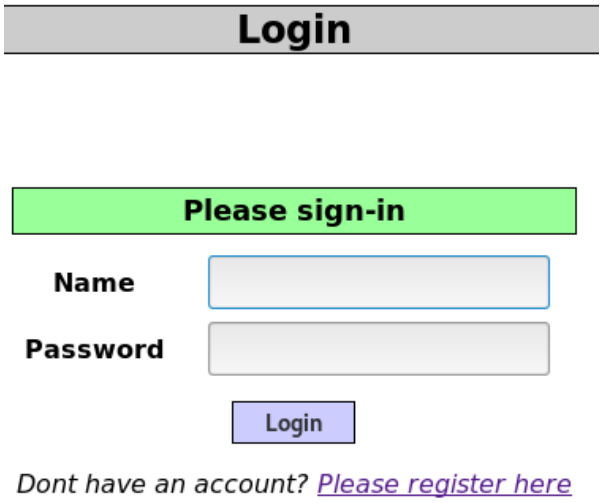
Durch unsere Manipulation der Query wurden die Benutzerdaten aller Benutzer, die in der Datenbank gespeichert wurden, an uns zurückgegeben.
Dies ist möglich, da die Query kein “*Where-Statement*” besitzt und dadurch jeden einzelnen Eintrag der Datenbank mit der Query vergleicht. Stimmt ein Eintrag mit dem Query-Werten überein, so wird dieser als Ausgabe zurückgegeben.

4.1.2 SQL-Injection – Authentication Bypass

In der Web-Applikation “Mutillidae” muss man nun folgenden Pfad folgen:



Dies leitet uns zu einem Web-Formular weiter.



Um jetzt die Benutzer-Authentifizierung umgehen zu können, müssen wir erst einmal herausfinden, wie die SQL-Query aussieht. Dazu geben wir in eines der beiden Formular-Felder eines der Sonderzeichen ein, welches für SQL-Queries reserviert wurde. Dies führt zu einem SQL-Fehler, sobald die Query ausgeführt wird. Wir versuchen es nochmals mit einem Apostroph.

Error: Failure is alwa	
Line	49
Code	0
File	/var/www/mutillidae/process-login-attempt.php
Message	Error executing query: You have an error in your SQL syntax; check th
Trace	#0 /var/www/mutillidae/index.php(96): include() #1 {main}
Diagnotic Information	SELECT * FROM accounts WHERE username='' AND password=''
Did	

Die Fehlermeldung besitzt zwei interessante Einträge, die uns beim Erreichen des Ziels weiterbringen werden: Die betroffene Datei und das SQL-Query.
Da wir über keine Benutzernamen verfügen, die in der Datenbank hinterlegt sein könnten, bedienen wir und einem Trick.
In das "Name"-Feld tragen wir ganz einfach den String ' OR 1=1 -- ein.

Dies hat zur Folge, dass das Statement, welches im "Name"-Feld steht, den Wert "TRUE" annimmt (denn 1=1 ist immer wahr) und alle Angaben, die nach dem "Name-Feld" kommen, ignoriert werden. Das Ignorieren des restlichen SQL-Queries wird durch das Hinzufügen von einem Doppel-Bindestrich ausgelöst.

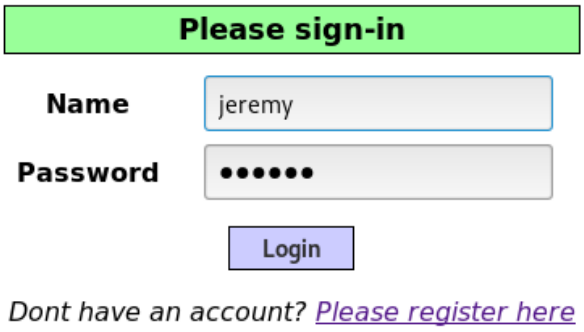
Durch die erfolgreiche Ausführung der SQL-Query werden wir am System angemeldet. Man wird an dieser Stelle als "*admin*" eingeloggt, da dies der erste Eintrag in der Datenbank ist und die SQL-Query den erstbesten Benutzer ausgewählt hat.

Falls man die Kontrolle über das Konto eines ganz bestimmten Benutzers erlangen möchte, so müssen wir anfänglich dessen Benutzernamen herausfinden. Den Account,

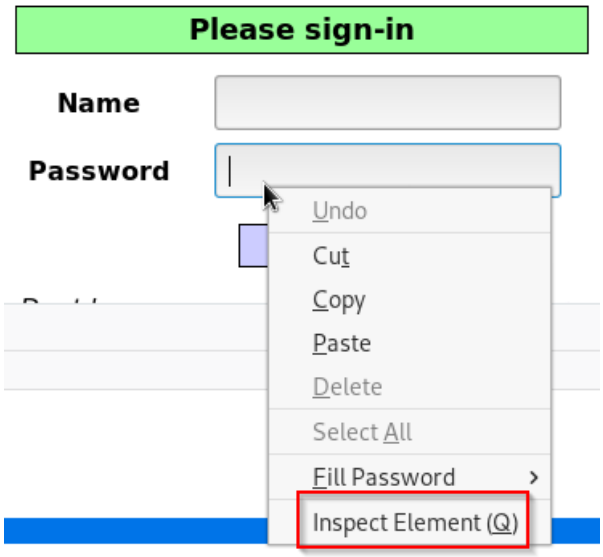
mit dem wir uns anmelden wollen, hat den Benutzernamen “jeremy”. Durch die erste SQL-Injection konnten wir den Benutzernamen der Person herausfinden.

Username=jeremy
Password=password
Signature=d1373 1337 speak

Wir ignorieren an dieser Stelle das Passwort, da wir eine Situation simulieren wollen, indem wir dieses noch nicht herausgefunden haben und kehren zum ursprünglichem Web-Formular zurück.



