INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Bachelorarbeit

Automated Success Verification of Exploits for Penetration Testing with Metasploit

Samuel Josef Sedlmeir

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Bachelorarbeit

Automated Success Verification of Exploits for Penetration Testing with Metasploit

Samuel Josef Sedlmeir

Aufgabensteller:	Prof. Dr. Helmut Reiser
Betreuer:	Tobias Appel
Abgabetermin:	07. Oktober 2019

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

München, den 03. Oktober 2019

. . . . . .
(Unterschrift des Kandidaten)

Abstract

Durch die Digitalisierung gewinnt das Management von Sicherheitsrisiken in der IT immer mehr an Bedeutung. Eine wichtige Methode zur Risikoanalyse in Unternehmen ist hierbei das *Penetration Testing*. Dabei versucht ein beauftragtes Team, Informationen über Sicherheitslücken eines Ziel-Systems zu sammeln, indem es Schwachstellen identifiziert und diese im Anschluss ausnutzt. Unterschieden wird hier zwischen manuellem und automatisiertem Penetration Testing. Ersteres liefert genauere Ergebnisse, ist aber deutlich kostenintensiver. Zur automatisierten Identifizierung von Schwachstellen werden häufig *Vulnerability Scanner* verwendet, welche als Ergebnis eine Liste potentiell vorhandener, aber gegebenenfalls nicht ausnutzbarer, Sicherheitslücken liefern.

In dieser Arbeit soll eine Lösung vorgestellt und evaluiert werden, die nicht nur die mögliche Existenz, sondern auch die Ausnutzbarkeit von Schwachstellen sowie die Effekte des zugehörigen Exploits automatisiert überprüft. Zuerst werden hierfür die Grundlagen des Penetration Testings sowie des *Metasploit Framework*s vorgestellt und verschiedene automatische Exploit-Tools und deren Eignung für den beschriebenen Einsatzzweck evaluiert. Anschließend wird prototypisch auf Basis des Metasploit Frameworks eine eigenständige Anwendung entwickelt, welche eine verlässlichere Aussage über die konkrete Gefährdung durch verschiedene Schwachstellen und den Erfolg dazu passender Exploits treffen kann. Diese Verbesserungen umfassen hauptsächlich Maßnahmen zur erfolgreicheren Automatisierung des Exploits-Vorganges sowie zur genaueren Bewertung der Ergebnisse. Nach einem Durchlauf wird ein Report generiert, der genau über die einzelnen Resultate informiert. Das entwickelte Tool wird anschließend auf verschiedene Testumgebungen angewandt und evaluiert.

Inhaltsverzeichnis

1. Einleitung

Die fortschreitende Digitalisierung nahezu sämtlicher Lebens- und Wirtschaftsbereiche führt zur Ersetzung bereits bestehender, analoger Abläufe durch digitale Lösungen und ermöglicht gleichermaßen das Erscheinen neuer digitaler Services. Damit einher geht aber auch eine stärkere Abhängigkeit von den dabei genutzten Computersystemen, ohne die der Service entweder ausfallen würde oder wieder durch langsamere oder umständlichere analoge Vorgänge ersetzt werden müsste.

Diese Abhängigkeit wurde beispielsweise im Mai 2017 deutlich, als Reisende der Deutschen Bahn vielerorts auf den Zug-Anzeigetafeln Warnmeldungen der Ransomware *WannaCry* beobachten konnten [Bö17].
Noch deutlich gravierender als dieser Fall war hingegen der Befall des Netzwerkes vieler Krankenhäuser im Vereinigten Königreich im selben Jahr. Er kostete das britische Gesundheitssystem *NHS* nicht nur über 73 Mio. GBP, sondern führte auch zur Absage von 19.000 Arztterminen [Fie18]. *WannaCry* hat sich zur weiteren Verbreitung nach erstmalig erfolgreicher Installation eine bereits länger bestehende Sicherheitslücke zunutze gemacht. Eine Ausbreitung im lokalen Netzwerk war aber nur möglich, wenn diese Lücke im SMB-Protokoll von Windows trotz verfügbarer Patches nicht geschlossen wurde [Uni18].

Die Beispiele zeigen, welche gravierenden Auswirkungen bestehende und bekannte Sicherheitslücken in Computer-Netzwerken sowohl bei manuellen als auch bei automatisierten Angriffen haben können. Zur Vermeidung derartiger Fälle mit teilweise weitreichenden Konsequenzen wird ein effektiver Schutz vor Malware und Angriffen durch Hacker besonders wichtig.

1.1. Motivation

Allgemein zählen Cyberangriffe heute zu den größten Geschäftsrisiken für Unternehmen, betreffen aber Regierungen und Individuen ebenso [Son18]. Für Unternehmen bringt ein erfolgter Angriff oft nicht nur Strafen aufgrund der Verletzung des Datenschutzes mit sich, sondern schädigt auch das öffentliche Image des Unternehmens sowie das Vertrauen der Kunden nachhaltig und kann bis zu einem temporären Erliegen des operativen Geschäftsbetriebs führen.
Mit einer zunehmend wachsenden Anzahl an Cyberangriffen wächst auch das Bedürfnis, sich gegen diese Attacken zu rüsten, indem eine solche schnell erkannt wird und durch passende Maßnahmen beispielsweise mittels eines *Intrusion Prevention System*s eine effektive Abwehr erfolgen kann. Derartige Angriffe können jedoch auch vorbeugend unterbunden oder zumindest erschwert werden, indem möglicherweise kritische Sicherheitslücken zeitnah erkannt und geschlossen werden.

Dabei ist eine möglichst regelmäßige Durchführung von *Penetration Testing* ein wirkungsvolles Mittel, um diese Sicherheitslücken möglichst zeitnah und vollständig zu identifizieren und dadurch passende Patches suchen und einspielen oder Updates durchführen zu können. Bis heute benötigt Penetration Testing jedoch nicht nur breites Fachwissen und Erfahrung,

sondern auch viel Manpower, obwohl dieser Prozess bereits durch zahlreiche Anwendungen unterstützt und vereinfacht wird. Dazu zählen unter anderem sogenannte *Vulnerability Scanner* wie beispielsweise *Nessus*, *OpenVAS* oder *Nexpose*, welche eine hilfreiche Unterstützung darstellen, indem sie die Suche nach möglicherweise vorhandenen Schwachstellen auf einem Ziel-Computer oder ganzen Netzwerkbereichen weitgehend automatisieren.
Eine von diesen Scannern gefundene Schwachstelle ist allerdings nicht notwendigerweise ausnutzbar, da der Erfolg von Exploits oft von Bedingungen abhängt, die von einem solchen Scanner nicht überprüft werden können. Viele Exploits basieren auf zur Laufzeit unterschiedlichen Bedingungen oder bestimmten, anfälligen oder fehlerhaften Konfigurationen. Diese Bedingungen können aber wiederum von den Scannern oft nicht vollständig überprüft werden, was zu einer gewissen Anzahl an „false positive“-Fehlern führt.

An dieser Stelle soll die hier entwickelte Lösung ansetzen und den Erfolg potentieller Exploit-Kandidaten so weit möglich durch Ausführung verifizieren. Dadurch ist zeitlich effizienteres Testen möglich, da sich der Penetrationstester mit weniger vermeintlichen Schwachstellen befassen muss und so sein Augenmerk auf die dringlicher zu behebenden Probleme richten kann. Außerdem wird der Prozess des Penetration Testings dadurch effizienter, was die Verbreitung und Anwendung auf mehr unterschiedliche Umgebungen bei gleichbleibenden Kosten und Qualität erhöhen kann.

1.2. Ziel der Arbeit

In dieser Arbeit soll eine Anwendung entwickelt werden, die in der Lage ist, die Ausgabe eines Vulnerability Scanners in einem festgelegten Format erst einzulesen, dann automatisiert zu den jeweiligen Reporteinträgen passende Exploits mit Hilfe einer öffentlich verfügbaren Exploit-Datenbank zu finden und diese im nächsten Schritt, nach passender Konfiguration, wiederholt auf den Zielrechner anzuwenden. Nach einer abgeschlossenen Exploit-Ausführung soll möglichst präzise dessen Erfolg und Auswirkungen mittels verschiedener Methoden analysiert und bewertet werden.
Am Ende soll automatisch ein Report generiert werden, welcher eine Übersicht über alle ausgeführten Exploits, inklusive der genauen Ausführungsdetails, und eine Analyse des jeweiligen Erfolgs liefert. Für die erfolgreichen Exploits sollen zudem genauere Infos gesammelt werden, dazu zählen z.B. die erlangten Nutzerrechte auf dem Target oder auch die Anzahl an benötigten Versuchen bis zur erfolgreichen Ausführung.

1.3. Vorgehen

Begonnen wird in Kapitel 3 damit, den genauen Use Case des zu entwickelnden Tools abzustecken und dabei die konkreten Anforderungen an eine für diesen Zweck geeignete Anwendung zu definieren.
Nach der Definition und der manuellen Analyse einer adäquaten Testumgebung, welche aus mehreren virtuellen Maschinen besteht und zur Evaluation herangezogen wird, werden zwei bestehende Anwendungen, Hail Mary (als Teil von Armitage) und AutoSploit, einer näheren Betrachtung hinsichtlich der Eignung für diese Arbeit unterzogen. Für die hierbei entdeckten Probleme werden verschiedene Lösungsmöglichkeiten skizziert.
In Kapitel 5 soll anschließend eine für den Einsatzzweck besser geeignete Anwendung entwickelt werden. Hierzu müssen zuerst verschiedene, mögliche Entwicklungsumgebungen hin-

sichtlich ihrer Eignung evaluiert werden. Auf Basis der gewählten Umgebung kann im nächsten Schritt das Anwendungsdesign festgelegt werden. Dies geschieht sowohl auf interner Ebene, als auch durch Definition der Schnittstellen. Hierzu zählen unter anderem die Anwendungsparameter, sowie das Input-Format (für einen Schwachstellenreport und die Exploit-Klassifikation) und das Output-Format für den Anwendungseigenen Report. Nach der Festlegung des Designs werden die bereits definierten Anforderungen in der Implementierung einer Anwendung auf Basis des Metasploit Frameworks umgesetzt.
Ist die Implementierung abgeschlossen, gilt es, die Anwendung in der zuvor definierten Testumgebung und im Vergleich zu den anderen Anwendungen zu evaluieren und auch die mögliche Erweiterbarkeit und entstandenen Einschränkungen zu betrachten.
Die Arbeit endet schließlich mit einer Übersicht über die erreichten Ziele, einer Auflistung potenzieller Erweiterungsmöglichkeiten und dem Fazit.

2. Grundlagen des Penetration Testings

Mit dem steigenden Bedarf an Sicherheit im IT-Umfeld, strengeren gesetzlichen Regelungen und ständig zunehmenden Bedrohungen wächst auch die Nachfrage nach professionellem Risikomanagement in der Informationstechnologie in Unternehmen.
Das Ziel des Risikomanagements ist es, die Nicht-Gefährdung der drei Schutzziele der Informationssicherheit zu gewährleisten: der Vertraulichkeit, Integrität und der Verfügbarkeit. Unter der Vertraulichkeit versteht man den „Schutz vor unbefugter Preisgabe von Informationen“, während für die Integrität von Daten die Unversehrtheit von Daten und Systemen sichergestellt sein muss [BSI13]. Ist das betrachtete System für autorisierte Subjekte jederzeit zugänglich, so ist auch das Ziel der Verfügbarkeit gewährleistet.
Als erster Schritt im Risikomanagementprozess gilt nach dem für das Risikomanagement der Informationssicherheit maßgeblichen Standard ISO/IEC 27005 die Festlegung des Kontextes. Dabei werden unter anderem der Anwendungsbereich und die zugehörigen Rollen sowie Verantwortlichkeiten im Prozess festgelegt. Bei der *Risikoidentifikation* wird versucht, mögliche Probleme organisatorischer oder technischer Art ausfindig zu machen. Dabei werden nicht nur Schwachstellen und Bedrohungen identifiziert, sondern auch bereits umgesetzte Maßnahmen und mögliche Schadensauswirkungen. Im Schritt der *Risikoanalyse* werden die potentiellen Konsequenzen und Eintrittswahrscheinlichkeiten abgeschätzt, welche direkt in die *Risikobewertung und Risikopriorisierung* mit einfließen. Anhand dieser Bewertungen können die untersuchten Risiken anschließend angemessen durch verschiedene Methoden behandelt werden. Bleibende Restrisiken können entweder akzeptiert werden oder den Prozess wieder von Anfang an durchlaufen. Der ganze dargestellte Prozess wird umrahmt von einer Risikokommunikation einerseits und einer stetigen Risikoüberwachung und -überprüfung andererseits [Kli15].
In der Praxis können technische Risiken in Unternehmen vielfältig geartet sein und dadurch auf unterschiedlichste Arten und Weisen *analysiert* und *behandelt* werden. Penetration Testing stellt hier eine von zahlreichen Möglichkeiten dar, Risiken schon vor dem Ernstfall, also *präventiv*, zu identifizieren und diese durch einen angemessenen Reaktionsprozess zu behandeln.

Allgemein wird bei Penetration Testing ein, oft externes, Team damit beauftragt, innerhalb eines klar definierten Rahmens einen gezielten Angriff auf das zu testende System zu simulieren, indem es Sicherheitslücken findet und ausnutzt. Damit handelt es sich um eine „Wirksamkeitsprüfung vorhandener Sicherheitsmaßnahmen“, die sich meist verschiedensten Techniken eines Hackers bedient [BSI13]. Die Beauftragung eines externen Teams bringt in der Regel den Vorteil der Unbefangenheit mit sich, was beispielsweise das Finden neuer Angriffsvektoren deutlich erleichtert. Außerdem ist so eine realitätsnähere Simulation eines Hacker-Angriffes möglich.

Im Rahmen des Risikomanagements in Unternehmen kommen oft auch andere Methodiken zum Einsatz, welche sich jedoch deutlich von Penetrationstests abgrenzen lassen.
Während bei *Informationssicherheitsrevisionen* in erster Linie die Einhaltung bestehender Sicherheitsmaßnahmen überprüft und verbessert wird, so versucht ein Penetration Tester die

Umgehung ebendieser, um so mindestens mittelfristig diese Maßnahmen wiederum verbessern zu können.
Penetrationstests lassen sich häufig auch in der Softwareentwicklung finden, wo sie – ebenso wie *Code Reviews* – eingesetzt werden können, um Sicherheitslücken für die entwickelte Anwendung schon vor dem Release zu finden. Ein Code Review einer Anwendung umfasst eine genaue Analyse des Quellcodes auf mögliche Schwachpunkte und Fehler und ist damit Teil der Qualitätssicherung. Ein Penetration Test dieser Anwendung versucht hingegen, beispielsweise durch unerwartete Eingaben ein ungewolltes und ausnutzbares Verhalten der Anwendung herbeizuführen und sich zu Nutze zu machen. Ein Penetration Test einer Anwendung ist somit auch ohne vorliegenden Sourcecode zur Laufzeit möglich, während Code Reviews statisch noch vor dem Compilen durchgeführt werden.
Neben den genannten Methodiken zur Erkennung und zum Umgang mit Schwachstellen wird in der Praxis auch auf weitere Vorgehensweisen, wie beispielsweise *Vulnerability Assessments*, zurückgegriffen.

2.1. Klassifikation

Allgemein findet man Penetration Testing hauptsächlich als Dienstleistung im Umfeld von Unternehmen. Dabei lassen sich die durchgeführten Tests nach verschiedenen Kriterien klassifizieren.

2.1.1. Automatisierungsgrad

Penetrationstests können in *manuelle* und *automatische* Tests unterschieden werden. Beide Arten von Tests haben Vorteile und sind in der Praxis oft in unterschiedlich stark vermischter Form anzutreffen.
Bei manuellen Tests werden grundsätzlich sämtliche Aktionen von einem erfahrenen Tester geplant und manuell ausgeführt, während der Penetrationstester bei automatischen Tests auf eine Vielzahl von Anwendungen zurückgreifen kann, welche im Besonderen redundante Aufgaben übernehmen können. Grundsätzlich sind manuelle Penetrationstests deutlich teurer und aufwändiger als automatische, allerdings liefern sie oft auch deutlich genauere und wertvollere Informationen, da auch auf Aktionen zurückgegriffen werden kann, die nicht von passenden Anwendungen angeboten werden.
Eine genauere Betrachtung der Anwendbarkeit automatischer Penetrationstests kann in Abschnitt 3.3 gefunden werden.

2.1.2. Informationsgrundlage

Eine weitere Möglichkeit der Klassifikation ist nach der vorhanden Informationsgrundlage, hauptsächlich in *white box* und *black box* Testing.
Bei *black box* Testing wird versucht, das Vorgehen eines *externen* Intruders anzuwenden und nur mit öffentlich verfügbaren und daher unvollständigen Informationen zu operieren.
Hingegen sind bei *white box* Tests alle benötigten, weiterführenden internen Informationen, wie zum Beispiel Netzwerk-Strukturen oder Quellcode, und/oder bereits Zugänge oder Berechtigungen verfügbar, die dem Angreifer ein tieferes Verständnis des zu testenden Systems ermöglichen.

Wenn nur ein bestimmter Teil der benötigten Informationen vom Auftraggeber zur Verfügung gestellt wird, spricht man hingegen von *gray box* Testing [BYCJ11].

2.1.3. Typ des Systems under Test

Penetrationstests können auch nach dem Typen des zu testenden Systems kategorisiert werden. Ein Test von selbst entwickelter Software wie beispielsweise einer neuen WebApp erfordert unter anderem andere Ziele, Werkzeuge und anderes Hintergrundwissen als ein Test der im Unternehmen eingesetzten Kombination und Konfiguration von Netzwerk-Komponenten. So lassen sich drei verschiedene Kategorien anhand des jeweils verfügbaren *Scopes* unterscheiden: Tests eines Netzwerks, einer Anwendung oder Social Engineering Tests [BYCJ11].
Bei Netzwerk-Tests wird vorrangig versucht, Schwachstellen in der vorhandenen Netzwerk-Infrastruktur und deren Konfiguration auszunutzen, um Zugang zum System under Test zu erhalten.
Anwendungstests sind hingegen darauf fokussiert, nur die Sicherheit der getesteten Anwendung in der vorliegenden Umgebung zu überprüfen. Wird hierbei beispielsweise ein Angriffsvektor gefunden, der die getestete Anwendung nicht betrifft, so ist dieser in diesem Kontext nicht relevant.
Das Ziel von Social Engineering Penetration Tests ist es, anstatt technischer Schwachstellen menschliche Schwächen, zum Beispiel Hilfsbereitschaft oder Angst vor Autoritäten, auszunutzen, um Zugriff auf eigentlich geschützte Daten und Systeme zu erhalten. Dadurch sind derartige Angriffe und Tests immer auf die Mitarbeit eines Menschen innerhalb der getesteten Organisation angewiesen und nicht durch rein technische Maßnahmen zu verhindern. Social Engineering Penetration Tests sind aber – neben anderen – eine wirksame Maßnahme, um das notwendige Bewusstsein für derartige Angriffe zu schaffen und weiter zu sensibilisieren.

2.1.4. Weitere

Aufgrund der weiten Bandbreite an Tests, die im Rahmen von Penetration Testing durchgeführt werden können, bieten sich einige weitere Klassifizierungsmerkmale an [BSI03].
So kann neben den bereits vorgestellten Attributen auch nach der *Aggressivität* des Tests differiert werden: Diese können rein passiv, beispielsweise durch reines Scannen, oder auch aggressiv, wie es beispielsweise bei einem reellen Angriff auch der Fall wäre, durchgeführt werden.
Auch anhand der *Art des Vorgehens* können verschiedene Testtypen inferiert werden: Wird ein Test vor und während der Ausführung völlig offen und transparent kommuniziert, so spricht man von einem offensichtlichen Test. Zur möglichst realitätsnahen Simulation eines echten Angriffes empfiehlt sich hier jedoch ein verdecktes Vorgehen.
Eine klare Definition des Umfangs eines Penetrationstests, unter anderem nach den hier genannten Klassifikationskriterien, ist aus Sicht der Penetrationstester besonders wichtig, um sich auf rechtlich sicherem Grund zu bewegen.

2.2. Rechtlicher Rahmen

Für Penetrationstester ist es von besonderer Relevanz, den genauen rechtlichen Rahmen ihrer Arbeit und des jeweiligen Auftrages zu kennen, da Aktionen außerhalb dessen einen Gesetzesverstoß bedeuten und somit rechtliche Konsequenzen haben können. Besonders wichtig „ist

daher die exakte Festlegung des beauftragten Handlungsrahmens zwischen dem Auftraggeber und dem Auftragnehmer“ [BSI03]. Dieser sichert den Penetrationstester vor juristischen Konsequenzen im Rahmen verschiedener Gesetze ab.

Im Folgenden sollen ein knapper Überblick über die in diesem Kontext wichtigsten gesetzlichen Regelungen gegeben werden.
Zu nennen sind hier insbesondere §202 und §303 StGB. §202a und §202b stellen das Abfangen von Daten unter Überwindung der Zugangssicherung sowie das Abfangen von Daten aus einer nichtöffentlichen Übermittlung unter Strafe. Nach §202c ist hierbei auch bereits jeder strafbar, der sich oder anderen Personen Computerprogramme zur Begehung einer solchen Straftat verschafft. Schwierig ist hierbei vor allem die Entscheidung, ob davon betroffene Programme zum Zwecke der Begehung einer Straftat beschafft werden, da auch legitime Programme aus dem Arsenal eines Penetrationstesters missbräuchlich zur Begehung einer Straftat verwendet werden können.
Die Paragraphen 303a und 303b StGB legen fest, dass auch die rechtswidrige Datenveränderung sowie die Computersabotage (erhebliche Störung einer Datenverarbeitung) sowie die Vorbereitung der beiden Straftaten strafbar ist.
Andere und weiterführende Regelungen finden sich beispielsweise auch im Bundesdatenschutzgesetz (BDSG), dem Telekommunikationsgesetz (TKG) und dem Zugangskontrolldiensteschutzgesetz (ZKDSG) sowie in Gesetzen anderer Verwaltungsebenen, beispielsweise der EU-Datenschutzgrundverordnung (EU-DSGVO) und dem Bayerischen Datenschutzgesetz (BayDSG).
In der EU-DSGVO findet sich beispielsweise in Artikel 32 für Datenverarbeiter die Pflicht, „ein Verfahren zur regelmäßigen Überprüfung, Bewertung und Evaluierung der Wirksamkeit der technischen und organisatorischen Maßnahmen zur Gewährleistung der Sicherheit der Verarbeitung“ [Eur18] durchzuführen und dies nachzuweisen. Penetration Testing ist in diesem Zuge eine adäquate Wahl sowohl zur Überprüfung der Maßnahmen als auch zur Erbringung des Nachweises.
Allgemein führt die derzeitige Gesetzgebung also nicht nur zu Unsicherheit unter ethischen Hackern bezüglich dem Besitz und der Benutzung dafür geschaffener Anwendungen, sondern begünstigt durch strengere Datenschutzauflagen mittlerweile auch einen verbreiteteren Einsatz von Penetrationstests in Unternehmen.

2.3. Ablauf

Ein typischer Penetration Test gliedert sich allgemein in fünf Phasen [BSI03]. In allen diesen Phasen ist eine kontinuierliche, umfassende und nachvollziehbare Dokumentation von größter Bedeutung für den Auftraggeber, um die gewonnenen Erkenntnisse später auch in der Praxis effizient nutzen zu können.

Vorbereitung

In der Regel wird ein Penetrationstest mit der *Vorbereitung* begonnen, während welcher in Zusammenarbeit mit dem Auftraggeber die zu erreichenden Ziele des Tests definiert werden. Auch festgelegt wird der Prüfumfang, welcher sich konkret aus der Prüftiefe, dem Prüfort, dem Prüfzeitraum und den Prüfbedingungen zusammensetzt [BSI16].
Allgemein müssen die schriftlich dokumentierten Bedingungen, wie die Ziele des Tests erreicht werden dürfen, zwingend eingehalten werden, um beispielsweise nicht den operativen

Betrieb der kritischen firmeneigenen IT zu gefährden und nicht Gefahr zu laufen, strafbar zu handeln, wie in Abschnitt 2.2 dargestellt. Für den Fall eines unbeabsichtigten, negativen Effekts eines Tests in einer für den operativen Betrieb relevanten Umgebung müssen außerdem Maßnahmen vereinbart werden, die in einem solchen Notfall strikt zu befolgen sind.

Informationsgewinnung und -analyse

Während der Phase der *Informationsgewinnung* und *-analyse* werden möglichst viele und präzise Daten über das Ziel-System gesammelt, die je nach Art des Tests aus unterschiedlichen Quellen stammen können. Am Ende dieses Prozesses sollten ausreichende Informationen über das System beziehungsweise Netzwerk, vorhandene Dienste sowie Schwachstellen verfügbar sein und damit bereits mögliche Ansatzpunkte für einen ersten Angriff identifizierbar sein.

Bewertung der Informationen und Risikoanalyse

Nach dem Sammeln der Informationen müssen diese *bewertet* und eine *Risikoanalyse* durchgeführt werden. Dafür werden mehrere Faktoren zu Rate gezogen, wie beispielsweise die potentiellen negativen Konsequenzen für das System under Test, die vorher definierten Restriktionen des Auftraggebers sowie der geschätzte Aufwand sowohl für Planung als auch für die Durchführung.
Die für den nächsten Schritt verfügbaren „primären“ Angriffsziele werden mithilfe dieser Kriterien ausgewählt und priorisiert. Auch wenn diese weiteren Einschränkungen zwar letztlich ein effizienteres und präziseres Vorgehen ermöglichen, müssen sie nachvollziehbar dokumentiert sein, da sie den Scope des Tests verändern und gewonnene Erkenntnisse damit gegebenenfalls weniger allgemeingültig und anders zu interpretieren sind.

Aktives Testen

Diese so gewonnenen und bewerteten Informationen über vorhandene Schwachstellen werden im dritten Schritt dazu genutzt, Zugang zum Ziel-System zu erhalten und dadurch weitere nützliche Informationen für zukünftige Angriffsarten und -wege zu sammeln. Dieser Schritt muss sorgfältig geplant und unter Beachtung der Bedingungen des Auftraggebers durchgeführt werden, da die in diesem Schritt übliche Anwendung von Exploits oft Risiken bezüglich der Systemverfügbarkeit mit sich bringt.
Von einem reellen Hackerangriff unterscheidet diese Phase, dass selbst nach dem Erreichen des Ziels der Angriff nicht beendet wird, sondern versucht wird, weitere Schwachstellen ausfindig zu machen und zu verifizieren.

Abschlussanalyse

Im Anschluss an den Test werden in der *Abschlussanalyse* alle während des Tests gesammelten Informationen und Erkenntnisse analysiert und aufbereitet. Daraus wird ein Report generiert, der wichtige Informationen unter anderem zu den gefundenen Sicherheitslücken und deren Ausnutzung enthält. Ebenfalls einbezogen werden sollten Bewertungen der entdeckten Risiken sowie vorgeschlagene Maßnahmen zu deren Behandlung.
Den in diesem Schritt erstellten Report erhält der Auftraggeber des Tests, um auf dessen Basis die Sicherheit des getesteten Systems verbessern zu können.

2.4. Werkzeuge

Im Rahmen eines Penetrationstests kommen oft zahlreiche, verschiedene Tools zum Einsatz, die sich nach dem Schritt, in dem sie üblicherweise eingesetzt werden, kategorisieren lassen. In diesem Abschnitt werden hauptsächlich Anwendungen aufgezeigt, die für die weitere Arbeit hilfreich sind. Von einer Aufzählung aller verbreiteten Tools wird hier allerdings, aufgrund des Umfanges und einer fehlenden Relevanz für diese Arbeit, abgesehen.

Informationsgewinnung

Beliebte Tools im Rahmen der *Informationsgewinnung* sind unter anderem Port- und Vulnerability Scanner.
Portscanner, wie beispielsweise *nmap*, werden benutzt, um die geöffneten Ports eines oder mehrerer Ziel-Systeme mittels TCP oder UDP zu ermitteln. Nmap bietet darüber hinaus weitere Funktionen wie eine *Service Detection* und *Operating System Detection*. Diese stellen dem Tester automatisiert wertvolle Informationen, die über die Nennung der Portnummer herausgehen, zur auf dem Ziel-System vorhandenen Software bereit. Bei der System Detection kann beispielsweise anhand eines Fingerprint-Vergleichs in vielen Fällen das auf dem Ziel-System vorhandene Betriebssystem richtig ermittelt werden. Wird die Service Detection aktiviert, kann nmap den Namen sowie oft die ungefähre Versionsnummer des auf dem jeweiligen Port laufenden Dienstes beispielsweise mithilfe von Banner Grabbing ermitteln. Dabei wird die Willkommensnachricht des jeweiligen Dienstes mit einer Liste von gespeicherten „Fingerabdrücken“verglichen und dabei die gewünschten Informationen erhalten. Mit nmap lassen sich jedoch nicht nur einzelne Zielsystem auf offene Ports testen, sondern im Rahmen eines Netzwerktests auch ganze Netzwerkbereiche auf erreichbare Hosts überprüfen. Nmap wird standardmäßig direkt über eine Kommandozeile bedient, allerdings stehen auch graphische Oberflächen, wie beispielsweise *Zenmap* hierfür zur Verfügung. Weitere Beispiele für gängige Portscanner sind *Unicornscan*, *AngryIP* oder auch die bekannte Linux-Anwendung *netcat*. *ZMap* ist ein geläufiger Netzwerkscanner, der sich besonders für die schnelle Host Discovery in einem großen Adressbereich eignet: Der gesamte öffentliche IPv4-Adressraum kann hiermit (bei gegebener Gigabit-Netzwerkverbindung) in unter 45 Minuten gescannt werden [DWH13].
Vulnerability Scanner werden hingegen benutzt, um auf dem Ziel-System nach (den im Scanner vorhandenen) bekannten Schwachstellen zu suchen. Beispiele hierfür sind *OpenVAS*, *Nexpose* und *Nessus* oder *OWASP ZAP (Zed Attack Proxy)* für WebApps. Schwachstellenscanner bieten damit eine über Portscanner hinausgehende Funktionalität, stellen jedoch ebenso ein passives Tool dar, da die gefundenen Schwachstellen in der Regel nicht aktiv überprüft werden. Von Schwachstellenscannern erstellte Reports enthalten dadurch häufig auch False Positives, deren Überprüfung dem Tester obliegt.
Zur Informationsgewinnung lassen sich auch eine Vielzahl anderer Tools nutzen, die teilweise auch sehr spezifische Anwendungsfälle abdecken.

Aktives Testen

Sind nun mithilfe der bereits vorgestellten Werkzeuge erste Ansatzpunkte für einen *aktiven Test* vorhanden, können erneut verschiedene Tools zum Einsatz kommen. Sehr beliebt in diesem Schritt ist das *Metasploit Framework*, das in Abschnitt 2.6 genauer vorgestellt wird

und dem Penetrationstester durch eine umfangreiche Modulsammlung und vielfältige Integrationsmöglichkeiten viel Arbeit abnimmt.
In dieser Phase ist aber oft auch ein manuelles Vorgehen nötig, um auch besonders spezifische Angriffsszenarien ausreichend abzudecken. In diesem Fall kann beispielsweise die *Exploit-DB*[1], eine Online-Datenbank mit einer Vielzahl von öffentlichen Exploits, hilfreich sein.
Eine Fülle weiterer Anwendungen unterstützt den Penetrationstester auch in diesem Schritt, in welchem professionelle Tester aber auch selbst entwickelte Programme einsetzen.

Abschlussanalyse

Auch für die *Abschlussanalyse* steht eine ganze Reihe hilfreicher Tools für den Tester zur Verfügung.
Für eine hohe Nachvollziehbarkeit des Vorgehens des Testers ist nicht nur ein informativer Abschlussbericht, sondern auch eine umfassende *kontinuierliche* Dokumentation der Aktivitäten und Erkenntnisse von Vorteil. Hierbei können die in den vorigen Schritten verwendeten Tools eine große Hilfestellung sein, indem sie, wie beispielsweise OpenVAS, Nessus und Metasploit Pro, eine automatische Reportgenerierung unterstützen. Auch andere Dokumentationssysteme, zum Beispiel *Dradis*, können in diesem Zug einem Penetrationstester helfen, einen Report in festgelegten Format zu produzieren sowie dabei redundante manuelle Aufgaben zu reduzieren.

Allgemein

Viele der genannten Anwendungen sind bereits in *Kali Linux* (dem Nachfolger von *Backtrack Linux*) enthalten, einer besonders für Ethical Hackers und Penetration Tester vorbereiteten und auf Debian basierenden Linux-Distribution. Kali umfasst bereits standardmäßig über 60 Anwendungen zur Informationsgewinnung, sowie jeweils über 20 Exploitation Tools und Anwendungen zur Schwachstellenanalyse.[2]

2.5. Schwachstellen

Sämtlichen Penetrationstests gemeinsam ist die Ausnutzung von Schwachstellen – hierbei sind in der Praxis oft besonders aktuelle Schwachstellen relevant, da bei diesen die Wahrscheinlichkeit höher ist, dass sie noch nicht geschlossen wurden. Allgemein definiert sich eine *Schwachstelle* nach ISO 27005 als „eine Schwachstelle bezüglich eines Werts oder mehrerer Werte, die durch eine oder mehrere Bedrohungen ausgenutzt werden kann“ [Kli15].
Schwachstellen können hierbei unterschiedlich geartet sein und klassifiziert werden. Der Fokus der Arbeit soll im Folgenden in Abgrenzung zu Hardware-, Netzwerk- oder organisatorischen Schwachstellen besonders auf Software-Schwachstellen liegen.
Zur Vereinfachung der Handhabung von Schwachstellen sollen im Folgenden zwei relevante und weit verbreitete Standards vorgestellt werden.

[1] `https://www.exploit-db.com/`
[2] `https://www.kali.org/tools-listing`, Stand: September 2019

2.5.1. Common Vulnerability Scoring System (CVSS)

Eine Einschätzung der von einer Schwachstelle ausgehenden Bedrohung kann beispielsweise mithilfe des *Common Vulnerability Scoring System*s (CVSS) vorgenommen werden. Anhand verschiedener Input-Parameter wird dabei ein *Score* zwischen 0 (keine Gefährdung) und 10 (kritisch) errechnet. Die genaue Bedeutung der konkreten Score-Werte kann Tabelle 2.1 entnommen werden.

CVSS-Score	Schweregrad
0.0	keiner
0.1 - 3.9	niedrig
4.0 - 6.9	mittel
7.0 - 8.9	hoch
9.0 - 10.0	kritisch

Abbildung 2.1.: Bewertungsskala für den Schweregrad von CVSS-Scores[3]

Der CVSS-Score setzt sich in der Version 3 allgemein aus drei Bestandteilen zusammen: den *Base Metrics*, den *Temporal Metrics* und den *Environmental Metrics*. Die Base Metrics werden nur durch grundlegende, rein schwachstellenspezifische Parameter beeinflusst, welche sich auch nicht im Laufe der Zeit oder in verschiedenen Umgebungen verändern. Die von der Zeit abhängigen Komponenten des Schweregrads einer Schwachstelle werden mit den Temporal Metrics eingeschätzt, um so beispielsweise einen höheren Score für brandaktuelle Lücken, für die noch kein Patch verfügbar ist, zu ermöglichen. Die Environmental Metrics hingegen spiegeln den Einfluss der lokalen Umgebung auf die Schwere der Sicherheitslücke durch verschiedene Parameter bezüglich dem lokalen Einsatz der verwundbaren Software wider und sind damit nur für eine konkrete Umgebung gültig.
Neben dem Score enthält der Output einer CVSS-Berechnung in der Regel außerdem einen Vektor-String, der die für die Berechnung verwendeten Werte aller Variablen in einer kompakteren Form enthält, und damit einen einfacheren Export einer detaillierten Beurteilung der Schwachstelle ermöglicht.
Zur allgemeinen Einschätzung von Sicherheitslücken ist besonders der Base Score relevant, da dieser eine Einschätzung der Schwachstelle völlig unabhängig von der zeitlichen und lokalen Situation ermöglicht und so eine allgemeingültige Aussage über eine Schwachstelle treffen kann. Ist im Folgenden vom CVSS-Score einer Schwachstelle die Rede, so ist hiermit, sofern nicht anders angegeben, der CVSS Base Score gemeint.

2.5.2. Common Vulnerabilities and Exposures (CVE)

Common Vulnerabilities and Exposures (CVE) ist eine umfassende, 1999 begonnene Auflistung öffentlich bekannter Schwachstellen. Jeder hierbei erfassten Schwachstelle wird eine einzigartige ID zugewiesen, die sogenannte CVE-Nummer. Diese hat in der Regel das Format CVE-YYYY-NNNNNNN, wobei *YYYY* für das Jahr steht, in dem die CVE-Nummer

[3] `https://www.first.org/cvss/specification-document#5-Qualitative-Severity-Rating-Scale`
[4] `https://www.cvedetails.com/top-50-products.php`, Stand: 22.04.2019
[5] `https://www.cvedetails.com/browse-by-date.php`, Stand: 24.04.2019
[6] `https://www.cvedetails.com/vulnerabilities-by-types.php`, Stand: 24.04.2019

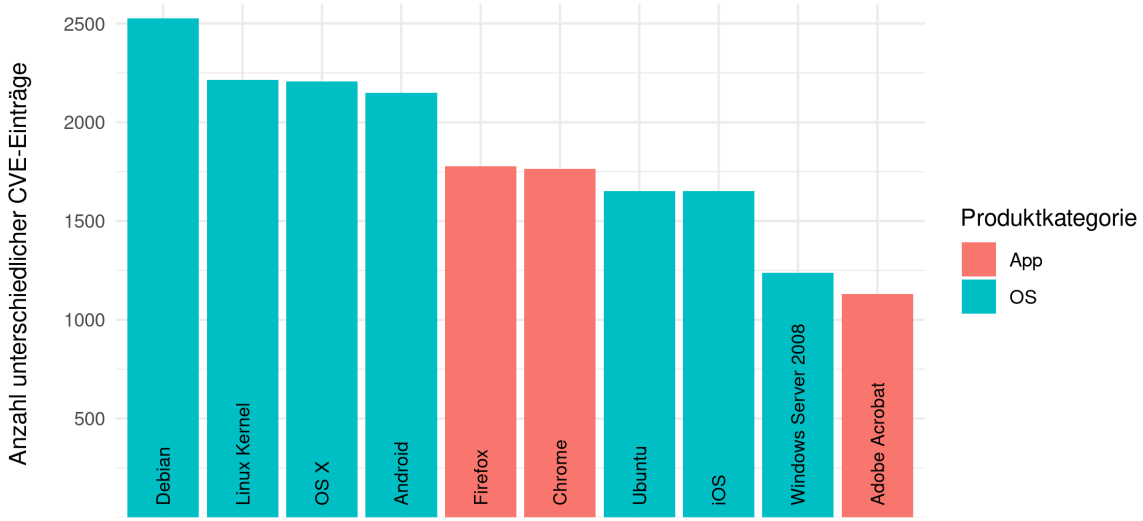

Abbildung 2.2.: Übersicht der zehn Produkte mit den meisten CVE-Einträgen seit Beginn der Aufzeichnung, kategorisiert nach Produktklasse.[4]

zugewiesen wurde, und *NNNNNNN* nur aus beliebigen vier bis sieben Ziffern zur eindeutigen Identifikation besteht. Jeder CVE-Eintrag umfasst neben der ID auch eine standardisierte, kurze Beschreibung der Schwachstelle sowie relevante Referenzen.
Um welche Größenordnungen an CVE-Einträgen es sich bei sehr verbreiteten Software-Produkten handelt, soll die Abbildung 2.2 verdeutlichen. Auf die vier häufigsten Produkte in der Kategorie *Betriebssysteme* entfallen jeweils über 2.000 eingetragene Schwachstellen. Anwendungssoftware, die für diese Systeme entwickelt wurde, ist hierin nicht eingeschlossen. Teilweise deutlich über 1.000 Einträge entfallen jeweils auch auf die fünf häufigsten Produkte im Bereich *Anwendungen*. Dass Windows sich hier nicht an erster Stelle befindet, liegt unter anderem daran, dass hierbei (im Gegensatz zu anderen Betriebssystemen) nach den einzelnen Betriebssystem-Versionen unterschieden wird. Außerdem sind bei beispielsweise Debian auch Schwachstellen eingeschlossen sind, die sich in Software befinden, die mit dem Betriebssystem ausgeliefert wird, aber möglicherweise von anderen, unabhängigen Entwicklern stammt.
In 2017 und 2018 wurden, wie in Abbildung 2.3 dargestellt, jeweils zwischen 14.000 und 17.000 neue Schwachstellen der Liste hinzugefügt. Nicht nur der totale jährliche Zuwachs hat damit einen Höchststand erreicht, sondern auch der Zuwachs an besonders kritischen Sicherheitslücken (CVSS-Score größer als 9,0)[7]. Das steigende Wachstum dieser Liste sowie der stetige Anstieg der Anzahl kritischer Schwachstellen machen Penetration Testing zu einem immer wichtigeren Mittel, um die Nicht-Gefährdung der Sicherheitsziele der Informationssicherheit zu gewährleisten.
Dank der durch CVE gegebenen, praktikablen und universellen Identifikationsmöglichkeit von Schwachstellen sowie deren weiten Verbreitung finden sich CVE-Nummern häufig im Output von Schwachstellenscannern und können wiederum zur Suche nach passenden Exploits in verschiedenen Datenbanken für Exploits benutzt werden. Anhand des CVE-Eintra-

[7] `https://www.cvedetails.com/cvss-score-charts.php?fromform=1&startdate=1999-01-01&enddate=2019-09-28&groupbyyear=1`

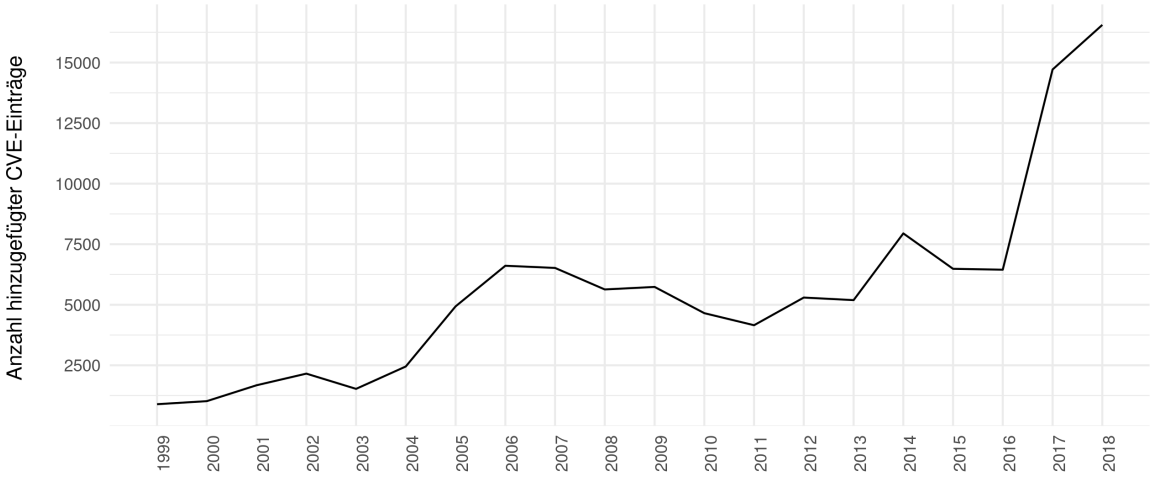

Abbildung 2.3.: Anzahl der neu hinzugekommenen CVE-Einträge pro Jahr seit Beginn der Nummernvergabe 1999.[5]

ges kann auch bereits ermittelt werden, um welchen Schwachstellentypen es sich handelt. Unter allen CVE-Einträgen sind besonders vier Schwachstellentypen mit einem Anteil von über zehn Prozent vorhanden: Code Execution, Denial of Service, Overflow und XSS (Cross Site Scripting) Vulnerabilities (siehe Abbildung 2.4). Besonders die ersten drei Typen sind hierbei für diese Arbeit besonders interessant, da sie sich oft auch remote mit Exploit-Modulen aus dem Metasploit-Framework ausnutzen lassen.
Zur Identifikation von Schwachstellen werden oft auch die vom *United States Computer Emergency Readiness Team (CERT)* ausgegebene Kennung oder auch die *Bugtraq ID (BID)* verwendet, welche im Zusammenhang mit der Veröffentlichung der zugehörigen Schwachstelle auf der Bugtraq-Mailingliste von *SecurityFocus* vergeben wird. Die Zuordnung einer Schwachstelle zwischen diesen drei Standards ist oft durchaus möglich, aber nicht immer eindeutig.

2.6. Metasploit

Ein weiteres Werkzeug, das auch in Kali Linux enthalten ist, ist *Metasploit*. Dabei handelt es sich um ein sehr verbreitetes und häufig genutztes, umfassendes Exploiting-Framework, das heute von der US-amerikanischen Firma *Rapid7* weiterentwickelt und sehr häufig im Rahmen von Penetrationstests verwendet wird. Dieses in Ruby implementierte Open-Source-Projekt verfügt über viele verschiedene Tools, um dem Penetrationstester die Arbeit zu erleichtern. Unter einem Exploiting-Framework versteht man in der Regel „eine definierte Plattform zur Entwicklung, zum Testen und zum Einsatz von Exploits“ [Mes15]. Metasploit hingegen bringt noch zahlreiche weitere Funktionalitäten mit sich, besonders im Bereich der Informationsgewinnung und der Post-Exploitation.
Metasploit ist derzeit in drei verschiedenen Versionen[8] erhältlich: während das *Metasploit Framework* (kurz *MSF*) quelloffen ist, so bietet die kommerzielle Lösung *Metasploit Pro*

[8]Stand April 2019, siehe auch: `https://www.rapid7.com/products/metasploit/download/editions/`

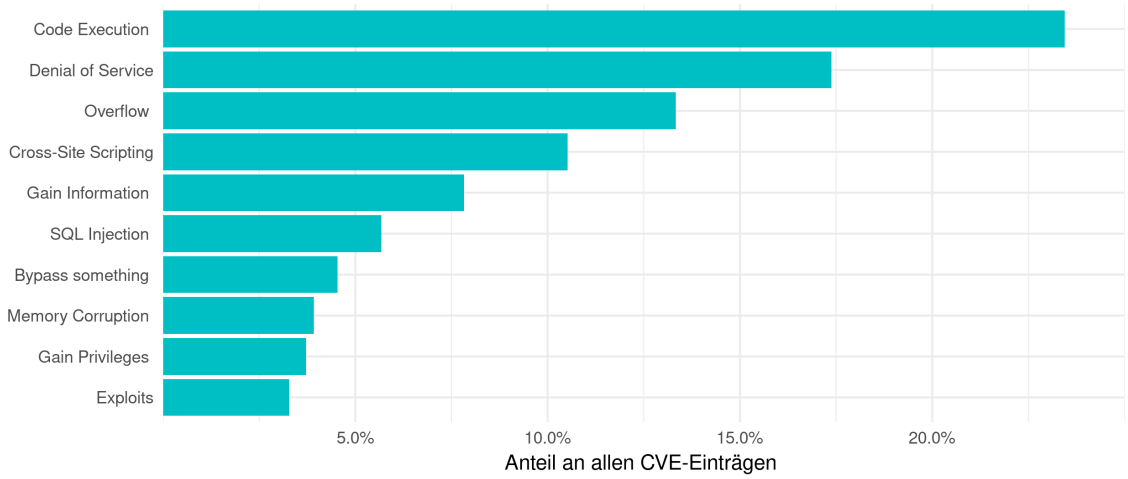

Abbildung 2.4.: Übersicht der Anteile der häufigsten Schwachstellenarten im CVE-Verzeichnis.[6]

unter anderem Support für Unternehmen sowie erweiterte Automatisierungsmöglichkeiten. Dazwischen angesiedelt ist die *Community* Version, welche das Framework beispielsweise um ein Web-Interface erweitert. Das Framework und die Community-Edition sind derzeit kostenlos erhältlich, die Pro-Version kann kostenfrei nur getestet werden.
Metasploit verfügt außerdem über eine *PostgreSQL*-Datenbankanbindung, welche der Speicherung von Workspaces dient. In diesen können wiederum bereits gesammelte Daten, wie beispielsweise Ergebnisse eines Port-/Servicescans oder gültige Credentials persistent gespeichert werden.
Das Metasploit Framework zeichnet sich besonders durch einen sehr modularen und einfach zu erweiternden Aufbau aus. Dadurch wird nicht nur häufige Codeduplikation verhindert, sondern auch eine einfache Aktualisier- und Erweiterbarkeit gewährleistet.

2.6.1. Module

Metasploit verfügt derzeit über mehr als 3.900 verschiedene Module[9], die sich nach Anwendungszweck in sieben verschiedene Kategorien einteilen lassen. Für diese Arbeit sind besonders die Exploit-, Payload- und Auxiliary-Module von Relevanz, auf welche im Folgenden genauer eingegangen wird. Daneben sind auch Encoder-, Evasion-, Nop- und Post-Module integriert.

Exploits

Das Metasploit-Framework bringt eine Reihe von *Exploit*-Modulen für verschiedenste Anwendungen mit sich. Allgemein ist ein Exploit ein Mittel, mit dem eine Schwachstelle systematisch für einen Angriff ausgenutzt werden kann[10]. Nach dieser, weiter gefassten, Definition finden sich in Metasploit drei verschiedene Arten von Exploits: Remote, Local und Denial of Service Exploits. Während Remote Exploits von einem angreifenden Rechner (nachfolgend

[9]Stand: September 2019
[10]`https://www.rapid7.com/fundamentals/vulnerabilities-exploits-threats/`

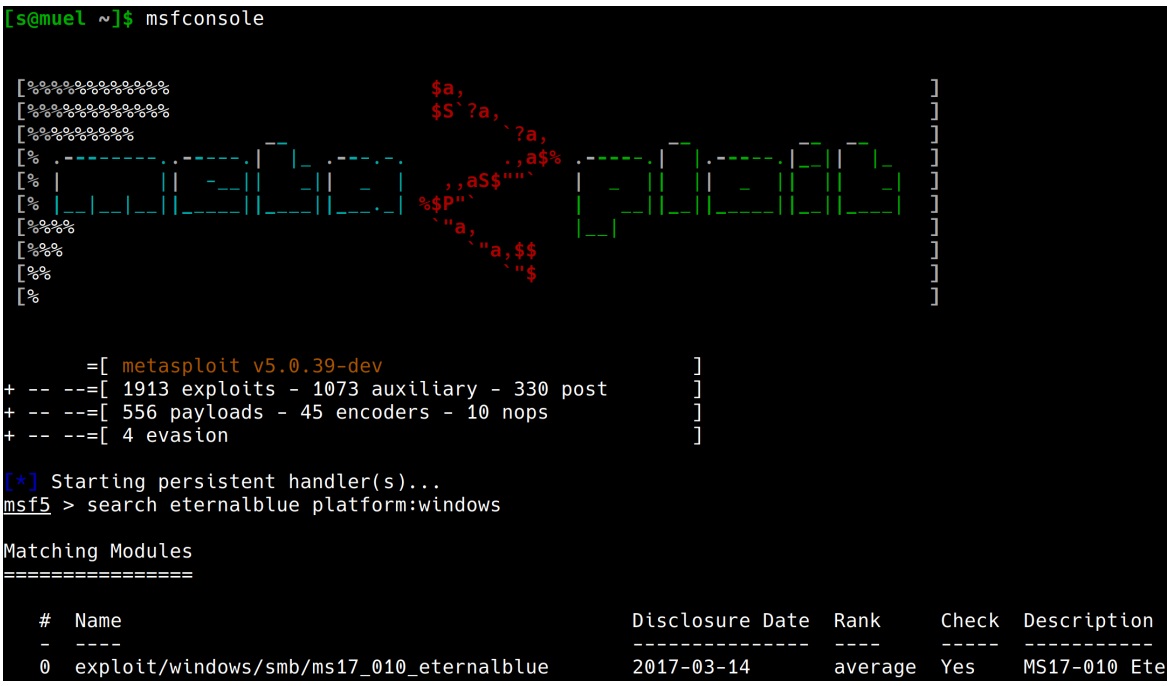

Abbildung 2.5.: Beispielhafte Ausführung von Metasploit: Begrüßungsbanner mit Modul-Statistiken, anschließende Suche nach dem Eternalblue-Exploit für Metasploitable 3.

Host genannt) über eine bestehende Netzwerkverbindung auf ein entferntes Ziel-System (im Folgenden auch *Target* oder *Remote Host* genannt) angewandt werden können, ist bei lokalen Exploits bereits ein Zugang zu ebendiesem nötig. Im Gegensatz zu DoS-Exploits, welche eine Beeinträchtigung der Verfügbarkeit des Ziel-Systems beabsichtigen, zielen lokale Exploits in der Regel nur auf die Ausweitung der eigenen, lokalen Zugriffsrechte ab.
Im Metasploit Framework ist diese Definition etwas enger gefasst, da hier nur Module als Exploits gelten, die eine Schwachstelle ausnutzen und dabei einen Payload ausführen können[11]. Denial of Service-Module zählen in Metasploit daher zu den *Auxiliary*-Modulen.
Die Exploits in Metasploit lassen sich außerdem noch nach weiteren Kriterien kategorisieren, beispielsweise nach dem *Betriebssystem* des Zielrechners und in zweiter Ebene nach der *Anwendung*, welche die ausgenutzte Schwachstelle zur Verfügung stellt.

Anhand der *intendierten Funktion* können Exploits auch in nicht-intrusive, intrusiv-harmlose oder intrusiv-destruktive Exploits unterschieden werden. Während destruktive Exploits versuchen, den ausgenutzten Dienst, erreichbare Dienste oder das ganze Zielsystem zum Absturz zu bringen oder einen solchen Ausfall zumindest hervorrufen können, ist das bei intrusiv-harmlosen und nicht-intrusiven Exploits nicht der Fall. Letztere versuchen in der Regel, die Schwachstelle zu nutzen, um remote Zugang zum Zielsystem zu ermöglichen. Wie dies wiederum ausgeführt wird, hängt vom jeweils gewählten Payload ab.

Exploits weisen in Metasploit neben dem Ziel-Betriebssystem, dem Veröffentlichungsdatum und weiteren Metadaten auch ein *Rank*-Attribut auf. Letzteres soll die Verlässlichkeit eines Exploits abbilden, indem es mehrere andere Attribute kombiniert: die Existenz und

[11] `https://www.offensive-security.com/metasploit-unleashed/modules-and-locations/`

automatische Auswahl eines Default-Targets, die übliche Erfolgsquote und die Wahrscheinlichkeit eines unbeabsichtigten Denial of Service. Das Ranking wird hierbei in sieben Stufen unterteilt: von *Excellent* über *Great* und *Good* zu *Normal*, *Average*, *Low* und schließlich *Manual*. Die genaue Bedeutung der jeweiligen Werte ist in Tabelle 2.6 dargestellt.

Eine Vielzahl von Exploit-Modulen bietet neben dem Rank-Attribut auch die *Check*-

Rank	**Beschreibung**
Excellent	Serviceabsturz ausgeschlossen und keinerlei Speicherkorruption.
Great	Default-Target und automatische Erkennung des Targets vorhanden.
Good	Default-Target, welches der am weitesten verbreiteten Version entspricht, vorhanden.
Normal	Exploit normalerweise verlässlich, aber abhängig von einer bestimmten Serviceversion, oder automatische Targeterkennung nicht zuverlässig.
Average	Exploit allgemein unverlässlich oder schwierig auszuführen.
Low	Erfolgsrate kleiner als 50% für gängige Targets.
Manual	Unsicherer oder sehr schwer auszuführender Exploit, der einem DoS gleichkommt.