An dieser Stelle kommt es zu einem Problem, da zum einem der SQL-Befehl nicht in das Password-Feld eingetragen werden kann, da eine Begrenzung der Eintragslänge festgelegt wurde. Außerdem handelt es sich hier um ein Feld des Typs “*password*”, was den String beim Eintrag in das Feld unkenntlich macht.
Mit einem Rechtsklick auf das Password-Feld öffnet sich das Optionen-Menü.



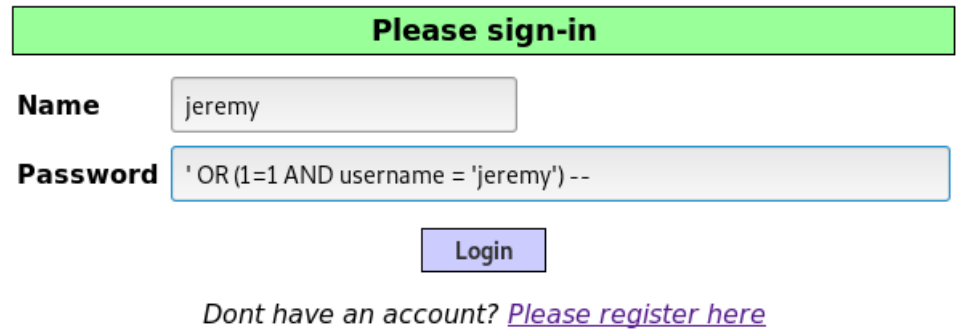
Wählen wir nun “Inspect Element”, werden wir im Seitenquelltext an die richtige Stelle weitergeleitet.

```
<input name="password" maxlength="20" size="20" type="password">
```

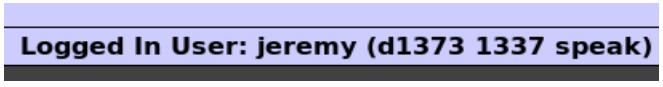
Nun ändern wir den Eintrag type=”*password*” auf type=”*input*” und “*maxlength”* von 20 auf 100.

```
<input name="password" maxlength="100" size="20" type="input">
```

Zum Schluss füllen müssen wir das Query folgendermaßen ausfüllen:



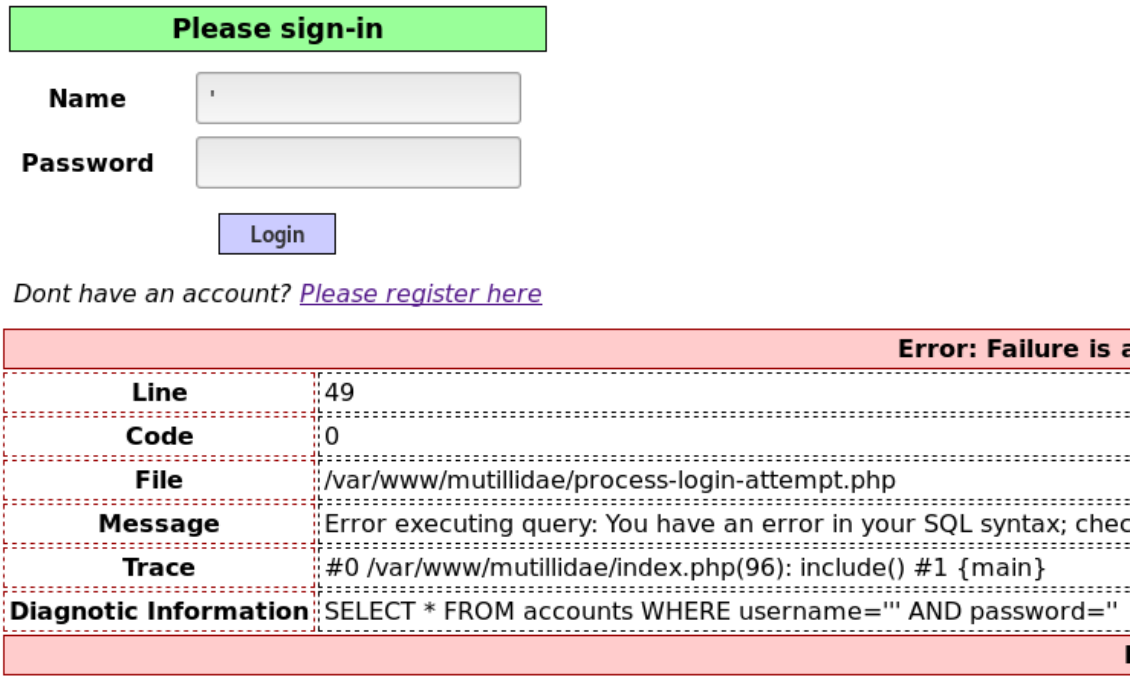
Diese Anweisung verhilft uns anschließend, als *“jeremy”* am System angemeldet zu werden.



4.1.3 SQL-Injection – Timing-Attack

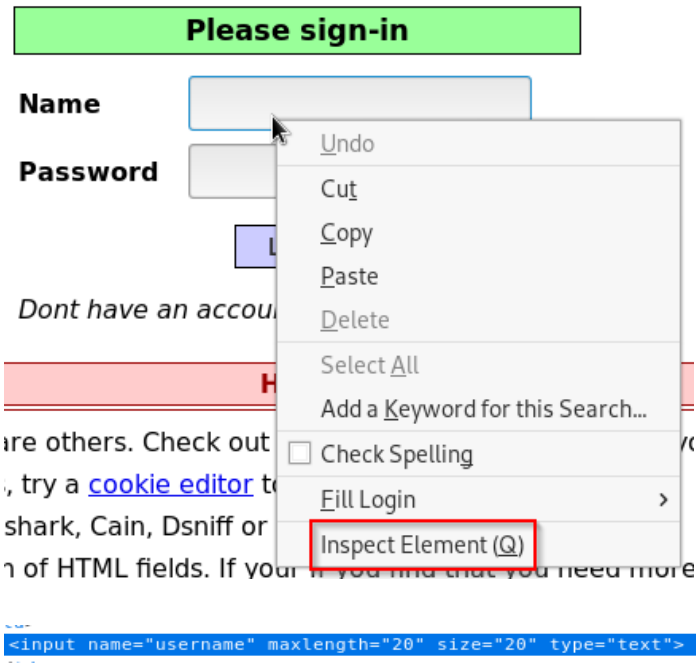
Obwohl Timing-Attacks in der Web-Applikation “Mutillidae” nicht nötig sind, da wir die Fehlermeldungen einsehen können, wird dieser Angriff aus Zwecken der Demonstration des Ablaufs trotzdem vorgestellt.
Wir kehren zu Beginn zu einem der Log-in Web-Formulare zurück.