Abbildung 2.6.: Übersicht über alle Werte des Rank-Attributs von Exploit-Modulen in Metasploit, sowie deren Bedeutungen.[12]

Funktion an, welche eine Überprüfung der Anwendbarkeit eines Exploits auf das Ziel-System anhand einer geeigneten Methode durchführt. Das Ergebnis der Ausführung von check kann hierbei sechs verschiedene Werte zurückgeben[13]. Die Werte reichen hierbei von *Unknown* (keine Aussage möglich), über *Safe* (nicht verwundbar), *Detected* (unbekannt, aber betroffener Dienst ist vorhanden) und *Appears* (anscheinend verwundbar, nicht überprüft, aber betroffene Version vorhanden) bis hin zu *Vulnerable* (definitiv verwundbar). *Unsupported* gibt an, dass die check-Funktion nicht zur Verfügung steht. Diese Methode ist dabei behilflich, bereits vor der Exploit-Ausführung eine Aussage über die Verwundbarkeit des Targets treffen zu können, auch wenn diese Aussage noch mit einer gewissen Unsicherheit behaftet ist.

Payloads

Payloads stellen den bei erfolgreicher Anwendung des Exploits ausgeführten Code dar. Auch sie lassen sich nach Funktion und Aufbau in verschiedene Typen unterscheiden.
Single Payloads sind eigenständige Payloads mit einer festgelegten Funktion. Aufgrund dieser Eigenständigkeit können Verbindungsversuche von single Payloads problemlos mit anderen Verbindungs-Handlern, wie beispielsweise *netcat*, entgegen genommen werden. *Staged* Payloads hingegen bestehen aus zwei Komponenten: Dem Dropper und der „Endstufe“ mit der ursprünglich gewünschten Funktionalität. Mit dem Exploit wird nur der Dropper ausgeliefert, welcher zuerst eine Netzwerkverbindung vom Ziel zum Angreifer aufbauen muss, um die Endstufe nachzuladen, bevor diese ausgeführt werden kann. Dadurch sind Payloads von diesem Typ normalerweise deutlich schlanker und verlässlicher. Allerdings kann das Nachladen der Stage beispielsweise von einer Firewall blockiert werden.

[12] `https://github.com/rapid7/metasploit-framework/wiki/Exploit-Ranking`, Stand: Mai 2019
[13] `https://www.rubydoc.info/github/rapid7/metasploit-framework/Msf/Exploit/CheckCode`

Ebenso kann zwischen einer *bind-* und einer *reverse*-Verbindung unterschieden werden. Bei einer bind-Verbindung wird in der Regel eine Shell auf einem lokalen Port des Ziel-Systems geöffnet, mit welcher sich der Angreifer verbinden kann. Dies bietet den Vorteil, dass diese Remote Shell auch zu einem späteren Zeitpunkt noch offen für Verbindungen sein kann. Allerdings ist es sehr wahrscheinlich, dass bei einer richtig konfigurierten und aktiven Firewall derartige Verbindungsversuche komplett unterbunden werden, und damit kein Zugriff auf das (eigentlich erfolgreich) angegriffene System möglich ist. Dieses Problem wird bei einer reverse-Verbindung dadurch umgangen, dass das Target die Verbindung zum Angreifer aufbaut. Dieser muss mit einem wartenden Handler auf einem festgelegten Port bereit sein, diese Verbindungsanfrage anzunehmen. Ist dieser Handler zum Zeitpunkt der Anfrage nicht bereit, kann keine Verbindung aufgebaut werden und der Zugang zum Ziel-System bleibt verwehrt [Mes15].
Auch Payloads lassen sich nach der Ziel-Umgebung unterscheiden, in der sie ausgeführt werden können. Diese ist nicht nur von dem Betriebssystem des Zielrechners abhängig, sondern gleichermaßen von dem genutzten Exploit, da die dadurch herbeigeführte Code-Ausführung oft nur eine bestimmte Programmiersprache (für ein bestimmtes Betriebssystem) zulässt.
Payloads können außerdem sehr unterschiedliche Intentionen verfolgen. Dieses Spektrum reicht vom Hinzufügen eines Benutzers oder dem Ändern von Berechtigungen bis zum Öffnen einer umfangreichen meterpreter-Session. Remote-Zugänge zum Ziel-System können ebenfalls über verschiedene Wege realisiert werden, zum Beispiel über eine Standard *Shell* oder eine *meterpreter*-Verbindung, welche zahlreiche weiterführende Funktionen mit sich bringt. Eine solche Verbindung kann wiederum über verschiedene Protokolle, beispielsweise über TCP, HTTP oder HTTPS aufgebaut werden.
Eine Besonderheit des Metasploit-Frameworks ist die bereits genannte, sehr umfangreiche Payload-Option: *meterpreter*. Dieser Metasploit-spezifische Payload bietet den Vorteil, dass er eine Kommandozeile mit vom Betriebssystem des Targets unabhängigen Befehlen anbieten kann und durch das Nachladen von Modulen sehr umfangreiche Funktionalitäten besitzt. Diese Module decken ein weites Spektrum an Aufgaben ab, welches von der lokalen Privilege Escalation über die Aufnahme von Screenshots bis hin zu einem Keylogger reicht. Die Möglichkeit des Ausführens von meterpreter Skripten sorgt wiederum für weitere Automatisierung.
Standardmäßig nutzt Metasploit – falls kein Payload explizit ausgewählt wurde – nach einer fest definierten Liste den besten verfügbaren Payload.

Auxiliary

Metasploit stellt auch eine Vielzahl an *Auxiliary*-Modulen bereit, die unterschiedlichste Aufgaben übernehmen können. Für diese Arbeit interessant sind hier in erster Linie *Scanner*-Module, welche für verschiedenste Dienste verfügbar sind und im Rahmen der Informationsgewinnung hilfreich sein und auch bereits erste Angriffe, wie Dictionary Attacks, beinhalten können. Außerdem finden sich unter den Auxiliary-Modulen auch eine Reihe von Fuzzern und DoS-Modulen. Letztere zählen im weiteren Sinne eigentlich zu den Exploits, werden in Metasploit aber aufgrund der fehlenden Payload-Funktionalität den Auxiliary-Modulen zugeordnet.

Weitere

Wie bereits erwähnt, stellt Metasploit noch vier weitere Modul-Typen zur Verfügung: Encoder-, Evasion-, Nop- und Post-Module.
Encoders dienen der Umgang mit gegebenenfalls unerwünschten Zeichen im Payload, die beispielsweise je nach gewähltem Exploit und Protokoll unerwünschte Effekte haben und den Vorgang auch komplett fehlschlagen lassen können. Encoders können damit, auch wenn das nicht ihre primäre Aufgabe ist, zur Umgehung von Antiviren-Software auf dem Zielsystem nützlich sein, da sie die Signatur-basierte Detektion des Payloads verhindern.
Der Modul-Typ *Evasion* wurde erst 2018 eingeführt und dient primär der Umgehung von Antiviren-Software durch den Payload.
Nops hingegen dienen dem Generieren und Einfügen von zufälligen Bytes in den Shellcode, um durch dieses Padding eine gleichbleibende Länge des Payloads zu gewährleisten. Außerdem kann bei einem Buffer Overflow mit statischer Rücksprungadresse sicher gestellt werden, dass durch das Einfügen von NOPs letztendlich immer an die richtige Stelle im Code gesprungen wird.
Nach einem erfolgreichen Exploit-Vorgang bieten *Post*-Module verschiedene Funktionalitäten, um beispielsweise weitere Informationen zu sammeln, die erlangten Rechte zu erweitern oder das System zu verwalten.

2.6.2. Anwendung

Zur Anwendung von Metasploit gibt es je nach gewählter Edition verschiedene Möglichkeiten. Während Metasploit Community und Pro über ein Web-Interface verfügen, so bleibt dem Framework-Benutzer nur die Interaktion über die *msfconsole* genannte Konsole.

Im Folgenden werden übliche Schritte im Rahmen eines Exploiting-Vorgangs über die *msfconsole* vorgestellt.
Sobald der PostgreSQL-Service und die Metasploit-Konsole über `msfconsole` erfolgreich gestartet sind, kann mit `workspace -a <workspace_name>` ein neuer Workspace erstellt werden. Über das Kommando `workspace <workspace_name>` kann direkt in eine bestehende Arbeitsfläche gewechselt werden.
Zuerst kann mit `db_nmap -p 1-65535 -sV <ip_address>` ein Nmap-Service-Scan über alle Ports des Ziel-Systems durchgeführt werden. Die Scan-Ergebnisse werden direkt in die Metasploit-Datenbank geschrieben und können mit `services` abgerufen werden. db_nmap nutzt hierbei dieselbe Parameterstruktur wie nmap.
Anhand dieses Scans kann nun mit `search <keyword> type:exploit` nach Exploits gesucht werden, die zum Suchbegriff <*keyword*> passen. Weitere Einschränkungen sind unter anderem mit `platform:<os_name>` oder `cve:<CVE>` auch nach Betriebssystemfamilie oder CVE möglich.
Wenn ein passender Exploit gefunden wurde, kann dieser mit `use <exploit_name>` direkt ausgewählt werden. Zum Bearbeiten der Optionen können diese und ihre Standardwerte zuerst mit `show options` angesehen werden und dann lokal einzeln über den Befehl `set <option_name> <new_value>` auf einen neuen Wert gesetzt werden. Eine globale Änderung eines Exploit-Parameters ist über `setg <option_name> <new_value>` möglich.
Der Befehl `info` gibt außerdem weitere Informationen, wie zum Beispiel eine Beschreibung, den Autor und verschiedene Referenzen, Preis.
Mit dem Befehl `exploit` kann der Exploit ausgeführt werden, die Option `-j` erlaubt zudem

eine Ausführung als Job im Hintergrund.
War der Exploit erfolgreich, kann mit `sessions` eine Übersicht über alle bestehenden Sessions ausgegeben werden und mit dem Befehl `sessions -i <session-id>` mit der Session mit der ID <*session-id*> kommuniziert werden. Mit `background` kann diese in den Hintergrund verschoben werden, sodass wieder eine direkte Interaktion mit der Metasploit-Konsole möglich ist. Der Befehl `exit` schließt eine bestehende Remote Session.
Die komplette Metasploit-Session wiederum kann mit `exit` beendet werden (`exit -y` im Falle noch bestehender Sessions).
Die Abbildung 2.7 zeigt eine Ausführung von Metasploit anhand einer erfolgreichen Anwendung des in Kapitel 1 beschriebenen Exploits, der sich einen Buffer Overflow im SMB-Protokoll verschiedener Windows-Versionen zunutze macht.

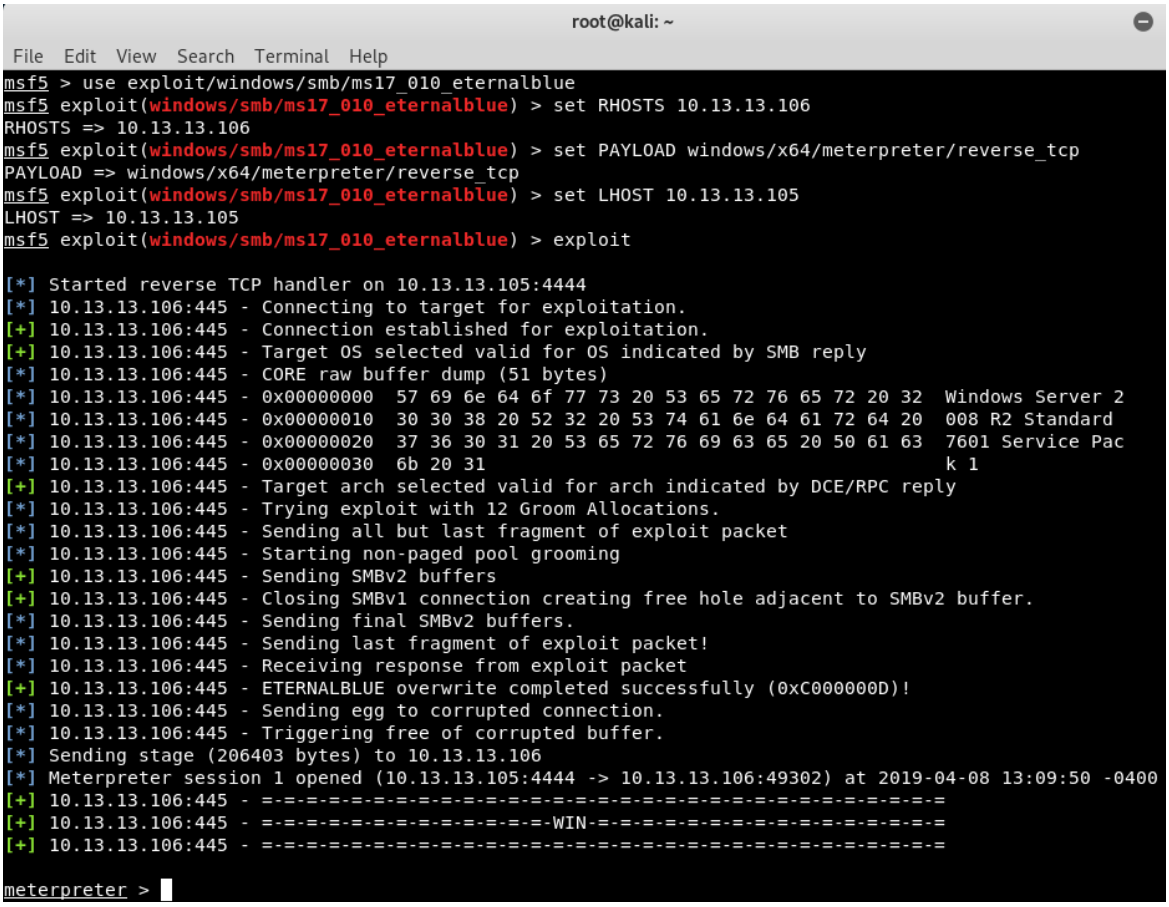

Abbildung 2.7.: Beispielhafte, erfolgreiche Ausführung des *Eternalblue*-Exploits mit Metasploit auf einem Remote Host mit Windows Server 2008 und Erlangen einer *meterpreter*-Session

2.6.3. Armitage

Armitage ist eine GUI für Metasploit, die einige weitere Funktionen mit sich bringt. Die Anwendung unterstützt den üblichen Pentesting-Workflow von der Informationsakquise bis zum aktiven Test. Zur Informationsakquise werden verschiedene Typen von nmap-Scans

angeboten, welche auch das Scannen eines ganzen Netzwerkbereichs erlauben. Bei einem aktiven Test können Ziele manuell oder, wie in Abschnitt 2.7.1 beschrieben, auch automatisiert angegriffen werden. Armitage kann auch besonders bei der Wahl passender Exploits für ein bereits gescanntes Target hilfreich sein. Weitere Vorteile von Armitage liegen unter anderem in der übersichtlichen Darstellung und einfachen Verwaltung der Netzwerk-Struktur, der verfügbaren Remote Hosts und deren Sessions sowie weiterer Entitäten. Ebenso verfügt Armitage über eine Integration der Datenbank von Metasploit, sodass die von der Anwendung gewonnenen Informationen auch nahtlos in der Metasploit Konsole verfügbar sind und vice versa. Armitage wird aktuell nicht mehr aktiv weiterentwickelt oder gewartet, die aktuellste Version 1.4.11 stammt von August 2015.[14]

2.7. Verwandte Arbeiten

Neben Metasploit hat sich bereits eine Vielzahl praktischer und auch wissenschaftlicher Arbeiten mit verwandten Aufgaben beschäftigt. Sie werden im Folgenden kurz vorgestellt und gemäß ihrer Relevanz in den Kontext eingeordnet.

2.7.1. Anwendungen

Für diese Arbeit relevante, bestehende Anwendungen können generell in zwei Kategorien unterteilt werden: *Schwachstellenscanner* und *automatische Exploiting-Tools*.

Schwachstellenscanner

Vulnerability Scanner wie beispielsweise *OpenVAS*, *Nessus* und *Nexpose* erlauben das einfache Auffinden potentiell bestehender Sicherheitslücken auf einem anderen Rechner oder in einem ganzen Netzwerk anhand der im Scanner enthaltenen Schwachstellendaten. Für jede Schwachstelle wird zu deren Detektion ein spezifischer Test in den Scanner integriert. Das bedeutet aber auch, dass solche Sicherheitslücken, für die keine Informationen und Tests integriert sind, auf diesem Wege auch nicht gefunden werden können. OpenVAS verfügt hierfür beispielsweise über tägliche Updates der Netzwerk-Schwachstellen-Tests.
Allgemein ist OpenVAS einer der umfangreichsten, kostenlosen Schwachstellenscanner, der unter anderem auch eine Behandlung von False-Positive-Fehlern umfasst [WY17]. Am Ende eines Scans liefert die Anwendung, wie in Abbildung 2.8 zu sehen, eine Liste der gefundenen, nach dem CVSS-Score sortierbaren Schwachstellen. Das Ergebnis umfasst ebenfalls eine Maßzahl für die „Quality of Detection“ (QoD), welche darüber informiert, wie verlässlich die Erkennung der potentiellen Schwachstelle grundsätzlich ist. Der Wert rangiert hierbei zwischen 1% und 100%, wobei 100% eine Detektion durch einen Exploit darstellt, womit die Ausnutzbarkeit dieser Schwachstelle auch verifiziert wurde, und 1% nur eine allgemeine Bemerkung ohne die Detektion einer verwundbaren Anwendung darstellt.
Außerdem wird, falls verfügbar, eine Lösungsmöglichkeit für ein gefundenes Problem vorgeschlagen, beispielsweise erfolgt ein Hinweis auf ein Versionsupgrade, einen verfügbaren Patch oder eine andere Konfiguration zur Minimierung des durch die Schwachstelle entstandenen Risikos.
Für alle drei der genannten Schwachstellenscanner besteht eine Integrationsmöglichkeit in

[14] Stand: September 2019

Metasploit, sodass diese direkt von der Konsole aus gestartet und ihre Ergebnisse in die Datenbank von Metasploit übernommen werden können.

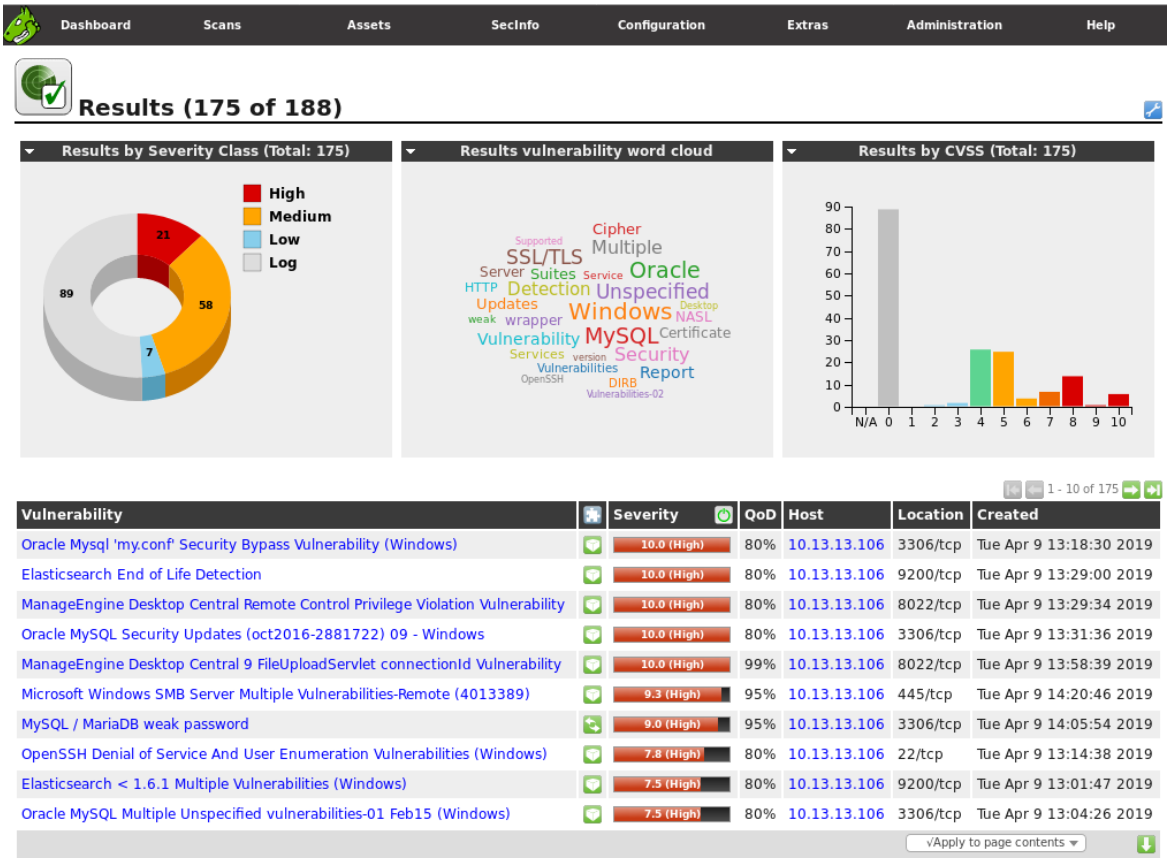

Abbildung 2.8.: Ergebnis eines ausführlichen Schwachstellenscans von Metasploitable 3 (Windows 2008 Server) mit OpenVAS; Übersicht der zehn nach CVSS-Score schwerwiegendsten Schwachstellen

Automatische Exploiting-Tools

Neben den Schwachstellenscannern haben auch automatische Exploiting-Tools eine erhebliche Relevanz für diese Arbeit. Diese Anwendungen versuchen, mit möglichst wenigen Benutzer-Interaktionen durch die massenhafte, automatisierte Ausführung von Exploits ein oder mehrere Ziel-System(e) zu kompromittieren.

Ein Beispiel für eine solche Anwendung ist der bereits in Abschnitt 2.6.3 erwähnte, in Armitage integrierte *Hail Mary*-Angriff. Die erste Version von Hail Mary basierte auf der inzwischen entfernten *db_autopwn*-Funktion des Metasploit Frameworks. Aktuelle Versionen von Armitage verfügen heute über eine intelligentere Hail-Mary-Funktion, welche verschiedene Verbesserungen, wie beispielsweise eine passendere Exploit-Auswahl, enthält. Allgemein werden bei Hail Mary anhand eines vorhandenen Portscans passende Exploits ausgeführt, und überprüft, ob die Ausführung eine Remote Session ermöglichen konnte.
Neben Hail Mary erfüllt auch das Tool ***AutoSploit***[15] einen ähnlichen Einsatzzweck. Es wur-

[15] https://github.com/NullArray/AutoSploit/

de 2018 veröffentlicht und in Python entwickelt. Die Veröffentlichung von AutoSploit hat zu einer kontroversen Debatte geführt, ob die Veröffentlichung eines solchen Tools sinnvoll ist, da es aufgrund der einfachen Nutzung schnell Einzug in das Toolset von „Script Kiddies“ finden könnte [Gal18]. Zu testende Targets können hier nicht nur manuell hinzugefügt, sondern auch über verschiedene IoT-Suchmaschinen gefunden werden. Anschließend wird eine Liste von Exploits ausgeführt und die Ergebnisse in einem Report dokumentiert.

Die beiden hier vorgestellten Tools werden in Abschnitt 4.2 einer genaueren Betrachtung und Evaluation anhand der in Abschnitt 3 definierten Anforderungen für den hier benötigten Einsatzzweck unterzogen.

2.7.2. Wissenschaftliche Arbeiten

Zu dieser Arbeit verwandte Themen wurden auch in zahlreichen wissenschaftlichen Arbeiten mit unterschiedlichsten Hintergründen aufgegriffen.
Das Spektrum reicht hierbei von der Konstruktion eines nicht-intrusiven automatischen Exploiting-Tools bis zum Einsatz von Machine-Learning-Methoden zum Abschätzen des Risikos der Erkennung eines automatischen Angriffes durch ein Intrusion Detection System (IDS).

Mit der Unterscheidung von Exploits mit intrusiver und überwiegend destruktiver Absicht beschäftigt sich die Arbeit [Gam19] und schlägt zur Einteilung der Exploits drei Kategorien vor: intrusiv-harmlos, intrusiv-destruktiv und nicht-intrusiv. Hierzu wird ein Entscheidungsbaum konzipiert, welcher anhand einer Betrachtung verschiedener Attribute und einer anschließenden Exploit-Ausführung eine solche Kategorisierung zulässt. Zu den betrachteten Attributen zählen unter anderem das Rank-Attribut des Metasploit-Frameworks, die CVSS-Base-Werte für die Bereiche Integrität und Verfügbarkeit sowie die Zuordnung als DoS-Modul in Metasploit. In drei Fällen kann bereits anhand dieser Betrachtung eine endgültige Aussage über die Absicht getroffen werden. Ist dies nicht der Fall, so muss der Exploit ausgeführt werden, um lokal auf dem Target ermitteln zu können, wie erheblich die jeweilige Verletzung der Verfügbarkeit und Integrität war.
Ein reiner Scanvorgang von Hosts auf nicht-intrusive Art kann beispielsweise durch die Kombination verschiedener, öffentlich verfügbarer und nicht-intrusiver Methoden erreicht werden [GGE14]. Eine Kombination der IoT-Suchmaschine *Shodan* mit der Schwachstellen-Datenbank NVD (*National Vulnerability Database*) ermöglicht umfangreiches Sammeln der öffentlich verfügbaren Informationen über ein Target. Kombiniert mit Einträgen von DNS-Servern und vom Host selbst (sofern verfügbar) sowie mit Daten aus dem Netzwerk-Traffic können bereits Schwachstellen entdeckt werden sowie potentiell gefährliche Verkettungen von Schwachstellen konstruiert werden. Diese können unter Umständen genutzt werden, um noch weitere Rechte auf dem Target zu erlangen, als es bei Ausnutzung einer einzigen Schwachstelle der Fall wäre.
Wie das Wissen über die nicht-destruktive Absicht von Exploits für ein Design-Konzept und die Implementierung eines nicht-aggressiven, teilweise automatischen Tools für Penetration Tests auf Basis von Metasploit, genutzt werden kann, kann unter [Vig13] gefunden werden. Als oberste Priorität wurde für diese Arbeit die Nicht-Gefährdung des Systems under Test definiert, indem beispielsweise dessen Stabilität und Integrität durch eine Ausführung der Anwendung nicht beeinträchtigt werden. Zur Bewertung des Risikos und damit zum Ausschluss riskanter Operationen wurde hierfür ein eigener Index eingeführt. Der Nutzer kann dabei anhand verschiedener Parameter das Verhalten der Anwendung an die Bedürfnisse

seines zu testenden Systems anpassen.
Ein andere, besonders modulare Architektur zur Penetrationstest-Automatisierung wird in [Dep13] vorgeschlagen. Sie folgt dem „Detect, Identify, Predict, and React“ (DIPR) Prinzip der „Intelligence Automation“. Hierbei wird jede dieser vier Aufgaben an ein dediziertes Modul übergeben, welches wiederum je nach Bedarf mit der GUI und der Netzwerk- und Exploit-Datenstruktur interagieren kann. Das *Detect*-Modul sammelt Informationen über das Target, welche vom *Identify*-Modul zur Identifikation von Schwachstellen benutzt werden. Das *Predict*-Modul ermittelt den empfohlenen Angriffsweg, welcher vom *React*-Modul letztendlich ausgeführt werden kann.

Das in [EKMM11] vorgestellte *FIDIUS*-Framework verfolgt hingegen einen ganz anderen Ansatz und nutzt verschiedene Machine-Learning-Techniken und das Metasploit-Framework zum Testen von Schwachstellen. Es versucht damit, Angriffe weiter zu automatisieren und dabei eine mögliche Entdeckung durch ein Intrusion Detection System (IDS) zu umgehen. Hierfür wird ein passendes Design entworfen und anschließend implementiert. Durch die Möglichkeit, vorab den Grad der Detektion des Angriffes durch ein IDS abzuschätzen, eignet es sich besonders für eine automatisierte Exploit-Ausführung, da eine solche oft viel Noise verursacht und leicht zu erkennen ist.

3. Anforderungen

Anhand der für den beschriebenen Anwendungszweck benötigten Funktionalitäten sollen nun die konkreten Anforderungen an die bestehenden sowie die neu zu entwickelnde Anwendung erarbeitet werden.