Um nun herauszufinden, ob die Datenbank eine Schwachstelle hat, nutzen wir wieder eines der Sonderzeichen.



Die Eingabe erzeugt, wie erwartet, eine Fehlermeldung, was auf eine SQL-Injection-Schwachstelle deutet. Der Fehlermeldung können wir wieder den Aufbau der SQL-Query entnehmen.

Um den Timing-Angriff nun ausführen zu können, gibt es mehrere Wege. Einer dieser Wege nutzt den Befehl "*sleep*" der mySQL-Datenbank, um seinen Betrieb für eine gewisse Zeit zu pausieren, um eine Antwort der Web-Applikation zu verzögern.
Da der Befehl relativ lang ist, muss "*maxlength*" des "*Name*"-Feldes erhöht werden. Dafür führen wir einen Rechtsklick auf das "Name"-Feld aus und wählen "*Inspect Element*".



Der "*maxlength*"-Wert muss nun auf einen höheren Wert geändert werden.

```
<input name="username" maxlength="100" size="50" type="text">
```

Anschließend wird der SQL-Query in das "*Name*"-Feld eingegeben, um den Timing-Angriff zu starten.

Diese SQL-Query führt zu einem Fehler, da die Spalten der Datenbank mit den Spalten, die wir versuchen aufzurufen, nicht übereinstimmen. Der Befehl "*sleep(5)*" wird hier als eine Tabelle mit einer Spalte interpretiert. Die eigentliche Tabelle der Benutzer hat weitaus mehr, deswegen auch die Fehlermeldung.
Fügen wir unserer Tabelle nun leere Spalten hinzu, sollte die Anfrage funktionieren. Dies ist möglich, indem wir "null"-Einträge in die SQL-Query einbauen.

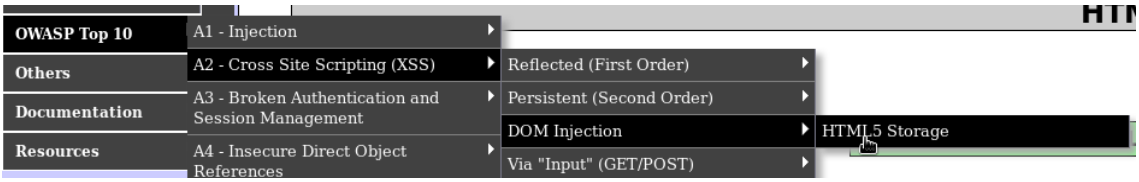
Wenn es nun zu einer Verzögerung bei dem Erhalt einer Antwort gibt, so war der Timing-Angriff erfolgreich.

4.2 Szenario #1: Cross-Site-Scripting

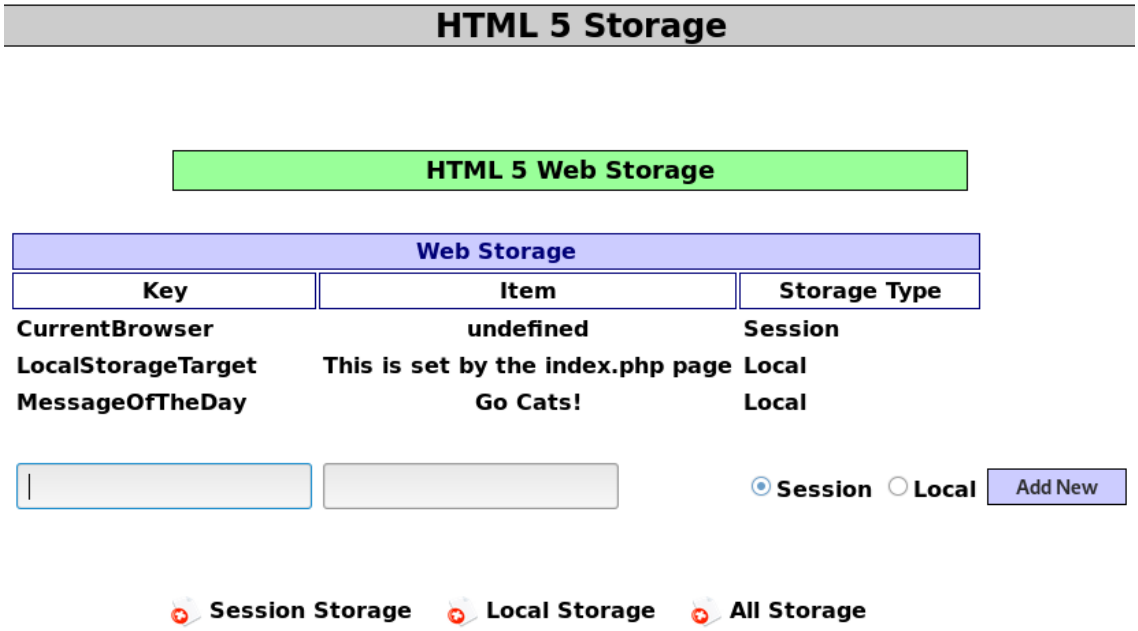
In diesem Szenario gehe ich auf drei verschiedene Formen des Cross-Site-Scriptings ein, welche sich in der Durchführung leicht voneinander unterscheiden.