3.1. Funktionale Anforderungen

Eine hierfür geeignete Anwendung nutzt einen gegebenen Input, um nach bestimmten Kriterien Exploits *auszuwählen* und diese gemäß den Vorgaben *auszuführen*. Anschließend sollen die *Auswirkungen der Exploits erkannt* und der *Erfolg bewertet* werden, was wiederum in den *abschließenden Report* einfließt. Die Anforderungen werden im Folgenden anhand der übergeordneten Ausführungsschritte eingeteilt.
Eine vollständige Übersicht über alle funktionalen Anforderungen kann in Tabelle 3.1 auf Seite 30 eingesehen werden.

3.1.1. Exploit-Auswahl

Mithilfe des eingelesenen Inputs sollen nun gemäß der vom Nutzer vorgegebenen Parameter passende Exploits für das Ziel-System ausgewählt werden. Das Matching von Exploits auf Schwachstellen soll ein zentraler Bestandteil der Anwendung sein und komplett automatisch, ohne eine direkte Interaktion mit dem Benutzer, erfolgen.

→ **Funktionale Anforderung 1**: Automatische Auswahl ohne Benutzer-Interaktion

Die Anwendung muss über einen Zugriff auf die Modul-Sammlung von Metasploit verfügen, um so beispielsweise nach Exploits suchen sowie weitere zugehörige Informationen abrufen zu können.

→ **Funktionale Anforderung 2**: Zugriff auf die Modul-Sammlung von Metasploit

Die Verwendung verschiedener Merkmale einer Schwachstelle (wie z.B. CVE-Nummer, Bugtraq-ID, Alias-Name, Name und Version des betroffenen Services etc.) soll bei der Exploit-Suche sicherstellen, eine möglichst hohe Abdeckung zu erzielen. Damit soll beispielsweise der Fall ausgeschlossen werden, dass eine bestehende Schwachstelle nicht überprüft werden kann, da der zugehörige Exploit nicht ausgewählt wurde.
Eine Anwendung zu weniger Exploits ist hierbei allgemein ein deutlich schwerwiegenderes Problem als die Anwendung falscher Exploits, da letztere schlimmstenfalls nur die Ausführungsdauer verlängern oder eine andere als die eigentlich beabsichtigte Schwachstelle ausnutzen. Wird dagegen eine Schwachstelle als „nicht von der Anwendung ausnutzbar“ markiert, kann dies zu einem falschen Gefühl der Sicherheit verleiten.

→ **Funktionale Anforderung 3**: Auswahl mindestens aller passenden Exploits anhand mehrere Kriterien

Der Benutzer soll hierbei kontrollieren können, welche Mindestverlässlichkeit ein auszuführender Exploit aufweisen soll. Dies soll sicherstellen, dass einerseits – falls gewünscht – nur Exploits ausgeführt werden können, die besonders in Umgebungen mit hohen Anforderungen an die Verfügbarkeit die Stabilität des Systems nicht absichtlich beeinträchtigen und andererseits, dass bei einem Durchlauf mit begrenzter Zeit oder begrenztem Umfang nur die verlässlichsten Exploits zu Rate gezogen werden.

→ **Funktionale Anforderung 4**: Einschränkung der Auswahl bezüglich des Mindestrankings

Außerdem soll der Benutzer auch in der Lage sein, die Auswahl der zu verwendenden Exploits hinsichtlich des Ziel-Betriebssystems einzuschränken. So sollen entweder nur Exploits ausgeführt werden können, die für das vorhandene Betriebssystem entwickelt wurden, oder auch solche, die auf mehreren Betriebssystemen funktionieren. Letzteres erhöht zwar die Genauigkeit der Anwendung, aber auch den Zeitbedarf.
Wird vom Nutzer bezüglich beider Parameter keine Einschränkung vorgenommen, so sollen sämtliche verfügbaren Exploits benutzt werden.

→ **Funktionale Anforderung 5**: Einschränkung der Auswahl bezüglich des Ziel-Betriebssystems

Bevor die ausgewählten Exploits im nächsten Schritt auf das Ziel-System angewandt werden können, ist es wichtig, deren Absicht zu ermitteln, da ohne diese auch keine präzise Bewertung des Erfolgs möglich ist.

→ **Funktionale Anforderung 6**: Ermittlung der Absicht des jeweiligen Exploits

3.1.2. Exploit-Ausführung

Sobald die Auswahl der Exploits abgeschlossen ist, sollen diese möglichst automatisiert auf das gewählte Target angewandt werden. Für eine möglichst störungsfreie Ausführung und den Fall eines Anwendungsabbruches durch den Benutzer sollen die kritischsten Schwachstellen als Erste überprüft werden, was wiederum zuerst mit den verlässlichsten der dafür verfügbaren Exploits erfolgen soll. Letzteres ist besonders wichtig, da bei der Ausführung mehrerer Exploits für eine einzelne Schwachstelle sichergestellt werden muss, dass die Ergebnisse späterer Exploit-Vorgänge nicht von vorhergehenden beeinflusst werden. Folglich kann die Ausnutzung einer Schwachstelle eines Dienstes mit verlässlichen Exploits nicht mehr erfolgen, wenn selbiger bereits aufgrund der Ausführung eines unverlässlichen Exploits beendet wurde. Insofern ist die Sortierung der Exploits vor der Ausführung von großer Relevanz. Die kritischsten Schwachstellen sind zudem sowohl für den Tester als auch den Auftraggeber am interessantesten und sollen deswegen zuerst überprüft werden.

→ **Funktionale Anforderung 7**: Sortierung der Schwachstellen nach Schweregrad

→ **Funktionale Anforderung 8**: Sortierung der Exploits nach Verlässlichkeit

Wichtig ist in diesem Schritt besonders eine richtige Konfiguration der obligatorischen und Exploit-spezifischen Parameter, da eine Fehlkonfiguration eine erfolgreiche Ausführung stark beeinträchtigt und gegebenenfalls komplett verhindert.

→ **Funktionale Anforderung 9**: Automatisches Setzen der Exploit-Parameter

Ebenso automatisiert soll der für diesen Einsatzzweck am besten passende Payload für den auszuführenden Exploit ausgewählt werden. Die Eignung definiert sich hierbei einerseits aus

der Zuverlässigkeit des Payloads und andererseits aus der Unterstützung der im nächsten Schritt vorzunehmenden Erfolgsbewertung und Informationsakquise.

→ **Funktionale Anforderung 10**: Automatische Auswahl des am besten geeigneten Payloads

Können jedoch nicht alle Parameter automatisch ermittelt und gesetzt werden, so soll der Benutzer gefragt werden, ob er den vorhandenen Standardwert beibehalten oder selbst einen Wert dafür setzen möchte.

→ **Funktionale Anforderung 11**: Anfrage an den Nutzer, falls keine automatische Ermittlung der Exploit-Parameter möglich

Sind alle benötigten Parameter gesetzt, soll der Exploit automatisch und gegebenenfalls wiederholt auf das Target angewandt werden. Eine mehrfache Ausführung ist unter Umständen wichtig, um die Erfolgswahrscheinlichkeit für Schwachstellen, bei denen der Erfolg von einem präzisen Timing abhängt, zu erhöhen.

→ **Funktionale Anforderung 12**: Wiederholte Anwendung der Exploits auf das festgelegte Target

Da die Exploit-Ausführung neben der Erfolgsverifikation der Schritt im Exploit-Prozess mit dem größten Einfluss auf die Ausführungsgeschwindigkeit ist, soll anhand eines Parameters festgelegt werden können, wie schnell und präzise dieser Schritt durchgeführt wird. Zur Verbesserung der Genauigkeit sollen die Exploits beispielsweise mehrfach in definierten Intervallen angewandt werden. Der Nutzer soll hierbei die Größe der Intervalle und die Anzahl der Wiederholungen beeinflussen können. Eine höhere Genauigkeit geht hier jedoch auf Kosten einer längeren Laufzeit.

→ **Funktionale Anforderung 13**: Steuerung der Geschwindigkeit der Exploit-Ausführung

3.1.3. Automatische Erkennung der Auswirkungen und Bewertung

Nach der Ausführung eines Exploits sollen dessen Auswirkungen auf den Remote Host erkannt werten und der Erfolg anschließend bewertet werden. Eine effektive Umsetzung dieser Anforderung fußt hierbei auf mehreren Methoden, die allgemein eine möglichst präzise Einschätzung der überprüften Verwundbarkeit ermöglichen, sodass eine erneute, manuelle Ausführung des Exploits zur weiteren Informationsbeschaffung nicht mehr notwendig ist.

Um eine präzise Erkennung der Auswirkungen auf verschiedenen Ebenen zu ermöglichen, ist der Einsatz mehrerer unterschiedlicher Methoden notwendig. Hierfür effektive Möglichkeiten sind beispielsweise:

a) eine Überprüfung, ob das Target noch aktiv und erreichbar ist mittels *ping*,

b) eine Überprüfung, ob der für den Angriff genutzte Port noch geöffnet ist, mittels eines *TCP-SYN-Scans*,

c) bei Exploits für WebApps eine Überprüfung, ob der jeweils zuständige Webserver noch aktiv ist, beispielsweise mittels einer einfachen *HTTP-Anfrage*,

d) eine Überprüfung, ob von dem genutzten Payload erfolgreich eine *Session* auf dem Ziel-System geöffnet werden konnte (hier reicht es bei reverse Payloads bereits, die Verbindungsanfrage zu registrieren) und

e) bei staged Payloads eine Überprüfung, ob der Dropper die zweite *Stage nachlädt.*

→ **Funktionale Anforderung 14**: Automatische Erkennung der Auswirkungen mithilfe verschiedener Methoden

Auch die Geschwindigkeit der Auswirkungserkennung soll zum Beispiel stufenweise steuerbar sein, um analog zur schnelleren Exploit-Ausführung auch einen schnelleren Modus zur Auswirkungserkennung und Erfolgsbewertung wählen zu können.

→ **Funktionale Anforderung 15**: Steuerung der Geschwindigkeit der Erkennung

Eine Erfolgsbewertung eines Exploits ist allerdings nur möglich, wenn vorher dessen Absicht klar definierbar ist. Hierfür ist eine Kategorisierung von Exploits, wie in [Gam19] konzipiert, nötig.
Anhand der detektierten Auswirkungen und abhängig von der vorhandenen Kategorisierung des Exploits soll anschließend eine Bewertung des Erfolgs durchgeführt werden. Hierbei müssen generell drei Fälle unterschieden werden:

a) der Exploit hat seine intendierte Funktion erfolgreich ausgeführt,

b) der Exploit zeigte keinerlei Auswirkungen auf dem Ziel-System oder

c) der Exploit erzielte eine Wirkung, die allerdings nicht seinem Zweck entsprach.

Falls ein Exploit seine Funktion erreicht, aber als Nebeneffekt noch eine andere Wirkung erzielt, muss eine Bewertung wie in Fall c) vorgenommen werden, da unerwünschte Seiteneffekte besonders auf produktiven Systemen fatale, unvorhergesehene Auswirkungen haben können. Insofern wäre es für den Auftraggeber riskant, einen solchen Exploit im Zuge der Reproduktion des Tests auf einer produktiven Instanz anzuwenden.

→ **Funktionale Anforderung 16**: Präzise Bewertung des Erfolgs

Grundsätzlich soll aber nicht nur die beschriebene Bewertung des Erfolgs vorgenommen werden, sondern, falls möglich, auch weitergehende Informationen gesammelt werden, die für eine spätere Auswertung und gegebenenfalls eine Reproduktion des dokumentierten Testfalls hilfreich sein können. Hierzu zählen zum Beispiel Informationen über erlangte Rechte und das hierfür verwendete Benutzerkonto auf dem Ziel-System, falls eine Remote Shell geöffnet werden konnte oder auch eine Auflistung unerwünschter Wirkungen, falls solche auftreten.

→ **Funktionale Anforderung 17**: Weiterführendes Sammeln von Informationen

3.1.4. Reportgenerierung

Am Ende des Durchlaufes soll von der Anwendung automatisch ein Report generiert werden, der im Detail über die einzelnen Resultate informiert.

→ **Funktionale Anforderung 18**: Automatische Reportgenerierung

Er soll insbesondere folgende Informationen beinhalten: die ausgewählten Exploits, die dafür gewählten Parameterwerte, den angegriffenen Service, die Kategorisierung der Exploits sowie ein Protokoll zu jeder Ausführung. Außerdem soll die Erfolgsbewertung, sowie alle dafür genutzten Daten und darüber hinaus erhobene Informationen enthalten.

→ **Funktionale Anforderung 19**: Genaues Protokoll jeder Exploit-Ausführung im Report

→ **Funktionale Anforderung 20**: Erfolgsbewertung und dafür verwendete Daten im Report

Die jeweiligen Reporteinträge sollen außerdem in Bezug zum gegebenen Input eines Schwachstellenscanners gesetzt werden, um dadurch beispielsweise auch die CVE-Nummer sowie den CVSS-Score der zugehörigen Schwachstelle bereitzustellen.

→ **Funktionale Anforderung 21**: Bezug zum eingelesenen Report des Schwachstellenscanners für jeden Eintrag

Der Report soll in einem weit verbreiteten Dateiformat, wie beispielsweise CSV oder XML, gespeichert werden können und einen schnellen Überblick über die wichtigsten Ergebnisse der Anwendungsausführung bieten. Die einzelnen Ergebnisse sollen gemäß eines adäquaten Parameters nach Priorität sortiert werden, beispielsweise anhand des CVSS-Scores, sodass hiermit Einträge, welcher aus Sicht eines Administrators unbedingt einer Reaktion bedürfen, schnell gefunden werden können.

→ **Funktionale Anforderung 22**: Speichern des Reports in einem verbreiteten Format

→ **Funktionale Anforderung 23**: Sortieren und Filtern der Einträge im Report nach Priorität

3.1.5. Allgemein

In die Liste der nach Ausführungsschritt gruppierten funktionalen Anforderungen reihen sich noch mehrere nicht-funktionale Anforderungen ein, die eine benutzerfreundliche Ausführung sicherstellen sollen.
Ein Abbrechen der Anwendung soll jederzeit möglich sein. Nach einem Abbruch soll trotzdem jederzeit ein Report erstellt werden können, sodass sämtliche bisher gesammelten Informationen noch dokumentiert werden und nicht ungewollt verloren gehen.

→ **Funktionale Anforderung 24**: Abbrechen jederzeit ohne Verlust bisher gesammelter Daten möglich

Bei Anwendungsabbruch soll dem Nutzer neben der Möglichkeit zur Reportgenerierung auch die Option angeboten werden, den Anwendungsstatus zu speichern, um damit die Ausführung zu einem späteren Zeitpunkt fortsetzen zu können. Dies ist insbesondere dazu nötig, Testergebnisse beispielsweise nach gegebenenfalls unbeabsichtigtem Ausfall des Zielsystems nicht zu verfälschen, indem alle nachfolgend ausgeführten Exploit-Module gegen ein nicht-erreichbares System ausgeführt werden.

→ **Funktionale Anforderung 25**: Fortsetzung der Ausführung jederzeit nach Anwendungsabbruch möglich

Die Anwendung soll dabei behilflich sein, eine versehentliche Ausführung über das Internet möglichst auszuschließen, beispielsweise durch eine Überprüfung, ob sich die IP-Adresse des Ziel-Systems in einem öffentlichen IP-Adressbereich befindet. Dafür werden Switches zum Abbrechen und an passenden Stellen Prompts zur Bestätigung in das Tool integriert, welche aber deaktiviert werden können, um so den Workflow erfahrenerer Benutzer nicht zu stören.

→ **Funktionale Anforderung 26**: Überprüfungen zum Verhindern versehentlicher Angriffe

Nr.	Beschreibung
	Exploit-Auswahl
1	Automatische Auswahl ohne Benutzer-Interaktion
2	Zugriff auf die Modul-Sammlung von Metasploit
3	Auswahl mindestens aller passenden Exploits anhand mehrerer Kriterien
4	Einschränkung der Auswahl bezüglich des Mindestrankings
5	Einschränkung der Auswahl bezüglich des Ziel-Betriebssystems
6	Ermittlung der Absicht des jeweiligen Exploits
	Exploit-Ausführung
7	Sortierung der Schwachstellen nach Schweregrad
8	Sortierung der Exploits nach Verlässlichkeit
9	Automatisches Setzen der Exploit-Parameter
10	Automatische Auswahl des am besten geeigneten Payloads
11	Anfrage an den Nutzer, falls keine automatische Ermittlung der Exploit-Parameter möglich
12	Wiederholte Anwendung der Exploits auf das festgelegte Target
13	Steuerung der Geschwindigkeit der Exploit-Ausführung
	Automatische Erkennung der Auswirkungen und Bewertung
14	Automatische Erkennung der Auswirkungen mithilfe verschiedener Methoden
14.1	Überprüfung mittels ping
14.2	Überprüfung mittels TCP-SYN-Scan
14.3	Überprüfung mittels HTTP-Anfrage
14.4	Erkennung einer erfolgreich aufgebauten Remote Session
14.5	Erkennung des Nachladens der zweiten Stage durch den Dropper
15	Steuerung der Genauigkeit der Erkennung
16	Präzise Bewertung des Erfolgs
17	Weiterführendes Sammeln von Informationen
	Reportgenerierung
18	Automatische Reportgenerierung
19	Genaues Protokoll jeder Exploit-Ausführung im Report
20	Erfolgsbewertung und dafür verwendete Daten im Report
21	Bezug zum eingelesenen Report des Schwachstellenscanners für jeden Eintrag
22	Speichern des Reports in einem verbreiteten Format
23	Sortieren und Filtern der Einträge im Report nach Priorität
	Allgemein
24	Abbrechen jederzeit ohne Verlust bisher gesammelter Daten möglich
25	Fortsetzung der Ausführung jederzeit nach Anwendungsabbruch möglich
26	Überprüfungen zum Verhindern versehentlicher Angriffe

Abbildung 3.1.: Übersicht über die funktionalen Anforderungen, nach Ausführungsschritt gruppiert

3.2. Nicht-funktionale Anforderungen

Neben den genannten funktionalen Anforderungen soll die Anwendung auch mehreren nicht-funktionalen Anforderungen genügen. Tabelle 3.2 auf Seite 33 listet alle nicht-funktionalen Anforderungen in kompakterer Fassung auf.

Unterstützte Betriebssysteme

Die zu entwickelnde Anwendung soll mindestens auf aktuellen Linux-Betriebssystemen lauffähig sein. Angesichts der Popularität von Linux im Umfeld von Penetrationstests empfiehlt sich selbiges als Referenz-Plattform.

→ **Nicht-funktionale Anforderung 1**: Ausführbarkeit der Anwendung auf Linux

Außerdem soll die Anwendung Penetration Tests auf Hosts mit verschiedenen Betriebssystemen ausführen können. Als Ziel-Betriebssysteme sollen aufgrund der hohen Verbreitung mindestens Windows und Linux unterstützt werden. Gemeinsam besitzen diese eine Marktanteil von über 80% bei Desktops[1] und teilen den Markt für Webserver unter sich auf.[2] Summiert man die Anzahl der CVE-Einträge über die verschiedenen Versionen beziehungsweise Distributionen, so zeigt sich, dass für Windows und Linux außerdem die meisten Schwachstellen bekannt sind. Von den derzeit verfügbaren 1913 Exploit-Modulen, lassen sich 337 auf Linux-Hosts und 1208 auf Windows-Rechner anwenden[3]. Durch diese Gegebenheiten ist sichergestellt, dass die Einschränkung bezüglich des Ziel-Betriebssystems die Verwendbarkeit der Anwendung kaum einschränkt.

→ **Nicht-funktionale Anforderung 2**: Unterstützung von Windows und Linux als Ziel-Betriebssysteme für Exploits

Zeitverhalten

Besonders wichtig ist nicht nur das Terminieren der Anwendung, sondern auch ein benutzerfreundliches Zeitverhalten, um einen wirklichen Mehrwert gegenüber einer manuellen Durchführung derselben Aufgaben zu bieten. Der Nutzer soll hierbei mittels mehrerer Parameter Anpassungen an seine Bedürfnisse vornehmen können.

→ **Nicht-funktionale Anforderung 3**: Benutzerfreundliches Zeitverhalten

Ebenso soll für Fälle, in denen ein höherer Automatisierungsgrad gewünscht ist, ein Schnelldurchlauf der Anwendung möglich sein. Auch wenn die Wahl der Parameter die Genauigkeit der Anwendung beeinflussen kann, soll in allen Fällen eine angemessen genaue Durchführung gewährleistet sein.

→ **Nicht-funktionale Anforderung 4**: Schnelldurchlauf möglich

Anwendungssteuerung

Die Anwendung soll von einem Benutzer direkt über die *Kommandozeile* gesteuert werden können. Dies ist ausreichend, da viele Tools, die häufig für Penetrationstests verwendet werden, keine graphische Benutzerschnittstelle anbieten.

[1] `http://gs.statcounter.com/os-market-share/desktop/worldwide/2019`, Stand: März 2019
[2] `https://w3techs.com/technologies/overview/operating_system/all`, Stand: April 2019
[3] Stand: September 2019

→ **Nicht-funktionale Anforderung 5**: Bedienung über die Kommandozeile

Der Nutzer wird hierbei von einer Art *Wizard* während des Ablaufs begleitet. Es soll aber auch möglich sein, die Anwendung mit allen erforderlichen Parameter direkt aufzurufen, und den Assistenten abzuschalten, sodass ein Anwendungsdurchlauf auch ohne weitere Abfragen möglich ist, was beispielsweise eine Skript-gesteuerte Ausführung ermöglicht.

→ **Nicht-funktionale Anforderung 6**: Führung des Nutzers von einem Wizard, falls nicht alle benötigten Parameter angegeben wurden

Erweiterbarkeit

Aufgrund der Volatilität im Bereich der IT-Sicherheit ist die einfache Erweiterbarkeit und Modifizierbarkeit der Anwendung sehr wichtig, um einen hohen Anpassungsgrad auch an zukünftige Einsatzzwecke zu gewährleisten.
Die Erweiterbarkeit soll sich hierbei auf zwei Dimensionen erstrecken: Es sollen auf einfache Weise neue Exploits und weitere Ziel-Betriebssysteme hinzugefügt sowie zusätzliche Module zur weiteren Automatisierung integriert werden können. Letzteres soll beispielsweise Module umfassen, die dem Benutzer die richtigen Exploit-spezifischen Parameter vorschlagen können, indem sie dafür benötigte Scan-Funktionen implementieren.
Damit kann die Anwendung nicht nur aktualisiert, sondern auch auf einfache Weise um weitere Funktionen erweitert werden.

→ **Nicht-funktionale Anforderung 7**: Einfache Erweiterbarkeit bezüglich neuer Betriebssysteme und Exploits

→ **Nicht-funktionale Anforderung 8**: Einfache Erweiterbarkeit bezüglich weiterer Automatisierungsfunktionalitäten

Bedienbarkeit

Die Anwendung soll außerdem einfach zu benutzen sein, was in erster Linie durch drei Maßnahmen sichergestellt werden soll: Eine Anleitung hilft dabei, die Einrichtung Schritt für Schritt sowie die Installation der Abhängigkeiten – auch für unerfahrene Python- und Metasploit-Nutzer – einfach und erfolgreich durchzuführen. Ebenso existiert eine Anleitung, wie die Anwendung und davon benötigte Dienste zu starten sind.

→ **Nicht-funktionale Anforderung 9**: Anleitung zur Einrichtung, Installation der Abhängigkeiten und zum Anwendungsstart

Hilfetexte jenseits des Wizards sollen den Benutzer vor der Ausführung unterstützen, indem sie beispielsweise eine Übersicht über die jeweiligen verfügbaren Parameter und deren Funktionen geben.

→ **Nicht-funktionale Anforderung 10**: Hilfetexte verfügbar

3.3. Anwendbarkeit automatischer Tests

Automatische Exploit-Vorgänge bieten eine Reihe von Vorteilen, müssen aufgrund diverser Einschränkungen und Nachteile aber mit Vernunft sinnvoll angewandt werden.

Nr.	Beschreibung
1	Ausführbarkeit der Anwendung auf Linux
2	Unterstützung von Windows und Linux als Ziel-Betriebssysteme für Exploits
3	Benutzerfreundliches Zeitverhalten
4	Schnelldurchlauf möglich
5	Bedienung über die Kommandozeile
6	Führung des Nutzers von einem Wizard, falls nicht alle benötigten Parameter angegeben wurden
7	Einfache Erweiterbarkeit bezüglich neuer Betriebssysteme und Exploits
8	Einfache Erweiterbarkeit bezüglich weiterer Automatisierungsfunktionalitäten
9	Anleitung zur Einrichtung, Installation der Abhängigkeiten und zum Anwendungsstart
10	Hilfetexte verfügbar

Abbildung 3.2.: Übersicht über alle nicht-funktionalen Anforderungen

3.3.1. Vorteile

Neben den bereits genannten Vorteilen der Zeit- und Kostenersparnis können automatische Penetrationstests noch weitere Verbesserungen gegenüber einem manuellen Vorgehen aufweisen.
So können diese beispielsweise auch automatisch geplant und wiederholt werden und liefern dabei in jedem Durchlauf ein konstantes Ergebnisformat. Dadurch wird die regelmäßige Durchführung und damit die Überwachung der vorhandenen Infrastruktur deutlich einfacher und konsistenter ermöglicht, da die Ergebnisse dieser Tests einfach miteinander verglichen werden können. Beispielsweise können für jede bereits manuell geschlossene Sicherheitslücke geeignete Tests zur Liste der geplanten und automatisch durchgeführten Tests hinzugefügt werden, um so ein Wiederauftreten der Schwachstelle, zum Beispiel aufgrund eines Versionswechsels oder einer geänderten Konfiguration, möglichst schnell zu erkennen.
Für den ausführenden Penetrationstester bedeuten automatische Tests die Möglichkeit, sich mehr auf komplexere und manuell auszuführende Szenarien fokussieren zu können. Indem dem Tester damit besonders häufige Aufgaben abgenommen und er bei anderen ausreichend unterstützt wird, ist so beispielsweise eine Kostensenkung oder bei gleichen Kosten eine höhere Prüftiefe möglich.
Aufgrund der Kostenvorteile und des niedrigeren erforderlichen Wissensstandes senken automatische Tests die Schwelle zur erstmaligen Einführung von Penetrationstests. Hilfreich ist dies für eine flächendeckendere Verbreitung besonders in kleineren Unternehmen, die sich beispielsweise weder ein professionelles, externes Red Team, noch eine dedizierte Sicherheitsabteilung leisten können.
Ähnlich wie gewöhnliche Softwaretests lassen sich auch automatische Penetrationstests für Anwendungen einfacher in den Softwareentwicklungsprozess integrieren und können so bereits vor dem Release bei der Detektion und Schließung von Sicherheitslücken behilflich sein. Besonders bei agilen Softwareprojekten ist dies ein großer Vorteil, da regelmäßige manuelle Audits hierbei oft zu aufwändig und langsam sein können – und beispielsweise bei Verwendung von Scrum auch gar nicht vorgesehen sind.

3.3.2. Probleme

Dass menschliche Penetrationstester noch längst nicht überflüssig sind, liegt an einer Reihe von Problemen und Einschränkungen von automatischen Penetrationstests. Diese Nachteile können allgemein in *aktive* und *passive* Risiken unterschieden werden.
Unter aktiven Risiken versteht man solche, die direkt von der Ausführung des Tests ausgehen: beispielsweise der Ausfall eines nicht-redundanten Systems oder ein potentieller Datenverlust. Trotz verschiedener Sicherheitsmaßnahmen fehlt bei automatisierten Penetrationstests die Um- und Rücksicht, die ein ethischer Hacker, zum Beispiel hinsichtlich besonderer Bedürfnisse bei der System-Verfügbarkeit, aufbringen kann [Oak19].
Passive Risiken gehen einher mit dem Ersetzen von menschlichen Penetrationstestern durch automatisierte Software. Davon abgesehen, dass solche automatisierten Tests nur *technische* Risiken abdecken können, besteht das Risiko, dass ein Einsatz einer solchen Anwendung ein „falsches Gefühl von Sicherheit“ entstehen lässt. Allerdings ist man nach einem Penetrationstest nur gegen Angriffsvektoren geschützt, die von der Software auch abgedeckt werden. Das Niveau eines reellen Hackers kann mit einer solchen Software jedoch in keinem Fall erreicht werden [Oak19].
Ebenso erhöhen automatisierte Tests im Falle eines verdeckten Vorgehens oft die Wahrscheinlichkeit, entdeckt zu werden. Die Vielzahl an Scans, versuchten Exploit-Vorgängen und Verbindungsaufbauten, die hierbei durchgeführt werden, machen beispielsweise eine Detektion durch ein Intrusion Detection System oder bei der Auswertung der Logdatei deutlich wahrscheinlicher.
Auf der anderen Seite ist ein gewisser Grad an Automatisierung insbesondere aufgrund der Ressourcenersparnisse durchaus wünschenswert.
Besonders sinnvoll ist damit ein unterstützender Einsatz von automatisierten Tools, der in einem vernünftigen Verhältnis zum Einsatz von manuellen Pentests steht und diese keinesfalls ersetzen soll.