4.2.1 Cross-Site-Scripting – DOM-based

In der Web-Applikation “Mutillidae” muss man folgenden Pfad folgen:



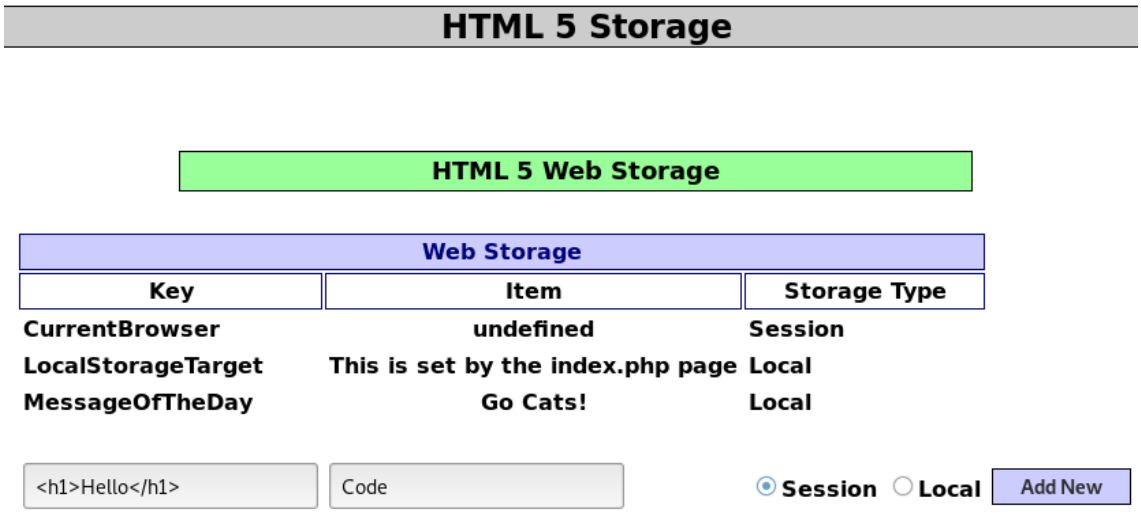
Dies öffnet eine HTML5 Web-Storage Seite.

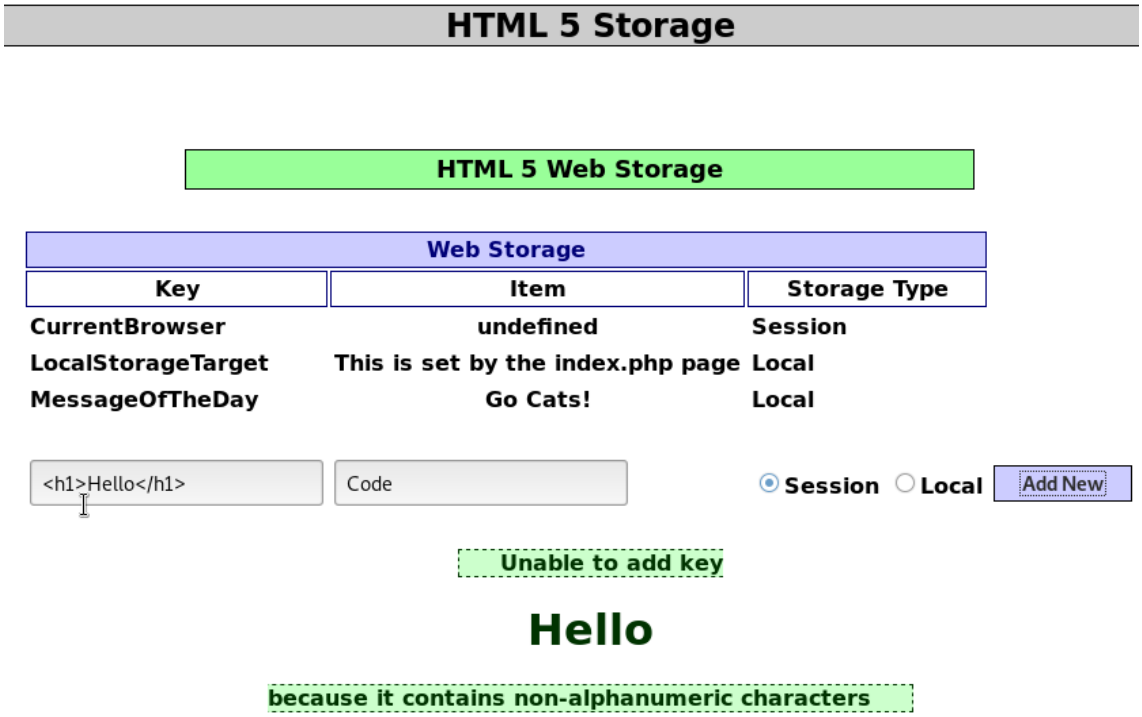


Die Webseite verfügt über zwei Eingabefelder, mit der neue Objekte in den Web-Storage aufgenommen werden können. Die Einträge werden jedoch nicht an den Server weitergeleitet, sondern nur auf der Webseite durch das lokale JavaScript-Skript gespeichert.
Dies kann überprüft werden, indem man den Seitenquelltext der Webseite aufruft und nach den JavaScript-Funktionen sucht.

```
var setMessage = function(/* String */ pMessage){
    var lMessageSpan = document.getElementById("idAddItemMessageSpan");
    lMessageSpan.innerHTML = pMessage;
    lMessageSpan.setAttribute("class","success-message");
};// end function setMessage
```

Durch die Analyse des Seitenquelltextes wurde letzten Endes auch die Schwachstelle der Webseite entdeckt. Durch die oben gezeigte "*setMessage*"-Funktion, wird das Element, was gerade erstellt wird, an die Webseite gesendet. Die "inner.HTML"-Funktion sendet das Element jedoch nicht als Text, sondern als Code, was die Funktion für Cross-Site-Scripting anfällig macht. Dieser Code kann dann vom lokale JavaScript ausgeführt werden.
Geben wir nun JavaScript-Code in das Eingabefeld ein, so können wir die Webseite verändern.

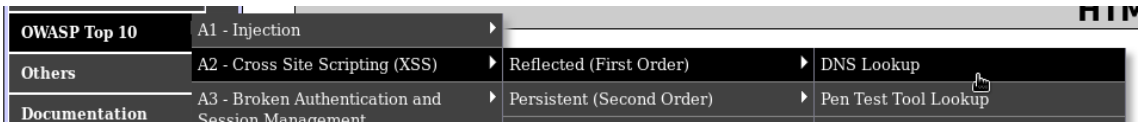




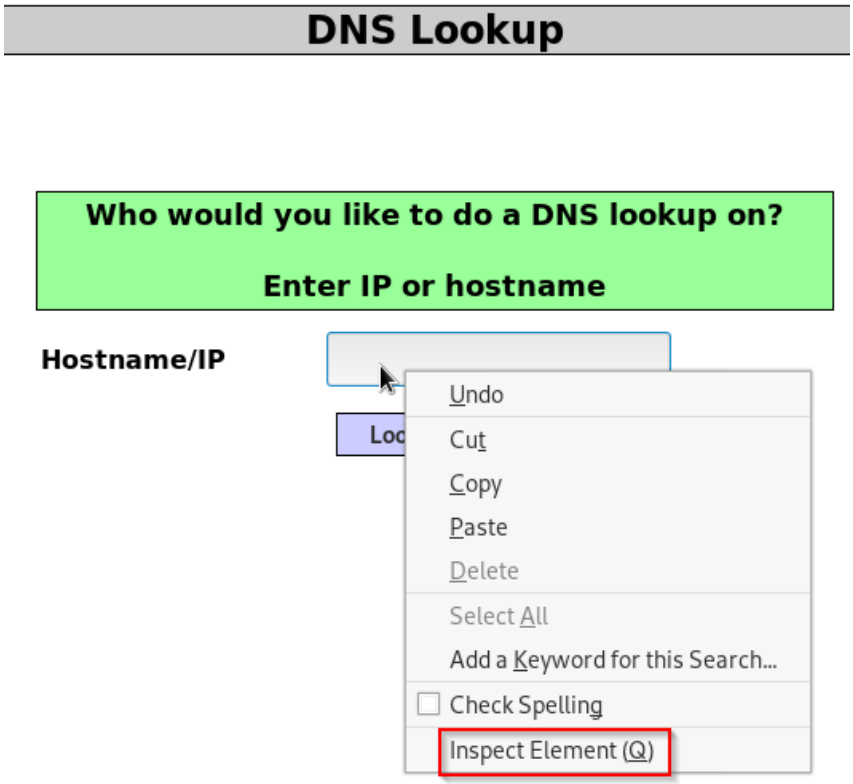
Da der Browser nicht zwischen Text und Code unterscheiden kann, wurde die JavaScript-Anweisung beim Hinzufügen in das Web Storage ausgeführt, was zu einer Veränderung der Webseite führte.
Mit dieser Methode können sogar ganze Skripte in die Webseite eingeschleust werden, die großen Schaden anrichten können.

4.2.2 Cross-Site-Scripting – Reflektiert

In der Web-Applikation “Mutillidae” muss man folgenden Pfad folgen:



Ob die Web-Applikation gegen diese Art von Cross-Site-Scripting anfällig ist, finden wir heraus, indem wir vor dem Senden einer Anfrage zuallererst das Eingabefeld inspizieren.



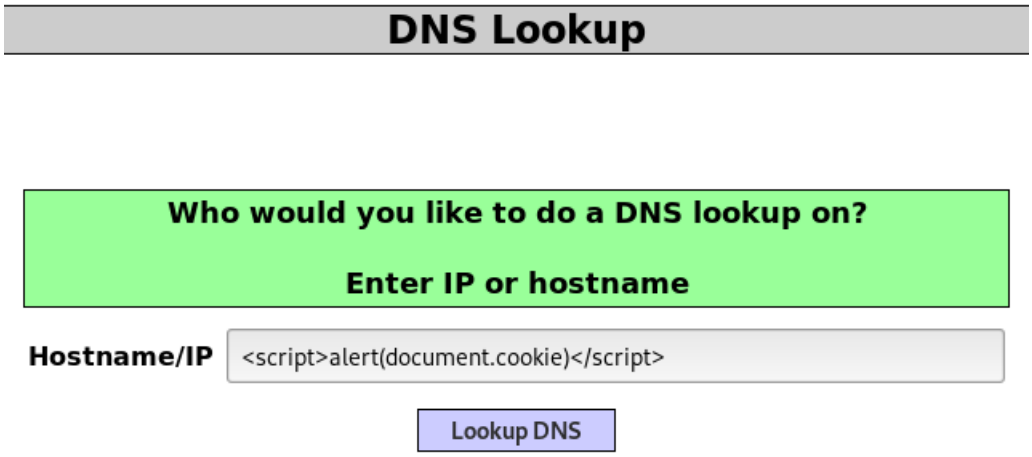
Es ist nun ersichtlich, wie die Bezeichnung des Eingabefeldes lautet. Wenn wir im oberen Teil des Seitenquelltextes nachschauen, sehen wir folgende Informationen:

```
<form id="idDNSLookupForm" action="index.php?page=dns-lookup.php" method="post"
  <table style="margin-left:auto; margin-right:auto;">
```

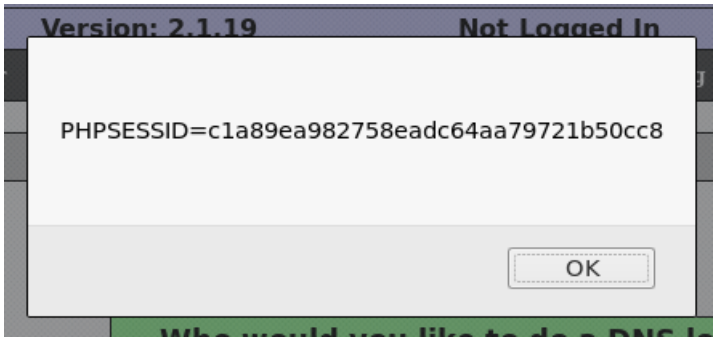
Das Formular wird als “POST” an die Webseite gesendet. Wird nun eine Eingabe getätigt, wird diese an den Server geschickt, der diese dann auswertet. Nach einer kurzen Verzögerung schickt der Server eine Antwort mit den Ergebnissen an den Client.
Wird die Antwort abgefangen, kann folgendes festgestellt werden:

```
<div class="report-header" ReflectedXSSExecutionPoint="1">Results for canary</div><pre
```

Der im Eingabefeld eingegebene Wert wurde vom Server an die Web-Applikation zurück reflektiert. Die Web-Applikation wertet die Antwort aus und bindet diese auf die Webseite ein. Die eröffnet Möglichkeiten für Angriffe durch reflektiertes Cross-Site-Scripting.