4. Evaluation bestehender Anwendungen

Im Folgenden soll zuerst eine passende Testumgebung, bestehend aus drei virtuellen Maschinen, definiert werden. Anschließend werden mit dieser, und anhand der nun formulierten Anforderungen, die bestehenden und gegebenenfalls geeigneten Anwendungen, Hail Mary und AutoSploit, evaluiert.

4.1. Testumgebung

Für die Testumgebung ist es wichtig, eine möglichst realitätsnahe Umgebung zu schaffen. Gleichzeitig soll durch die Verwendung von virtuellen Maschinen sichergestellt werden, dass sämtliche Tests lokal auf einem einzigen Rechner ausgeführt werden können, welcher durch die bei Virtualisierung gegebene Isolation des Gast-Betriebssystems auch nicht gefährdet wird.
Für den geforderten Einsatzzweck stehen verschiedene virtuelle Maschinen zur Verfügung, welche eine Reihe absichtlich eingebauter Schwachstellen enthalten. Im Folgenden werden vier verschiedene virtuelle Maschinen vorgestellt, sowie teilweise eine Übersicht über die mit Exploit-Modulen von Metasploit ausnutzbaren Schwachstellen gegeben.

4.1.1. Metasploitable 2

Metasploitable 2 ist eine virtuelle Maschine, die auf Ubuntu 8.04 basiert, und neben einer Reihe absichtlich integrierter Schwachstellen und Backdoors besonders auch schwache Passwörter enthält, die mit verschiedenen Scanner-Modulen des Metasploit-Frameworks und geeigneten Wörterbüchern auf einfache Weise entdeckt und genutzt werden können.
Die Tabelle 4.1 zeigt eine Liste vorhandener, verwundbarer Dienste sowie die darauf anwendbaren Exploit-Module des Metasploit-Frameworks. Bei den drei folgenden Exploit-Listen ist zu beachten, dass diese gegebenenfalls nicht vollständig sind, da sie nur Einträge enthalten, für welche ein dazu passendes Exploit-Modul aus Metasploit erfolgreich manuell angewandt werden konnte.

4.1.2. Metasploitable 3

Auch der Nachfolger von Metasploitable 2, *Metasploitable 3*, wurde von Rapid7 entwickelt, aber existiert im Gegensatz zum Vorgänger in zwei verschiedenen Versionen: eine virtuelle Maschine basiert auf Ubuntu 14.04, die zweite auf Windows Server 2008. Beide wurden 2016 veröffentlicht und können nahezu automatisch mithilfe von Vagrant zu Oracle VirtualBox oder VMWare hinzugefügt und entsprechend konfiguriert werden.

[1] siehe auch `https://metasploit.help.rapid7.com/docs/metasploitable-2-exploitability-guide`, `https://stuffwithaurum.blog/2015/06/14/metasploitable-2-walkthrough-an-exploitation-guide`

Dienstname	Port	CVE(s)	passende Exploit-Module im MSF
FTP	21		exploit/unix/ftp/vsftpd_234_backdoor
PHP	80	CVE-2012-1823	exploit/multi/http/php_cgi_arg_injection
TWiki	80	CVE-2005-2877	exploit/unix/webapp/twiki_history
Samba	139	CVE-2007-2447	exploit/multi/samba/usermap_script
Java RMI	1099	CVE-2011-3556	exploit/multi/misc/java_rmi_server
DistCC	3632	CVE-2004-2687	exploit/unix/misc/distcc_exec
PostgreSQL	5432	CVE-2007-3280	exploit/linux/postgres/postgres_payload
IRC	6667	CVE-2010-2075	exploit/unix/irc/unreal_ircd_3281_backdoor
Tomcat	8180	CVE-2009-3843 CVE-2009-4188 CVE-2009-4189 CVE-2009-3548 CVE-2010-0557 CVE-2010-4094	exploit/multi/http/tomcat_mgr_deploy exploit/multi/http/tomcat_mgr_upload
Distributed Ruby (DRb)	8787		exploit/linux/misc/drb_remote_codeexec

Abbildung 4.1.: Liste vorhandener Schwachstellen in Metasploitable 2, für die mindestens ein funktionierender Exploit in Metasploit enthalten ist.[1]

Windows

Die auf Windows Server 2008 basierende Version von Metasploitable 3 weist eine ganze Reihe verschiedenster, ausnutzbarer Schwachstellen auf. Diese betreffen nicht nur die installierten Anwendungen, wie beispielsweise *GlassFish*, *Tomcat*, *Jenkins* oder *ElasticSearch*, sondern auch verschiedene Dienste des Betriebssystems, wie beispielsweise den in Abbildung 2.5 angegriffenen *SMB-Server*.
Eine Liste vorhandener Schwachstellen, welche mindestens ein darauf erfolgreich anwendbares Exploit-Modul in Metasploit besitzen, kann in Abbildung 4.2 eingesehen werden.
Neben diesen wurden in die virtuelle Maschine auch verschiedene verbreitete Angriffspunkte, wie zum Beispiel schwache Passwörter, integriert. Diese können beispielsweise zur Ausführung lokaler Exploits oder zum Zugriff über das Remote Desktop Protocol (RDP) genutzt werden. Auch Dienste, die keinerlei Authentifizierung verlangen, wie beispielsweise der vorhandene WebDAV-Server, wurden auf der Maschine präpariert.

Ubuntu

Etwas weniger mit Exploit-Modulen aus Metasploit ausnutzbare Schwachstellen finden sich hingegen auf der Ubuntu 14.04-Version von Metasploitable 3. Da bereits auf eine verwundbare Linux-VM zurückgegriffen werden kann und hierfür auch die öffentlich verfügbare Dokumentation deutlich weniger umfangreich als bei den anderen virtuellen Maschinen ist, wird von einer Verwendung der Ubuntu-Version abgesehen.
Wenn im Folgenden die Rede von Metasploitable 3 ist, so ist damit ausschließlich die Windows-Version gemeint.

[2] siehe auch `https://github.com/rapid7/metasploitable3/wiki/Vulnerabilities`

Dienstname	Port	CVE(s)	passende Exploit-Module im MSF
psexec	139 445		exploit/windows/smb/psexec exploit/windows/smb/psexec_psh
SMB	445	CVE-2017-0143 CVE-2017-0144 CVE-2017-0145 CVE-2017-0146 CVE-2017-0147 CVE-2017-0148	exploit/windows/smb/ms17_010_eternal-blue
JMX	1617	CVE-2015-2342	exploit/multi/misc/java_jmx_server
Ruby on Rails	3000	CVE-2015-3224	exploit/multi/http/rails_web_console_v2_code_exec
MySQL	3306		exploit/multi/mysql/mysql_udf_payload
GlassFish	4848 8080 8181	CVE-2011-0807	exploit/multi/http/glassfish_deployer
WinRM	5985		exploit/windows/winrm/winrm_script_exec
ManageEngine	8020	CVE-2015-8249	exploit/windows/http/manageengine_connectionid_write
Apache Struts	8282	CVE-2016-3087	exploit/multi/http/struts_dmi_rest_exec
Tomcat	8282	CVE-2009-3843 CVE-2009-4188 CVE-2009-4189 CVE-2009-3548 CVE-2010-0557 CVE-2010-4094	exploit/multi/http/tomcat_mgr_deploy exploit/multi/http/tomcat_mgr_upload
Apache Axis2	8282	CVE-2010-0219	exploit/multi/http/axis2_deployer
Jenkins	8484		exploit/multi/http/jenkins_script_console
Wordpress	8585	CVE-2016-1209	exploit/unix/webapp/wp_ninja_forms_unauthenticated_file_upload
PHPMyAdmin	8585	CVE-2013-3238	exploit/multi/http/phpmyadmin_preg_replace
ElasticSearch	9200	CVE-2014-3120	exploit/multi/elasticsearch/script_mvel_rce

Abbildung 4.2.: Liste vorhandener Schwachstellen in Metasploitable 3 (Windows Server 2008), für die mindestens ein passendes Exploit-Modul in Metasploit enthalten ist.[2]

4.1.3. Typhoon

Die *Typhoon* VM ist eine von der türkischen Consulting-Firma *Prisma CSI* veröffentlichte und ebenfalls auf Ubuntu 14.04 basierende virtuelle Maschine, welche Schwachstellen verschiedenster Arten aufweist. Einige hiervon sind samt der dafür verfügbaren Exploit-Module des Metasploit-Frameworks in Tabelle 4.3 aufgelistet.

Dienstname	Port	CVE(s)	passende Exploit-Module im MSF
Drupal	80	CVE-2018-7600	exploit/unix/webapp/drupal_drupalgeddon2
LotusCMS	80	CVE-2011-0518	exploit/multi/http/lcms_php_exec
Apache	80	CVE-2014-6271 CVE-2014-6278	exploit/multi/http/apache_mod_cgi_bash_env_exec
Samba	445	CVE-2017-7494	exploit/linux/samba/is_known_pipename
PostgreSQL	5432	CVE-2019-9193	exploit/multi/postgres/postgres_copy_from_program_cmd_exec
Tomcat	8080	CVE-2009-3843 CVE-2009-4188 CVE-2009-4189 CVE-2009-3548 CVE-2010-0557 CVE-2010-4094	exploit/multi/http/tomcat_mgr_upload

Abbildung 4.3.: Liste vorhandener Schwachstellen in der Typhoon VM (Ubuntu 14.04), auf die mindestens ein Exploit-Modul aus Metasploit erfolgreich angewandt werden kann.[3]

4.2. Evaluation

Anhand der in Kapitel 3 definierten Anforderungen an eine für den gegebenen Einsatzzweck geeignete Anwendung sollen hier die bereits in Abschnitt 2.7.1 auf Seite 22 vorgestellten Tools *Hail Mary* (Seite 22) und *AutoSploit* (Seite 22) genauer evaluiert werden. Die Evaluierung erfolgt in den im selbigen Kapitel definierten Ausführungsschritten. Für jede der betrachteten Anwendungen muss hierzu noch die jeweilige Definition einer erfolgreichen Exploit-Ausführung festgelegt werden.

4.2.1. AutoSploit

Die kontrovers diskutierte Anwendung *AutoSploit* könnte anhand der öffentlich verfügbaren Beschreibung ein geeignetes Tool für den hier geforderten Einsatzzweck sein.
AutoSploit ist eine offen verfügbare Kommandozeilen-basierte Python-Anwendung und verfügt über mehrere integrierte Wege zur Informationsgewinnung, zum Beispiel können potentielle Targets über mehrere Internet-of-Things-Suchmaschinen, wie *shodan.io*, *censys.io* oder *zoomeye.org*, gesammelt werden.
Anschließend können diese Targets anhand einer vorgegebenen, aber veränderbaren statischen Liste an Exploits angegriffen werden. Jeder Exploit-Vorgang wird hierbei genau protokolliert und kann so nach Ausführungsende genauer analysiert werden. AutoSploit führt jedoch keine Überprüfung der Eignung der einzelnen Exploits durch, sondern arbeitet völlig statisch mit der vorgegebenen Liste.
Da ein von AutoSploit generierter Report kein Attribut bezüglich des Erfolgs eines Exploits enthält, werden im Folgenden alle Exploits als erfolgreich gewertet, bei welchen der zugehörige Reporteintrag Daten im Feld „Successful log“ enthält.

```
root@kali-vm:~/AutoSploit# python2 autosploit.py -C autosploit 10.13.13.110 4567 -e
                                                                                
#--Author : Vector/NullArray
#--Twitter: @Real__Vector
#--Type   : Mass Exploiter
#--Version: 4.0
##########################################

[+] welcome to autosploit, give us a little bit while we configure
[i] checking your running platform
[i] checking for disabled services
[+] attempting to load API keys
[+] Shodan API token loaded from /root/AutoSploit/etc/tokens/shodan.key
[+] Censys API token loaded from /root/AutoSploit/etc/tokens/censys.key
[i] checking if there are multiple exploit files
[+] total of 3 exploit files discovered for use, select one:
1. 'windows_modules_msa3'
2. 'msa2_eval_modules'
3. 'linux_modules_typhoon'
root@autosploit# 2
[+] Launching exploits against 1 hosts:
[+] launching exploit 'exploit/unix/webapp/twiki_maketext' against host '10.13.13.104'
```

Abbildung 4.4.: Beispielhafte Ausführung von AutoSploit: Angriff von Target 10.13.13.104 (definiert in der Datei `hosts.txt`) mit der selbst erstellten Exploit-Liste `msa2_eval_modules`.

Exploit-Auswahl

Die Auswahl der Exploits geschieht bei AutoSploit nicht automatisch. Vielmehr liest AutoSploit nur eine vom Nutzer statisch festgelegte Liste an Exploits ein und verwendet diese. Genauere Einschränkungen bezüglich des Mindestrankings oder des Ziel-Betriebssystems muss der Benutzer bereits beim Erstellen dieser Datei vornehmen. Für AutoSploit stehen allgemein aber alle in Metasploit integrierten Module zur Verfügung.

Exploit-Ausführung

Sämtliche ausgewählten Exploits werden von AutoSploit rein automatisch mit den Vorgabewerten für alle obligatorischen Parameter ausgeführt. Eine Änderung findet hierbei nur für relevante Netzwerk-Adressen (wie zum Beispiel `LHOST` und `RHOST`) statt und ist für den Anwendungsbenutzer zur Laufzeit nicht möglich.
Auch der Payload wird hierbei nicht geändert, sondern die standardmäßige Reihenfolge, die vom Metasploit Framework vorgegeben wird, zur Payload-Auswahl genutzt.
Die Exploits werden streng sequentiell über einzelne Konsolenkommandos ausgeführt, eine sitzungsartige Kommunikation mit dem Metasploit-Framework wird hierbei nicht durchgeführt, da jeder Exploit-Aufruf eine neue Instanz der Metasploit Konsole startet.
Auch die mehrfache Ausführung eines Exploits ist bei AutoSploit ohne einen Neustart der Anwendung nicht möglich.

Automatische Erkennung der Auswirkungen und Bewertung

Die Erkennung der Auswirkungen und die Erfolgsbewertung findet bei AutoSploit in einem Schritt statt. Da das Tool über keinen Input bezüglich der Exploit-Intention verfügt, findet

nur eine ungenaue und unvollständige Bewertung des Ausführungserfolges statt.
Eine Quellcodeanalyse ergab, dass hierbei vier verschiedene Methoden zur Markierung eines Exploits als „erfolgreich“ verwendet werden: Ein Exploit wird in diesem Sinne als erfolgreich betrachtet, sobald in dessen Ausführungslog mindestens ein bestimmter Ausdruck enthalten ist. Diese statisch definierte Liste umfasst die Ausdrücke `[+]`, `Meterpreter`, `Session` und `Sending stage`. Da es sich hierbei zumindest teilweise um sehr allgemeine Schlüsselwörter handelt, ist die Rate an False-Positive-Fehlern entsprechend hoch. In Abbildung 4.5 ist die bei Ausführungsende angegebene Übersicht über die Ergebnisse zu sehen.

```
(msf)>> [-] 10.13.13.104: - Exploit failed: The following options failed to validate: RPORT.
(msf)>> [*] Exploit completed, but no session was created.
(msf)>> resource (/root/.autosploit_home/autosploit_out/2019-09-11_19h56m22s/10.13.13.104/explo
(msf)>>

[+] **********RESULTS**********
[+]     14 exploits run against 1 hosts.
[+]     10 exploit successful (Check report.csv to validate!).
[+]     4 exploit failed.
[+]     Exploit run saved to /root/.autosploit_home/autosploit_out/2019-09-11_19h56m22s
[+]     Report saved to /root/.autosploit_home/autosploit_out/2019-09-11_19h56m22s/report.csv
root@kali-vm:~/AutoSploit#
```

Abbildung 4.5.: Beispielhaftes Ende einer Ausführung von AutoSploit mit Übersicht über die Anzahl erfolgreich ausgeführter Exploits.

Reportgenerierung

AutoSploit erstellt nach jedem Durchlauf rein automatisch einen Bericht im CSV-Format, der neben verschiedenen Metadaten nur die Metasploit-Logs für die Ausführung der einzelnen Exploits enthält, aber keine Informationen zu den zugehörigen Schwachstellen oder zu den aus der Exploit-Ausführung resultierenden Auswirkungen auf das Target liefert. Die Logs werden hierbei in erfolgreiche Logs und fehlgeschlagene Logs unterschieden, sodass eine schnelle manuelle Überprüfung der von AutoSploit vorgenommenen Bewertung möglich ist. Eine Übersicht über alle Spalten in einem von AutoSploit erstellten Report befindet sich im Anhang in Abbildung D.1.

Anwendung auf die Testumgebung

Um eine sinnvolle und zeitlich angemessene Anwendung von AutoSploit auf die Testumgebung zu gewährleisten, ist es nötig, sicher zu stellen, dass AutoSploit eine adäquate Auswahl an Exploits verwendet.
Hierfür wird für die Anwendungsausführung im Folgenden eine entsprechende Liste von Exploits erstellt, die sowohl Betriebssystemspezifische als auch universelle Exploits beinhaltet. Die Default-Liste von AutoSploit wurde um die bekannten, funktionierenden Exploits erweitert, um so die Genauigkeit präzise bewerten zu können. Die Liste enthält Exploits für verschiedene Betriebssysteme, um mithilfe dieser, mit Sicherheit fehlschlagenden Exploits, Falsch-Positiv-Fehler einfach aufdecken zu können.
Allgemein zeigt sich, dass AutoSploit zwar bei einer beachtlichen Menge von Exploits mindestens einen Log-Eintrag als erfolgreich klassifiziert, diese Einschätzung allerdings eine hohe Fehlerrate aufweist. Dies wiederum liegt nicht nur an der sehr einfachen Methode der

Erfolgsbewertung, sondern auch an Fehlern im Programm. Beispielsweise wurde bei der Anwendung auf Metasploitable 3 mindestens ein Exploit für das Datenbank-Management-Tool *PhpMyAdmin* als erfolgreich markiert, obwohl für die Ausführung der (bei Webservern übliche) Standard-Port 80 ausgewählt wurde, unter welchem allerdings kein Webserver erreichbar ist. Weiterhin ersichtlich wird dies vor allem an der Tatsache, dass selbst Exploits, die für ein nicht-vorhandenes System konstruiert wurden, als erfolgreich markiert werden. Eine manuelle Ausführung mit denselben Parametern hingegen führt zu keinem Erfolg. Beispielsweise enthält der Exploit `exploit/windows/iis/iis_webdav_upload_asp` bei einer Anwendung gegen die Metasploitable 2-VM „erfolgreiche“ Log-Einträge, die eine erfolgreich aufgebaute Meterpreter-Session versprechen – obwohl der ausgeführte Exploit nur auf Windows-Hosts anwendbar ist.

Weiteres

AutoSploit kann aufgrund der hohen Verbreitung von Python auf allen gängigen Betriebssystemen ausgeführt werden.
Da bei AutoSploit keine direkte Interaktion mit dem Metasploit Framework stattfindet, sondern sämtliche Befehle einzeln als Konsolenkommandos ausgeführt werden, ist AutoSploit bei der Ausführung der in der Standard-Liste enthaltenen Exploits besonders im Vergleich zu Hail Mary auffällig langsam. Insofern ist es besonders ratsam, garantiert unpassende Exploits vor der Ausführung von der benutzten Exploit-Liste zu entfernen, was den Vorbereitungsaufwand eines Penetrationstests mit AutoSploit deutlich erhöht. Ein Schnelldurchlauf ist nicht möglich.
AutoSploit besitzt über die hier definierten Anforderungen hinaus auch die Möglichkeit, mehrere Systeme in einem Durchlauf anzugreifen. Die Möglichkeit, diese Remote Hosts davor mithilfe verschiedener Suchmaschinen aufzufinden, ist für diese Arbeit nicht von Relevanz und wurde folglich auch nicht getestet.

4.2.2. Hail Mary (db_autopwn)

Direkt aus der ehemaligen *db_autopwn*-Funktion des Metasploit Frameworks hervorgegangen ist die *Hail Mary*-Funktion von Armitage, die diese in mehreren Punkten verbessert und erweitert.
Jede Ausführung von Hail Mary setzt sich allgemein aus fünf einzelnen Schritten zusammen [Mud13b]. Zu Beginn werden für jeden verfügbaren Remote Host passende Exploits für jeden vorhandenen Service ausgewählt. Exploits, welche ein schlechteres Ranking als den vorgegebenen Mindestwert aufweisen, bleiben hierbei unberücksichtigt. Gleiches gilt für Exploits, welche für ein anderes Betriebssystem entwickelt wurden. Danach werden die übrigen Exploits nach Verlässlichkeit und Veröffentlichungsdatum absteigend sortiert, sodass die neuesten und verlässlichsten Exploits im nächsten Schritt als Erstes ausgeführt werden. Anschließend werden nach kurzer Wartezeit sämtliche dadurch erlangte Remote Sessions angezeigt und eine Interaktion mit ebendiesen ermöglicht.
Da sich Hail Mary stark auf die bei der Exploit-Ausführung erzeugten Remote Sessions konzentriert, ist im Folgenden die Definition eines erfolgreichen Exploits mit dem Erlangen einer Remote Session gleichzusetzen. Als erste Übersicht über die bei einer erfolgreichen Ausführung von Hail Mary erzielten Ergebnisse dient das Ende der Konsolenausgabe, wel-

ches in Abbildung 4.6 abgebildet ist.

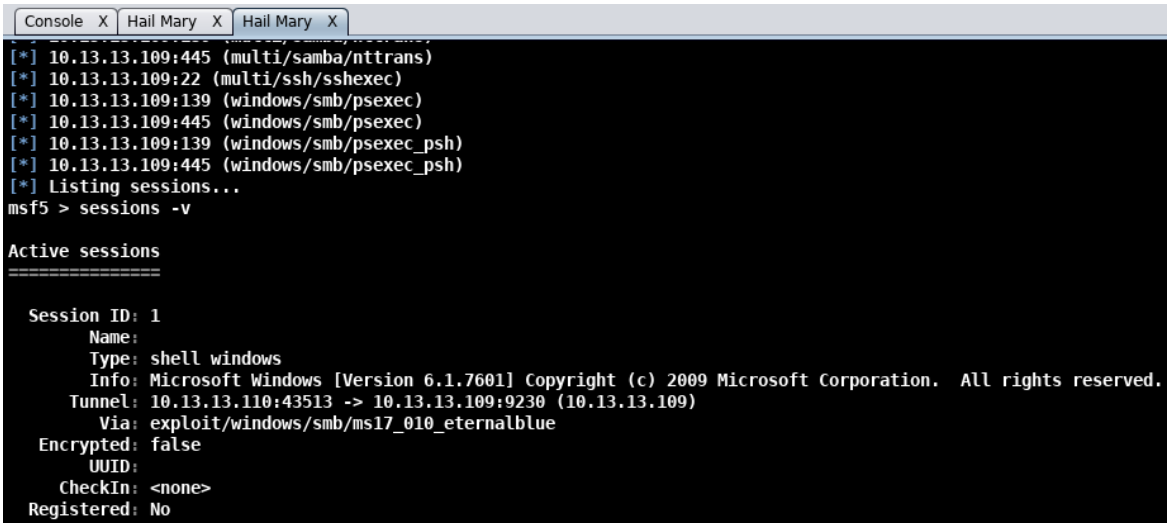

Abbildung 4.6.: Ende einer beispielhaften Ausführung von Hail Mary. Es wurde das Target 10.13.13.109 angegriffen und eine Remote Shell erlangt. Oben zu sehen sind die zuletzt ausgeführten Exploits, unten Details zur Session.

Exploit-Auswahl

Hail Mary führt eine automatische Exploit-Auswahl durch, die keine Benutzerinteraktion erfordert. Zum Auffinden passender Exploits werden diese auf Basis eines vorher vom Nutzer über die graphische Oberfläche gestarteten Service-Scans ausgewählt. Dabei werden alle im Metasploit Framework integrierten Exploits zu Rate gezogen und diese nach dem von Armitage detektierten Betriebssystem gefiltert. Ebenso kann in Armitage global ein Mindestranking für die Exploit-Auswahl festgelegt werden, welches anschließend auch für Hail Mary gilt.
Eine Analyse des Quellcodes ergibt, dass zur Exploit-Auswahl allerdings nur für jeden als offen erkannten Port überprüft wird, ob Exploit-Module in Metasploit existieren, die standardmäßig diesen Port als Zielport benutzen. Sobald eine Anwendung folglich auf einem anderen Port als üblich verfügbar ist, werden keine passenden Exploits hierfür mehr gefunden. Eine Abschätzung der Absicht eines Exploits findet nicht statt.

Exploit-Ausführung

Da Hail Mary keinen Zugriff auf einen Schwachstellenscan hat, werden nur die Exploits nach Verlässlichkeit sortiert und mit den Standardwerten für fast alle Parameter ausgeführt. Die Ausnahmen davon sind – analog zu AutoSploit – nur für die Ausführung kritische Parameter, wie zum Beispiel `LHOST` und `RHOST`. Für die Exploits wird jeweils der aus Sicht von Hail Mary beste verfügbare Payload benutzt, um eine Remote Session zu erzeugen.
Eine wiederholte Ausführung von Exploits ist hierbei nur über eine wiederholte Ausführung der gesamten Hail-Mary-Funktion möglich, eine Steuerung der Geschwindigkeit der Ausführung kann nicht vorgenommen werden.

Automatische Erkennung der Auswirkungen und Bewertung

Hail Mary kann den Erfolg eines Exploits nur über eine erfolgreich aufgebaute Session bewerten. Wurde von dem genutzten Payload keine Remote Session geöffnet, kann über den Exploit keine Aussage getroffen werden.
Werden weitergehende Informationen über das erfolgreich angegriffene Target benötigt, so muss die weitere Informationsakquise vom Benutzer selbst durchgeführt werden oder gegebenenfalls ein geeignetes anderes Modul zu Rate gezogen werden.
Im Gegensatz zu AutoSploit bietet Armitage aber die Möglichkeit der Interaktion mit von Hail Mary erzeugten Sessions über eine integrierte Metasploit-Kommandozeile.

Reportgenerierung

Hail Mary generiert nach der Ausführung keinen Report, wie es beispielsweise bei AutoSploit der Fall ist. Eine Rekonstruktion des Vorgehens ist sehr eingeschränkt nur anhand des Anwendungslogs möglich. Dieser enthält auch keine Logs für die einzelnen Exploit-Vorgänge, sondern nur eine Liste der ausgeführten Exploits und die erlangten Sessions sowie jeweils Informationen zu Letzteren.

Anwendung auf die Testumgebung

Eine Ausführung von Hail Mary ergibt eine deutlich geringere Anzahl an erfolgreich ausgeführten Exploits, was an der gegenüber AutoSploit stark eingeschränkten Erfolgsevaluation liegt. Für alle Tests wurde hierbei das Mindestranking zu Beginn auf die niedrigste Stufe („poor“) festgelegt.
Auf Metasploitable 2 konnte Hail Mary nur genau eine Session erlangen, die virtuelle Maschine mit Metasploitable 3 hingegen stürzte als Folge einer Exploit-Ausführung mehrfach, allerdings nicht immer, ab. Setzte man das Mindestranking auf die nächsthöhere Stufe („normal“), so blieb das Ziel-System in allen Fällen stabil und lieferte genau eine meterpreter-Session.
Bei einer Anwendung von Hail Mary auf die Typhoon VM wird ebenfalls nur eine Session geöffnet.