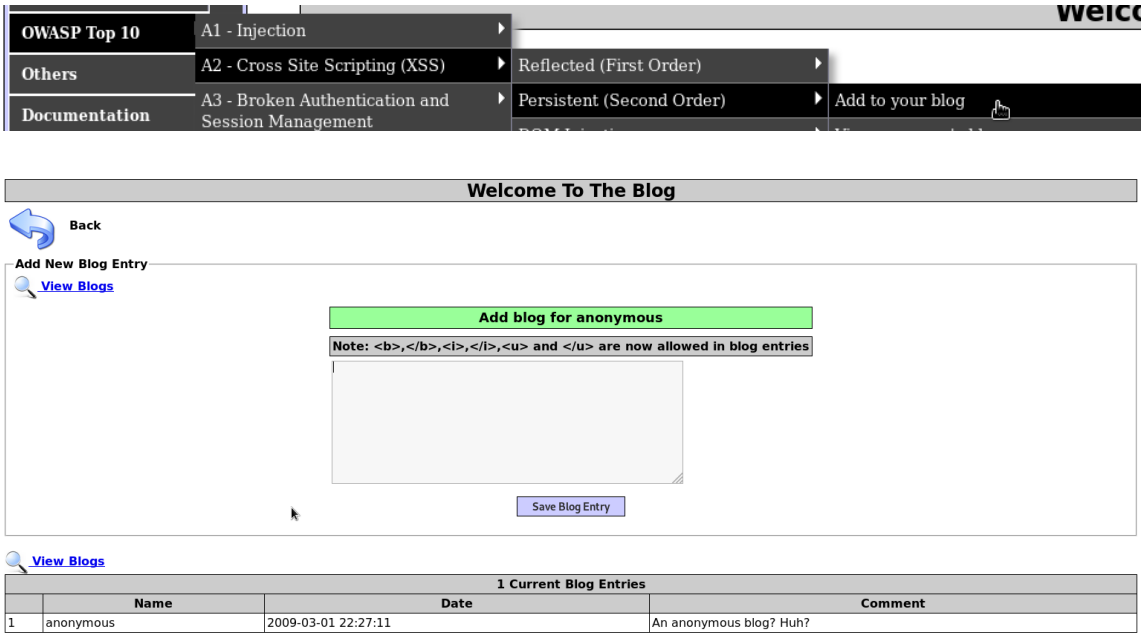


Durch diese Eingabe wird eine “Alert-Box” erstellt, die das Cookie der Webseite abruft. Wird dieses Skript in die Webseite eingebunden, wird der Browser nicht erkennen können, dass es sich hier nicht um Text, sondern um ein Skript handelt. Dies führt dazu, dass der Browser das Skript ausführt und das Cookie über ein separates Fenster anzeigt.

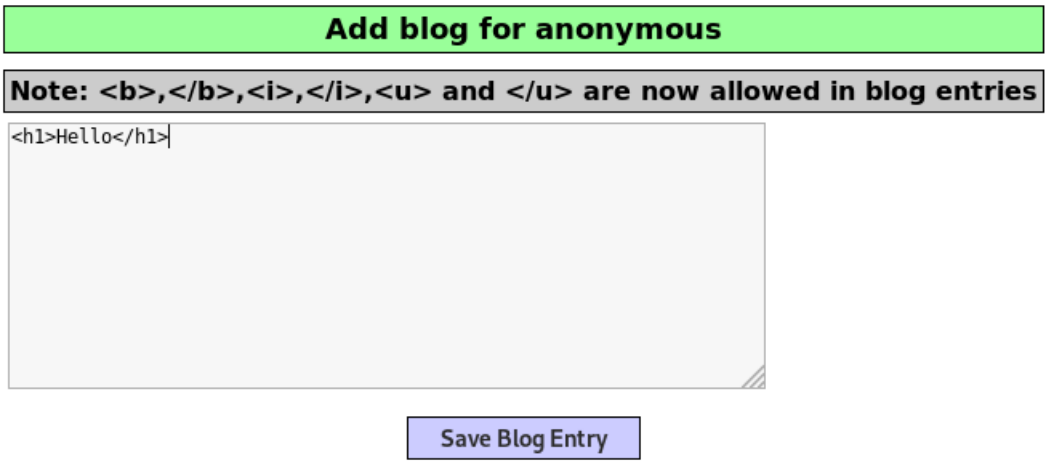


4.2.3 Cross-Sites-Scripting – Stored

In der Web-Applikation “Mutillidae” muss man folgenden Pfad folgen:



Geben wir in das Eingabefeld irgendwelchen Text ein, wird dieser im Kommentarbereich ausgegeben.
Da die Ausgabe nicht verschlüsselt wird, kann der persistentes Cross-Site-Scripting-Angriff hier angesetzt werden.
Wenn wir also statt Text Code in das Eingabefeld einfügen, wird dieser von der Web-Applikation ausgeführt.



Der eingegebene Code veränderte einen Teil der Webseite, da der Browser den Unterschied zwischen Text und Code nicht erkennen kann. Dieser Code bleibt auf der Webseite bestehen und jeder Benutzer, der diese Webseite besucht, fällt dem gespeicherten Skript zum Opfer, da dieses bei Besuch der Webseite ausgeführt wird.