Weiteres

Aufgrund der Verwendung der Java Virtual Machine ist Hail Mary auf verschiedensten Betriebssystem ausführbar. Für Exploits stehen sämtliche von Metasploit angebotenen Ziel-Betriebssysteme zur Verfügung.
Hail Mary bietet gegenüber AutoSploit ein deutlich benutzerfreundlicheres Zeitverhalten: ein einzelner Durchlauf dauert in der Regel nur wenige Minuten. Außerdem ist über die GUI eine Fortschrittsanzeige möglich. Dies liegt nicht nur an der parallelen Ausführung der einzelnen Exploits, sondern auch an der Zeitbeschränkung: mitunter haben Exploits nur 30 Sekunden Zeit zur Ausführung. Während das für die meisten Exploits in der Regel ausreicht, kann dies besonders kritisch bei Exploits sein, die beispielsweise erst einen bestimmten Parameter (wie zum Beispiel eine Rücksprungadresse) per Brute Force erraten müssen. In solchen Fällen ist es daher ratsam, die Liste aller erlangten Sessions zu einem späteren Zeitpunkt noch einmal manuell zu überprüfen.

4.3. Probleme der evaluierten Anwendungen

Bei der vorangegangenen Evaluation der beiden Anwendungen haben sich jeweils durchaus ähnliche Probleme gezeigt, die AutoSploit und Hail Mary für den hier vorgestellten Use Case eher ungeeignet machen.
Beide Anwendungen passen ihren Ablauf nur sehr bedingt an die Ziel-Umgebung an, da das Target (im Falle von AutoSploit) entweder gar nicht oder nur eher oberflächlich (im Falle von Hail Mary) analysiert wird. Dies beginnt bei der statischen Exploit-Auswahl von AutoSploit und führt bis hin zur Verwendung von Standardwerten für die Exploit-Parameter.
Bei Letzterem werden von beiden Anwendungen sämtliche Exploits größtenteils mit den von Metasploit vorgegebenen Standardwerten ausgeführt, was die Wahrscheinlichkeit einer erfolgreichen Exploit-Ausführung, besonders in der Praxis, deutlich verringert.
Die Erkennung der Ausführung der jeweiligen Metasploit-Module ist sowohl bei AutoSploit als auch bei Hail Mary sehr eingeschränkt, beispielsweise fokussiert sich Hail Mary nur auf Remote Sessions.
Beiden Anwendungen gemeinsam ist eine mangelhafte und unvollständige Erkennung der Auswirkungen der Exploit-Ausführung auf dem Target – keines der beiden Tools analysiert mögliche Seiteneffekte.
Während eine Erfolgsbewertung durch eine oberflächliche Auswirkungserkennung deutlich erschwert wird, wird sie bei fehlenden Informationen über die jeweiligen Exploit-Intentionen unmöglich. Aus diesem Grund wird eine solche Bewertung im engeren Sinne auch von keiner der beiden getesteten Anwendungen durchgeführt, beide Tools bewerten den Erfolg im weiteren Sinne anhand ihrer eigenen Kriterien.
Allgemein wurde anhand der durchgeführten Analyse und Evaluation von Hail Mary und AutoSploit deutlich, dass beide Anwendungen für den gegebenen Einsatzzweck nur so eingeschränkt nutzbar sind.

4.4. Lösungsmöglichkeiten

Diese aufgezeigten Probleme sollen bei der zu entwickelnden Anwendung durch geeignete Maßnahmen gelöst werden.
Das Problem der nicht oder nicht ausreichend gründlich durchgeführten Target-Analyse wird durch einen davor erfolgten Schwachstellenscan behoben. Dessen Informationen können für eine präzisere Exploit-Ausführung und Erfolgsbewertung genutzt werden.
Außerdem können sie ebenso zur Auswahl der Werte für bestimmte Exploit-Parameter genutzt werden. Dem Nutzer muss trotzdem die Möglichkeit der Änderung der Parameter gegeben werden.
Mithilfe einer Überwachung der Ausführung des gewählten Moduls kann bereits während dessen Laufzeit eine besonders schnelle und genaue Erkennung der erfolgreichen Ausführung auf dem Target vorgenommen werden.
Eine präzise Detektion der Auswirkungen der Exploit-Ausführung kann nur mithilfe mehrerer Methoden erfolgen. Hierzu gehören in erster Linie Überprüfungen des Remote Hosts auf Veränderungen durch den ausgeführten Exploit.
Anhand einer vorhergehenden Exploit-Kategorisierung und einer umfassenden Erkennung der Auswirkungen kann eine präzise Erfolgsbewertung ermöglicht werden.

5. Implementierung

In diesem Kapitel soll nun mithilfe der erarbeiteten Anforderungen und den Problemen bestehender, vergleichbarer Tools die Anwendung **pyperpwn** („Python Penetration Testing autopwn“) prototypisch entwickelt werden. pyperpwn soll für den beschriebenen Anwendungszweck deutlich besser geeignet sein.
Hierfür muss allerdings zuerst eine passende Entwicklungsumgebung gefunden und das Anwendungsdesign entworfen werden.

5.1. Evaluation verfügbarer Entwicklungsumgebungen

Vor der Festlegung des spezifischen Anwendungsdesigns werden zuerst verschiedene Entwicklungsmöglichkeiten näher betrachtet und hinsichtlich verschiedener Kriterien bezüglich ihrer Eignung evaluiert.
Die Entwicklung einer komplett eigenständigen Anwendung wäre hier nicht zielführend, weswegen nun vier Möglichkeiten untersucht werden, eine Anwendung auf der Basis des Metasploit-Frameworks zu entwickeln, um damit nicht nur dessen bewährte Funktionalitäten nutzen, sondern auch in Zukunft von Erweiterungen des Frameworks um beispielsweise neue Module oder Funktionen profitieren zu können.

5.1.1. Metasploit Resource Script

Die erste Möglichkeit der Umsetzung der Anwendung ist die Entwicklung als *Metasploit Resource Script*. Diese Skripte können sowohl normale Metasploit-Befehle, als auch Ruby-Code mit Zugriff auf die Metasploit-Datenbank enthalten. Standardmäßig enthält das Metasploit Framework bereits über 20 vordefinierte Skripte, die überwiegend kleinere Aufgaben, wie beispielsweise das Aufräumen der Datenbank oder auch Login- oder Brute-Force-Funktionalitäten.
Besonders interessant ist in diesem Kontext das Autoexploit-Skript, welches unter dem Pfad `scripts/resources/autoexploit.rc` zu finden ist und ebenfalls automatisch Exploits basierend auf den Schwachstelleneinträgen in der Metasploit-Datenbank auswählen und ausführen kann.
Dies zeigt, dass der hier benötigte Funktionsumfang bezüglich des Exploit-Handlings und der -Ausführung unterstützt wird. Durch die besonders tiefe Integration in das Metasploit-Framework können Resource Skripte allerdings auch Funktionen nutzen, welche nicht durch eine API öffentlich zugänglich sind.
Resource Skripte haben jedoch den Nachteil, dass sie entweder innerhalb von Metasploit (oder auch Armitage) oder zusammen mit Metasploit ausgeführt werden müssen. Eine aktive, lokale Metasploit-Instanz wird somit in jedem Fall benötigt.

5.1.2. msfconsole Plugin

Eine weitere Möglichkeit der Entwicklung einer sehr eng mit dem Metasploit Framework verbundenen Anwendung ist die Implementierung eines *msfconsole Plugin*s, welches direkt über die API mit Metasploit kommuniziert. Über ein solches Plugin kann effektiv ein neues Kommando zu Metasploit hinzugefügt werden[1]. Beispielsweise ist das eigentlich aus dem Metasploit-Funktionsumfang entfernte `db_autopwn`-Kommando jetzt als Plugin verfügbar. Eine Standard-Installation des Metasploit-Frameworks umfasst bereits mehrere Dutzend Plugins, welche unter anderem erweiterte Integrationsfunktionalitäten bieten, um beispielsweise Schwachstellenscanner direkt aus Metasploit heraus ansteuern und deren Ergebnisse in die Metasploit-Datenbank übernehmen zu können.
Auch Plugins haben, wie Resource Skripte, Zugriff auf die Metasploit Datenbank und müssen in Ruby geschrieben werden, sind aber als eigenständiger und umfangreicher als die Skripte zu betrachten. Plugins sind außerdem direkt in das Event-System von Metasploit integriert. Für ihre Ausführung ist aber, wie bei den Resource Skripten, eine lokale Instanz des Metasploit-Frameworks nötig.
Allgemein sind Plugins der vom Metasploit-Hersteller *Rapid7* empfohlene Weg der Erweiterung von Metasploit und des Teilens solcher Erweiterungen[2].

5.1.3. Cortana Script

Allerdings muss für den hier gegebenen Einsatzzweck nicht zwingend das Metasploit-Framework erweitert werden, gegebenenfalls ist die Wahl einer unabhängigeren Umgebung eine sinnvollere Möglichkeit.
Eine hierzu passende Option bietet sich in Form der Entwicklung eines *Cortana Skripts* an. Cortana Skripte werden in der *Sleep Scripting Language*[3] entwickelt, welche vom Entwickler von Armitage veröffentlicht wurde und Ähnlichkeiten zu *Perl* aufweist. Cortana Skripte können entweder völlig selbstständig als jar-Datei oder direkt in Armitage ausgeführt werden. Cortana wurde ausdrücklich zur Automatisierung des Metasploit-Frameworks entwickelt – als Möglichkeit zur Erweiterung von Armitage oder zur Simulation eines weiteren Penetrationstesters.
Der angebotene, große Funktionsumfang reicht von den benötigten Basis-Interaktionen mit Metasploit bis hin zur automatischen Synchronisierung der Metasploit-Datenbank und der Erstellung einer graphischen Oberfläche [Mud13a].
Trotz der zahlreichen Funktionalitäten wird von einer Verwendung aufgrund der geringen Verbreitung sowie der letztmaligen Aktualisierung im Jahre 2012 abgesehen. Gegen die Verwendung von Cortana sprechen auch eine im Vergleich mit Python deutlich schwierigere Erlernbarkeit, weniger verfügbare Referenzen und Dokumentationen sowie das Fehlen der Integration externer Bibliotheken zur Erweiterung des Funktionsumfangs.

5.1.4. MSFRPC und Python

Ebenso bietet sich die Möglichkeit der Nutzung der *Metasploit Framework Remote Procedure Call (MSFRPC)*-Schnittstelle. Der RPC-Server kann entweder als Daemon (*msfrpcd*) direkt über die Kommandozeile oder als Plugin (*msgrpc*) mit einer *MessagePack*-Schnittstelle aus

[1] `https://www.offensive-security.com/metasploit-unleashed/mixins-plugins/`
[2] `https://blog.rapid7.com/2011/12/08/six-ways-to-automate-metasploit/`
[3] `https://github.com/rsmudge/sleep`

der Metasploit-Konsole heraus gestartet werden. Prinzipiell ist die Kommunikation mit dieser Schnittstelle mithilfe jeder Programmiersprache möglich, welche HTTPS und MessagePack, ein Format für schnellen Datenaustausch[4], unterstützt.
Die Steuerung des Metasploit-Frameworks durch eine Python-Anwendung ist dank einer offen verfügbaren Bibliothek wie beispielsweise *pymetasploit*[5] einfach möglich. Allgemein bietet die Verwendung von Python die Möglichkeit der Entwicklung einer komplett eigenständigen Anwendung, welche zwar nach wie vor auf Metasploit basiert, aber dank der Verwendung der MSFRPC-Schnittstelle auch auf eine remote ausgeführte Metasploit-Instanz zugreifen kann. Dadurch kann die Anwendung auch auf Plattformen ausgeführt werden, für welche das Metasploit-Framework nicht verfügbar ist.
Eine Implementierung außerhalb der Metasploit-Umgebung ermöglicht ebenso auch flexiblere Datenstrukturen, unabhängig von der Metasploit-Datenbank.
Vom Funktionsumfang her unterstützt pymetasploit die Exploit-Auswahl, -Konfiguration und -Ausführung sowie die direkte Interaktion mit erzeugten Sessions und der Metasploit-Konsole und damit alle hier benötigten Funktionalitäten.

5.1.5. Auswahl

Da sich die Kombination aus Python und einer Bibliothek zur Ansteuerung der MSFRPC-API gut für schnelles Prototyping einer Anwendung eignet, soll im Folgenden diese Umgebung zur Entwicklung benutzt werden. Zudem ist damit sichergestellt, dass sich pyperpwn möglichst Plattformunabhängig sowie eigenständig ausführen lässt und darüber hinaus einfach in ein bereits bestehendes Toolset für Penetrationstests integriert werden kann. Letzteres ist besonders von Vorteil, da pyperpwn nur einen Teil des Pentetrationstest-Prozesses teilweise automatisiert und durch weitere Integration ein noch höherer Automatisierungsgrad erzielt werden kann.
Die Verwendung der im Penetration-Testing-Umfeld sehr verbreiteten Entwicklungssprache Python empfiehlt sich nicht nur wegen der einfachen Erlernbarkeit und der guten Codelesbarkeit, sondern auch aufgrund der immensen Anzahl an dafür verfügbaren Bibliotheken, die die Integration weiterer für Penetrationstests nützlicher Funktionalitäten deutlich vereinfachen.

5.2. Anwendungsdesign

Nach der Wahl der passenden Umgebung für die Entwicklung soll nun das Design der Anwendung festgelegt werden. Zuerst müssen hierfür der erwartete Input und die benötigten Anwendungsparameter, sowie das Output-Format des Tools festgelegt werden. Anschließend kann der Aufbau entworfen und skizziert werden.

5.2.1. Input

Nach dem Anwendungsstart müssen verschiedene Input-Werte und -Formate entgegengenommen werden, da die darin enthaltenen Informationen die Steuerung der Anwendung ermöglichen oder schlichtweg außerhalb der Funktionalität der Anwendung liegen.

[4] `https://msgpack.org/index.html`
[5] `https://github.com/DanMcInerney/pymetasploit3`

Zu Beginn soll die *Ausgabe eines weit verbreiteten Schwachstellenscanners* in dessen Ausgabeformat eingelesen werden können. Aufgrund der hohen Verbreitung wird hierfür der Output im CSV-Format bevorzugt. Welche Spalten für eine korrekte und vollständige Ausführung des Tools hierfür mindestens benötigt werden, ist in Anhang A tabellarisch dargestellt. Durch die Wahl universeller Datenfelder und eines verbreiteten Dateiformats ist sichergestellt, dass Reports verschiedenster Schwachstellenscanner einfach in das erwartete Format konvertiert werden können. Bei einem von OpenVAS erstellten Report müssen beispielsweise nur einzelne Spalten umbenannt werden, bei einem Nessus-Report müssen manuell wenige Spalten mit Titel, aber ohne Werte, hinzugefügt werden, da Nessus standardmäßig im CSV-Report weniger Informationen zur Verfügung stellt.
Ebenfalls benötigt wird eine *Kategorisierung der Exploit-Intention*, wie in Abschnitt 2.7.2 vorgestellt, um den Erfolg der Exploits angemessen bewerten zu können. Hierfür ist eine Einteilung in die drei genannten Klassen (intrusiv-destruktiv, intrusiv-harmlos, nicht-intrusiv) ausreichend. Auch diese Kategorisierung muss im CSV-Format zur Verfügung gestellt werden, um anschließend eingelesen zu werden.
Der Nutzer soll außerdem das Verhalten der Anwendung über *sechs Parameter* beeinflussen können: jeweils drei zur Einschränkung der Exploit-Auswahl und drei zur Beeinflussung des Zeitverhaltens der Anwendung.

5.2.2. Parameter

Alle für den Anwendungsstart unbedingt benötigten, obligatorischen Parameter sind in Tabelle 5.1 aufgelistet. Sie umfassen nur Optionen zur Kommunikation mit dem Metasploit-Framework über MSFRPCD. Ohne diese Werte kann keine Verbindung zum Framework hergestellt werden und somit auch keine Exploits gesucht und ausgeführt werden, weswegen ein Anwendungsabbruch erfolgt.

Flag	Langform	Format	Beschreibung
-a	--msfhost	nnn.nnn.nnn.nnn	Die IPv4-Adresse der MSFRPCD-Instanz
-P	--port	nnnn	Von MSFRPCD genutzter Port
-u	--user	*String*	Benutzername zur Verbindung mit MSFRPCD
-p	--password	*String*	Passwort zur Verbindung mit MSFRPCD

Abbildung 5.1.: Liste aller für pyperpwn obligatorischen Parameter

Die Tabelle 5.2 zeigt alle für pyperpwn verfügbaren optionalen Parameter. Die Anwendung verfügt über eine Vielzahl optionaler Parameter, da diese, falls kein Wert angegeben wurde, entweder vom Wizard abgefragt oder durch die Standardwerte ersetzt werden.

5.2.3. Output

Der Anhang C zeigt die Struktur des von der Anwendung generierten Reports.
Dieser wird in jedem Fall, also auch bei einem vorzeitigen Anwendungsabbruch erstellt.

Flag	Langform	Format	Beschreibung
-h	--help		Zeigt den Hilfetext an; führt die Anwendung nicht aus
-v	--vulns	<Pfad>	Der Pfad zum Output eines Schwachstellenscanners
-c	--class	<Pfad>	Der Pfad zur Datei der Exploit-Kategorisierung
-x	--export	<Pfad>	Der Pfad für den Report der Anwendung
-t	--target	nnn.nnn.nnn.nnn	Die IPv4-Adresse des zu testenden Hosts
-l	--lhost	nnn.nnn.nnn.nnn	Die IPv4-Adresse des ausführenden Systems
-w	--lport	nnnn	Nummer des für Metasploit-Handler genutzten Ports
-r	--rank	∈ [manual, low, average, normal, good, great, excellent]	Das für die Exploit-Auswahl geltende Mindestranking
-o	--os	∈ [linux, windows]	Das für die Exploit-Auswahl relevante Ziel-Betriebssystem. Universelle Exploits werden standardmäßig eingeschlossen.
	--no-multi		Keine hinsichtlich des Betriebssystems universellen Exploits benutzen.
-xs	--execspeed	∈ [1, 2, 3]	Die für die Ausführung der Exploits anzuwendende Geschwindigkeit (je höher, desto schneller)
-es	--evalspeed	∈ [1, 2, 3]	Die für die Erkennung der Auswirkungen anzuwendende Geschwindigkeit (je höher, desto schneller)
-e			Exploits auch wirklich ausführen
-XX	--express		Anwendung mit möglichst wenig Rückfragen ausführen

Abbildung 5.2.: Liste aller für pyperpwn verfügbaren optionalen Parameter

5.2.4. Aufbau

Um auf mögliche, zukünftige Änderungen gut vorbereitet zu sein, ist ein modularer Aufbau der Anwendung wichtig. Im Folgenden soll der Aufbau von pyperpwn aus interner und auch externer Perspektive dargestellt und die Funktion wichtiger verwendeter Abhängigkeiten aufgezeigt werden.

Intern

Der interne Aufbau von pyperpwn ist gekennzeichnet durch die Aufteilung in mehrere Python-Pakete. Die Organisation der einzelnen Module in Paketen dient der besseren Kapselung und zusammen mit der Definition eines universellen Datenformates der einfachen Austauschbarkeit der Input-/Output-Komponenten im Besonderen. Falls in Zukunft beispielsweise auch Reports in einem anderen Datenformat als CSV eingelesen werden sollen,

so muss lediglich eine neue Klasse hierfür implementiert werden, der Rest kann unverändert bleiben.

Die Anwendung umfasst insgesamt fünf separate Pakete:

- **core**: Beinhaltet die eigentliche Anwendungslogik, wie beispielsweise die Exploit-Auswahl und die Erfolgsbewertung.
- **entities**: Enthält hauptsächlich die intern verwendeten Datenstrukturen sowie hierfür benötigte Konvertierungen.
- **inout**: Enthält alle wichtigen Module zur Steuerung der In- und Output-Prozesse, beispielsweise für das Einlesen und Schreiben des Reports sowie die Steuerung der Anwendung über die Kommandozeile.
- **msf**: Ist alleine verantwortlich für die Kommunikation mit dem Metasploit Framework, übernimmt die nötige Fehlerbehandlung und Kapselung der jeweiligen Funktionen.
- **db**: Zuständig für das Einfügen und Abrufen von Objekten in und aus der anwendungseigenen Datenbank.

Extern

Die Abbildung 5.3 zeigt schematisch alle mit der Anwendung verbundenen Komponenten, die wichtigsten Interaktionen und eine Einordnung der entwickelten Anwendung in die Systemlandschaft.

pymetasploit3

Wie auch in Abbildung 5.3 dargestellt, wird die pymetasploit-Bibliothek zur Kommunikation mit der MSFRPC-API benutzt.
Grundsätzlich bestehen zur Ausführung einer Aktion in Metasploit mit pymetasploit zwei verschiedene Möglichkeiten: Metasploit-Funktionen können über bibliothekseigene Funktionen aufgerufen, welche eine Funktion des Metasploit Frameworks abstrahieren. Diese Funktionen bieten den Vorteil, dass sie in der Regel die zugehörige Fehlerbehandlung und das Parsen der Rückgabewerte bereits übernehmen, sodass diese in einem einfach zu verwendenden Format vorliegen. Außerdem ist dadurch oft eine einfache Integration eines Aufrufs einer asynchronen Funktion in die Anwendung in Form eines synchronen, aber blockierenden, Funktionsaufrufes möglich. Ist für die gewünschte Funktion keine solche Methode im Framework vorhanden, bietet sich auch die Möglichkeit der direkten Ausführung des gewünschten Kommandos auf der Konsole des Metasploit-Frameworks. Das Parsen der Rückgabewerte und Handling des gegebenenfalls asynchronen Aufrufs muss in diesem Fall selbst übernommen werden.
Da pymetasploit3 jedoch alle für diese Anwendung benötigten Funktionalitäten abstrahiert, wurde letztere Option ausschließlich zu Entwicklungszwecken benutzt und ist nicht in der finalen Implementierung vorzufinden.
Bei der Entwicklung der Exploit-Ausführung wurden mehrere Fehler im Code von pymetasploit3 gefunden, welche eine erfolgreiche Exploit-Ausführung, das Empfangen des Outputs

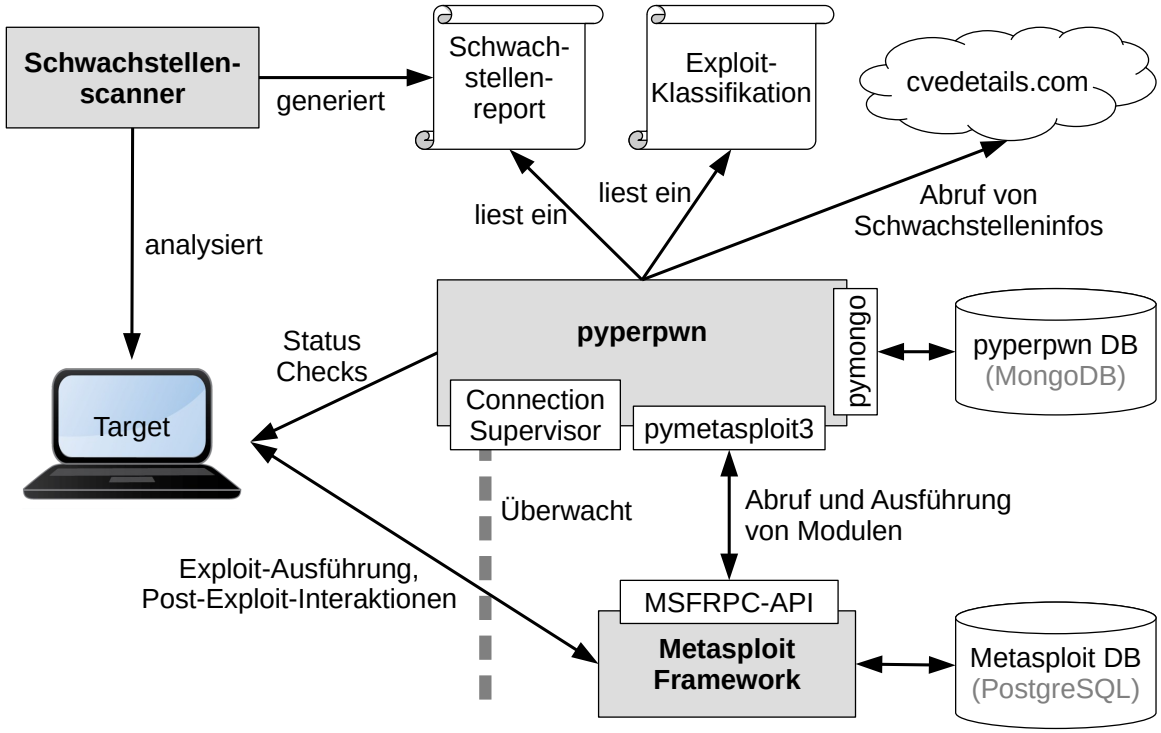

Abbildung 5.3.: Übersicht über die Systemarchitektur

des ausgeführten Exploits sowie das Setzen der Payload-Parameter vor Ausführung unter bestimmten Umständen verhinderten. Nach selbstständiger Entwicklung einer besseren Lösung, mehrfacher Rücksprache mit dem Entwickler und Öffnen mehrerer Pull Requests auf GitHub konnte dieses Fehlverhalten inzwischen beseitigt werden[6]. Auf Nachfrage beim zuständigen Maintainer wurde von diesem die Version 1.0.2 von pymetasploit3 veröffentlicht, sodass die eingereichten Verbesserungen nun direkt aus dem *Python Package Index (PyPI)* mittels *pip* installiert werden können[7].

pymongo und MongoDB

Zur persistenten Datenspeicherung ist pyperpwn an eine eigene Datenbank angebunden. Hierbei ist die Wahl aus mehreren Gründen auf eine NoSQL-Datenbank, konkret auf *MongoDB*, gefallen. Besonders geeignet sind solche Dokumenten-basierte Datenbanken für Rapid Prototyping, da für das Setup einer solchen Datenbank kein konkretes Datenschema im Voraus definiert werden muss. Die einzelnen *Dokumente* (entspricht einem Datensatz in einer relationalen Datenbank) werden in sogenannten *Collections* (analog zu einer Tabelle) organisiert. Die Dokumente wiederum werden als JSON-Objekt in der Datenbank gespeichert. NoSQL-Datenbanken erfreuen sich aufgrund der großen Flexibilität und der einfachen Handhabung großer Beliebtheit. Relationale Datenbanken hingegen garantieren durch die strikte Einhaltung des Datenschemas ein gewisses Maß an Datenkonsistenz, was für diesen Anwendungsfall allerdings zu vernachlässigen ist, da es sich hier nur um relativ kurzlebige Daten

[6] `https://github.com/DanMcInerney/pymetasploit3/pulls`; Issues Nr. 9, 14, 16; Pull Requests 10, 15, 17
[7] `https://pypi.org/project/pymetasploit3/1.0.2/`

handelt.
Die Anbindung der Datenbank geschieht hier über die *pymongo*-Bibliothek, welche eine einfache Integration sämtlicher Datenbankoperationen in die Python-Anwendung ermöglicht.
Hier wird die Datenbank konkret dazu genutzt, verschiedene Entitäten im anwendungseigenen Format zu speichern, um diese anschließend schnell abrufen und durchsuchen zu können. Beispielsweise wird bei Anwendungsstart eine kompakte Repräsentation eines jeden Exploit-Moduls in Metasploit in der Datenbank gespeichert, um später schnell und mittels eines synchronen Aufrufs beispielsweise anhand des Namens passende Exploits suchen zu können. Ebenso werden die aus dem Internet abgerufenen Schwachstellendetails in der Datenbank zwischengespeichert, um nicht bei jedem Abruf erneut eine deutlich langsamere HTTP-Anfrage stellen zu müssen. Anhand der Datenbank kann auch sichergestellt werden, dass eine Ausführung von pyperpwn zu jedem Zeitpunkt ohne Datenverlust unterbrochen werden und später wieder fortgesetzt werden kann.