5. Schlusswort/Fazit

Das Penetrationtesting stellt eine geeignete Methode dar, Schwachstellen eines Systems ausfindig zu machen und deren Schadenspotential aufzuzeigen. Beim Zusammenspiel von mehreren Methoden wird außerdem die Wirksamkeit von bereits bestehenden Sicherheitsmaßnahmen besser kontrolliert und bei Mängeln Verbesserungsvorschläge unterbreitet, als dies mit Penetrationtesting allein möglich wäre.

In Kapitel 3 konnte eine Reihe an Angriffsvektoren aufgezeigt werden, die zum Eindringen in ein System genutzt werden können. Die Effektivität diverser Angriffe reicht von einer geringen Belästigung bis hin zur kompletten Komprimittierung der Systeme eines Unternehmens. Jedoch wurde auf die Betrachtung aller existierender Angriffsvektoren verzichtet, da dies den Rahmen der Arbeit um ein Vielfaches sprengen würde, da der damit verbundene Aufwand einfach zu groß gewesen wäre.

Bei dem Penetrationstest handelt es sich jedoch nicht um die beste der Sicherheitstests. Im Kapitel 2 zeigte sich, dass es auch andere Sicherheitsprüfungen gibt, die in manchen Bereichen dem Penetrationtesting überlegen sind. Somit muss vor jeder Sicherheitsprüfung festgelegt werden, was das eigentliche Ziel dieses Tests ist, um die Verschwendung von Ressourcen zu verhindern. Nichtsdestotrotz sind die exakte Planung und der ordnungsgemäße Ablauf des Tests immer noch von größter Bedeutung.

Mit dieser Arbeit als Grundlage lägen die nächsten Schritte in der tieferen Behandlung von den Themen “Testautomatisierung” und “Penetrationtesting von SSL/TLS”.
SSL/TLS ist als Verschlüsselungs-Protokoll ein Zeichen der Privatsphäre und sicherer Kommunikation. Das Vorhandensein von Schwachstellen, die mit SSL/TLS in Verbindung stehen, können nicht nur Schaden am System, sondern auch Misstrauen bei den Benutzern des Systems schüren. Die Erstellung von Prozeduren und Checklisten sollten also in Erwägung gezogen werden.
Die Automatisierung der Testprozesse hilft Unternehmen, mit geringem Fachwissen und Zeitaufwand, Web-Applikationen und die dazugehörigen Web-Server vor Angriffen zu schützen. Die Erstellung und der Einsatz von Tools für das automatische Testen sollten zur schnelleren Erkennung von Schwachstellen verhelfen.
Zudem wird die Angriffsmethode “Social Engineering” in der Arbeit nicht behandelt und wird im Kontext “Penetrationtesting” gerne übersehen. Dies schafft eine Gefahr für Infrastruktur und Unternehmen, da der Faktor “Mensch” eine der größten Sicherheitslücken darstellt.

Anhang

Quellenverzeichnis

[Acros;1] acros.si, *Session Fixation Vulnerability in Web-based Applications*, http://www.acros.si/papers/session_fixation.pdf

[Barracuda;1] Barracuda, Barracud We Application Firewall, https://www.barracuda.com/products/webapplicationfirewall

[Cole;1] Eric Cole, *Hackers Beware*, ISBN 978-0735710092

[cybrary;1] cybrary, The Three Types of Penetration Tests, https://www.cybrary.it/study-guides/cissp/the-three-types-of-penetration-tests/

[EC-Council;1] EC-Council University, https://www.eccouncil.org/

[Eckert;1] Claudia Eckert, *IT-Sicherheit: Konzepte – Verfahren – Protokolle*, ISBN 978-3-486-77848-9

[exploit-db;1] Exploit Database, https://www.exploit-db.com/

[Hanse, Grossman;1] Robert Hansen, Jeremiah Grossmann, Clickjacking, http://sectheory.com/clickjacking.htm

[Harper, et al.;1] Harper, et al., Grey Hat Hacking: The Ethical Hacker´s Handbook, ISBN 978-1-260-10841-5

[Hope, Walther;1] Paco Hope, Ben Walther, Web Security Testing Cookbook, ISBN 978-0-596-51483-9

[Imperva;1] Imperva, *Man in the middle (MITM) attack*, https://www.imperva.com/learn/application-security/man-in-the-middle-attack-mitm/

[Kali;1] SQL-ninja package, Kali Tools,

https://tools.kali.org/vulnerability-analysis/sqlninja

[Kaspersky;1] Kaspesky, *What is a Replay Attack*, https://www.kaspersky.com/resource-center/definitions/replay-attack

[Kofler, et al.;1] Michael Kofler, et al., *Hacking & Security – Das umfassende Handbuch*, ISBN 978 -38362-5546-2

[Kurtz;1] George Kurtz, Chris Prosise, Penetration Testing: Myth vs Reality

[Kurtz;2] George Kurtz, Chris Prosise, *Penetration Testing Exposed*

[Metasploit;1] Metasploit – Penetration Testing Framework, https://www.metasploit.com/

[Merriam-Webster;1] Definition: "reconnaissance", https://www.merriam-webster.com/dictionary/reconnaissance

[Microsoft;1] Huang, et al., *Clickjacking: Attacks and Defenses*, https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/clickjacking.pdf

[Mobarhan;1] Mojitaba Ayoubi Mobarhan, Mostafa Ayoubi Morbahan, *Evaluation of Security Attacks on UMTS Authentication Mechanism*, https://www.idc-online.com/technical_references/pdfs/data_communications/EVALUATION%20OF%20SECURITY.pdf

[Moore;1] Robert Moore, *Cybercrime: Investigating High Technology Computer Crime*, ISBN 978-1437755824

[Moyer;1] Phillip R. Moyer, *What to demand from penetration testers*

[Moyer, Schultz;1] Phillip R. Moyer, E. Eugene Schultz, A systematic methodology for firewall penetration testing, https://www.sciencedirect.com/science/article/pii/S1353485800900060#!