5.3. Umsetzung der Anforderungen

Der gewählte Aufbau muss letztendlich jedoch der Umsetzung der benötigten Anforderungen dienen. Wie das genau in der finalen Version der Implementierung, welche sich auch auf der beigelegten DVD befindet, erreicht wurde, wird im Folgenden aufgezeigt.

Exploit-Auswahl

Alle eingelesenen Schwachstellen werden zu Beginn mithilfe des zugeordneten CVSS-Scores absteigend nach Schweregrad sortiert.
Die für eine Schwachstelle jeweils geeigneten Exploits werden anschließend unter Benutzung der eigenen Datenbankanbindung und des bei Anwendungsstart eingerichteten Exploit-Caches anhand mehrerer Kriterien ausgewählt. Die Datenbank wird dabei nach Exploits durchsucht, deren CVE- oder BID-Referenz mit der des Eintrages im Report des Schwachstellenscanners übereinstimmt. Außerdem wurde ein Index auf dem Namensattribut der Exploits definiert, sodass auch bei einer großen Anzahl an Dokumenten in der Collection schnell nach einem passenden Exploit mit ähnlichem Namen gesucht werden kann. Dies ist insbesondere dann hilfreich, wenn der betreffende Exploit oder das zugehörige Scanner-Modul nicht über die für ein eindeutiges Matching benötigten Referenznummern verfügt. Sollte bei der Namenssuche eine Vielzahl von Exploits einen ähnlichen Namen aufweisen, so wird nur eine zuvor festgelegte Anzahl an Exploits mit dem höchsten Score berücksichtigt.
Die jeweils gefundenen Exploit-Module werden vor der Ausführung anhand des Rank-Attributs von Metasploit nach der Verlässlichkeit absteigend sortiert.

Exploit-Ausführung

Um die für die Ausführung unbedingt benötigten Werte der Modul-Parameter korrekt zuzuweisen, wurden verschiedene Maßnahmen in der Implementierung umgesetzt.
Die obligatorischen Parameter `RHOST`/`RHOSTS`, `LHOST`, `RPORT` und `LPORT` können bereits anhand der bei Anwendungsstart angegebenen Parameter beziehungsweise mithilfe des Reports des Schwachstellenscanners gesetzt werden.
Anschließend werden die für den jeweiligen Exploit verfügbaren Parameter mit einer statischen Liste an Parameternamen verglichen. Diese Liste enthält Einträge für Parameter, die

häufig eine korrekte Exploit-Ausführung verhindern, falls hierfür die Standardwerte benutzt werden, wie beispielsweise `URI` oder `TARGETURI`. Wurde festgestellt, dass für den Exploit Parameter aus dieser Liste verfügbar sind, wird dem Nutzer die Möglichkeit gegeben, deren Werte zu ändern. Falls anschließend, vor der Ausführung, für mindestens einen obligatorischen Parameter noch kein Wert zugewiesen ist, so wird eine Warnung ausgegeben.
Die darauf folgende Exploit-Ausführung erfolgt über eine synchrone, blockierende Methode von pymetasploit3, welche nach dem Terminieren den Output der Ausführung zurück liefert, sodass dieser später analysiert und in den Report integriert werden kann.

Erkennung der Ausführung

Zur Erkennung der Exploit-Ausführung werden mehrere Methoden verwendet, um eine möglichst sicherere Aussage treffen zu können.
Ein intrusiver Exploit ohne destruktive Absicht wurde definitiv ausgeführt, wenn bei reverse und staged Payloads ein Verbindungsversuch zurück zum ausführenden System erfolgt. Gleiches gilt, sobald bei einem bind Payload ein erfolgreicher Verbindungsaufbau vom Host zum Target möglich ist. Ein intrusiv-destruktiver Exploit hingegen kann auch als erfolgreich angesehen werden, wenn der angegriffene Dienst nicht mehr verfügbar ist.
Um anhand der Funktion des Payloads den Erfolg des Exploits bewerten zu können, ist eine Unterscheidung der verschiedenen Payload-Typen nötig. Grundsätzlich unterscheiden sich die einzelnen Payload-Typen in der Anzahl und den Richtungen der aufgebauten Verbindungen.
Während folgender Schritte des Exploit-Vorganges werden üblicherweise TCP-Verbindungen aufgebaut:

- **Exploit-Übermittlung**: Die erste aufgebaute Verbindung geht in jedem Fall vom Host (dem Metasploit-ausführenden Rechner) zum Target (dem zu überprüfenden Rechner) und dient der Übertragung des Exploits. Die darauf folgenden Verbindungen sind Payload-spezifisch.

- **gegebenenfalls Laden der zweiten Payload-Stage**: Handelt es sich um einen *staged* Payload, so wird eine Verbindung vom Target zurück zum Host zwecks der Auslieferung der zweiten Payload-Stage aufgebaut. Bei *single*-Payloads entfällt dieser Schritt, da diese bereits zusammen mit dem Exploit vollständig ausgeliefert werden.

- **Verbindung zur Remote Shell**: Im letzten Schritt wird nun eine Verbindung zwischen dem Host und dem ausgeführten Payload hergestellt. Handelt es sich um einen *bind* Payload, wird diese vom Host initiiert, indem sich dieser auf den auf dem Target geöffneten Port verbindet. Bei einem *reverse*-Payload stellt der Host auf einem vorher festgelegten Port einen Payload-Handler bereit, welcher auf Verbindungsanfragen vom Target wartet.

Ein beliebtes Linux-Tool, um den TCP/IP-Traffic zu protokollieren, ist *tcpdump*. Folgende Listings zeigen jeweils den relevantesten Teil des Outputs von tcpdump: In Abbildung 5.4 wird der Ablauf bei Verwendung eines single bind Payloads dargestellt, während Listing 5.5 selbiges für einen staged reverse Payload darstellt. In beiden Fällen ist das Target über die IP-Adresse `172.28.128.13` zu erreichen, während `muel` den angreifenden Rechner bezeichnet. Zu beachten sind die sich ändernden Verbindungsrichtungen und Portnummern (hier teilweise durch den Dienstnamen ersetzt) sowie die einzelnen jeweils gesetzten TCP-Flags

('S' entspricht einem `SYN`, '.' einem `ACK` und 'P' einem `PUSH`).

```
1  IP muel.40649 > 172.28.128.13.distcc: Flags [S], seq 1210965050, win 29200
2  IP 172.28.128.13.distcc > muel.40649: Flags [S.], seq 780666887, ack 1210965051, win 5792
3  IP muel.40649 > 172.28.128.13.distcc: Flags [.], ack 1, win 229
4  IP muel.40649 > 172.28.128.13.distcc: Flags [P.], seq 1:279, ack 1, win 229, length 278
5  IP 172.28.128.13.distcc > muel.40649: Flags [.], ack 279, win 54
6  IP muel.40649 > 172.28.128.13.distcc: Flags [P.], seq 279:302, ack 1, win 229, length 23
7  IP 172.28.128.13.distcc > muel.40649: Flags [.], ack 302, win 54
8  ...
9  IP muel.40273 > 172.28.128.13.9999: Flags [S], seq 2076525657, win 29200
10 IP 172.28.128.13.9999 > muel.40273: Flags [S.], seq 781283357, ack 2076525658, win 5792
11 IP muel.40273 > 172.28.128.13.9999: Flags [.], ack 1, win 229
```

Abbildung 5.4.: Gekürzter Output von tcpdump während der Exploit-Ausführung bei Verwendung eines single bind Payloads (hier `cmd/unix/bind_ruby`).

In Abbildung 5.4 wurde eine Schwachstelle im *DistCC Daemon* von Metasploitable 2 mit dem Exploit `exploit/unix/misc/distcc_exec` ausgenutzt. Die Zeilen 1-3 zeigen hierbei die Verbindung vom Host zum Target zur Übertragung des Payloads (in den Zeilen 4-8) sowie eine weitere vom Host zum Target (Zeilen 9-10) zum Aufbau der Remote Session. Der auf dem Target für die Remote Session benutzte Port (in diesem Fall 9999) muss vor der Ausführung explizit (in Form des Parameters `LHOST`) gesetzt werden.

```
1  IP muel.35957 > 172.28.128.13.rmiregistry: Flags [S], seq 2549384642, win 29200
2  IP 172.28.128.13.rmiregistry > muel.35957: Flags [S.], seq 1621102601, ack 2549384643, win 5792
3  IP muel.35957 > 172.28.128.13.rmiregistry: Flags [.], ack 1, win 229,
4  IP muel.35957 > 172.28.128.13.rmiregistry: Flags [P.], seq 1:14, ack 1, win 229, length 13
5  IP 172.28.128.13.rmiregistry > muel.35957: Flags [.], ack 14, win 46
6  IP 172.28.128.13.rmiregistry > muel.35957: Flags [P.], seq 1:17, ack 14, win 46, length 16
7  ...
8  IP 172.28.128.13.53827 > muel.http-alt: Flags [S], seq 1638118692, win 5840
9  IP muel.http-alt > 172.28.128.13.53827: Flags [S.], seq 2580595145, ack 1638118693, win 28960
10 IP 172.28.128.13.53827 > muel.http-alt: Flags [.], ack 1, win 46
11 IP 172.28.128.13.53827 > muel.http-alt: Flags [P.], seq 1:245, ack 1, win 46, length 244:
12         HTTP: GET /TH1NouuFIAGCOMB/YiyV.jar HTTP/1.1
13 IP muel.http-alt > 172.28.128.13.53827: Flags [.], ack 245, win 235
14 IP muel.http-alt > 172.28.128.13.53827: Flags [.], seq 1:1449, ack 245, win 235, length 1448:
15         HTTP: HTTP/1.1 200 OK
16 IP muel.http-alt > 172.28.128.13.53827: Flags [.], seq 1449:2897, ack 245, win 235, length 1448:
17         HTTP
18 ...
19 IP 172.28.128.13.37419 > muel.4444: Flags [S], seq 1628625323, win 5840
20 IP muel.4444 > 172.28.128.13.37419: Flags [S.], seq 808646792, ack 1628625324, win 28960
21 IP 172.28.128.13.37419 > muel.4444: Flags [.], ack 1, win 46
```

Abbildung 5.5.: Gekürzter Output von tcpdump während der Exploit-Ausführung bei Verwendung eines staged reverse Payloads (in diesem Fall `java/meterpreter/reverse_tcp`).

Das zweite Listing (Abbildung 5.5) zeigt den Ausschnitt einer Exploit-Ausführung, bei der der staged reverse Payload `java/meterpreter/reverse_tcp` zur Anwendung kam. Ausgenutzt wurde in diesem Fall eine Schwachstelle im Java RMI Server von Metasploitable 2, weswegen auch ein für Java entwickelter Payload zum Einsatz kam (siehe beispielsweise Da-

teiendung in Zeile 12). Die Zeilen 1-3 zeigen den Verbindungsaufbau des Hosts zum Target zur Übermittlung des Exploits, was in den Zeilen 4-7 geschieht. Danach (in den Zeilen 8-10) ist der Verbindungsaufbau des Targets zum lokalen HTTP-Server des Hosts zu sehen, welcher zum Nachladen der zweiten Stage des Payloads (die per HTTP angefragte jar-Datei) durchgeführt wird. In den Zeilen 11-18 wird die Stage schließlich übertragen. Sobald diese nachgeladen ist, versucht sie eine Verbindung zurück zur Maschine des Angreifers herzustellen. Der dazu gehörende TCP-Handshake ist in den Zeilen 19-21 zu erkennen. Zu beachten ist hierbei auch, dass diese letzte Anfrage von einem anderen Port als zuvor geschieht, da der erste Port nur vom Payload zum Nachladen der Stage benutzt wurde.

Durch die Integration von tcpdump in die entwickelte Anwendung können sämtliche zwischen dem Host und dem Target aufgebauten Netzwerk-Verbindungen überwacht werden und so besonders zuverlässig und schnell die Aktivitäten des im Erfolgsfall ausgeführten Payloads registriert werden.
pyperpwn kann hierbei eine komplett passive Rolle einnehmen, da Verbindungsversuche auf das Target im Falle eines bind Payloads von Metasploit automatisch mehrfach wiederholt durchgeführt werden. Bei reverse Payloads stellt Metasploit automatisch einen passenden Handler auf dem Host bereit, auf den sich der Payload verbinden kann. Dadurch ist eine für diesen Zweck rein überwachende Funktion der *ConnectionSupervisor* genannten Klasse ausreichend.

Erfolgsbewertung

Die Bewertung des Erfolges der einzelnen Exploits wird anhand dreier Kriterien vorgenommen. Diese enthalten die Informationen, ob der Exploit erfolgreich ausgeführt, ein Zugang zum Target erreicht und ob die Verfügbarkeit des Ziel-Systems beeinträchtigt wurde.
Als erste wichtige Entscheidungsregel über den Erfolg muss hierbei die Exploit-Klassifikation zu Rate gezogen werden. Da von pyperpwn keine Überprüfung der Integrität des Targets durchgeführt werden kann, werden alle in [Gam19] als „nicht-intrusiv“ oder „intrusiv-harmlos“ eingestuften Exploits hier als „intrusiv“ betrachtet.
Anhand der jeweiligen Klassifikation ergeben sich drei Entscheidungsregeln:

- Handelt es sich um einen *nicht-intrusiven* Exploit, so war dieser erfolgreich, wenn er die Verfügbarkeit des Ziel-Systems nicht beeinflusst hat und eine erfolgte Ausführung auf dem Target erkannt werden konnte.

- Ein *intrusiver* Exploit hat seine Aufgabe erfüllt, wenn er ausgeführt wurde und eine Session zum Target erzeugt hat, aber keinen Schaden am Ziel-Rechner herbeigeführt hat.

- Wurde die Verfügbarkeit des Targets durch die Exploit-Ausführung beeinträchtigt, so kann der Exploits nur als erfolgreich gelten, wenn er zuvor als *intrusiv-destruktiv* klassifiziert wurde. Ein solcher Exploit gilt allerdings auch als erfolgreich, wenn er keinerlei negative Auswirkungen auf das Target hat und einen Zugang zu diesem ermöglicht.

6. Evaluation und Test

Die in den vorhergehenden Kapiteln erarbeitete Anwendung soll nun in diesem Schritt einer genaueren Evaluation unterzogen und mit den beiden anderen, vorgestellten Anwendungen verglichen werden.

6.1. Anwendung des Tools

Bevor das Tool ausgeführt werden kann, sind die dafür benötigten Abhängigkeiten zu installieren. Genauere Informationen zum Setup und Start finden sich in der bereitgestellten Readme-Datei.
Die folgenden Abbildungen 6.1, 6.2, 6.3 und 6.4 zeigen chronologisch geordnete Ausschnitte aus einer beispielhaften Ausführung von pyperpwn, jeweils während der Anwendung auf die Typhoon VM.

```
[s@muel pyperpwn]$ sudo ./pyperpwn.py -p pynetrator -v vuln_reports/typhoon.csv -c expl_classes.csv

================================================================================

 $$$$$$\  $$\   $$\  $$$$$$\   $$$$$$\   $$$$$$\   $$$$$$\  $$\  $$\  $$\ $$$$$$$\
$$  __$$\ $$ |  $$ |$$  __$$\ $$  __$$\ $$  __$$\ $$  __$$\ $$ | $$ | $$ |$$  __$$\
$$ /  $$ |$$ |  $$ |$$ /  $$ |$$$$$$$$ |$$ |  \__|$$ /  $$ |$$ | $$ | $$ |$$ |  $$ |
$$ |  $$ |$$ |  $$ |$$ |  $$ |$$   ____|$$ |      $$ |  $$ |$$ | $$ | $$ |$$ |  $$ |
$$$$$$$  |\$$$$$$$ |$$$$$$$  |\$$$$$$$\ $$ |      $$$$$$$  |\$$$$$\$$$$  |$$ |  $$ |
$$  ____/  \____$$ |$$  ____/  \_______|\__|      $$  ____/  \_____\____/ \__|  \__|
$$ |      $$\   $$ |$$ |                          $$ |
$$ |      \$$$$$$  |$$ |                          $$ |
\__|       \______/ \__|                          \__|

================================================================================

Welcome to pyperpwn!
Please restart the MSFRPC daemon everytime you restart this application.

================================================================================

08:25:43 Wizard       [?] Required option [Remote Host IP Address] not specified. Enter a value: 172.28.128.6
08:25:48 Wizard       [?] Required option [Local Host IP Address] not specified. Enter a value: 172.28.128.1
08:25:50 Wizard       [?] Required option [Rhost operating system] not specified. Enter a value: linux
08:25:52 MsfHandler   [+] Connecting to MSFRPC server...
08:25:52 MsfHandler   [+]       Successfully connected to MSFRPC at 127.0.0.1:55553
08:25:52 DBHandler    [+] Checking connection to MongoDB...
08:25:52 DBHandler    [+] Checking state of Exploit Cache...
08:25:52 DBHandler    [+]       Found 1913 exploits in DB, while MSF currently offers 1913 in total
08:25:52 DBHandler    [+]       No need to fill cache again. Proceeding...
08:25:52 ExplClassR.  [+] Start reading csv file: expl_classes.csv
08:25:52 ExplClassR.  [+] Written 66 entries from Exploit Classification file to DB.
08:25:52 VulnReader   [+] Start reading csv file: vuln_reports/typhoon.csv
08:25:52 VulnReader   [+] Read 42 entries from Vulnerability Scanner Output
```

Abbildung 6.1.: Ausschnitt des Beginns einer beispielhaften Ausführung von pyperpwn. Zu erkennen sind besonders die Abfragen aufgrund fehlender Parameterwerte, die Verbindungsaufbauten zu MSFRPC und der MongoDB-Datenbank sowie das Einlesen des Schwachstellenreports.

```
NOW TREATING VULNERABILITY: 'LotusCMS PHP Eval Remote Command Execution Vulnerability' (CVSS: 10.0)

09:41:38 ExplMatcher  [+] Searching by keywords 'lotuscms php eval remote command execution'
09:41:38 ExplMatcher  [+]       Found 1 exploit(s).
09:41:38 ExplMatcher  [+] Searching by CVEs ['CVE-2011-0518']
09:41:38 ExplMatcher  [+]       Found 1 exploit(s).
09:41:38 ExplMatcher  [+] Searching by BIDs ['52349']
09:41:38 ExplMatcher  [!]       None found.
09:41:38 ExplMatcher  [+] Removed 1 duplicate exploits.
09:41:38 ExplMatcher  [+] Filtered exploits by OS 'linux': 1 left.
09:41:38 ExplMatcher  [+] Filtered exploits by Rank 'manual': 1 left.
09:41:38 ExplMatcher  [+] Sorted exploits.
09:41:38 pyperpwn     [+] All exploits found: multi/http/lcms_php_exec

Exploit 'multi/http/lcms_php_exec' will now be executed
09:41:38 ExploitExec  [?] Exploit has 3 non-common params. Do you want to change them? [y/n]: y
09:41:40 Wizard       [?]       [URI], current value: '/lcms/'. Enter a new value (leave blank for default): /cms
09:41:42 Wizard       [?]       [HttpPassword], current value: ''. Enter a new value (leave blank for default):
09:41:42 Wizard       [?]       [HttpUsername], current value: ''. Enter a new value (leave blank for default):
09:41:43 Wizard       [?] Do you want to execute exploit 'multi/http/lcms_php_exec' against 172.28.128.6? [y/n]: y
09:41:45 Wizard       [?] Do you want to change the default payload (8 alternatives available)? [y/n]: y
09:41:46 Wizard       [+]       Available payloads:
09:41:46 Wizard       [+]        (0)   php/exec
09:41:46 Wizard       [+]        (1)   php/meterpreter/bind_tcp
09:41:46 Wizard       [+]        (2)   php/meterpreter/bind_tcp_ipv6
09:41:46 Wizard       [+]        (3)   php/meterpreter/bind_tcp_ipv6_uuid
09:41:46 Wizard       [+]        (4)   php/meterpreter/bind_tcp_uuid
09:41:46 Wizard       [+]        (5)   php/meterpreter/reverse_tcp
09:41:46 Wizard       [+]        (6)   php/meterpreter/reverse_tcp_uuid
09:41:46 Wizard       [+]        (7)   php/reverse_perl
09:41:46 Wizard       [+]        (8)   php/reverse_php
09:41:46 Wizard       [?] Enter the number of the payload to use instead (blank for default): 5
09:41:50 ExploitExec  [+]       Selected payload: php/meterpreter/reverse_tcp
09:41:50 ExploitExec  [+] Set payload 'php/meterpreter/reverse_tcp'
09:41:50 ConnSuperv   [+] Start to capture network traffic with host 172.28.128.6...
09:41:50 ExploitExec  [+] Exploit is being executed using msf console 8. This might take a while.
```

Abbildung 6.2.: Ausschnitt aus der automatischen Exploit- und der – auf Wunsch des Nutzers – manuellen Payload-Auswahl von pyperpwn. Zuerst werden anhand des Namens, der CVE-Nummern und der Bugtraq-IDs passende Exploits gesucht, wovon hier der Erste direkt im Anschluss mit dem ausgewählten Payload und den teils manuell geänderten Parameterwerten konfiguriert und ausgeführt wird.

Mit der Anwendungsoption `-h` können Hilfsinformationen ausgegeben werden, die die jeweils verfügbaren Parameter auflisten und beschreiben. Die gesamte Konsolenausgabe der Anwendung wird in die Datei `pyperpwn_log.txt` geschrieben, sodass der Anwendungslog auch später noch genauer analysiert werden kann.
Mittels `STRG + C` kann die Ausführung der Anwendung jederzeit unterbrochen werden. Dabei kann sofort ein Report erstellt werden, womit die Ausführung als beendet betrachtet wird, oder aber der Anwendungszustand in die Datenbank geschrieben werden, so dass diese unterbrochene Ausführung später fortgesetzt werden kann. Ebenso ist ein sofortiges Terminieren ohne Speichern der gesammelten Information möglich. Findet das Tool bei erneutem Start eine früher gespeicherte Ausführung, so wird der Nutzer direkt nach dem Start gefragt, ob er diese fortsetzen möchte.
Die Ausführung von pyperpwn auf anderen Betriebssystemen (außer Linux) ist möglich, allerdings sind dann andere, jeweils online verfügbare Installationsanweisungen und Ausführungsschritte zu beachten.

```
09:29:30 ConnSuperv   [+] Start to capture network traffic with host 172.28.128.6...
09:29:30 ExploitExec  [+] Set payload 'php/meterpreter/reverse_tcp'
09:29:30 ExploitExec  [+] Exploit is being executed using msf console 3. This might take a while.
09:29:31 ConnSuperv   [+] Detected exploit delivery to target
09:29:31 ConnSuperv   [+] Detected reverse payload request.
09:29:31 ConnSuperv   [!] EXPLOIT HAS BEEN EXECUTED ON THE TARGET!
09:29:31 ConnSuperv   [+]        detected with: REVERSE_PAYLOAD_REQUEST
09:29:31 ConnSuperv   [+] Reverse payload has been delivered
09:29:31 ConnSuperv   [!] EXPLOIT HAS BEEN EXECUTED ON THE TARGET!
09:29:31 ConnSuperv   [+]        detected with: PAYLOAD_STAGE_CONNECTION
09:29:51 ExploitExec  [+] Received exploit output
09:29:51 ConnSuperv   [+] Stop capturing network traffic...
09:29:51 ExploitExec  [+] Exploit has been run without errors.
09:29:51 SuccessCheck [+] Run #0 of success checker
09:29:58 SuccessCheck [+]        intentionally waiting...
09:30:03 SuccessCheck [+] Run #1 of success checker
09:30:09 SuccessCheck [+]        intentionally waiting...
09:30:14 Sess.Handler [+] Start Session checker for exploit/multi/http/lcms_php_exec
09:30:14 Sess.Handler [+]        Got session 2 via exploit exploit/multi/http/lcms_php_exec
09:30:16 Sess.Handler [+] Received shell response to command 'getuid'.
09:30:17 Sess.Handler [+] Received shell response to command 'route'.
09:30:19 Sess.Handler [+] Received shell response to command 'whoami'.
09:30:20 Sess.Handler [+] Received shell response to command 'uname -a'.
09:30:21 Sess.Handler [+] Received shell response to command 'ifconfig'.
09:30:21 Sess.Handler [+] Automatically closed session with id 2
09:30:21 pyperpwn     [+] Waiting intentionally...
```

Abbildung 6.3.: Ausschnitt aus der erfolgreichen Anwendung des Exploits `exploit/multi/http/lcms_php_exec` von pyperpwn auf die Typhoon VM. Zu erkennen ist hier, dass die erfolgreiche Ausführung des Exploits mithilfe der Überwachung des Traffics bereits richtig erkannt wird, bevor der Exploit-Output zur Verfügung steht. Im Anschluss werden verschiedene Überprüfungen auf das Target angewandt und mehrere Kommandos zur Informationsgewinnung ausgeführt, sowie die erzeugte Session automatisch geschlossen.

6.2. Evaluation

Ein Vergleich mit den beiden bereits vorgestellten Tools soll im Folgenden die Vorteile der entwickelten Anwendung sowie daraus resultierende Einsatzmöglichkeiten für den beschriebenen Einsatzzweck aufzeigen.

6.2.1. Vergleich mit bestehenden Anwendungen

Für den Vergleich mit den beiden bestehenden Anwendungen wurden jeweils unterschiedliche Metriken gewählt, da sich aufgrund unterschiedlicher Inputs nicht die gleichen Testbedingungen konstruieren lassen.
Ein Exploit gilt nachfolgend als von pyperpwn erfolgreich ausgeführt, wenn sich im Report im Feld „Success“ der Wert `True` befindet.

Vergleich mit AutoSploit

Die Tabelle 6.5 zeigt einen direkten Vergleich der Anzahl der als erfolgreich gekennzeichneten Exploits bei einer Ausführung der beiden Anwendungen auf die drei verfügbaren Testumgebungen. Hierbei wurde für AutoSploit eine selbst erstellte Exploit-Liste und für pyperpwn ein von OpenVAS erstellter, aber manuell veränderter Schwachstellenreport verwendet, um

```
09:45:07 ExploitExec  [?] Exploit has 3 non-common params. Do you want to change them? [y/n]: ^C

09:45:22 pyperpwn     [!] Gracefully stopping application.
09:45:22 Sess.Handler [+] Automatically closed 1 sessions.
09:45:22 Wizard       [?] Save this execution for later [c]ontinuation, [g]enerate a report now or e[x]it at any cost?: g

09:45:30 pyperpwn     [+] Application execution successfully finished.
09:45:30 pyperpwn     [+] Creating result CSV file...
09:45:30 CSVWriter    [+] You can find the generated report at 'hyper_pyper_report.csv'.
09:45:30 pyperpwn     [+] Execution finished successfully.
```

Abbildung 6.4.: Beispielhaftes Ende einer erfolgreichen Anwendungsausführung von pyperpwn. Dargestellt ist der Abbruch einer Exploit-Ausführung durch den Benutzer und das anschließende Speichern des Reports, bevor die Anwendung terminiert.

Testumgebung	AutoSploit	pyperpwn
Metasploitable 2	15 (7)	11 (0)
Metasploitable 3	6 (2)	7 (0)
Typhoon VM	2 (1)	5 (0)

Abbildung 6.5.: Evaluationsdetails beim Vergleich von pyperpwn mit AutoSploit: Anzahl der jeweils als erfolgreich erkannten Exploits und Anzahl der False-Positive-Fehler (in Klammern). Beide Anwendungen wurden so konfiguriert, dass sie dieselben Exploits ausführten.

sicherzustellen, dass beide Anwendungen die gleichen Exploits ausführen.
Besonders deutlich wird anhand der Übersicht auch hier die hohe Fehlerrate von AutoSploit, das teilweise funktionierende Exploits nicht erfolgreich ausführen kann und nicht-kompatible oder nicht-erfolgreiche Exploits als erfolgreich markiert. pyperpwn kann hingegen aufgrund der zusätzlichen Daten aus dem Schwachstellenreport und den Rückfragen beim Benutzer mehr Exploits erfolgreich ausführen. Auch diese Erfolgsbewertung ist dank der gezeigten Maßnahmen wesentlich präziser. Bei der Anwendung von pyperpwn auf Metasploitable 3 ist seitens pymetasploit3 ein Fehler aufgetreten, sodass alle zu dem Zeitpunkt noch ausstehenden Exploits nicht korrekt ausgeführt werden konnten.
Die von pyperpwn während des Exploit-Vorganges gesammelten Informationen sind deutlich umfangreicher als die von AutoSploit, wie der Vergleich der Tabellen C.1 und D.1 im Anhang zeigt. Dies liegt nicht nur an der Verfügbarkeit eines Schwachstellenscans und der Integration der darin enthaltenen Informationen, sondern auch an der Durchführung weitergehender Informationsakquise bei erfolgreicher Ausnutzung einer Schwachstelle.