[Mozilla;1] Mozilla Wiki, *Content Security Policy*, https://wiki.mozilla.org/index.php?title=Security/CSP/Spec&oldid=133465

[Nielson;1] Riis Nielson, Hanne Gollmann, Dieter (Eds.), *Secure IT Systems*, ISBN 978-3-642-41488-6

[Norton;1] Norton, *What is the Difference Between Black. White and Grey Hat Hacker*, https://us.norton.com/internetsecurity-emerging-threats-what-is-the-difference-between-black-white-and-grey-hat-hackers.html

[noscript;1] NoScript, Firefox-Extension, https://noscript.net

[nvd-nist.gov;1] National Institute of Standards and Technology, National Vulnerability Database, https://nvd.nist.gov/vuln

[Openwall;1] John the Ripper – Password Cracker, https://www.openwall.com/john/

[OSSTMM;1] Open Source Security Testing Methodology Manual, http://www.isecom.org/research/

[OWASP;1] OWASP Zed Attack Proxy (ZAP), https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project

[OWASP;2] OWASP Xenotix XSS Exploit Framework, https://www.owasp.org/index.php/OWASP_Xenotix_XSS_Exploit_Framework

[OWASP;3] OWASP, *Cross-Site Request Forgery*,

https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)

[OWASP;4] OWASP, *CSRF-Cheatsheet*, https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html

[OWASP;5] OWASP, *Cross-Site Scripting*, https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)

[OWASP;6] OWASP, *DOM based XSS*, https://www.owasp.org/index.php/DOM_Based_XSS

[OWASP;7] OWASP, *Periodic Table of Vulnerabilities – Null Byte Injection*, https://www.owasp.org/index.php/OWASP_Periodic_Table_of_Vulnerabilities_-_Null_Byte_Injection

[OWASP;8] OWASP, *Session fixation*, https://www.owasp.org/index.php/Session_fixation

[OWASP;9] OWASP, *Session hijacking attack*, https://www.owasp.org/index.php/Session_hijacking_attack

[OWASP;10] OWASP, *Testing for Cross site scripting*, https://www.owasp.org/index.php/Testing_for_Cross_site_scripting

[PHP;1] php.net, *What are Magic Quotes*, https://www.php.net/manual/en/security.magicquotes.what.php

[PHP;2] php.net, *mysql_escape_string*, https://www.php.net/manual/en/function.mysql-escape-string.php

[PortSwigger;1] Burp Suite – Web Application Security Testing Software, https://portswigger.net/burp

[project.webappsec;1] project.webappsec, *Cross Site Scripting*, http://projects.webappsec.org/w/page/13246920/Cross%20Site%20Scripting

[searchsecurity;1] Margaret Rouse, *White Hat,* https://searchsecurity.techtarget.com/definition/white-hat

[securepedia;1] securepedia, *mTAN*, https://www.secupedia.info/wiki/MTAN

[selfhtml;1] selfhtml-wiki, *HTML/Zeichenvorrat und HTML-eigene Zeichen* (Archiviert bei "*web-archive.org*"), https://web.archive.org/web/20150311152922/http://wiki.selfhtml.org/wiki/HTML/Zeichenvorrat_und_HTML-eigene_Zeichen#HTML-eigene_Zeichen_maskieren

[SET;1] Social-Engineer Toolkit https://www.trustedsec.com/social-engineer-toolkit-set/

[Schäfer;1] Tim Schäfer, *Hacking im Web*, ISBN 978-3645603768

[Symantec;1] Symantec, Symantec Internet Security Threat Report, http://eval.symantec.com/mktginfo/enterprise/white_papers/b-whitepaper_exec_summary_internet_security_threat_report_xiii_04-2008.en-us.pdf

[Symantec;2] Symantec, *SSH Host Key Protection*, https://www.symantec.com/connect/articles/ssh-host-key-protection

[Taylor;1] Paul A. Taylor, *Hackers: Crime in the Digital* Sublime, ISBN 978-0415180724

[thesecuritybuddy;1] thesecuritybuddy, *What is replay attack*, https://www.thesecuritybuddy.com/vulnerabilities/what-is-replay-attack/

[thesecuritybuddy;2] thesecuritybuddy, *What is CSRF or Cross Site Request Forgery Attack*, https://www.thesecuritybuddy.com/vulnerabilities/what-is-csrf-or-cross-site-request-forgery-attack/

[thesecuritybuddy;3] thesecuritybuddy; *What is Session Fixation Attack*, https://www.thesecuritybuddy.com/malware-prevention/what-is-session-fixation-attack/

[thesecuritybuddy;4] thesecuritybuddy, *What is Blind SQL Injection Attack*, https://www.thesecuritybuddy.com/vulnerabilities/what-is-blind-sql-injection-attack/

[thesecuritybuddy;5] thesecuritybuddy; *What is SQL Injetion Attack*, https://www.thesecuritybuddy.com/vulnerabilities/what-is-sql-injection-attack/3/

[Vegh;1] Sandor Vegh, *The media´s portrayal of hacking, hackers, and hacktivism before and after September 11,* https://firstmonday.org/ojs/index.php/fm/article/view/1206/1126

[w3;1] w3, *The Original HTTP as defined in 1991*, https://www.w3.org/Protocols/HTTP/AsImplemented.html

[webmaster;1] webmaster-tipps.de, PHP-Sicherheit: Local & Remote File Inclusion, https://www.webmaster-tipps.de/php-sicherheit-local-file-inclusion-und-remote-file-inclusion/

[Wikipedia;1] https://en.wikipedia.org/wiki/Security_hacker

[Wikipedia;2] https://de.wikipedia.org/wiki/Penetrationstest

[Wikipedia;3] https://de.wikipedia.org/wiki/Brutus_(Software)

[Wikipedia;4] Wikipedia, *Man in the Middle Angriff*, https://de.wikipedia.org/wiki/Man-in-the-Middle-Angriff

[Wireshark;1] https://www.wireshark.org/

[zdnet;1] zdnet, *At $30,000 for a flaw, bug bounties are big and getting bigger*, https://www.zdnet.com/article/at-30000-for-a-flaw-bug-bounties-are-big-and-getting-bigger/