Vergleich mit Hail Mary

Da eine Einflussnahme auf die Exploit-Auswahl bei Hail Mary nicht ohne Weiteres möglich ist, wird hierbei nur die Anzahl erfolgreich ausgeführter Exploits mit der von pyperpwn verglichen. Zugrunde gelegt werden für Hail Mary zwei zuvor ausgeführte nmap-Scans („*Intense Scan, all TCP ports*“ und „*Intense Scan + UDP*“), für pyperpwn ein von OpenVAS generierter Schwachstellenreport in der höchsten vom OpenVAS-Wizard angebotenen Genauigkeitsstufe sowie eine manuell erstellte Exploit-Kategorisierung. Ausgeführt wird Hail Mary, ebenso wie pyperpwn, ohne eine Beschränkung des Exploit-Ranks.

Testumgebung	Hail Mary	pyperpwn
Metasploitable 2	3 (446)	7 (27)
Metasploitable 3	0 (128) 1 (89)*)	3 (23)
Typhoon VM	1 (470)	4 (27)

Abbildung 6.6.: Evaluationsdetails beim Vergleich von pyperpwn mit Hail Mary: Anzahl der jeweils als erfolgreich erkannten Exploits und Gesamtanzahl der ausgeführten Exploits (in Klammern).
∗) Die Ausführung von Hail Mary auf Metasploitable 3 wurde mit einem höheren Exploit-Ranking (*Good* statt *Poor*) wiederholt, da nach einem Systemabsturz des Targets keinerlei Sessions erkannt werden konnten.

Die Erfolgsevaluation läuft bei Hail Mary ausschließlich über das Prüfen erzeugter Remote Sessions ab, weswegen False-Positive-Fehler (im Gegensatz zu AutoSploit) beinahe auszuschließen sind. Die Tabelle 6.6 zeigt deswegen die Anzahl aller und aller erfolgreich ausgeführten Exploits. Damit wird deutlich, dass pyperpwn mithilfe des Reports und der Rückfragen beim Nutzer deutlich mehr Exploits erfolgreich anwenden kann, während dafür insgesamt eine erheblich geringere Anzahl an Exploits ausgeführt werden muss. pyperpwn ist hiermit bezüglich der Exploit-Ausführung nicht nur deutlich effizienter als Hail Mary, sondern angesichts einer geringeren Anzahl an Log-Einträgen und TCP-Verbindungen auch deutlich unauffälliger sowie weniger gefährlich für die Verfügbarkeit und Integrität des Targets.
Da Hail Mary keinen Abschlussreport als solchen erstellt, ist pyperpwn auch bezüglich der gesammelten und dokumentierten Informationen deutlich überlegen.

6.2.2. Einsatzmöglichkeiten

Aus diesen Vorzügen von pyperpwn ergeben sich verschiedene, mögliche Einsatzszenarien der Anwendung in der Praxis.
In erster Linie wurde pyperpwn für teilautomatische Penetrationstests entworfen. Für einen sinnvollen Einsatz von pyperpwn ist die Kombination mit anderen Anwendungen, wie beispielsweise einem vorhergehenden Schwachstellenscanner notwendig.
Die Anwendung ist im Express-Modus auch für vollautomatische Penetrationstests geeignet. In diesem Fall bietet pyperpwn einen ähnlichen Funktionsumfang wie die beiden vorgestellten Anwendungen.
Darüber hinaus ließe sich die Anwendung aber auch einsetzen, um vollautomatisch das Schließen von Sicherheitslücken zu überprüfen. So kann der behobene Eintrag des Schwachstellenscannerreports in einen anderen Report übernommen werden, anhand dessen automatisiert in regelmäßigen Abständen diese Schwachstelle wieder überprüft wird, um so das Wiederauftreten einer Lücke, beispielsweise aufgrund einer fehlerhaften automatischen Rekonfiguration der verwundbaren Anwendung, möglichst zeitnah zu erkennen.

6.3. Erweiterbarkeit

Die einfache Erweiterbarkeit des Tools ist auf mehreren Ebenen gegeben.
Neue Exploit- oder Payload-Module in Metasploit können ohne weitere Änderungen am

Programm angewandt werden, da sich diese grundsätzlich an die von Rapid7 vorgegebene Schnittstelle halten müssen. Ein Einsatz neuer Modul-Funktionen ist auch einfach möglich, sobald diese neue Funktion in pymetasploit3 integriert wurde.
Weitere Module zur Überprüfung des Target-Zustandes während und nach der Exploit-Ausführung lassen sich ebenfalls leicht in die Anwendung und den am Ende erstellten Report integrieren. Hierbei müssen die neuen Module nicht auch in Python geschrieben sein, da sie (beispielsweise analog zu tcpdump) auch separat über einen eigenen Prozess ausgeführt werden können.
Durch den modularen Aufbau ist das Tool auch um weitere Funktionen, beispielsweise um weitergehende Scanner-Module aus Metasploit, einfach zu erweitern.
Aufgrund der einfachen Erweiterbarkeit ist pyperpwn nicht nur einfach weiterzuentwickeln und hinsichtlich neuer Umgebungen und Funktionalitäten anzupassen, sondern auch gut gerüstet für die Zukunft.

6.4. Einschränkungen

Bei der Konzeptionierung und Implementierung der prototypischen Anwendung mussten allerdings auch mehrere Einschränkungen in Kauf genommen werden.
Beispielsweise werden bisher von pyperpwn nur von remote ausführbare Exploits korrekt ausgeführt und anschließend analysiert. Ein Einsatz beispielsweise lokaler Exploits oder anderer Modul-Arten ist nicht möglich. Dies gilt insbesondere für Exploits, die im weiteren Sinne als Exploits gelten, von Metasploit aber zu den Auxiliary-Modulen gezählt werden, wie beispielsweise DoS-Exploits.
Ebenso wird jeder Exploit möglichst isoliert ausgeführt und betrachtet. Dadurch ist das in der Praxis häufig genutzte, gezielte Verketten von Exploits nicht zu simulieren. Derartige Eskalationsketten wurden hier nicht näher berücksichtigt, da der Fokus auf einer möglichst getrennten, von der Ausführungsreihenfolge unabhängigen Erfolgsbewertung lag und nur remote Exploits von pyperpwn unterstützt werden.
Bei der Erkennung der Auswirkungen nach der Exploit-Ausführung wurden nur Methoden implementiert, die das Target remote vom Host aus überprüfen. Dies ist gegebenenfalls jedoch nicht ausreichend, da hiermit interne Änderungen des Remote Hosts, beispielsweise eine Beeinträchtigung der Integrität, nicht erkannt werden können.

7. Zusammenfassung und Ausblick

Nachdem die Anwendung nun entwickelt, vorgestellt und evaluiert wurde, soll im folgenden Kapitel noch eine Übersicht über die in dieser Arbeit erreichten Ziele sowie die daraus entstehenden Erweiterungsmöglichkeiten gegeben werden.

7.1. Erreichte Ziele

Die nachfolgende Tabelle 7.1 zeigt die ursprüngliche, in Kapitel 3 definierte Liste an funktionalen Anforderungen inklusive einer zusätzlichen Spalte, die angibt, ob die jeweilige Anforderung auch von der finalen Version der Implementierung erfüllt wird („✓“ bedeutet „umgesetzt“, „○“ steht für „ausstehend“).
Wie der Tabelle zu entnehmen ist, werden beinahe alle formulierten Anforderungen von pyperpwn erfüllt. Lediglich bei der Ermittlung der Absicht des jeweiligen Exploits ist die Zuhilfenahme einer externen Quelle, die von pyperpwn eingelesen wird, nötig.
Außerdem kann der Report nicht innerhalb der Anwendung nach der Priorität der Einträge sortiert und gefiltert werden. Aufgrund der Wahl eines universellen Ausgabeformats (CSV) ist dies jedoch einfach und sinnvoller mithilfe anderer Anwendungen (z.B. LibreOffice Calc) zu erledigen.
Analog dazu zeigt Tabelle 7.2 den Status aller nicht-funktionalen Anforderungen in der finalen Version von pyperpwn. Hier wird deutlich, dass pyperpwn alle formulierten Punkte erfüllt. Die Anwendung wurde auf Linux als Host-System sowie mit Windows- und Linux-VMs getestet. Ein benutzerfreundliches Zeitverhalten wurde durch möglichst wenige Rückfragen durch den durch den Programmablauf führenden Wizard ermöglicht. pyperpwn kann über die Kommandozeile gesteuert werden und mittels des Express-Modus' ohne Rückfragen ausgeführt werden. Auf die einfache Erweiterbarkeit wurde bereits in Abschnitt 6.3 eingegangen.

7.2. Erweiterungsmöglichkeiten

Auch wenn pyperpwn im Vergleich mit AutoSploit und Hail Mary für den beschriebenen Einsatzzweck einige Vorteile bietet, kann diese bisher nur prototypisch entwickelte Anwendung noch hinsichtlich verschiedenster Funktionalitäten erweitert werden, um damit ihren produktiven Einsatz komfortabler zu gestalten.
Die Anwendung bietet bisher nur ansatzweise Funktionalitäten, die für Pivoting hilfreich wären. Allgemein bezeichnet Pivoting den Prozess, eine bereits zugängliche Maschine als „Sprungbrett“ für das weitere Vordringen in nicht-öffentliche Teile des mit diesem Rechner verbundenen Netzwerkes zu nutzen. pyperpwn ist in dieser Hinsicht nur behilflich, indem es bei erfolgreichem Erlangen einer Remote Shell nützliche Informationen über das Target, wie beispielsweise konfigurierte Netzwerk-Schnittstellen, sammelt – darauf aufbauende Aktionen müssen bisher manuell initiiert werden.

Nr.	Beschreibung	Status
	Exploit-Auswahl	
1	Automatische Auswahl ohne Benutzer-Interaktion	✓
2	Zugriff auf die Modul-Sammlung von Metasploit	✓
3	Auswahl mindestens aller passenden Exploits anhand mehrerer Kriterien	✓
4	Einschränkung der Auswahl bezüglich des Mindestrankings	✓
5	Einschränkung der Auswahl bezüglich des Ziel-Betriebssystems	✓
6	Ermittlung der Absicht des jeweiligen Exploits	○
	Exploit-Ausführung	
7	Sortierung der Schwachstellen nach Schweregrad	✓
8	Sortierung der Exploits nach Verlässlichkeit	✓
9	Automatisches Setzen der Exploit-Parameter	✓
10	Automatische Auswahl des am besten geeigneten Payloads	✓
11	Anfrage an den Nutzer, falls keine automatische Ermittlung der Exploit-Parameter möglich	✓
12	Wiederholte Anwendung der Exploits auf das festgelegte Target	✓
13	Steuerung der Geschwindigkeit der Exploit-Ausführung	✓
	Automatische Erkennung der Auswirkungen und Bewertung	
14	Automatische Erkennung der Auswirkungen mithilfe verschiedener Methoden	✓
14.1	Überprüfung mittels ping	✓
14.2	Überprüfung mittels TCP-SYN-Scan	✓
14.3	Überprüfung mittels HTTP-Anfrage	✓
14.4	Erkennung einer erfolgreich aufgebauten Remote Session	✓
14.5	Erkennung des Nachladens der zweiten Stage durch den Dropper	✓
15	Steuerung der Genauigkeit der Erkennung	✓
16	Präzise Bewertung des Erfolgs	✓
17	Weiterführendes Sammeln von Informationen	✓
	Reportgenerierung	
18	Automatische Reportgenerierung	✓
19	Genaues Protokoll jeder Exploit-Ausführung im Report	✓
20	Erfolgsbewertung und dafür verwendete Daten im Report	✓
21	Bezug zum eingelesenen Report des Schwachstellenscanners für jeden Eintrag	✓
22	Speichern des Reports in einem verbreiteten Format	✓
23	Sortieren und Filtern der Einträge im Report nach Priorität	○
	Allgemein	
24	Abbrechen jederzeit ohne Verlust bisher gesammelter Daten möglich	✓
25	Fortsetzung der Ausführung jederzeit nach Anwendungsabbruch möglich	✓
26	Überprüfungen zum Verhindern versehentlicher Angriffe	✓

Abbildung 7.1.: Übersicht über den Status der funktionalen Anforderungen in der finalen Version von pyperpwn

Nr.	Beschreibung	Status
1	Ausführbarkeit der Anwendung auf Linux	✓
2	Unterstützung von Windows und Linux als Ziel-Betriebssysteme für Exploits	✓
3	Benutzerfreundliches Zeitverhalten	✓
4	Schnelldurchlauf möglich	✓
5	Bedienung über die Kommandozeile	✓
6	Führung des Nutzers von einem Wizard, falls nicht alle benötigten Parameter angegeben wurden	✓
7	Einfache Erweiterbarkeit bzgl. neuer Betriebssysteme und Exploits	✓
8	Einfache Erweiterbarkeit bzgl. weiterer Automatisierungsfunktionalitäten	✓
9	Anleitung zur Einrichtung, Installation der Abhängigkeiten und zum Anwendungsstart	✓
10	Hilfetexte verfügbar	✓

Abbildung 7.2.: Übersicht über den Status der nicht-funktionalen Anforderungen in der finalen Version von pyperpwn

Die Anwendung wäre außerdem universeller einsetzbar, wenn auch andere Exploit-Quellen miteinbezogen werden würden. Beispielsweise könnten auch Exploits aus der online und frei verfügbaren ExploitDB[1] direkt angewandt werden. Dies würde einen deutlichen Vorteil gegenüber den verglichenen Anwendungen darstellen, aber mehr Anpassungsarbeit benötigen. Dieser Aufwand, wie beispielsweise die einheitliche Konfiguration von Parametern und die eigenständige, automatische Ausführung von Payload-Handlern, wird normalerweise direkt vom Metasploit-Framework übernommen.
Neben dem Hinzufügen weiterer Exploit-Quellen wäre es auch hilfreich, eine präzisere Erkennung der Auswirkungen auf dem Target vorzunehmen. Die bisherigen Methoden beschränken sich allein auf eine Überprüfung der Verfügbarkeit des Zielsystems, können also interne Veränderungen auf dem Target nicht erkennen, wenn diese keine Änderung der Verfügbarkeit zur Folge haben. Um derartige Zustandsänderungen zu identifizieren, wäre eine Überprüfung der Integrität des Targets möglich, welche allerdings nur lokal auf selbigem ausgeführt werden kann. Folglich wäre für diese deutlich genauere Möglichkeit der Erfolgsbewertung eine Erweiterung der bestehenden Architektur notwendig, welche die Anwendung um einen Remote-Client zur Integritätsüberprüfung auf dem Remote Host ergänzt.
Da pyperpwn bisher einzig über die Kommandozeile steuerbar ist, böte sich die Implementierung einer Schnittstelle beispielsweise mit dem für Python verfügbaren Webanwendungsframework *Flask* an, sodass die Anwendung auch mittels einer graphischen Weboberfläche remote gesteuert werden könnte.
Für den in Abschnitt 6.2.2 erwähnten Einsatz von pyperpwn zur regelmäßigen Überprüfung von bereits geschlossenen Sicherheitslücken wäre es hilfreich, wenn sich die für jedes Target spezifischen Exploit-Parameter einfach exportieren und speichern, sowie bei einer erneuten Ausführung importieren ließen. Dies würde ermöglichen, dass das Tool auch bei vollautomatischer Ausführung im Express-Modus hier nicht die Standard-Werte verwenden muss.

[1] `https://www.exploit-db.com/`

Eine zusätzliche Erweiterungsmöglichkeit bestünde auch hinsichtlich der Unterstützung mehrerer verschiedener Zielsysteme während einer einzigen Ausführung, sodass ein Schwachstellenscan eines ganzen Netzwerkbereiches eingelesen und beispielsweise entsprechend sequenziell verarbeitet werden kann.

7.3. Fazit

Im Rahmen dieser Arbeit wurde eine eigenständige, auf dem Metasploit Framework basierende Anwendung entwickelt, die für ein Ziel-System auf Basis eines gegebenen Schwachstellenscans passende Exploits auswählt, ausführt und anschließend deren Erfolg bewertet.
Dabei wurde deutlich, dass vergleichbare, gegebenenfalls für diesen Einsatzzweck verfügbare Anwendungen jeweils auf sehr einfachen Ansätzen basieren, die in allen drei genannten Ausführungsschritten zu deutlich schlechteren Ergebnissen führen.
Mit gezielten Maßnahmen konnte in allen diesen Bereichen nennenswerte Verbesserungen erzielt werden, sodass die entwickelte Anwendung pyperpwn für den gegebenen Einsatzzweck deutlich besser geeignet ist.
Die Anwendung zeichnet sich besonders durch einen viele Details umfassenden Report, eine tiefgreifende Überwachung des Traffics mit dem Target sowie die Kombination mehrerer, verschiedener Informationsquellen aus. Bei ihrer Entwicklung wurde außerdem zur Verbesserung der genutzten Bibliothek für die Steuerung des Metasploit Frameworks beigetragen. Die Wahl des Metasploit Frameworks als Basis der Anwendung stellt sicher, dass auch zukünftig aktuelle Exploits für die Anwendung zur Verfügung stehen werden. Eine Einbindung einer weiteren Quelle würde allerdings eine noch größere Auswahl ermöglichen.
Erweitert werden könnte die Anwendung beispielsweise auch um einen Client auf dem Target zur lokalen Überprüfung der Dienst-Verfügbarkeit und Integrität des Remote Hosts nach der Exploit-Ausführung, was eine präzisere Erkennung der Auswirkungen und damit Erfolgsbewertung ermöglichen würde.
Die einfache Erweiter- und Wartbarkeit von pyperpwn ermöglicht den einfachen Einsatz des Tools in einer benutzerdefinierten Umgebung. Dadurch kann die Anwendung in Verbindung mit anderen geeigneten Tools bereits jetzt einen wertvollen Beitrag zu einem teilautomatisierten Penetrationstest leisten.
Die bereits geschehene Veröffentlichung der Anwendung auf GitHub[2] ermöglicht in Verbindung mit der Wahl einer passenden, freien Software-Lizenz auch anderen Sicherheitsforschern und Penetrationstestern den freien Einsatz, die Verteilung sowie die Weiterentwicklung von pyperpwn.

[2] `https://github.com/dial25sd/pyperpwn`

Abbildungsverzeichnis

Literaturverzeichnis

[BSI03] BSI Bundesamt für Sicherheit in der Informationstechnik: *Durchführungskonzept für Penetrationstests*. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Studien/Penetrationstest/penetrationstest.pdf?__blob=publicationFile&v=3. Version: November 2003, Abruf: 29.03.2019

[BSI13] BSI Bundesamt für Sicherheit in der Informationstechnik: *IT-Grundschutz-Kompendium - Glossar*. https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKompendium/vorkapitel/Glossar_.html. Version: 2013, Abruf: 17.09.2019

[BSI16] BSI Bundesamt für Sicherheit in der Informationstechnik: *Ein Praxis-Leitfaden für IS-Penetrationstests*. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Sicherheitsberatung/Pentest_Webcheck/Leitfaden_Penetrationstest.pdf?__blob=publicationFile&v=10. Version: November 2016, Abruf: 29.03.2019

[BYCJ11] Bacudio, Aileen G. ; Yuan, Xiaohong ; Chu, Bei-Tseng B. ; Jones, Monique: An Overview Of Penetration Testing. In: *International Journal of Network Security & Its Applications (IJNSA)* (2011)

[Bö17] Böhm, Markus: Ransomware WannaCry befällt Rechner der Deutschen Bahn. In: *Spiegel Online* (2017). http://www.spiegel.de/netzwelt/web/wannacry-450-bahn-computer-von-cyber-attacke-betroffen-a-1147921.html, Abruf: 03.01.2019

[Dep13] Deptula, Kendra: *Automation of cyber penetration testing using the detect, identify, predict, react intelligence automation model*, Naval Postgraduate School, Masterarbeit, 2013

[DWH13] Durumeric, Zakir ; Wustrow, Eric ; Halderman, J. A.: ZMap: Fast Internet-Wide Scanning and its Security Applications. In: *Proceedings of the 22ndUSENIXSecurity Symposium* (2013), August

[EKMM11] Elsbroek, Dominik ; Kohlsdorf, Daniel ; Menke, Dominik ; Meyer, Lars: FIDIUS: Intelligent Support for Vulnerability Testing. In: *Working Notes for the 2011 IJCAI Workshop on Intelligent Security* (2011)

[Eur18] Europäische Union: *Datenschutz-Grundverordnung: Art. 32 Sicherheit der Verarbeitung*. https://dejure.org/gesetze/DSGVO/32.html. Version: 2018, Abruf: 17.09.2019

[Fie18] Field, Matthew: WannaCry cyber attack cost the NHS £92m as 19,000 appointments cancelled. In: *The Telegraph* (2018). `https://tech.newstatesman.com/security/cost-wannacry-ransomware-attack-nhs`, Abruf: 03.01.2019

[Gal18] Gallagher, Sean: *Threat or menace? "Autosploit" tool sparks fears of empowered "script kiddies".* `https://arstechnica.com/information-technology/2018/02/threat-or-menace-autosploit-tool-sparks-fears-of-empowered-script-kiddies/`. Version: Februar 2018, Abruf: 17.09.2019

[Gam19] Gamisch, Laura: *Kategorisierung von Exploits zur Durchführung von nicht-intrusiven Penetrationstests*, Ludwig-Maximilians-Universität München, Bachelorarbeit, September 2019

[GGE14] Genge, Béla ; Graur, Flavius ; Enăchescu, Călin: Non-intrusive Techniques for Vulnerability Assessment of Services in Distributed Systems. In: *8th International Conference Interdisciplinarity in Engineering* (2014)

[Kli15] *Kapitel* ISO/IEC 27005. In: Klipper, Sebastian: *Information Security Risk Management*. Springer, 2015. – ISBN 978–3–658–08774–6, S. 59–94

[Mes15] Messner, Michael: *Hacking mit Metasploit*. 2. Auflage. dpunkt.verlag, 2015. – ISBN 978–3–86490–224–6

[Mud13a] Mudge, Raphael: *Cortana Tutorial*. `http://www.fastandeasyhacking.com/download/cortana/cortana_tutorial.pdf`. Version: Mai 2013, Abruf: 19.01.2019

[Mud13b] Mudge, Raphael: *The Origin of Armitage's Hail Mary Mass Exploitation Feature*. `https://blog.cobaltstrike.com/2013/07/17/the-origin-of-armitages-hail-mary-mass-exploitation-feature/`. Version: Juli 2013, Abruf: 17.09.2019

[Oak19] *Kapitel* Why Human Hackers? In: Oakley, Jacob G.: *Professional Red Teaming*. Springer, 2019. – ISBN 978–1–4842–4309–1, S. 15–28

[Son18] SonicWall Inc.: *2018 SonicWall Cyber Threat Report*. 2018

[Uni18] United States Computer Emergency Readiness Team: *Alert (TA17-132A) Indicators Associated With WannaCry Ransomware*. `https://www.us-cert.gov/ncas/alerts/TA17-132A`. Version: 2018, Abruf: 17.09.2019

[Vig13] Viggiani, Fabio: *Design and implementation of a non-aggressive automated penetration testing tool*, NTNU Trondheim, Masterarbeit, 2013

[WY17] Wang, Yien ; Yang, Jianhua: Ethical Hacking and Network Defense: Choose Your Best Network Vulnerability Scanning Tool. In: *31st International Conference on Advanced Information Networking and Applications Workshops* (2017)

Anhang

A. Input-Format eines Schwachstellenscanners

Die Anwendung kann Dateien im CSV-Format einlesen, solange die einzelnen Spalten mit einem Komma getrennt sind. Die in Tabelle A.1 dargestellten Spalten müssen zwingendermaßen vorhanden sein, gegebenenfalls weitere vorhandene Spalten werden ignoriert.

Spaltenname	**Beschreibung**
IP	IPv4-Adresse des gescannten Hosts
Port	Vom verwundbaren Dienst verwendeter Port
Protocol	Verwendetes Protokoll
Name	Bezeichnung der beschriebenen Schwachstelle
CVSS	CVSS-Score des Eintrages
Solution Type	Lösungsvorschlag des Scanners zur Behebung der Lücke
Detection Method	Verwendete Methode zur Entdeckung der Schwachstelle
CVEs	CVE-Nummer(n) der Schwachstelle. 'NOCVE' für keine Angabe, mehrere Nummern mit Komma separiert
BIDs	Bugtraq-ID(s) der Schwachstelle. Leeres Feld für keine Angabe, mehrere Nummern mit Komma separiert

Abbildung A.1.: Obligatorische Spalten des Input-Formats für den einzulesenden Report eines Schwachstellenscanners

Ein Report mit allen obligatorischen Spalten und einem beispielhaften Eintrag kann im für die Reports vorgesehenen Verzeichnis `vuln_reports` gefunden werden.

B. Input-Format einer Exploit-Klassifikation

Auch die einzulesende Exploit-Klassifikation muss im CSV-Format vorliegen.

Spaltenname	**Beschreibung**
Exploit	Vollständiger Pfad des Exploits (wie er im MSF verwendet wird)
Classification	Klassifikation des Exploits; muss entweder `not_intrusive`, `intrusive` oder `intrusive_destructive` sein.

Abbildung B.1.: Obligatorische Spalten des Input-Formats für die einzulesende Exploit-Klassifikation

Eine beispielhafte Datei einer Exploit-Klassifikation mit mehreren Einträgen kann im für diese Dateien vorgesehenen Verzeichnis `exploit_classification` gefunden werden.

C. Report-Format von pyperpwn

Der Report der Anwendung wird im verbreiteten CSV-Format am gewünschten Pfad gespeichert und enthält alle in Tabelle C.1 dargestellten Spalten. Eine Erweiterung des Reports ist durch die Anpassung der verwendeten Datenklassen einfach zu erledigen.

Spaltenname	Beschreibung
Entry ID	Nummer des Eintrages im Report
Vuln ID	Interne Nummer der Schwachstelle
Timestamp	Zeitpunkt des Ausführungsbeginns
Vulnerability Name	Name der überprüften Schwachstelle
Exploit	Pfad des verwendeten Exploits aus dem Metasploit-Framework
CVEs	Zur Schwachstelle gehörende CVE-Nummer(n)
CVSS	CVSS-Score der Schwachstelle
Vulnerability Type	Typ der Schwachstelle laut cvedetails.com
Vulnerability Classification	Kategorisierung der Schwachstelle
Executed	Angabe, ob die Exploit-Ausführung begonnen wurde
Exploit Classification	Kategorisierung des Exploits
Execution Result	Ergebnis der Exploit-Ausführung
Execution Detection Method	Verwendete Methode zur Erkennung der Exploit-Ausführung
Execution Impact	Detaillierte detektierte Auswirkung des Exploits
Session Info	Über die erlangte Session gesammelte Informationen
Success	Angabe, ob der Exploit erfolgreich war
Output	Output der Exploit-Ausführung
Solution Type	Vom Schwachstellenscanner vorgeschlagene Möglichkeit zur Behebung der Schwachstelle

Abbildung C.1.: Übersicht der Spalten des von pyperpwn automatisch generierten Reports

D. Report-Format von AutoSploit

Spaltenname	Beschreibung
Target Host	IP-Adresse des Targets
Date (UTC)	Zeitpunkt der Exploit-Ausführung
MSF Module	Pfad des ausgeführten Exploit-Moduls
LocalHost	IP-Adresse des Hosts
Listening Port	Auf dem Host benutzter Port für Handler
Successful Logs	Erfolgreich Logeinträge der Exploit-Ausführung
Failure Logs	Negative Logeinträge er Exploit-Ausführung
All Logs	Gesamter Ausführungslog

Abbildung D.1.: Übersicht der Spalten des von AutoSploit generierten Reports